IBM MQ

Reference

Version 9 Release 0

Note
Before using this information and the product it supports, read the information in "Notices" on page 5295.

This edition applies to version 9 release 0 of IBM MQ and to all subsequent releases and modifications until otherwise indicated in new editions.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

IBM MQ Reference

Use the reference information in this section to accomplish the tasks that address your business needs.

- Syntax diagrams

Configuration reference

Use the reference information in this section to help you configure IBM® MQ.

The configuration reference information is provided in the following subtopics:

Related information:

Configuring

z/OS Configuring z/OS®

Example IBM MQ configuration for all platforms

The configuration examples describe tasks performed to establish a working IBM MQ network. The tasks are to establish IBM MQ sender and receiver channels to enable bidirectional message flow between the platforms over all supported protocols.

To use channel types other than sender-receiver, see the DEFINE CHANNEL command.

Figure 1 is a conceptual representation of a single channel and the IBM MQ objects associated with it.

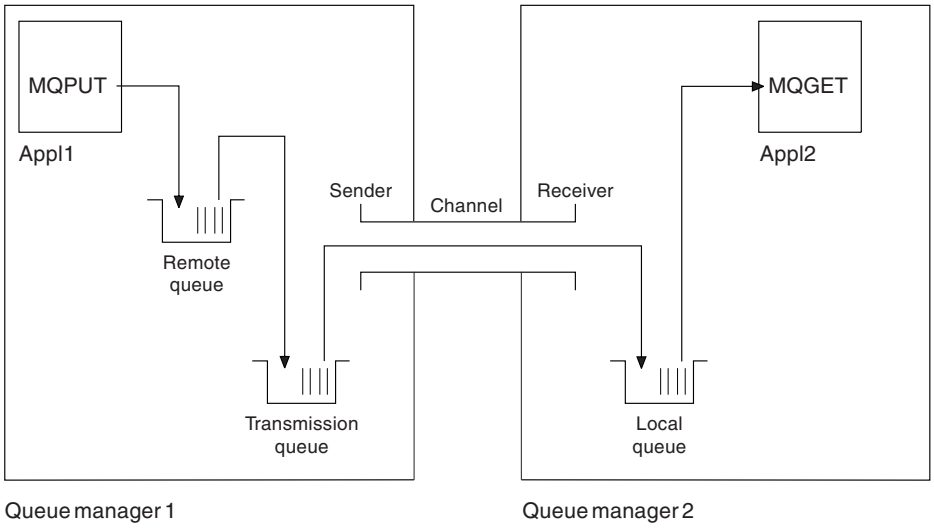


Figure 1. IBM MQ channel to be set up in the example configuration

This example is a simple one, intended to introduce only the basic elements of the IBM MQ network. It does not demonstrate the use of triggering which is described in Triggering channels.

The objects in this network are:

- A remote queue
- A transmission queue
- A local queue
- A sender channel

- A receiver channel

Appl1 and Appl2 are both application programs; Appl1 is putting messages and Appl2 is receiving them.

Appl1 puts messages to a remote queue. The definition for this remote queue specifies the name of a target queue manager, a local queue on that queue manager, and a transmission queue on this local queue manager.

When the queue manager receives the request from Appl1 to put a message to the remote queue, the queue manager determines from the queue definition that the destination is remote. It therefore puts the message, along with a transmission header, straight onto the transmission queue specified in the definition. The message remains on the transmission queue until the channel becomes available, which might happen immediately.

A sender channel has in its definition a reference to one, and one only, transmission queue. When a channel is started, and at other times during its normal operation, it looks at this transmission queue and send any messages on it to the target system. The message has in its transmission header details of the destination queue and queue manager.

The intercommunication examples describe in detail the creation of each of the preceding objects described, for various platform combinations.

On the target queue manager, definitions are required for the local queue and the receiver side of the channel. These objects operate independently of each other and so can be created in any sequence.

On the local queue manager, definitions are required for the remote queue, the transmission queue, and the sender side of the channel. Since both the remote queue definition and the channel definition refer to the transmission queue name, it is advisable to create the transmission queue first.

Network infrastructure in the example

The configuration examples assume that particular network infrastructures are in place for particular platforms:

- z/OS z/OS communicates by using a 3745 network controller (or equivalent) that is attached to a token ring
- Solaris Solaris is on an adjacent local area network (LAN) also attached to a 3745 network controller (or equivalent)
- All other platforms are connected to a token-ring network

It is also assumed that, for SNA, all the required definitions in VTAM and network control program (NCP) are in place and activated for the LAN-attached platforms to communicate over the wide area network (WAN).

Similarly, for TCP, it is assumed that name server function is available, either by using a domain name server or by using locally held tables (for example a host file).

Communications software in the example

Working configurations are given in the examples for the following network software products:

- SNA
  - IBM Personal Communications for Windows V5.9
  - IBM Communications Server for AIX®, V6.3
  - Hewlett-Packard SNAplus2

  - IBM i
  - Data Connection SNAP-IX Version 7 or later
  - OS/390® Version 2 Release 4
- TCP
  - Microsoft Windows
  - AIX Version 4 Release 1.4
  - HP-UX Version 10.2 or later
  - Sun Solaris Release 2.4 or later
  - IBM i
  - TCP for z/OS
  - HP Tru64 UNIX
- NetBIOS
- SPX

Related information:

Configuring

z/OS Configuring z/OS

How to use the communication examples

The example-configurations describe the tasks that are carried out on a single platform to set up communication to another of the platforms. Then they describe the tasks to establish a working channel to that platform.

Wherever possible, the intention is to make the information as generic as possible. Thus, to connect any two queue managers on different platforms, you need to refer to only the relevant two sections. Any deviations or special cases are highlighted as such. You can also connect two queue managers running on the same platform (on different machines or on the same machine). In this case, all the information can be derived from the one section.

ULW On UNIX, Linux, and Windows, before you begin to follow the instructions for your platform you must set various environment variables. Do this by entering one of the following commands:

- Windows On Windows:

  ```
  MQ_INSTALLATION_PATH/bin/setmqenv
  ```

  where *MQ_INSTALLATION_PATH* refers to the location where IBM MQ is installed.

- UNIX Linux On UNIX and Linux systems:

  ```
  . MQ_INSTALLATION_PATH/bin/setmqenv
  ```

  where *MQ_INSTALLATION_PATH* refers to the location where IBM MQ is installed. This command sets the environment variables for the shell you are currently working in. If you open another shell, you must enter the command again.

There are worksheets in which you can find the parameters used in the example configurations. There is a short description of each parameter and some guidance on where to find the equivalent values in your system. When you have a set of values of your own, record these values in the spaces on the worksheet. As you proceed through the section, you will find cross-references to these values as you need them.

The examples do not cover how to set up communications where clustering is being used. For information about setting up communications while using clustering, see Configuring a queue manager cluster. The communication configuration values given here still apply.

There are example configurations for the following platforms:

- AIX "Example IBM MQ configuration on AIX"
- HP-UX "Example IBM MQ configuration for HP-UX" on page 11
- IBM i "Example MQ configuration for IBM i" on page 17
- Linux "Example MQ configuration for Linux" on page 35
- Solaris "Example MQ configuration for Solaris" on page 41
- Windows "Example IBM MQ configuration for Windows" on page 47
- z/OS "Example MQ configuration for z/OS" on page 56
- z/OS "Example MQ configuration for z/OS using QSGs" on page 61
- z/OS "Example MQ configuration for z/OS using intra-group queuing" on page 68

IT responsibilities

To understand the terminology used in the examples, consider the following guidelines as a starting point.

- System administrator: The person (or group of people) who installs and configures the software for a specific platform.
- Network administrator: The person who controls LAN connectivity, LAN address assignments, network naming conventions, and other network tasks. This person can be in a separate group or can be part of the system administration group.

  In most z/OS installations, there is a group responsible for updating the ACF/VTAM, ACF/NCP, and TCP/IP software to support the network configuration. The people in this group are the main source of information needed when connecting any IBM MQ platform to IBM MQ for z/OS. They can also influence or mandate network naming conventions on LANs and you must verify their span of control before creating your definitions.
- A specific type of administrator, for example CICS® administrator, is indicated in cases where we can more clearly describe the responsibilities of the person.

The example-configuration sections do not attempt to indicate who is responsible for and able to set each parameter. In general, several different people might be involved.

Related concepts:

"Example IBM MQ configuration for all platforms" on page 1
The configuration examples describe tasks performed to establish a working IBM MQ network. The tasks are to establish IBM MQ sender and receiver channels to enable bidirectional message flow between the platforms over all supported protocols.

Related information:

setmqenv
Use the **setmqenv** command to set up the IBM MQ environment on UNIX, Linux, and Windows.

Example IBM MQ configuration on AIX

AIX

This section gives an example of how to set up communication links from IBM MQ for AIX to IBM MQ products.

The following platforms are covered in the examples:

- Windows Windows

- UNIX HP Tru64 UNIX
- HP-UX HP-UX
- Solaris Solaris
- Linux Linux
- IBM i IBM i
- z/OS z/OS
- VSE/ESA

See "Example IBM MQ configuration for all platforms" on page 1 for background information about this section and how to use it.

Establishing an LU 6.2 connection: AIX

Describes the parameters needed for an LU 6.2 connection.

For the latest information about configuring SNA over TCP/IP, refer to the following online IBM documentation: Communications Server for AIX.

Establishing a TCP connection: AIX

The listener must be started explicitly before any channels are started. It enables receiving channels to start automatically in response to a request from an inbound sending channel.

The IBM MQ command used to start the IBM MQ for TCP listener is:

```
runmqlsr -t tcp
```

Alternatively, if you want to use the UNIX supplied TCP/IP listener, complete the following steps:

1. Edit the file /etc/services.

   Note: To edit the /etc/services file, you must be logged in as a superuser or root. If you do not have the following line in that file, add it as shown:

   ```
   MQSeries       1414/tcp      # MQSeries channel listener
   ```

2. Edit the file /etc/inetd.conf. If you do not have the following line in that file, add it as shown, replacing *MQ_INSTALLATION_PATH* with the high-level directory in which IBM MQ is installed:

   ```
   MQSeries stream tcp nowait root MQ_INSTALLATION_PATH/bin/amqcrsta amqcrsta
   [-m queue.manager.name]
   ```

3. Enter the command `refresh -s inetd`.

Note: You must add **root** to the mqm group. You need not have the primary group set to mqm. As long as mqm is in the set of groups, you can use the commands. If you are running only applications that use the queue manager you do not need mqm group authority.

What next?

The connection is now established. You are ready to complete the configuration. Go to "IBM MQ for AIX configuration" on page 6.

IBM MQ for AIX configuration: AIX

Defining channels to complete the configuration.

Note:

1. Before beginning the installation process ensure that you have first created the *mqm* user and group, and set the password.
2. If installation fails as a result of insufficient space in the file system you can increase the size as follows, using the command `smit C sna`. (Use `df` to display the status of the file system. This indicates the logical volume that is full.)

   ```
   -- Physical and Logical Storage
     -- File Systems
       -- Add / Change / Show / Delete File Systems
         -- Journaled File Systems
           -- Change/Show Characteristics of a Journaled File System
   ```

3. Start any channel using the command:

   ```
   runmqchl -c channel.name
   ```

4. Sample programs are installed in *MQ_INSTALLATION_PATH*/samp, where *MQ_INSTALLATION_PATH* represents the high-level directory in which IBM MQ is installed.
5. Error logs are stored in /var/mqm/qmgrs/ *qmgrname* /errors.
6. On AIX, you can start a trace of the IBM MQ components by using standard IBM MQ trace commands, or using AIX system trace. See Using trace for more information about IBM MQ Trace and AIX system trace.
7. When you are using the command interpreter **runmqsc** to enter administration commands, a + at the end of a line indicates that the next line is a continuation. Ensure that there is a space between the last parameter and the continuation character.

Basic configuration

1. Create the queue manager from the AIX command line using the command:

   ```
   crtmqm -u dlqname -q aix
   ```

   where:

   aix Is the name of the queue manager

   -q Indicates that this is to become the default queue manager

   -u *dlqname*
   Specifies the name of the undeliverable message queue

   This command creates a queue manager and a set of default objects.
2. Start the queue manager from the AIX command line using the command:

   ```
   strmqm aix
   ```

   where *aix* is the name given to the queue manager when it was created.
3. Start **runmqsc** from the AIX command line and use it to create the undeliverable message queue by entering the command:

   ```
   def ql (dlqname)
   ```

   where *dlqname* is the name given to the undeliverable message queue when the queue manager was created.

Channel configuration for AIX: 

Includes information about configuring a queue manager for a given channel and platform.

The following section details the configuration to be performed on the AIX queue manager to implement the channel described in "Example IBM MQ configuration for all platforms" on page 1.

In each case the MQSC command is shown. Either start **runmqsc** from an AIX command line and enter each command in turn, or build the commands into a command file.

Examples are given for connecting IBM MQ for AIX and IBM MQ for Windows. To connect to IBM MQ on another platform use the appropriate set of values from the table in place of those for Windows.

Note: The words in **bold** are user-specified and reflect the names of IBM MQ objects used throughout these examples. If you change the names used here, ensure that you also change the other references made to these objects throughout this section. All others are keywords and should be entered as shown.

Table 1. Configuration worksheet for IBM MQ for AIX

ID	Parameter Name	Reference	Example Used	User Value
Definition for local node				
A	Queue Manager Name		**AIX**	
B	Local queue name		**AIX.LOCALQ**	
Connection to IBM MQ for Windows The values in this section of the table must match those used in "Channel configuration for Windows" on page 50, as indicated.				
C	Remote queue manager name	A	**WINNT**	
D	Remote queue name		**WINNT.REMOTEQ**	
E	Queue name at remote system	B	**WINNT.LOCALQ**	
F	Transmission queue name		**WINNT**	
G	Sender (SNA) channel name		**AIX.WINNT.SNA**	
H	Sender (TCP/IP) channel name		**AIX.WINNT.TCP**	
I	Receiver (SNA) channel name	G	**WINNT.AIX.SNA**	
J	Receiver (TCP) channel name	H	**WINNT.AIX.TCP**	
Connection to IBM MQ for HP Tru64 UNIX The values in this section of the table must match those used in your HP Tru64 UNIX system.				
C	Remote queue manager name	A	**DECUX**	
D	Remote queue name		**DECUX.REMOTEQ**	
E	Queue name at remote system	B	**DECUX.LOCALQ**	
F	Transmission queue name		**DECUX**	
H	Sender (TCP) channel name		**DECUX.AIX.TCP**	
J	Receiver (TCP) channel name	H	**AIX.DECUX.TCP**	
Connection to IBM MQ for HP-UX The values in this section of the table must match those used in "Channel configuration for HP-UX" on page 13, as indicated.				
C	Remote queue manager name	A	**HPUX**	
D	Remote queue name		**HPUX.REMOTEQ**	

Table 1. Configuration worksheet for IBM MQ for AIX (continued)

ID	Parameter Name	Reference	Example Used	User Value
E	Queue name at remote system	B	**HPUX.LOCALQ**	
F	Transmission queue name		**HPUX**	
G	Sender (SNA) channel name		**AIX.HPUX.SNA**	
H	Sender (TCP) channel name		**AIX.HPUX.TCP**	
I	Receiver (SNA) channel name	G	**HPUX.AIX.SNA**	
J	Receiver (TCP) channel name	H	**HPUX.AIX.TCP**	
Connection to IBM MQ for Solaris The values in this section of the table must match those used in "Channel configuration for Solaris" on page 43, as indicated.				
C	Remote queue manager name	A	**SOLARIS**	
D	Remote queue name		**SOLARIS.REMOTEQ**	
E	Queue name at remote system	B	**SOLARIS.LOCALQ**	
F	Transmission queue name		**SOLARIS**	
G	Sender (SNA) channel name		**AIX.SOLARIS.SNA**	
H	Sender (TCP/IP) channel name		**AIX.SOLARIS.TCP**	
I	Receiver (SNA) channel name	G	**SOLARIS.AIX.SNA**	
J	Receiver (TCP/IP) channel name	H	**SOLARIS.AIX.TCP**	
Connection to IBM MQ for Linux The values in this section of the table must match those used in "Channel configuration for Linux" on page 38, as indicated.				
C	Remote queue manager name	A	**LINUX**	
D	Remote queue name		**LINUX.REMOTEQ**	
E	Queue name at remote system	B	**LINUX.LOCALQ**	
F	Transmission queue name		**LINUX**	
G	Sender (SNA) channel name		**AIX.LINUX.SNA**	
H	Sender (TCP/IP) channel name		**AIX.LINUX.TCP**	
I	Receiver (SNA) channel name	G	**LINUX.AIX.SNA**	
J	Receiver (TCP/IP) channel name	H	**LINUX.AIX.TCP**	
IBM i ***Connection to IBM MQ for IBM i*** IBM i The values in this section of the table must match those used in "Channel configuration for IBM i" on page 31, as indicated.				
C	Remote queue manager name	A	**AS400**	
D	Remote queue name		**AS400.REMOTEQ**	
E	Queue name at remote system	B	**AS400.LOCALQ**	
F	Transmission queue name		**AS400**	
G	Sender (SNA) channel name		**AIX.AS400.SNA**	
H	Sender (TCP) channel name		**AIX.AS400.TCP**	
I	Receiver (SNA) channel name	G	**AS400.AIX.SNA**	
IBM i J	Receiver (TCP) channel name	H	**AS400.AIX.TCP**	

Table 1. Configuration worksheet for IBM MQ for AIX (continued)

ID	Parameter Name	Reference	Example Used	User Value
z/OS ***Connection to IBM MQ for z/OS*** z/OS The values in this section of the table must match those used in "Channel configuration for z/OS" on page 57, as indicated.				
C	Remote queue manager name	A	**MVS**	
D	Remote queue name		**MVS.REMOTEQ**	
E	Queue name at remote system	B	**MVS.LOCALQ**	
F	Transmission queue name		**MVS**	
G	Sender (SNA) channel name		**AIX.MVS.SNA**	
H	Sender (TCP) channel name		**AIX.MVS.TCP**	
I	Receiver (SNA) channel name	G	**MVS.AIX.SNA**	
z/OS J	Receiver (TCP) channel name	H	**MVS.AIX.TCP**	
z/OS ***Connection to IBM MQ for z/OS using queue-sharing groups*** z/OS The values in this section of the table must match those used in "Shared channel configuration example" on page 66, as indicated.				
C	Remote queue manager name	A	**QSG**	
D	Remote queue name		**QSG.REMOTEQ**	
E	Queue name at remote system	B	**QSG.SHAREDQ**	
F	Transmission queue name		**QSG**	
G	Sender (SNA) channel name		**AIX.QSG.SNA**	
H	Sender (TCP) channel name		**AIX.QSG.TCP**	
I	Receiver (SNA) channel name	G	**QSG.AIX.SNA**	
z/OS J	Receiver (TCP) channel name	H	**QSG.AIX.TCP**	
Connection to MQSeries® for VSE/ESA The values in this section of the table must match those used in your VSE/ESA system.				
C	Remote queue manager name	A	**VSE**	
D	Remote queue name		**VSE.REMOTEQ**	
E	Queue name at remote system	B	**VSE.LOCALQ**	
F	Transmission queue name		**VSE**	
G	Sender channel name		**AIX.VSE.SNA**	
I	Receiver channel name	G	**VSE.AIX.SNA**	

IBM MQ for AIX sender-channel definitions using SNA: AIX

Example commands.

```
def ql (WINNT) +                              F
    usage(xmitq) +
    replace

def qr (WINNT.REMOTEQ) +                      D
    rname(WINNT.LOCALQ) +                     E
    rqmname(WINNT) +                          C
    xmitq(WINNT) +                            F
    replace

def chl (AIX.WINNT.SNA) chltype(sdr) +        G
    trptype(lu62) +
    conname('WINNTCPIC') +                    17
    xmitq(WINNT) +                            F
    replace
```

IBM MQ for AIX receiver-channel definitions using SNA: AIX

Example commands.

```
def ql (AIX.LOCALQ) replace                 B

def chl (WINNT.AIX.SNA) chltype(rcvr) +     I
    trptype(lu62) +
    replace
```

IBM MQ for AIX TPN setup: AIX

Alternative ways of ensuring that SNA receiver channels activate correctly when a sender channel initiates a conversation.

During the AIX Communications Server configuration process, an LU 6.2 TPN profile was created, which contained the full path to a TP executable program. In the example, the file was called u/interops/AIX.crs6a. You can choose a name, but consider including the name of your queue manager in it. The contents of the executable file must be:

```
#!/bin/sh
MQ_INSTALLATION_PATH/bin/amqcrs6a -m aix
```

where *aix* is the queue manager name (A) and *MQ_INSTALLATION_PATH* is the high-level directory in which IBM MQ is installed. After creating this file, enable it for execution by running the command:

```
chmod 755 /u/interops/AIX.crs6a
```

As an alternative to creating an executable file, you can specify the path on the Add LU 6.2 TPN Profile panel, using command-line parameters.

Specifying a path in one of these two ways ensures that SNA receiver channels activate correctly when a sender channel initiates a conversation.

IBM MQ for AIX sender-channel definitions using TCP: AIX

Example commands.

```
def ql (WINNT) +                              F
    usage(xmitq) +
    replace

def qr (WINNT.REMOTEQ) +                      D
    rname(WINNT.LOCALQ) +                     E
    rqmname(WINNT) +                          C
    xmitq(WINNT) +                            F
    replace

def chl (AIX.WINNT.TCP) chltype(sdr) +        H
    trptype(tcp) +
    conname(remote_tcpip_hostname) +
    xmitq(WINNT) +                            F
    replace
```

IBM MQ for AIX receiver-channel definitions using TCP: AIX

Example commands.

```
def ql (AIX.LOCALQ) replace                   B

def chl (WINNT.AIX.TCP) chltype(rcvr) +       J
    trptype(tcp) +
    replace
```

Example IBM MQ configuration for HP-UX

HP-UX

This section gives an example of how to set up communication links from IBM MQ for HP-UX to IBM MQ products.

The following platforms are included:

- Windows
- AIX
- HP Tru64 UNIX
- Solaris
- Linux
- IBM i
- z/OS
- VSE/ESA

See “Example IBM MQ configuration for all platforms” on page 1 for background information about this section and how to use it.

Establishing an LU 6.2 connection: HP-UX

Describes the parameters needed for an LU 6.2 connection

For the latest information about configuring SNA over TCP/IP, refer to the following online IBM documentation: Communications Server, and the following online HP documentation: HP-UX SNAplus2 Installation Guide.

Establishing a TCP connection: HP-UX

Alternative ways of establishing a connection and next steps.

The listener must be started explicitly before any channels are started. It enables receiving channels to start automatically in response to a request from an inbound sending channel.

Alternatively, if you want to use the UNIX supplied TCP/IP listener, complete the following steps:

1. Edit the file /etc/services.

   Note: To edit the /etc/services file, you must be logged in as a superuser or root. If you do not have the following line in that file, add it as shown:

   ```
   MQSeries       1414/tcp      # MQSeries channel listener
   ```

2. Edit the file /etc/inetd.conf. If you do not have the following line in that file, add it as shown, replacing *MQ_INSTALLATION_PATH* with the high-level directory in which IBM MQ is installed.

   ```
   MQSeries stream tcp nowait root MQ_INSTALLATION_PATH/bin/amqcrsta amqcrsta
   [-m queue.manager.name]
   ```

3. Find the process ID of the inetd with the command:

   ```
   ps -ef | grep inetd
   ```

4. Run the command:

   ```
   kill -1 inetd processid
   ```

Note: You must add **root** to the mqm group. You do not need not have the primary group set to mqm. As long as mqm is in the set of groups, you can use the commands. If you are running only applications that use the queue manager you do not need to have mqm group authority.

What next?

The connection is now established. You are ready to complete the configuration. Go to "IBM MQ for HP-UX configuration" on page 13.

IBM MQ for HP-UX configuration: HP-UX

Describes defining the channels to complete the configuration.

Before beginning the installation process ensure that you have first created the *mqm* user and group, and set the password.

Start any channel using the command:

```
runmqchl -c channel.name
```

Note:

1. Sample programs are installed in *MQ_INSTALLATION_PATH*/samp, where *MQ_INSTALLATION_PATH* represents the high-level directory in which IBM MQ is installed.
2. Error logs are stored in /var/mqm/qmgrs/ *qmgrname* /errors.
3. When you are using the command interpreter **runmqsc** to enter administration commands, a + at the end of a line indicates that the next line is a continuation. Ensure that there is a space between the last parameter and the continuation character.

Basic configuration

1. Create the queue manager from the UNIX prompt using the command:

   ```
   crtmqm -u dlqname -q hpux
   ```

   where:

   hpux Is the name of the queue manager

   -q Indicates that this is to become the default queue manager

   -u *dlqname*
   Specifies the name of the undeliverable message queue

   This command creates a queue manager and a set of default objects. It sets the DEADQ attribute of the queue manager but does not create the undeliverable message queue.
2. Start the queue manager from the UNIX prompt using the command:

   ```
   strmqm hpux
   ```

   where *hpux* is the name given to the queue manager when it was created.

Channel configuration for HP-UX: HP-UX

Includes information about configuring a queue manager for a given channel and platform.

The following section details the configuration to be performed on the HP-UX queue manager to implement the channel described in “Example IBM MQ configuration for all platforms” on page 1.

In each case the MQSC command is shown. Either start **runmqsc** from a UNIX prompt and enter each command in turn, or build the commands into a command file.

Examples are given for connecting IBM MQ for HP-UX and IBM MQ for Windows. To connect to IBM MQ on another platform use the appropriate set of values from the table in place of those for Windows.

Note: The words in **bold** are user-specified and reflect the names of IBM MQ objects used throughout these examples. If you change the names used here, ensure that you also change the other references made to these objects throughout this section. All others are keywords and should be entered as shown.

Table 2. Configuration worksheet for IBM MQ for HP-UX

ID	Parameter Name	Reference	Example Used	User Value
Definition for local node				
A	Queue Manager Name		**HPUX**	
B	Local queue name		**HPUX.LOCALQ**	
Connection to IBM MQ for Windows The values in this section of the table must match those used in "Channel configuration for Windows" on page 50, as indicated.				
C	Remote queue manager name	A	**WINNT**	
D	Remote queue name		**WINNT.REMOTEQ**	
E	Queue name at remote system	B	**WINNT.LOCALQ**	
F	Transmission queue name		**WINNT**	
G	Sender (SNA) channel name		**HPUX.WINNT.SNA**	
H	Sender (TCP/IP) channel name		**HPUX.WINNT.TCP**	
I	Receiver (SNA) channel name	G	**WINNT.HPUX.SNA**	
J	Receiver (TCP) channel name	H	**WINNT.HPUX.TCP**	
Connection to IBM MQ for AIX The values in this section of the table must match those used in Table 1 on page 7, as indicated.				
C	Remote queue manager name	A	**AIX**	
D	Remote queue name		**AIX.REMOTEQ**	
E	Queue name at remote system	B	**AIX.LOCALQ**	
F	Transmission queue name		**AIX**	
G	Sender (SNA) channel name		**HPUX.AIX.SNA**	
H	Sender (TCP) channel name		**HPUX.AIX.TCP**	
I	Receiver (SNA) channel name	G	**AIX.HPUX.SNA**	
J	Receiver (TCP) channel name	H	**AIX.HPUX.TCP**	
Connection to IBM MQ for HP Tru64 UNIX The values in this section of the table must match those used in your HP Tru64 UNIX system.				
C	Remote queue manager name	A	**DECUX**	
D	Remote queue name		**DECUX.REMOTEQ**	
E	Queue name at remote system	B	**DECUX.LOCALQ**	
F	Transmission queue name		**DECUX**	
H	Sender (TCP) channel name		**DECUX.HPUX.TCP**	
J	Receiver (TCP) channel name	H	**HPUX.DECUX.TCP**	
Connection to IBM MQ for Solaris The values in this section of the table must match those used in Table 6 on page 43, as indicated.				
C	Remote queue manager name	A	**SOLARIS**	
D	Remote queue name		**SOLARIS.REMOTEQ**	
E	Queue name at remote system	B	**SOLARIS.LOCALQ**	
F	Transmission queue name		**SOLARIS**	
G	Sender (SNA) channel name		**HPUX.SOLARIS.SNA**	

Table 2. Configuration worksheet for IBM MQ for HP-UX (continued)

ID	Parameter Name	Reference	Example Used	User Value
H	Sender (TCP/IP) channel name		**HPUX.SOLARIS.TCP**	
I	Receiver (SNA) channel name	G	**SOLARIS.HPUX.SNA**	
J	Receiver (TCP/IP) channel name	H	**SOLARIS.HPUX.TCP**	
Connection to IBM MQ for Linux The values in this section of the table must match those used in Table 5 on page 38, as indicated.				
C	Remote queue manager name	A	**LINUX**	
D	Remote queue name		**LINUX.REMOTEQ**	
E	Queue name at remote system	B	**LINUX.LOCALQ**	
F	Transmission queue name		**LINUX**	
G	Sender (SNA) channel name		**HPUX.LINUX.SNA**	
H	Sender (TCP/IP) channel name		**HPUX.LINUX.TCP**	
I	Receiver (SNA) channel name	G	**LINUX.HPUX.SNA**	
J	Receiver (TCP/IP) channel name	H	**LINUX.HPUX.TCP**	
IBM i ***Connection to IBM MQ for IBM i*** IBM i The values in this section of the table must match those used in Table 4 on page 31, as indicated.				
C	Remote queue manager name	A	**AS400**	
D	Remote queue name		**AS400.REMOTEQ**	
E	Queue name at remote system	B	**AS400.LOCALQ**	
F	Transmission queue name		**AS400**	
G	Sender (SNA) channel name		**HPUX.AS400.SNA**	
H	Sender (TCP/IP) channel name		**HPUX.AS400.TCP**	
I	Receiver (SNA) channel name	G	**AS400.HPUX.SNA**	
IBM i J	Receiver (TCP) channel name	H	**AS400.HPUX.TCP**	
z/OS ***Connection to IBM MQ for z/OS*** z/OS The values in this section of the table must match those used in Table 8 on page 57, as indicated.				
C	Remote queue manager name	A	**MVS**	
D	Remote queue name		**MVS.REMOTEQ**	
E	Queue name at remote system	B	**MVS.LOCALQ**	
F	Transmission queue name		**MVS**	
G	Sender (SNA) channel name		**HPUX.MVS.SNA**	
H	Sender (TCP) channel name		**HPUX.MVS.TCP**	
I	Receiver (SNA) channel name	G	**MVS.HPUX.SNA**	
z/OS J	Receiver (TCP) channel name	H	**MVS.HPUX.TCP**	
Connection to MQSeries for VSE/ESA The values in this section of the table must match those used in your VSE/ESA system.				
C	Remote queue manager name	A	**VSE**	
D	Remote queue name		**VSE.REMOTEQ**	

Table 2. Configuration worksheet for IBM MQ for HP-UX (continued)

ID	Parameter Name	Reference	Example Used	User Value
E	Queue name at remote system	B	**VSE.LOCALQ**	
F	Transmission queue name		**VSE**	
G	Sender channel name		**HPUX.VSE.SNA**	
I	Receiver channel name	G	**VSE.HPUX.SNA**	

IBM MQ for HP-UX sender-channel definitions using SNA: HP-UX

Example commands.

```
def ql (WINNT) +                                F
    usage(xmitq) +
    replace

def qr (WINNT.REMOTEQ) +                        D
    rname(WINNT.LOCALQ) +                       E
    rqmname(WINNT) +                            C
    xmitq(WINNT) +                              F
    replace

def chl (HPUX.WINNT.SNA) chltype(sdr) +         G
    trptype(lu62) +
    conname('WINNTCPIC') +                      16
    xmitq(WINNT) +                              F
    replace
```

IBM MQ for HP-UX receiver-channel definitions using SNA: HP-UX

Example commands.

```
def ql (HPUX.LOCALQ) replace                  B

def chl (WINNT.HPUX.SNA) chltype(rcvr) +      I
    trptype(lu62) +
    replace
```

IBM MQ for HP-UX invokable TP setup: HP-UX

Ensuring that SNA receiver channels activate correctly when a sender channel initiates a conversation.

This is not required for HP SNAplus2 Release 6.

During the HP SNAplus2 configuration process, you created an invokable TP definition, which points to an executable file. In the example, the file was called /users/interops/HPUX.crs6a. You can choose what you call this file, but consider including the name of your queue manager in the name. The contents of the executable file must be:

```
#!/bin/sh
MQ_INSTALLATION_PATH/bin/amqcrs6a -m hpux
```

where *hpux* is the name of your queue manager A and *MQ_INSTALLATION_PATH* is the high-level directory in which IBM MQ is installed.

This ensures that SNA receiver channels activate correctly when a sender channel initiates a conversation.

IBM MQ for HP-UX sender-channel definitions using TCP: HP-UX

Example commands.

```
def ql (WINNT) +                                F
    usage(xmitq) +
    replace

def qr (WINNT.REMOTEQ) +                        D
    rname(WINNT.LOCALQ) +                       E
    rqmname(WINNT) +                            C
    xmitq(WINNT) +                              F
    replace

def chl (HPUX.WINNT.TCP) chltype(sdr) +         H
    trptype(tcp) +
    conname(remote_tcpip_hostname) +
    xmitq(WINNT) +                              F
    replace
```

IBM MQ for HP-UX receiver-channel definitions using TCP/IP: HP-UX

Example commands.

```
def ql (HPUX.LOCALQ) replace                  B

def chl (WINNT.HPUX.TCP) chltype(rcvr) +      J
    trptype(tcp) +
    replace
```

Example MQ configuration for IBM i

IBM i

This section gives an example of how to set up communication links from IBM MQ for IBM i to IBM MQ products on other platforms.

Other platforms covered are the following platforms:

- Windows
- AIX
- Compaq Tru64 UNIX
- HP-UX
- Solaris
- Linux
- z/OS or MVS
- VSE/ESA

See “Example IBM MQ configuration for all platforms” on page 1 for background information about this section and how to use it.

Configuration parameters for an LU 6.2 connection: IBM i

The following worksheet lists all the parameters needed to set up communication from IBM i system to one of the other IBM MQ platforms. The worksheet shows examples of the parameters, which have been tested in a working environment, and leaves space for you to enter your own values.

Use the worksheet in this section to record the values for this configuration. Use the worksheet with the worksheet in the section for the platform to which you are connecting.

Where numbers appear in the Reference column they indicate that the value must match that in the appropriate worksheet elsewhere in this section. The examples that follow in this section refer to the values in the ID column of this table.

The entries in the Parameter Name column are explained in "Explanation of terms" on page 20.

Table 3. Configuration worksheet for SNA on an IBM i system

ID	Parameter Name	Reference	Example Used	User Value
Definition for local node				
1	Local network ID		**NETID**	
2	Local control point name		**AS400PU**	
3	LU name		**AS400LU**	
4	LAN destination address		**10005A5962EF**	
5	Subsystem description		**QCMN**	
6	Line description		**TOKENRINGL**	
7	Resource name		**LIN041**	
8	Local Transaction Program name		**MQSERIES**	
Connection to a Windows system				
9	Network ID	2	**NETID**	
10	Control point name	3	**WINNTCP**	
11	LU name	5	**WINNTLU**	
12	Controller description		**WINNTCP**	
13	Device		**WINNTLU**	
14	Side information		**NTCPIC**	
15	Transaction Program	7	**MQSERIES**	
16	LAN adapter address	9	**08005AA5FAB9**	
17	Mode	17	**#INTER**	
Connection to an AIX system				
9	Network ID	1	**NETID**	
10	Control point name	2	**AIXPU**	
11	LU name	4	**AIXLU**	
12	Controller description		**AIXPU**	
13	Device		**AIXLU**	
14	Side information		**AIXCPIC**	
15	Transaction Program	6	**MQSERIES**	
16	LAN adapter address	8	**123456789012**	
17	Mode	14	**#INTER**	

Table 3. Configuration worksheet for SNA on an IBM i system (continued)

ID	Parameter Name	Reference	Example Used	User Value
Connection to an HP-UX system				
9	Network ID	4	**NETID**	
10	Control point name	2	**HPUXPU**	
11	LU name	5	**HPUXLU**	
12	Controller description		**HPUXPU**	
13	Device		**HPUXLU**	
14	Side information		**HPUXCPIC**	
15	Transaction Program	7	**MQSERIES**	
16	LAN adapter address	8	**100090DC2C7C**	
17	Mode	17	**#INTER**	
Connection to a Solaris system				
9	Network ID	2	**NETID**	
10	Control point name	3	**SOLARPU**	
11	LU name	7	**SOLARLU**	
12	Controller description		**SOLARPU**	
13	Device		**SOLARLU**	
14	Side information		**SOLCPIC**	
15	Transaction Program	8	**MQSERIES**	
16	LAN adapter address	5	**08002071CC8A**	
17	Mode	17	**#INTER**	
Connection to a Linux (x86 platform) system				
9	Network ID	4	**NETID**	
10	Control point name	2	**LINUXPU**	
11	LU name	5	**LINUXLU**	
12	Controller description		**LINUXPU**	
13	Device		**LINUXLU**	
14	Side information		**LXCPIC**	
15	Transaction Program	7	**MQSERIES**	
16	LAN adapter address	8	**08005AC6DF33**	
17	Mode	6	**#INTER**	
Connection to an z/OS system				
9	Network ID	2	**NETID**	
10	Control point name	3	**MVSPU**	
11	LU name	4	**MVSLU**	
12	Controller description		**MVSPU**	
13	Device		**MVSLU**	
14	Side information		**MVSCPIC**	
15	Transaction Program	7	**MQSERIES**	
16	LAN adapter address	8	**400074511092**	
17	Mode	6	**#INTER**	

Table 3. Configuration worksheet for SNA on an IBM i system (continued)

ID	Parameter Name	Reference	Example Used	User Value
Connection to a VSE/ESA system ed				
9	Network ID	1	**NETID**	
10	Control point name	2	**VSEPU**	
11	LU name	3	**VSELU**	
12	Controller description		**VSEPU**	
13	Device		**VSELU**	
14	Side information		**VSECPIC**	
15	Transaction Program	4	**MQ01**	**MQ01**
16	LAN adapter address	5	**400074511092**	
17	Mode		**#INTER**	

Explanation of terms: IBM i

An explanation of the terms used in the configuration worksheet.

1 2 3 See "How to find network attributes" on page 21 for the details of how to find the configured values.

4 LAN destination address
The hardware address of the IBM i system token-ring adapter. You can find the value using the command DSPLIND *Line description* (6).

5 Subsystem description
This parameter is the name of any IBM i subsystem that is active while using the queue manager. The name QCMN has been used because it is the IBM i communications subsystem.

6 Line description
If this parameter has been specified it is indicated in the Description field of the resource Resource name. See "How to find the value of Resource name" on page 21 for details. If the value is not specified you need to create a line description.

7 Resource name
See "How to find the value of Resource name" on page 21 for details of how to find the configured value.

8 Local Transaction Program name
IBM MQ applications trying to converse with this workstation specify a symbolic name for the program to be run at the receiving end. This name is defined on the channel definition at the sender. For simplicity, wherever possible use a transaction program name of MQSERIES, or in the case of a connection to VSE/ESA, where the length is limited to 4 bytes, use MQTP.

See Settings on the local IBM i system for a remote queue manager platform for more information.

12 Controller description
This parameter is an alias for the Control Point name (or Node name) of the partner system. For convenience, we have used the actual name of the partner in this example.

13 Device
This parameter is an alias for the LU of the partner system. For convenience, we have used the LU name of the partner in this example.

14 Side information

This parameter is the name given to the CPI-C side information profile. You specify your own 8-character name.

How to find network attributes: IBM i

The local node has been partially configured as part of the IBM i installation. To display the current network attributes enter the command DSPNETA.

If you need to change these values use the command CHGNETA. An IPL might be required to apply your changes.

```
Display Network Attributes
System:  AS400PU
Current system name . . . . . . . . . . . . . . . :  AS400PU
Pending system name . . . . . . . . . . . . . . . :
Local network ID . . . . . . . . . . . . . . . . . :  NETID
Local control point name . . . . . . . . . . . . . :  AS400PU
Default local location . . . . . . . . . . . . . . :  AS400LU
Default mode . . . . . . . . . . . . . . . . . . . :  BLANK
APPN node type . . . . . . . . . . . . . . . . . . :  *ENDNODE
Data compression . . . . . . . . . . . . . . . . . :  *NONE
Intermediate data compression . . . . . . . . . . :  *NONE
Maximum number of intermediate sessions . . . . . :  200
Route addition resistance . . . . . . . . . . . . :  128
Server network ID/control point name . . . . . . . :  NETID    NETCP




More...
Press Enter to continue.

F3=Exit  F12=Cancel
```

Check that the values for **Local network ID** (1), **Local control point name** (2), and **Default local location** (3), correspond to the values on your worksheet.

How to find the value of Resource name: IBM i

To find the value of resource name, type `WRKHDWRSC TYPE(*CMN)` and press enter.

The Work with Communication Resources panel is displayed. The value for **Resource name** is found as the token-ring Port. It is LIN041 in this example.

```
Work with Communication Resources
System:  AS400PU
Type options, press Enter.
2=Edit  4=Remove  5=Work with configuration description
7=Add configuration description ...

Configuration
Opt Resource     Description  Type Description
CC02              2636 Comm Processor
LIN04            2636 LAN Adapter
LIN041    TOKEN-RING  2636 Token-ring Port









Bottom
F3=Exit   F5=Refresh  F6=Print  F11=Display resource addresses/statuses
F12=Cancel  F23=More options
```

Establishing an LU 6.2 connection: IBM i

This section describes how to establish an LU 6.2 connection.

Local node configuration: IBM i

To configure the local node you need to create a line description and add a routing entry.

Creating a line description

1. If the line description has not already been created use the command CRTLINTRN.
2. Specify values for **Line description** (6) and **Resource name** (7).

```
Create Line Desc (token-ring) (CRTLINTRN)

Type choices, press Enter.

Line description . . . . . .  TOKENRINGL  Name
Resource name . . . . . .  LIN041    Name, *NWID
NWI type . . . . . .  *FR      *FR, *ATM
Online at IPL . . . . . .  *YES     *YES, *NO
Vary on wait . . . . . .  *NOWAIT    *NOWAIT, 15-180 (1 second)
Maximum controllers . . . . . .  40      1-256
Attached NWI . . . . . .  *NONE     Name, *NONE



Bottom
F3=Exit  F4=Prompt  F5=Refresh  F10=Additional parameters  F12=Cancel
F13=How to use this display    F24=More keys
Parameter LIND required.                          +
```

Adding a routing entry

1. Type the command ADDRTGE and press enter.

```
Add Routing Entry (ADDRTGE)

Type choices, press Enter.

Subsystem description . . . . .  QCMN     Name
Library . . . . . .  *LIBL    Name, *LIBL, *CURLIB
Routing entry sequence number .  1       1-9999
Comparison data:
Compare value . . . . . .  'MQSERIES'

Starting position . . . . . .  37      1-80
Program to call . . . . . .  AMQCRC6B   Name, *RTGDTA
Library . . . . . .  QMAS400    Name, * LI BL, *CURLIB
Class . . . . . .  *SBSD     Name, *SBSD
Library . . . . . .  *LIBL     Name, *LIBL, *CURLIB
Maximum active routing steps . .  *NOMAX    0-1000, *NOMAX
Storage pool identifier . . . .  1       1-10



Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
Parameter SBSD required.                          +
```

2. Specify your value for **Subsystem description** (5), and the values shown here for **Routing entry sequence number**, **Compare value** (8), **Starting position**, **Program to call**, and the **Library** containing the program to call.
3. Type the command STRSBS *subsystem description* (5) and press enter.

Connection to partner node: 

To connect to a partner node, you need to: create a controller description, create a device description, create CPI-C side information, add a communications entry for APPC, and add a configuration list entry.

This example is for a connection to a Windows system, but the steps are the same for other nodes.

Creating a controller description

1. At a command-line, type `CRTCTLAPPC` and press enter.

```
Create Ctl Desc (APPC) (CRTCTLAPPC)

Type choices, press Enter.

Controller description . . . . .  WINNTCP     Name
Link type . . . . . .  *LAN     *FAX, *FR, *IDLC,
*LAN...
Online at IPL . . . . . .  *NO      *YES, *NO



Bottom
F3=Exit  F4=Prompt  F5=Refresh  F10=Additional parameters  F12=Cancel
F13=How to use this display    F24=More keys
Parameter CTLD required.                          +
```

2. Specify a value for **Controller description** (12), set **Link type** to `*LAN`, and set **Online at IPL** to `*NO`.
3. Press enter twice, followed by F10.

```
Create Ctl Desc (APPC) (CRTCTLAPPC)

Type choices, press Enter.

Controller description . . . . . > WINNTCP     Name
Link type . . . . . > *LAN     *FAX, *FR, *IDLC, *LAN...
Online at IPL . . . . . > *NO      *YES, *NO
APPN-capable . . . . . *YES     *YES, *NO
Switched line list . . . . . TOKENRINGL  Name
+ for more values
Maximum frame size . . . . . *LINKTYPE   265-16393, 256, 265, 512...
Remote network identifier . . . NETID     Name, *NETATR, *NONE, *ANY
Remote control point . . . . . WINNTCP     Name, *ANY
Exchange identifier . . . . . 00000000-FFFFFFFF
Initial connection . . . . . *DIAL     *DIAL, *ANS
Dial initiation . . . . . *LINKTYPE   *LINKTYPE, *IMMED, *DELAY
LAN remote adapter address . . . 10005AFC5D83 000000000001-FFFFFFFFFFFF
APPN CP session support . . . . *YES     *YES, *NO
APPN node type . . . . . *ENDNODE   *ENDNODE, *LENNODE...
APPN transmission group number   1       1-20, *CALC
More...
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
```

4. Specify values for **Switched line list** (6), **Remote network identifier** (9), **Remote control point** (10), and **LAN remote adapter address** (16).
5. Press enter.

Creating a device description

1. Type the command CRTDEVAPPC and press enter.

```
Create Device Desc (APPC) (CRTDEVAPPC)

Type choices, press Enter.

Device description . . . . . WINNTLU     Name
Remote location . . . . . WINNTLU     Name
Online at IPL . . . . . *YES     *YES, *NO
Local location . . . . . AS400LU    Name, *NETATR
Remote network identifier . . . NETID     Name, *NETATR, *NONE
Attached controller . . . . . WINNTCP     Name
Mode . . . . . *NETATR    Name, *NETATR
+ for more values
Message queue . . . . . QSYSOPR    Name, QSYSOPR
Library . . . . . *LIBL    Name, *LIBL, *CURLIB
APPN-capable . . . . . *YES     *YES, *NO
Single session:
Single session capable . . . . *NO      *NO, *YES
Number of conversations . . .         1-512


Bottom
F3=Exit  F4=Prompt  F5=Refresh  F10=Additional parameters  F12=Cancel
F13=How to use this display    F24=More keys
Parameter DEVD required.                          +
```

2. Specify values for **Device description** (13), **Remote location** (11), **Local location** (3), **Remote network identifier** (9), and **Attached controller** (12).

Note: You can avoid having to create controller and device descriptions manually by taking advantage of the IBM i auto-configuration service. Consult the IBM i documentation for details.

Creating CPI-C side information

1. Type CRTCSI and press F10.

```
Create Comm Side Information (CRTCSI)

Type choices, press Enter.

Side information . . . . . .  NTCPIC    Name
Library . . . . . .  *CURLIB   Name, *CURLIB
Remote location . . . . . .  WINNTLU     Name
Transaction program . . . . . .  MQSERIES

Text 'description' . . . . . .  *BLANK


Additional Parameters

Device . . . . . .  *LOC     Name, *LOC
Local location . . . . . .  AS400LU    Name, *LOC, *NETATR
Mode . . . . . .  #INTER    Name, *NETATR
Remote network identifier . . .  NETID     Name, *LOC, *NETATR, *NONE
Authority . . . . . .  *LIBCRTAUT  Name, *LIBCRTAUT, *CHANGE...

Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
Parameter CSI required.
```

2. Specify values for **Side information** (14), **Remote location** (11), **Transaction program** (15), **Local location** (3), **Mode**, and **Remote network identifier** (9).
3. Press enter.

Adding a communications entry for APPC

1. At a command-line, type ADDCMNE and press enter.

```
Add Communications Entry (ADDCMNE)

Type choices, press Enter.

Subsystem description . . . . .  QCMN     Name
Library . . . . . .  *LIBL    Name, *LIBL, *CURLIB
Device . . . . . .  WINNTLU     Name, generic*, *ALL...
Remote location . . . . . .         Name
Job description . . . . . .  *USRPRF    Name, *USRPRF, *SBSD
Library . . . . . .         Name, *LIBL, *CURLIB
Default user profile . . . . . .  *NONE     Name, *NONE, *SYS
Mode . . . . . .  *ANY     Name, *ANY
Maximum active jobs . . . . . .  *NOMAX    0-1000, *NOMAX



Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
Parameter SBSD required.
```

2. Specify values for **Subsystem description** (5) and **Device** (13), and press enter.

Adding a configuration list entry

1. Type ADDCFGLE *APPNRMT and press F4.

```
Add Configuration List Entries (ADDCFGLE)

Type choices, press Enter.

Configuration list type . . . . > *APPNRMT   *APPNLCL, *APPNRMT...
APPN remote location entry:
Remote location name . . . . .  WINNTLU     Name, generic*, *ANY
Remote network identifier . .  NETID     Name, *NETATR, *NONE
Local location name . . . . .  AS400LU    Name, *NETATR
Remote control point . . . . .  WINNTCP    Name, *NONE
Control point net ID . . . . .  NETID     Name, *NETATR, *NONE
Location password . . . . . .  *NONE
Secure location . . . . . .  *NO      *YES, *NO
Single session . . . . . .  *NO      *YES, *NO
Locally controlled session . .  *NO      *YES, *NO
Pre-established session . . .  *NO      *YES, *NO
Entry 'description' . . . . .  *BLANK
Number of conversations . . .  10      1-512
+ for more values

Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
```

2. Specify values for **Remote location name** (11), **Remote network identifier** (9), **Local location name** (3), **Remote control point** (10), and **Control point net ID** (9).
3. Press enter.

What next?: IBM i

The LU 6.2 connection is now established. You are ready to complete the configuration.

Go to "IBM MQ for IBM i configuration" on page 28.

Establishing a TCP connection: IBM i

If TCP is already configured there are no extra configuration tasks. If TCP/IP is not configured you need to: add a TCP/IP interface, add a TCP/IP loopback interface, and add a default route.

Adding a TCP/IP interface

1. At a command-line, type `ADDTCPIFC` and press enter.

```
Add TCP/IP Interface (ADDTCPIFC)

Type choices, press Enter.

Internet address . . . . . .  19.22.11.55
Line description . . . . . .  TOKENRINGL  Name, *LOOPBACK
Subnet mask . . . . . .  255.255.0.0
Type of service . . . . . .  *NORMAL    *MINDELAY, *MAXTHRPUT..
Maximum transmission unit . . .  *LIND     576-16388, *LIND
Autostart . . . . . .  *YES     *YES, *NO
PVC logical channel identifier          001-FFF
+ for more values
X.25 idle circuit timeout . . .  60      1-600
X.25 maximum virtual circuits .  64      0-64
X.25 DDN interface . . . . . .  *NO      *YES, *NO
TRLAN bit sequencing . . . . . .  *MSB     *MSB, *LSB




Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
```

2. Specify the **IP address** and **Line description**, and a **Subnet mask** of the machine.
3. Press enter.

Adding a TCP/IP loopback interface

1. At a command-line, type `ADDTCPIFC` and press enter.

```
Add TCP Interface (ADDTCPIFC)

Type choices, press Enter.

Internet address . . . . . .  127.0.0.1
Line description . . . . . .  *LOOPBACK   Name, *LOOPBACK
Subnet mask . . . . . .  255.0.0.0
Type of service . . . . . .  *NORMAL    *MINDELAY, *MAXTHRPUT..
Maximum transmission unit . . .  *LIND     576-16388, *LIND
Autostart . . . . . .  *YES     *YES, *NO
PVC logical channel identifier          001-FFF
+ for more values
X.25 idle circuit timeout . . .  60      1-600
X.25 maximum virtual circuits .  64      0-64
X.25 DDN interface . . . . . .  *NO      *YES, *NO
TRLAN bit sequencing . . . . . .  *MSB     *MSB, *LSB



Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
```

2. Specify the values for **IP address**, **Line description**, and **Subnet mask**.

Adding a default route

1. At a command-line, type `ADDTCPRTE` and press enter.

```
Add TCP Route (ADDTCPRTE)

Type choices, press Enter.

Route destination . . . . . .  *DFTROUTE
Subnet mask . . . . . .  *NONE
Type of service . . . . . .  *NORMAL    *MINDELAY, *MAXTHRPUT.
Next hop . . . . . .  19.2.3.4
Maximum transmission unit . . .  576      576-16388, *IFC



Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
Command prompting ended when user pressed F12.
```

2. Enter values appropriate to your network and press enter to create a default route entry.

What next?

The TCP connection is now established. You are ready to complete the configuration. Go to "IBM MQ for IBM i configuration."

IBM MQ for IBM i configuration: IBM i

To configure IBM MQ for IBM i, use the WRKMQMQ command to display the configuration menu.

Start the TCP channel listener using the command STRMQMLSR.

Start any sender channel using the command STRMQMCHL CHLNAME(*channel_name*).

Use the WRKMQMQ command to display the IBM MQ configuration menu.

Note: AMQ* errors are placed in the log relating to the job that found the error. Use the WRKACTJOB command to display the list of jobs. Under the subsystem name QSYSWRK, locate the job and enter `5` against it to work with that job. IBM MQ logs are prefixed `AMQ`.

Creating a queue manager: IBM i

Use the following steps to set up the basic configuration queue manager.

1. First you need to create a queue manager. Type `CRTMQM` and press enter.

```
                    Create Message Queue Manager (CRTMQM)

Type choices, press Enter.

Message Queue Manager name . . .

Text 'description' . . . . . .   *BLANK

Trigger interval . . . . . .   999999999     0-999999999
Undelivered message queue  . . .   *NONE

Default transmission queue . . .   *NONE

Maximum handle limit . . . . . .   256           1-999999999
Maximum uncommitted messages . .   1000          1-10000
Default Queue manager  . . . . .   *NO           *YES, *NO




                                                                      Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

2. In the **Message Queue Manager name** field, type `AS400`. In the **Undelivered message queue** field, type `DEAD.LETTER.QUEUE`.
3. Press enter.
4. Now start the queue manager by entering `STRMQM MQMNAME(AS400)`.
5. Create the undelivered message queue using the following parameters. (For details and an example refer to "Defining a queue.")

```
Local Queue
     Queue name :   DEAD.LETTER.QUEUE
     Queue type :   *LCL
```

Defining a queue: IBM i

You can define a queue using the CRTMQMQ command.

Type `CRTMQMQ` on the command line.

```
Create MQM Queue (CRTMQMQ)

Type choices, press Enter.

Queue name . . . . . .

Queue type . . . . . .        *ALS, *LCL, *RMT









Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
Parameter QNAME required.
```

Complete the two fields of this panel and press enter. Another panel is shown, with entry fields for the other parameters you have. Defaults can be taken for all other queue attributes.

Defining a channel: IBM i

You can define a channel using the CRTMQMCHL command.

Type `CRTMQMCHL` on the command line.

```
Create MQM Channel (CRTMQMCHL)

Type choices, press Enter.

Channel name . . . . . .
Channel type . . . . . .        *RCVR, *SDR, *SVR, *RQSTR









Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
Parameter CHLNAME required.
```

Complete the two fields of this panel and press enter. Another panel is displayed on which you can specify the values for the other parameters given earlier. Defaults can be taken for all other channel attributes.

Channel configuration for IBM i: 

You need to configure your channels to implement the example configuration channels.

This section details the configuration to be performed on the IBM i queue manager to implement the channel described in "Example IBM MQ configuration for all platforms" on page 1.

Examples are given for connecting IBM MQ for IBM i and IBM MQ for Windows. To connect to IBM MQ on another platform, use the appropriate values from the table in place of those values for Windows

Note:

1. The words in **bold** are user-specified and reflect the names of IBM MQ objects used throughout these examples. If you change the names used here, ensure that you also change the other references made to these objects throughout this section. All others are keywords and must be entered as shown.
2. The IBM MQ channel ping command (PNGMQMCHL) runs interactively, whereas starting a channel causes a batch job to be submitted. If a channel ping completes successfully but the channel does not start, the network and IBM MQ definitions are probably correct, but that the IBM i environment for the batch job is not. For example, make sure that QSYS2 is included in the system portion of the library list and not just your personal library list.

For details and examples of how to create the objects listed refer to "Defining a queue" on page 29 and "Defining a channel" on page 30.

Table 4. Configuration worksheet for IBM i

ID	Parameter Name	Reference	Example Used	User Value
Definition for local node				
A	Queue Manager Name		**AS400**	
B	Local queue name		**AS400.LOCALQ**	
Connection to IBM MQ for Windows The values in this section of the table must match the values used in "Channel configuration for Windows" on page 50, as indicated.				
C	Remote queue manager name	A	**WINNT**	
D	Remote queue name		**WINNT.REMOTEQ**	
E	Queue name at remote system	B	**WINNT.LOCALQ**	
F	Transmission queue name		**WINNT**	
G	Sender (SNA) channel name		**AS400.WINNT.SNA**	
H	Sender (TCP/IP) channel name		**AS400.WINNT.TCP**	
I	Receiver (SNA) channel name	G	**WINNT.AS400.SNA**	
J	Receiver (TCP/IP) channel name	H	**WINNT.AS400.TCP**	
Connection to IBM MQ for AIX The values in this section of the table must match the values used in "Channel configuration for AIX" on page 7, as indicated.				
C	Remote queue manager name	A	**AIX**	
D	Remote queue name		**AIX.REMOTEQ**	
E	Queue name at remote system	B	**AIX.LOCALQ**	
F	Transmission queue name		**AIX**	
G	Sender (SNA) channel name		**AS400.AIX.SNA**	

Table 4. Configuration worksheet for IBM i (continued)

ID	Parameter Name	Reference	Example Used	User Value
H	Sender (TCP/IP) channel name		**AS400.AIX.TCP**	
I	Receiver (SNA) channel name	G	**AIX.AS400.SNA**	
J	Receiver (TCP) channel name	H	**AIX.AS400.TCP**	
Connection to MQSeries for Compaq Tru64 Unix The values in this section of the table must match the values used in your Compaq Tru64 UNIX system.				
C	Remote queue manager name	A	**DECUX**	
D	Remote queue name		**DECUX.REMOTEQ**	
E	Queue name at remote system	B	**DECUX.LOCALQ**	
F	Transmission queue name		**DECUX**	
H	Sender (TCP) channel name		**DECUX.AS400.TCP**	
J	Receiver (TCP) channel name	H	**AS400.DECUX.TCP**	
Connection to IBM MQ for HP-UX The values in this section of the table must match the values used in "Channel configuration for HP-UX" on page 13, as indicated.				
C	Remote queue manager name	A	**HPUX**	
D	Remote queue name		**HPUX.REMOTEQ**	
E	Queue name at remote system	B	**HPUX.LOCALQ**	
F	Transmission queue name		**HPUX**	
G	Sender (SNA) channel name		**AS400.HPUX.SNA**	
H	Sender (TCP) channel name		**AS400.HPUX.TCP**	
I	Receiver (SNA) channel name	G	**HPUX.AS400.SNA**	
J	Receiver (TCP) channel name	H	**HPUX.AS400.TCP**	
Connection to IBM MQ for Solaris The values in this section of the table must match the values used in "Channel configuration for Solaris" on page 43, as indicated.				
C	Remote queue manager name	A	**SOLARIS**	
D	Remote queue name		**SOLARIS.REMOTEQ**	
E	Queue name at remote system	B	**SOLARIS.LOCALQ**	
F	Transmission queue name		**SOLARIS**	
G	Sender (SNA) channel name		**AS400.SOLARIS.SNA**	
H	Sender (TCP/IP) channel name		**AS400.SOLARIS.TCP**	
I	Receiver (SNA) channel name	G	**SOLARIS.AS400.SNA**	
J	Receiver (TCP/IP) channel name	H	**SOLARIS.AS400.TCP**	
Connection to IBM MQ for Linux The values in this section of the table must match the values used in "Channel configuration for Linux" on page 38, as indicated.				
C	Remote queue manager name	A	**LINUX**	
D	Remote queue name		**LINUX.REMOTEQ**	
E	Queue name at remote system	B	**LINUX.LOCALQ**	
F	Transmission queue name		**LINUX**	

Table 4. Configuration worksheet for IBM i (continued)

ID	Parameter Name	Reference	Example Used	User Value
G	Sender (SNA) channel name		**AS400.LINUX.SNA**	
H	Sender (TCP/IP) channel name		**AS400.LINUX.TCP**	
I	Receiver (SNA) channel name	G	**LINUX.AS400.SNA**	
J	Receiver (TCP/IP) channel name	H	**LINUX.AS400.TCP**	
Connection to IBM MQ for z/OS The values in this section of the table must match the values used in "Channel configuration for z/OS" on page 57, as indicated.				
C	Remote queue manager name	A	**MVS**	
D	Remote queue name		**MVS.REMOTEQ**	
E	Queue name at remote system	B	**MVS.LOCALQ**	
F	Transmission queue name		**MVS**	
G	Sender (SNA) channel name		**AS400.MVS.SNA**	
H	Sender (TCP) channel name		**AS400.MVS.TCP**	
I	Receiver (SNA) channel name	G	**MVS.AS400.SNA**	
J	Receiver (TCP) channel name	H	**MVS.AS400.TCP**	
Connection to MQSeries for VSE/ESA The values in this section of the table must match the values used in your VSE/ESA system.				
C	Remote queue manager name	A	**VSE**	
D	Remote queue name		**VSE.REMOTEQ**	
E	Queue name at remote system	B	**VSE.LOCALQ**	
F	Transmission queue name		**VSE**	
G	Sender channel name		**AS400.VSE.SNA**	
I	Receiver channel name	G	**VSE.AS400.SNA**	

Sender-channel definitions for IBM i: IBM i

Example sender-channel definitions for SNA and TCP.

Using SNA

```
     Local Queue
              Queue name :   WINNT                           F
              Queue type :   *LCL
                   Usage :   *TMQ

     Remote Queue
              Queue name :   WINNT.REMOTEQ                   D
              Queue type :   *RMT
            Remote queue :   WINNT.LOCALQ                    E
    Remote Queue Manager :   WINNT                           C
      Transmission queue :   WINNT                           F

     Sender Channel
            Channel Name :   AS400.WINNT.SNA                 G
            Channel Type :   *SDR
          Transport type :   *LU62
        Connection name :    WINNTCPIC                       14
      Transmission queue :   WINNT                           F
```

Using TCP

```
Local Queue
         Queue name :   WINNT                           F
         Queue type :   *LCL
              Usage :   *TMQ

Remote Queue
         Queue name :   WINNT.REMOTEQ                   D
         Queue type :   *RMT
       Remote queue :   WINNT.LOCALQ                    E
Remote Queue Manager :   WINNT                          C
  Transmission queue :   WINNT                          F

Sender Channel
       Channel Name :   AS400.WINNT.TCP                 H
       Channel Type :   *SDR
     Transport type :   *TCP
    Connection name :   WINNT.tcpip.hostname
 Transmission queue :   WINNT                          F
```

Receiver-channel definitions for IBM i: 

Example receiver-channel definitions for SNA and TCP

Using SNA

```
Local Queue
         Queue name :   AS400.LOCALQ                    B
         Queue type :   *LCL

Receiver Channel
       Channel Name :   WINNT.AS400.SNA                 I
       Channel Type :   *RCVR
     Transport type :   *LU62
```

Using TCP

```
Local Queue
         Queue name :   AS400.LOCALQ                    B
         Queue type :   *LCL

Receiver Channel
       Channel Name :   WINNT.AS400.TCP                 J
       Channel Type :   *RCVR
     Transport type :   *TCP
```

Example MQ configuration for Linux

Linux

This section gives an example of how to set up communication links from IBM MQ for Linux to IBM MQ products.

The examples given are on the following platforms:

- Windows
- AIX
- Compaq Tru64 UNIX
- HP-UX
- Solaris
- IBM i
- z/OS
- VSE/ESA

See "Example IBM MQ configuration for all platforms" on page 1 for background information about this section and how to use it.

Establishing an LU 6.2 connection: Linux

Use this worksheet to record the values you use for your configuration.

Note: The information in this section applies only to IBM MQ for Linux (x86 platform). It does not apply to IBM MQ for Linux (x86-64 platform), IBM MQ for Linux (zSeries s390x platform), or IBM MQ for Linux (Power platform).

For the latest information about configuring SNA over TCP/IP, refer to the the Administration Guide for your version of Linux from the following documentation: Communications Server for Linux library.

Establishing a TCP connection on Linux: Linux

Some Linux distributions now use the extended inet daemon (XINETD) instead of the inet daemon (INETD). The following instructions tell you how to establish a TCP connection using either the inet daemon or the extended inet daemon.

Using the inet daemon (INETD)

MQ_INSTALLATION_PATH represents the high-level directory in which IBM MQ is installed.

To establish a TCP connection, follow these steps.

1. Edit the file /etc/services. If you do not have the following line in the file, add it as shown:

   ```
   MQSeries    1414/tcp   # MQSeries channel listener
   ```

   Note: To edit this file, you must be logged in as a superuser or root.
2. Edit the file /etc/inetd.conf. If you do not have the following line in that file, add it as shown:

   ```
   MQSeries stream tcp nowait mqm MQ_INSTALLATION_PATH/bin/amqcrsta amqcrsta
   [-m queue.manager.name ]
   ```
3. Find the process ID of the inetd with the command:

   ```
   ps -ef | grep inetd
   ```
4. Run the command:

```
kill -1 inetd processid
```

If you have more than one queue manager on your system, and therefore require more than one service, you must add a line for each additional queue manager to both /etc/services and inetd.conf.

For example:

```
MQSeries1   1414/tcp
MQSeries2   1822/tcp
```

```
MQSeries1 stream tcp nowait mqm MQ_INSTALLATION_PATH/bin/amqcrsta amqcrsta -m QM1
MQSeries2 stream tcp nowait mqm MQ_INSTALLATION_PATH/bin/amqcrsta amqcrsta -m QM2
```

This avoids error messages being generated if there is a limitation on the number of outstanding connection requests queued at a single TCP port. For information about the number of outstanding connection requests, see Using the TCP listener backlog option.

The inetd process on Linux can limit the rate of inbound connections on a TCP port. The default is 40 connections in a 60 second interval. If you need a higher rate, specify a new limit on the number of inbound connections in a 60 second interval by appending a period (.) followed by the new limit to the nowait parameter of the appropriate service in inetd.conf. For example, for a limit of 500 connections in a 60 second interval use:

```
MQSeries stream tcp nowait.500 mqm / MQ_INSTALLATION_PATH/bin/amqcrsta amqcrsta -m QM1
```

MQ_INSTALLATION_PATH represents the high-level directory in which IBM MQ is installed.

Using the extended inet daemon (XINETD)

The following instructions describe how the extended inet daemon is implemented on Red Hat Linux. If you are using a different Linux distribution, you might have to adapt these instructions.

To establish a TCP connection, follow these steps.

1. Edit the file /etc/services. If you do not have the following line in the file, add it as shown:

   ```
   MQSeries    1414/tcp   # MQSeries channel listener
   ```

   Note: To edit this file, you must be logged in as a superuser or root.
2. Create a file called IBM MQ in the XINETD configuration directory, /etc/xinetd.d. Add the following stanza to the file:

   ```
   # IBM MQ service for XINETD
   service MQSeries
   {
     disable         = no
     flags           = REUSE
     socket_type     = stream
     wait            = no
     user            = mqm
     server          = MQ_INSTALLATION_PATH/bin/amqcrsta
     server_args     = -m queue.manager.name
     log_on_failure += USERID
   }
   ```
3. Restart the extended inet daemon by issuing the following command:

   ```
   /etc/rc.d/init.d/xinetd restart
   ```

If you have more than one queue manager on your system, and therefore require more than one service, you must add a line to /etc/services for each additional queue manager. You can create a file in the /etc/xinetd.d directory for each service, or you can add additional stanzas to the IBM MQ file you created previously.

The xinetd process on Linux can limit the rate of inbound connections on a TCP port. The default is 50 connections in a 10 second interval. If you need a higher rate, specify a new limit on the rate of inbound connections by specifying the 'cps' attribute in the xinetd configuration file. For example, for a limit of 500 connections in a 60 second interval use:

```
cps = 500 60
```

What next?

The TCP/IP connection is now established. You are ready to complete the configuration. Go to "IBM MQ for Linux configuration."

IBM MQ for Linux configuration: 

Before beginning the installation process ensure that you have first created the mqm user ID and the mqm group, and set the password.

Start any channel using the command:

```
runmqchl -c channel.name
```

Note:

1. Sample programs are installed in *MQ_INSTALLATION_PATH*/samp, where *MQ_INSTALLATION_PATH* represents the high-level directory in which IBM MQ is installed.
2. Error logs are stored in /var/mqm/qmgrs/ *qmgrname* /errors.
3. When you are using the command interpreter **runmqsc** to enter administration commands, a + at the end of a line indicates that the next line is a continuation. Ensure that there is a space between the last parameter and the continuation character.

Basic configuration

1. Create the queue manager from the UNIX prompt using the command:

   ```
   crtmqm -u dlqname -q linux
   ```

   where:

   linux Is the name of the queue manager

   -q Indicates that this is to become the default queue manager

   -u *dlqname*
   Specifies the name of the dead letter queue

   This command creates a queue manager and a set of default objects.
2. Start the queue manager from the UNIX prompt using the command:

   ```
   strmqm linux
   ```

   where *linux* is the name given to the queue manager when it was created.

Channel configuration for Linux: 

The following section details the configuration to be performed on the Linux queue manager to implement the channel described in "Example IBM MQ configuration for all platforms" on page 1.

The MQSC command to create each object is shown. Either start **runmqsc** from a UNIX prompt and enter each command in turn, or build the commands into a command file.

Examples are given for connecting IBM MQ for Linux and IBM MQ for HP-UX. To connect to IBM MQ on another platform use the appropriate set of values from the table in place of those for HP-UX.

Note: The words in **bold** are user-specified and reflect the names of IBM MQ objects used throughout these examples. If you change the names used here, ensure that you also change the other references made to these objects throughout this section. All others are keywords and should be entered as shown.

Table 5. Configuration worksheet for IBM MQ for Linux

ID	Parameter Name	Reference	Example Used	User Value
Definition for local node				
A	Queue Manager Name		**LINUX**	
B	Local queue name		**LINUX.LOCALQ**	
Connection to IBM MQ for Windows The values in this section of the table must match those used in "Channel configuration for Windows" on page 50, as indicated.				
C	Remote queue manager name	A	**WINNT**	
D	Remote queue name		**WINNT.REMOTEQ**	
E	Queue name at remote system	B	**WINNT.LOCALQ**	
F	Transmission queue name		**WINNT**	
G	Sender (SNA) channel name		**LINUX.WINNT.SNA**	
H	Sender (TCP/IP) channel name		**LINUX.WINNT.TCP**	
I	Receiver (SNA) channel name	G	**WINNT.LINUX.SNA**	
J	Receiver (TCP) channel name	H	**WINNT.LINUX.TCP**	
Connection to IBM MQ for AIX The values in this section of the table must match those used in "Channel configuration for AIX" on page 7, as indicated.				
C	Remote queue manager name	A	**AIX**	
D	Remote queue name		**AIX.REMOTEQ**	
E	Queue name at remote system	B	**AIX.LOCALQ**	
F	Transmission queue name		**AIX**	
G	Sender (SNA) channel name		**LINUX.AIX.SNA**	
H	Sender (TCP) channel name		**LINUX.AIX.TCP**	
I	Receiver (SNA) channel name	G	**AIX.LINUX.SNA**	
J	Receiver (TCP) channel name	H	**AIX.LINUX.TCP**	
Connection to MQSeries for Compaq Tru64 UNIX The values in this section of the table must match those used in your Compaq Tru64 UNIX system.				
C	Remote queue manager name	A	**DECUX**	

Table 5. Configuration worksheet for IBM MQ for Linux (continued)

ID	Parameter Name	Reference	Example Used	User Value
D	Remote queue name		**DECUX.REMOTEQ**	
E	Queue name at remote system	B	**DECUX.LOCALQ**	
F	Transmission queue name		**DECUX**	
H	Sender (TCP) channel name		**DECUX.LINUX.TCP**	
J	Receiver (TCP) channel name	H	**LINUX.DECUX.TCP**	
Connection to IBM MQ for HP-UX The values in this section of the table must match those used in Table 2 on page 14, as indicated.				
C	Remote queue manager name	A	**HPUX**	
D	Remote queue name		**HPUX.REMOTEQ**	
E	Queue name at remote system	B	**HPUX.LOCALQ**	
F	Transmission queue name		**HPUX**	
G	Sender (SNA) channel name		**LINUX.HPUX.SNA**	
H	Sender (TCP) channel name		**LINUX.HPUX.TCP**	
I	Receiver (SNA) channel name	G	**HPUX.LINUX.SNA**	
J	Receiver (TCP/IP) channel name	H	**HPUX.LINUX.TCP**	
Connection to IBM MQ for Solaris The values in this section of the table must match those used in Table 6 on page 43, as indicated.				
C	Remote queue manager name	A	**SOLARIS**	
D	Remote queue name		**SOLARIS.REMOTEQ**	
E	Queue name at remote system	B	**SOLARIS.LOCALQ**	
F	Transmission queue name		**GIS**	
G	Sender (SNA) channel name		**LINUX.SOLARIS.SNA**	
H	Sender (TCP/IP) channel name		**LINUX.SOLARIS.TCP**	
I	Receiver (SNA) channel name	G	**SOLARIS.LINUX.SNA**	
J	Receiver (TCP/IP) channel name	H	**SOLARIS.LINUX.TCP**	
IBM i ***Connection to IBM MQ for IBM i*** IBM i The values in this section of the table must match those used in Table 4 on page 31, as indicated.				
C	Remote queue manager name	A	**AS400**	
D	Remote queue name		**AS400.REMOTEQ**	
E	Queue name at remote system	B	**AS400.LOCALQ**	
F	Transmission queue name		**AS400**	
G	Sender (SNA) channel name		**LINUX.AS400.SNA**	
H	Sender (TCP) channel name		**LINUX.AS400.TCP**	
I	Receiver (SNA) channel name	G	**AS400.LINUX.SNA**	
J IBM i	Receiver (TCP) channel name	H	**AS400.LINUX.TCP**	
z/OS ***Connection to IBM MQ for z/OS*** z/OS The values in this section of the table must match those used in Table 8 on page 57, as indicated.				

Table 5. Configuration worksheet for IBM MQ for Linux (continued)

ID	Parameter Name	Reference	Example Used	User Value
C	Remote queue manager name	A	**MVS**	
D	Remote queue name		**MVS.REMOTEQ**	
E	Queue name at remote system	B	**MVS.LOCALQ**	
F	Transmission queue name		**MVS**	
G	Sender (SNA) channel name		**LINUX.MVS.SNA**	
H	Sender (TCP) channel name		**LINUX.MVS.TCP**	
I	Receiver (SNA) channel name	G	**MVS.LINUX.SNA**	
z/OS J	Receiver (TCP) channel name	H	**MVS.LINUX.TCP**	
Connection to MQSeries for VSE/ESA (IBM MQ for Linux (x86 platform) only) The values in this section of the table must match those used in your VSE/ESA system.				
C	Remote queue manager name	A	**VSE**	
D	Remote queue name		**VSE.REMOTEQ**	
E	Queue name at remote system	B	**VSE.LOCALQ**	
F	Transmission queue name		**VSE**	
G	Sender channel name		**LINUX.VSE.SNA**	
I	Receiver channel name	G	**VSE.LINUX.SNA**	

IBM MQ for Linux (x86 platform) sender-channel definitions using SNA: Linux

Example coding.

```
def ql (HPUX) +                                F
    usage(xmitq) +
    replace

def qr (HPUX.REMOTEQ) +                        D
    rname(HPUX.LOCALQ) +                       E
    rqmname(HPUX) +                            C
    xmitq(HPUX) +                              F
    replace

def chl (LINUX.HPUX.SNA) chltype(sdr) +        G
    trptype(lu62) +
    conname('HPUXCPIC') +                      14
    xmitq(HPUX) +                              F
    replace
```

IBM MQ for Linux (x86 platform) receiver-channel definitions using SNA: Linux

Example coding.

```
def ql (LINUX.LOCALQ) replace                    B

def chl (HPUX.LINUX.SNA) chltype(rcvr) +         I
    trptype(lu62) +
    replace
```

IBM MQ for Linux sender-channel definitions using TCP: Linux

Example coding.

```
def ql (HPUX) +                                  F
    usage(xmitq) +
    replace

def qr (HPUX.REMOTEQ) +                          D
    rname(HPUX.LOCALQ) +                         E
    rqmname(HPUX) +                              C
    xmitq(HPUX) +                                F
    replace

def chl (LINUX.HPUX.TCP) chltype(sdr) +          H
    trptype(tcp) +
    conname(remote_tcpip_hostname) +
    xmitq(HPUX) +                                F
    replace
```

IBM MQ for Linux receiver-channel definitions using TCP/IP: Linux

Example coding.

```
def ql (LINUX.LOCALQ) replace                    B

def chl (HPUX.LINUX.TCP) chltype(rcvr) +         J
    trptype(tcp) +
    replace
```

Example MQ configuration for Solaris

Solaris

This section gives an example of how to set up communication links from IBM MQ for Solaris to IBM MQ products.

Examples are given on the following platforms:

- Windows
- AIX
- HP Tru64 UNIX
- HP-UX
- Linux
- IBM i
- z/OS
- VSE/ESA

See “Example IBM MQ configuration for all platforms” on page 1 for background information about this section and how to use it.

Establishing an LU 6.2 connection using SNAP-IX: Solaris

Parameters for configuring an LU 6.2 connection using SNAP-IX.

For the latest information about configuring SNA over TCP/IP, refer to the following online IBM documentation: Communications Server, the following online MetaSwitch documentation: SNAP-IX Administration Guide, and the following online Sun documentation: Configuring Intersystem Communications (ISC).

Establishing a TCP connection: Solaris

Information about configuring a TCP connection and next steps.

To establish a TCP connection, follow these steps.

1. Edit the file /etc/services.

   Note: To edit the /etc/services file, you must be logged in as a superuser or root. If you do not have the following line in that file, add it as shown:

   ```
   MQSeries       1414/tcp      # MQSeries channel listener
   ```

2. Edit the file /etc/inetd.conf. If you do not have the following line in that file, add it as shown:

   ```
   MQSeries stream tcp nowait mqm MQ_INSTALLATION_PATH/bin/amqcrsta amqcrsta
   [-m queue.manager.name]
   ```

   MQ_INSTALLATION_PATH represents the high-level directory in which IBM MQ is installed.

3. Find the process ID of the inetd with the command:

   ```
   ps -ef | grep inetd
   ```

4. Run the appropriate command, as follows:

   - For Solaris 9:

     ```
     kill -1 inetd processid
     ```

   - For Solaris 10 or later:

     ```
     inetconv
     ```

What next?

The TCP/IP connection is now established. You are ready to complete the configuration. Go to "IBM MQ for Solaris configuration."

IBM MQ for Solaris configuration: Solaris

Describes channels to be defined to complete the configuration.

Before beginning the installation process ensure that you have first created the *mqm* user and group, and set the password.

Start any channel using the command:

```
runmqchl -c channel.name
```

Note:

1. Sample programs are installed in *MQ_INSTALLATION_PATH*/samp.
   MQ_INSTALLATION_PATH represents the high-level directory in which IBM MQ is installed.
2. Error logs are stored in /var/mqm/qmgrs/ *qmgrname* /errors.

3. When you are using the command interpreter **runmqsc** to enter administration commands, a + at the end of a line indicates that the next line is a continuation. Ensure that there is a space between the last parameter and the continuation character.
4. For an SNA or LU6.2 channel, if you experience an error when you try to load the communications library, probably file liblu62.so cannot be found. A likely solution to this problem is to add its location, which is probably /opt/SUNWlu62, to LD_LIBRARY_PATH.

Basic configuration

1. Create the queue manager from the UNIX prompt using the command:

   ```
   crtmqm -u dlqname -q solaris
   ```

   where:

   solaris
   : Is the name of the queue manager

   -q
   : Indicates that this is to become the default queue manager

   -u *dlqname*
   : Specifies the name of the undeliverable message queue

   This command creates a queue manager and a set of default objects.
2. Start the queue manager from the UNIX prompt using the command:

   ```
   strmqm solaris
   ```

   where *solaris* is the name given to the queue manager when it was created.

Channel configuration for Solaris: Solaris

The following section details the configuration to be performed on the Solaris queue manager to implement a channel.

The configuration described is to implement the channel described in Figure 1 on page 1.

The MQSC command to create each object is shown. Either start **runmqsc** from a UNIX prompt and enter each command in turn, or build the commands into a command file.

Examples are given for connecting IBM MQ for Solaris and IBM MQ for Windows. To connect to IBM MQ on another platform use the appropriate set of values from the table in place of those for Windows.

Note: The words in **bold** are user-specified and reflect the names of IBM MQ objects used throughout these examples. If you change the names used here, ensure that you also change the other references made to these objects throughout this section. All others are keywords and should be entered as shown.

Table 6. Configuration worksheet for IBM MQ for Solaris

ID	Parameter Name	Reference	Example Used	User Value
Definition for local node				
A	Queue Manager Name		**SOLARIS**	
B	Local queue name		**SOLARIS.LOCALQ**	
Connection to IBM MQ for Windows The values in this section of the table must match those used in Table 7 on page 50, as indicated.				
C	Remote queue manager name	A	**WINNT**	
D	Remote queue name		**WINNT.REMOTEQ**	

Table 6. Configuration worksheet for IBM MQ for Solaris (continued)

ID	Parameter Name	Reference	Example Used	User Value
E	Queue name at remote system	B	**WINNT.LOCALQ**	
F	Transmission queue name		**WINNT**	
G	Sender (SNA) channel name		**SOLARIS.WINNT.SNA**	
H	Sender (TCP/IP) channel name		**SOLARIS.WINNT.TCP**	
I	Receiver (SNA) channel name	G	**WINNT.SOLARIS.SNA**	
J	Receiver (TCP) channel name	H	**WINNT.SOLARIS.TCP**	
Connection to IBM MQ for AIX The values in this section of the table must match those used in Table 1 on page 7, as indicated.				
C	Remote queue manager name	A	**AIX**	
D	Remote queue name		**AIX.REMOTEQ**	
E	Queue name at remote system	B	**AIX.LOCALQ**	
F	Transmission queue name		**AIX**	
G	Sender (SNA) channel name		**SOLARIS.AIX.SNA**	
H	Sender (TCP) channel name		**SOLARIS.AIX.TCP**	
I	Receiver (SNA) channel name	G	**AIX.SOLARIS.SNA**	
J	Receiver (TCP) channel name	H	**AIX.SOLARIS.TCP**	
Connection to MQSeries for Compaq Tru64 Unix The values in this section of the table must match those used in your Compaq Tru64 UNIX system.				
C	Remote queue manager name	A	**DECUX**	
D	Remote queue name		**DECUX.REMOTEQ**	
E	Queue name at remote system	B	**DECUX.LOCALQ**	
F	Transmission queue name		**DECUX**	
H	Sender (TCP) channel name		**DECUX.SOLARIS.TCP**	
J	Receiver (TCP) channel name	H	**SOLARIS.DECUX.TCP**	
Connection to IBM MQ for HP-UX The values in this section of the table must match those used in Table 2 on page 14, as indicated.				
C	Remote queue manager name	A	**HPUX**	
D	Remote queue name		**HPUX.REMOTEQ**	
E	Queue name at remote system	B	**HPUX.LOCALQ**	
F	Transmission queue name		**HPUX**	
G	Sender (SNA) channel name		**SOLARIS.HPUX.SNA**	
H	Sender (TCP) channel name		**SOLARIS.HPUX.TCP**	
I	Receiver (SNA) channel name	G	**HPUX.SOLARIS.SNA**	
J	Receiver (TCP/IP) channel name	H	**HPUX.SOLARIS.TCP**	
Connection to IBM MQ for Linux The values in this section of the table must match those used in Table 5 on page 38, as indicated.				
C	Remote queue manager name	A	**LINUX**	
D	Remote queue name		**LINUX.REMOTEQ**	
E	Queue name at remote system	B	**LINUX.LOCALQ**	

Table 6. Configuration worksheet for IBM MQ for Solaris (continued)

ID	Parameter Name	Reference	Example Used	User Value
F	Transmission queue name		**LINUX**	
G	Sender (SNA) channel name		**SOLARIS.LINUX.SNA**	
H	Sender (TCP/IP) channel name		**SOLARIS.LINUX.TCP**	
I	Receiver (SNA) channel name	G	**LINUX.SOLARIS.SNA**	
J	Receiver (TCP/IP) channel name	H	**LINUX.SOLARIS.TCP**	
IBM i ***Connection to IBM MQ for IBM i***				
IBM i The values in this section of the table must match those used in Table 4 on page 31, as indicated.				
C	Remote queue manager name	A	**AS400**	
D	Remote queue name		**AS400.REMOTEQ**	
E	Queue name at remote system	B	**AS400.LOCALQ**	
F	Transmission queue name		**AS400**	
G	Sender (SNA) channel name		**SOLARIS.AS400.SNA**	
H	Sender (TCP) channel name		**SOLARIS.AS400.TCP**	
I	Receiver (SNA) channel name	G	**AS400.SOLARIS.SNA**	
IBM i J	Receiver (TCP) channel name	H	**AS400.SOLARIS.TCP**	
z/OS ***Connection to IBM MQ for z/OS***				
z/OS The values in this section of the table must match those used in Table 8 on page 57, as indicated.				
C	Remote queue manager name	A	**MVS**	
D	Remote queue name		**MVS.REMOTEQ**	
E	Queue name at remote system	B	**MVS.LOCALQ**	
F	Transmission queue name		**MVS**	
G	Sender (SNA) channel name		**SOLARIS.MVS.SNA**	
H	Sender (TCP) channel name		**SOLARIS.MVS.TCP**	
I	Receiver (SNA) channel name	G	**MVS.SOLARIS.SNA**	
z/OS J	Receiver (TCP) channel name	H	**MVS.SOLARIS.TCP**	
Connection to MQSeries for VSE/ESA				
The values in this section of the table must match those used in your VSE/ESA system.				
C	Remote queue manager name	A	**VSE**	
D	Remote queue name		**VSE.REMOTEQ**	
E	Queue name at remote system	B	**VSE.LOCALQ**	
F	Transmission queue name		**VSE**	
G	Sender channel name		**SOLARIS.VSE.SNA**	
I	Receiver channel name	G	**VSE.SOLARIS.SNA**	

IBM MQ for Solaris sender-channel definitions using SNAP-IX SNA: Solaris

Example coding.

```
def ql (WINNT) +                                F
    usage(xmitq) +
    replace

def qr (WINNT.REMOTEQ) +                        D
    rname(WINNT.LOCALQ) +                       E
    rqmname(WINNT) +                            C
    xmitq(WINNT) +                              F
    replace

def chl (SOLARIS.WINNT.SNA) chltype(sdr) +      G
    trptype(lu62) +
    conname('NTCPIC') +            14
    xmitq(WINNT) +                              F
    replace
```

IBM MQ for Solaris receiver-channel definitions using SNA: Solaris

Example coding.

```
def ql (SOLARIS.LOCALQ) replace                 B

def chl (WINNT.SOLARIS.SNA) chltype(rcvr) +     I
    trptype(lu62) +
    replace
```

IBM MQ for Solaris sender-channel definitions using TCP: Solaris

Example coding.

```
def ql (WINNT) +                                F
    usage(xmitq) +
    replace

def qr (WINNT.REMOTEQ) +                        D
    rname(WINNT.LOCALQ) +                       E
    rqmname(WINNT) +                            C
    xmitq(WINNT) +                              F
    replace

def chl (SOLARIS.WINNT.TCP) chltype(sdr) +      H
    trptype(tcp) +
    conname(remote_tcpip_hostname) +
    xmitq(WINNT) +                              F
    replace
```

IBM MQ for Solaris receiver-channel definitions using TCP/IP: Solaris

Example coding.

```
def ql (SOLARIS.LOCALQ) replace                  B

def chl (WINNT.SOLARIS.TCP) chltype(rcvr) +      J
    trptype(tcp) +
    replace
```

Example IBM MQ configuration for Windows

Windows

This section gives an example of how to set up communication links from IBM MQ for Windows to IBM MQ products on other platforms.

Setup of communication links is shown on the following platforms:

- AIX
- HP Tru64 UNIX
- HP-UX
- Solaris
- Linux
- IBM i
- z/OS
- VSE/ESA

When the connection is established, you must define some channels to complete the configuration. Example programs and commands for configuration are described in "IBM MQ for Windows configuration" on page 49.

See "Example IBM MQ configuration for all platforms" on page 1 for background information about this section and how to use it.

Establishing an LU 6.2 connection: Windows

Reference to information about configuring AnyNet® SNA over TCP/IP.

For the latest information about configuring AnyNet SNA over TCP/IP, see the following online IBM documentation: AnyNet SNA over TCP/IP, SNA Node Operations, and Communications Server for Windows

Establishing a TCP connection: Windows

The TCP stack that is shipped with Windows systems does not include an *inet* daemon or equivalent.

The IBM MQ command used to start the IBM MQ for TCP listener is:

```
runmqlsr -t tcp
```

The listener must be started explicitly before any channels are started. It enables receiving channels to start automatically in response to a request from an inbound sending channel.

What next?

When the TCP/IP connection is established, you are ready to complete the configuration. Go to "IBM MQ for Windows configuration" on page 49.

Establishing a NetBIOS connection: Windows

A NetBIOS connection is initiated from a queue manager that uses the ConnectionName parameter on its channel definition to connect to a target listener.

To set up a NetBIOS connection, follow these steps:

1. At each end of the channel specify the local NetBIOS name to be used by the IBM MQ channel processes in the queue manager configuration file qm.ini. For example, the NETBIOS stanza in Windows at the sending end might look like the following:

   ```
   NETBIOS:
    LocalName=WNTNETB1
   ```

   and at the receiving end:

   ```
   NETBIOS:
    LocalName=WNTNETB2
   ```

   Each IBM MQ process must use a different local NetBIOS name. Do not use your system name as the NetBIOS name because Windows already uses it.
2. At each end of the channel, verify the LAN adapter number being used on your system. The IBM MQ for Windows default for logical adapter number 0 is NetBIOS running over an Internet Protocol network. To use native NetBIOS you must select logical adapter number 1. See Establishing the LAN adapter number.

   Specify the correct LAN adapter number in the NETBIOS stanza of the Windows registry. For example:

   ```
   NETBIOS:
    AdapterNum=1
   ```
3. So that sender channel initiation works, specify the local NetBIOS name by the MQNAME environment variable:

   ```
   SET MQNAME=WNTNETB1I
   ```

   This name must be unique.
4. At the sending end, define a channel specifying the NetBIOS name being used at the other end of the channel. For example:

   ```
   DEFINE CHANNEL (WINNT.OS2.NET) CHLTYPE(SDR) +
          TRPTYPE(NETBIOS) +
          CONNAME(WNTNETB2) +
          XMITQ(OS2) +
          MCATYPE(THREAD) +
          REPLACE
   ```

   You must specify the option `MCATYPE(THREAD)` because, on Windows, sender channels must be run as threads.

5. At the receiving end, define the corresponding receiver channel. For example:
   ```
   DEFINE CHANNEL (WINNT.OS2.NET) CHLTYPE(RCVR) +
          TRPTYPE(NETBIOS) +
          REPLACE
   ```
6. Start the channel initiator because each new channel is started as a thread rather than as a new process.
   ```
   runmqchi
   ```
7. At the receiving end, start the IBM MQ listener:
   ```
   runmqlsr -t netbios
   ```
   Optionally you can specify values for the queue manager name, NetBIOS local name, number of sessions, number of names, and number of commands. See Defining a NetBIOS connection on Windows for more information about setting up NetBIOS connections.

IBM MQ for Windows configuration: Windows

Example programs and commands for configuration.

Note:

1. You can use the sample program, AMQSBCG, to show the contents and headers of all the messages in a queue. For example:
   ```
   AMQSBCG q_name qmgr_name
   ```
   shows the contents of the queue *q_name* defined in queue manager *qmgr_name*.
   Alternatively, you can use the message browser in the IBM MQ Explorer.
2. You can start any channel from the command prompt using the command
   ```
   runmqchl -c channel.name
   ```
3. Error logs can be found in the directories *MQ_INSTALLATION_PATH*\qmgrs\ *qmgrname* \errors and *MQ_INSTALLATION_PATH*\qmgrs\@system\errors. In both cases, the most recent messages are at the end of amqerr01.log.
   MQ_INSTALLATION_PATH represents the high-level directory in which IBM MQ is installed.
4. When you are using the command interpreter **runmqsc** to enter administration commands, a + at the end of a line indicates that the next line is a continuation. Ensure that there is a space between the last parameter and the continuation character.

Default configuration: Windows

You can create a default configuration by using the IBM MQ Postcard application to guide you through the process.

For information about using the Postcard application, see *Verify the installation using the Postcard application* for the platform that your enterprise uses.

Basic configuration: Windows

You can create and start a queue manager from the IBM MQ Explorer or from the command prompt.

.If you choose the command prompt:

1. Create the queue manager using the command:
   ```
   crtmqm -u dlqname -q winnt
   ```
   where:

   winnt Is the name of the queue manager

   -q Indicates that this is to become the default queue manager

   -u *dlqname*
   Specifies the name of the undeliverable message queue

   This command creates a queue manager and a set of default objects.
2. Start the queue manager using the command:
   ```
   strmqm winnt
   ```
   where *winnt* is the name given to the queue manager when it was created.

Channel configuration for Windows: Windows

Example configuration to be performed on the Windows queue manager to implement a given channel.

The following sections detail the configuration to be performed on the Windows queue manager to implement the channel described in "Example IBM MQ configuration for all platforms" on page 1.

In each case the MQSC command is shown. Either start **runmqsc** from a command prompt and enter each command in turn, or build the commands into a command file.

Examples are given for connecting IBM MQ for Windows and IBM MQ for AIX. To connect to IBM MQ on another platform use the appropriate set of values from the table in place of those for Windows.

Note: The words in **bold** are user-specified and reflect the names of IBM MQ objects used throughout these examples. If you change the names used here, ensure that you also change the other references made to these objects throughout this section. All others are keywords and should be entered as shown.

Table 7. Configuration worksheet for IBM MQ for Windows

	Parameter Name	Reference	Example Used	User Value
Definition for local node				
A	Queue Manager Name		**WINNT**	
B	Local queue name		**WINNT.LOCALQ**	
Connection to IBM MQ for AIX The values in this section of the table must match those used in "Channel configuration for AIX" on page 7, as indicated.				
C	Remote queue manager name	A	**AIX**	
D	Remote queue name		**AIX.REMOTEQ**	
E	Queue name at remote system	B	**AIX.LOCALQ**	
F	Transmission queue name		**AIX**	
G	Sender (SNA) channel name		**WINNT.AIX.SNA**	

Table 7. Configuration worksheet for IBM MQ for Windows (continued)

	Parameter Name	Reference	Example Used	User Value
H	Sender (TCP) channel name		**WINNT.AIX.TCP**	
I	Receiver (SNA) channel name	G	**AIX.WINNT.SNA**	
J	Receiver (TCP) channel name	H	**AIX.WINNT.TCP**	
Connection to IBM MQ for HP Tru64 UNIX The values in this section of the table must match those used in your HP Tru64 UNIX system.				
C	Remote queue manager name	A	**DECUX**	
D	Remote queue name		**DECUX.REMOTEQ**	
E	Queue name at remote system	B	**DECUX.LOCALQ**	
F	Transmission queue name		**DECUX**	
H	Sender (TCP) channel name		**DECUX.WINNT.TCP**	
J	Receiver (TCP) channel name	H	**WINNT.DECUX.TCP**	
Connection to IBM MQ for HP-UX The values in this section of the table must match those used in "Channel configuration for HP-UX" on page 13, as indicated.				
C	Remote queue manager name	A	**HPUX**	
D	Remote queue name		**HPUX.REMOTEQ**	
E	Queue name at remote system	B	**HPUX.LOCALQ**	
F	Transmission queue name		**HPUX**	
G	Sender (SNA) channel name		**WINNT.HPUX.SNA**	
H	Sender (TCP) channel name		**WINNT.HPUX.TCP**	
I	Receiver (SNA) channel name	G	**HPUX.WINNT.SNA**	
J	Receiver (TCP/IP) channel name	H	**HPUX.WINNT.TCP**	
Connection to IBM MQ for Solaris The values in this section of the table must match those used in "Channel configuration for Solaris" on page 43, as indicated.				
C	Remote queue manager name	A	**SOLARIS**	
D	Remote queue name		**SOLARIS.REMOTEQ**	
E	Queue name at remote system	B	**SOLARIS.LOCALQ**	
F	Transmission queue name		**SOLARIS**	
G	Sender (SNA) channel name		**WINNT.SOLARIS.SNA**	
H	Sender (TCP) channel name		**WINNT.SOLARIS.TCP**	
I	Receiver (SNA) channel name	G	**SOLARIS.WINNT.SNA**	
J	Receiver (TCP) channel name	H	**SOLARIS.WINNT.TCP**	
Connection to IBM MQ for Linux The values in this section of the table must match those used in "Channel configuration for Linux" on page 38, as indicated.				
C	Remote queue manager name	A	**LINUX**	
D	Remote queue name		**LINUX.REMOTEQ**	
E	Queue name at remote system	B	**LINUX.LOCALQ**	
F	Transmission queue name		**LINUX**	

Table 7. Configuration worksheet for IBM MQ for Windows (continued)

	Parameter Name	Reference	Example Used	User Value
G	Sender (SNA) channel name		**WINNT.LINUX.SNA**	
H	Sender (TCP) channel name		**WINNT.LINUX.TCP**	
I	Receiver (SNA) channel name	G	**LINUX.WINNT.SNA**	
J	Receiver (TCP) channel name	H	**LINUX.WINNT.TCP**	
IBM i *Connection to IBM MQ for IBM i* IBM i The values in this section of the table must match those used in "Channel configuration for IBM i" on page 31, as indicated.				
C	Remote queue manager name	A	**AS400**	
D	Remote queue name		**AS400.REMOTEQ**	
E	Queue name at remote system	B	**AS400.LOCALQ**	
F	Transmission queue name		**AS400**	
G	Sender (SNA) channel name		**WINNT.AS400.SNA**	
H	Sender (TCP) channel name		**WINNT.AS400.TCP**	
I	Receiver (SNA) channel name	G	**AS400.WINNT.SNA**	
J IBM i	Receiver (TCP) channel name	H	**AS400.WINNT.TCP**	
z/OS *Connection to IBM MQ for z/OS* z/OS The values in this section of the table must match those used in "Channel configuration for z/OS" on page 57, as indicated.				
C	Remote queue manager name	A	**MVS**	
D	Remote queue name		**MVS.REMOTEQ**	
E	Queue name at remote system	B	**MVS.LOCALQ**	
F	Transmission queue name		**MVS**	
G	Sender (SNA) channel name		**WINNT.MVS.SNA**	
H	Sender (TCP) channel name		**WINNT.MVS.TCP**	
I	Receiver (SNA) channel name	G	**MVS.WINNT.SNA**	
J z/OS	Receiver (TCP/IP) channel name	H	**MVS.WINNT.TCP**	
z/OS *Connection to IBM MQ for z/OS using queue-sharing groups* z/OS The values in this section of the table must match those used in "Shared channel configuration example" on page 66, as indicated.				
C	Remote queue manager name	A	**QSG**	
D	Remote queue name		**QSG.REMOTEQ**	
E	Queue name at remote system	B	**QSG.SHAREDQ**	
F	Transmission queue name		**QSG**	
G	Sender (SNA) channel name		**WINNT.QSG.SNA**	
H	Sender (TCP) channel name		**WINNT.QSG.TCP**	
I	Receiver (SNA) channel name	G	**QSG.WINNT.SNA**	
J z/OS	Receiver (TCP/IP) channel name	H	**QSG.WINNT.TCP**	

Table 7. Configuration worksheet for IBM MQ for Windows (continued)

	Parameter Name	Reference	Example Used	User Value
Connection to MQSeries for VSE/ESA The values in this section of the table must match those used in your VSE/ESA system.				
C	Remote queue manager name	A	**VSE**	
D	Remote queue name		**VSE.REMOTEQ**	
E	Queue name at remote system	B	**VSE.LOCALQ**	
F	Transmission queue name		**VSE**	
G	Sender channel name		**WINNT.VSE.SNA**	
I	Receiver channel name	G	**VSE.WINNT.SNA**	

IBM MQ for Windows sender-channel definitions using SNA: Windows

A code sample.

```
def ql (AIX) +                              F
    usage(xmitq) +
    replace

def qr (AIX.REMOTEQ) +                      D
    rname(AIX.LOCALQ) +                     E
    rqmname(AIX) +                          C
    xmitq(AIX) +                            F
    replace

def chl (WINNT.AIX.SNA) chltype(sdr) +      G
    trptype(lu62) +
    conname(AIXCPIC) +                      18
    xmitq(AIX) +                            F
    replace
```

IBM MQ for Windows receiver-channel definitions using SNA: Windows

A code sample.

```
def ql (WINNT.LOCALQ) replace               B

def chl (AIX.WINNT.SNA) chltype(rcvr) +     I
    trptype(lu62) +
    replace
```

IBM MQ for Windows sender-channel definitions using TCP/IP: Windows

A code sample.

```
def ql (AIX) +                                  F
    usage(xmitq) +
    replace

def qr (AIX.REMOTEQ) +                          D
    rname(AIX.LOCALQ) +                         E
    rqmname(AIX) +                              C
    xmitq(AIX) +                                F
    replace

def chl (WINNT.AIX.TCP) chltype(sdr) +          H
    trptype(tcp) +
    conname(remote_tcpip_hostname) +
    xmitq(AIX) +                                F
    replace
```

IBM MQ for Windows receiver-channel definitions using TCP: Windows

A code sample.

```
def ql (WINNT.LOCALQ) replace                   B

def chl (AIX.WINNT.TCP) chltype(rcvr) +         J
    trptype(tcp) +
    replace
```

Automatic startup: Windows

IBM MQ for Windows allows you to automate the startup of a queue manager and its channel initiator, channels, listeners, and command servers.

Use the IBM MQ Services snap-in to define the services for the queue manager. When you have successfully completed testing of your communications setup, set the relevant services to **automatic** within the snap-in. This file can be read by the supplied IBM MQ service when the system is started.

For more information, see Administering IBM MQ.

Running channels as processes or threads: Windows

IBM MQ for Windows provides the flexibility to run sending channels as Windows processes or Windows threads. This is specified in the MCATYPE parameter on the sender channel definition.

Most installations run their sending channels as threads, because the virtual and real memory required to support many concurrent channel connections is reduced. However, a NetBIOS connection needs a separate process for the sending Message Channel Agent.

Multiple thread support - pipelining: Windows

You can optionally allow a message channel agent (MCA) to transfer messages using multiple threads. This process, called *pipelining*, enables the MCA to transfer messages more efficiently, with fewer wait states, which improves channel performance. Each MCA is limited to a maximum of two threads.

You control pipelining with the *PipeLineLength* parameter in the qm.ini file. This parameter is added to the `CHANNELS` stanza:

PipeLineLength= 1 | *number*
This attribute specifies the maximum number of concurrent threads a channel uses. The default is 1. Any value greater than 1 is treated as 2.

With IBM MQ for Windows, use the IBM MQ Explorer to set the *PipeLineLength* parameter in the registry.

IBM i See The Channels stanza for a complete description of the `CHANNELS` stanza.

Note:

1. *PipeLineLength* applies only to V5.2 or later products.
2. Pipelining is effective only for TCP/IP channels.

When you use pipelining, the queue managers at both ends of the channel must be configured to have a *PipeLineLength* greater than 1.

Channel exit considerations

Pipelining can cause some exit programs to fail, because:

- Exits might not be called serially.
- Exits might be called alternately from different threads.

Check the design of your exit programs before you use pipelining:

- Exits must be reentrant at all stages of their execution.
- When you use MQI calls, remember that you cannot use the same MQI handle when the exit is invoked from different threads.

Consider a message exit that opens a queue and uses its handle for MQPUT calls on all subsequent invocations of the exit. This fails in pipelining mode because the exit is called from different threads. To avoid this failure, keep a queue handle for each thread and check the thread identifier each time the exit is invoked.

Example MQ configuration for z/OS

This section gives an example of how to set up communication links from IBM MQ for z/OS to IBM MQ products on other platforms.

These are the other platforms covered by this example:

- Windows
- AIX
- Compaq Tru64 UNIX
- HP-UX
- Solaris
- Linux
- IBM i
- VSE/ESA

You can also connect any of the following:

- z/OS to z/OS
- z/OS to MVS
- MVS to MVS

See "Example IBM MQ configuration for all platforms" on page 1 for background information about this section and how to use it.

Establishing a connection: z/OS

To establish a connection there are a number of things to configure.

Establishing an LU 6.2 connection

For the latest information about configuring SNA over TCP/IP, refer to the following online IBM documentation: Communications Server for z/OS.

Establishing a TCP connection

Alter the queue manager object to use the correct distributed queuing parameters using the following command. You must add the name of the TCP address space to the TCPNAME queue manager attribute.

```
ALTER QMGR TCPNAME(TCPIP)
```

The TCP connection is now established. You are ready to complete the configuration.

IBM MQ for z/OS configuration:

The following steps outline how to configure IBM MQ; starting and configuring channels and listeners.

1. Start the channel initiator using the command:
   ```
   /cpf   START CHINIT  1
   ```
2. Start an LU 6.2 listener using the command:
   ```
   /cpf   START LSTR LUNAME( M1 ) TRPTYPE(LU62)
   ```
   The LUNAME of M1 refers to the symbolic name you gave your LU (5). You must specify TRPTYPE(LU62), otherwise the listener assumes that you want TCP.
3. Start a TCP listener using the command:
   ```
   /cpf   START LSTR
   ```
   If you want to use a port other than 1414 (the default IBM MQ port), use the command:
   ```
   /cpf   START LSTR PORT( 1555 )
   ```

IBM MQ channels do not initialize successfully if the channel negotiation detects that the message sequence number is different at each end. You might need to reset these channels manually.

Channel configuration for z/OS:

To implement the example channels, there is some configuration necessary on the z/OS queue manager.

The following sections detail the configuration to be performed on the z/OS queue manager to implement the channel described in "Example IBM MQ configuration for all platforms" on page 1.

Examples are given for connecting IBM MQ for z/OS and IBM MQ for Windows. To connect to IBM MQ on another platform use the appropriate set of values from the table in place of the values for Windows.

Note: The words in **bold** are user-specified and reflect the names of IBM MQ objects used throughout these examples. If you change the names used here, ensure that you also change the other references made to these objects throughout this section. All others are keywords and must be entered as shown.

Table 8. Configuration worksheet for IBM MQ for z/OS

ID	Parameter Name	Reference	Example Used	User Value
Definition for local node				
A	Queue Manager Name		**MVS**	
B	Local queue name		**MVS.LOCALQ**	
Connection to IBM MQ for Windows The values in this section of the table must match the values used in "Channel configuration for Windows" on page 50, as indicated.				
C	Remote queue manager name	A	**WINNT**	
D	Remote queue name		**WINNT.REMOTEQ**	
E	Queue name at remote system	B	**WINNT.LOCALQ**	
F	Transmission queue name		**WINNT**	
G	Sender (LU 6.2) channel name		**MVS.WINNT.SNA**	
H	Sender (TCP) channel name		**MVS.WINNT.TCP**	
I	Receiver (LU 6.2) channel name	G	**WINNT.MVS.SNA**	
J	Receiver (TCP/IP) channel name	H	**WINNT.MVS.TCP**	

Table 8. Configuration worksheet for IBM MQ for z/OS (continued)

ID	Parameter Name	Reference	Example Used	User Value
Connection to IBM MQ for AIX The values in this section of the table must match the values used in "Channel configuration for AIX" on page 7, as indicated.				
C	Remote queue manager name	A	**AIX**	
D	Remote queue name		**AIX.REMOTEQ**	
E	Queue name at remote system	B	**AIX.LOCALQ**	
F	Transmission queue name		**AIX**	
G	Sender (LU 6.2) channel name		**MVS.AIX.SNA**	
H	Sender (TCP/IP) channel name		**MVS.AIX.TCP**	
I	Receiver (LU 6.2) channel name	G	**AIX.MVS.SNA**	
J	Receiver (TCP/IP) channel name	H	**AIX.MVS.TCP**	
Connection to MQSeries for Compaq Tru64 Unix The values in this section of the table must match the values used in your Compaq Tru64 UNIX system.				
C	Remote queue manager name	A	**DECUX**	
D	Remote queue name		**DECUX.REMOTEQ**	
E	Queue name at remote system	B	**DECUX.LOCALQ**	
F	Transmission queue name		**DECUX**	
H	Sender (TCP) channel name		**DECUX.MVS.TCP**	
J	Receiver (TCP) channel name	H	**MVS.DECUX.TCP**	
Connection to IBM MQ for HP-UX The values in this section of the table must match the values used in "Channel configuration for HP-UX" on page 13, as indicated.				
C	Remote queue manager name	A	**HPUX**	
D	Remote queue name		**HPUX.REMOTEQ**	
E	Queue name at remote system	B	**HPUX.LOCALQ**	
F	Transmission queue name		**HPUX**	
G	Sender (LU 6.2) channel name		**MVS.HPUX.SNA**	
H	Sender (TCP) channel name		**MVS.HPUX.TCP**	
I	Receiver (LU 6.2) channel name	G	**HPUX.MVS.SNA**	
J	Receiver (TCP) channel name	H	**HPUX.MVS.TCP**	
Connection to IBM MQ for Solaris The values in this section of the table must match the values used in "Channel configuration for Solaris" on page 43, as indicated.				
C	Remote queue manager name	A	**SOLARIS**	
D	Remote queue name		**SOLARIS.REMOTEQ**	
E	Queue name at remote system	B	**SOLARIS.LOCALQ**	
F	Transmission queue name		**SOLARIS**	
G	Sender (LU 6.2) channel name		**MVS.SOLARIS.SNA**	
H	Sender (TCP) channel name		**MVS.SOLARIS.TCP**	
I	Receiver (LU 6.2) channel name	G	**SOLARIS.MVS.SNA**	

Table 8. Configuration worksheet for IBM MQ for z/OS (continued)

ID	Parameter Name	Reference	Example Used	User Value
J	Receiver (TCP/IP) channel name	H	**SOLARIS.MVS.TCP**	
Connection to IBM MQ for Linux The values in this section of the table must match the values used in "Channel configuration for Linux" on page 38, as indicated.				
C	Remote queue manager name	A	**LINUX**	
D	Remote queue name		**LINUX.REMOTEQ**	
E	Queue name at remote system	B	**LINUX.LOCALQ**	
F	Transmission queue name		**LINUX**	
G	Sender (LU 6.2) channel name		**MVS.LINUX.SNA**	
H	Sender (TCP) channel name		**MVS.LINUX.TCP**	
I	Receiver (LU 6.2) channel name	G	**LINUX.MVS.SNA**	
J	Receiver (TCP/IP) channel name	H	**LINUX.MVS.TCP**	
Connection to IBM MQ for IBM i The values in this section of the table must match the values used in "Channel configuration for IBM i" on page 31, as indicated.				
C	Remote queue manager name	A	**AS400**	
D	Remote queue name		**AS400.REMOTEQ**	
E	Queue name at remote system	B	**AS400.LOCALQ**	
F	Transmission queue name		**AS400**	
G	Sender (LU 6.2) channel name		**MVS.AS400.SNA**	
H	Sender (TCP/IP) channel name		**MVS.AS400.TCP**	
I	Receiver (LU 6.2) channel name	G	**AS400.MVS.SNA**	
J	Receiver (TCP/IP) channel name	H	**AS400.MVS.TCP**	
Connection to MQSeries for VSE/ESA The values in this section of the table must match the values used in your VSE/ESA system.				
C	Remote queue manager name	A	**VSE**	
D	Remote queue name		**VSE.REMOTEQ**	
E	Queue name at remote system	B	**VSE.LOCALQ**	
F	Transmission queue name		**VSE**	
G	Sender channel name		**MVS.VSE.SNA**	
I	Receiver channel name	G	**VSE.MVS.SNA**	

IBM MQ for z/OS sender-channel definitions:

This topic details the sender-channel definitions required to configure IBM MQ for z/OS using LU 6.2 or TCP.

For LU 6.2:

```
     Local Queue
            Object type :   QLOCAL
                   Name :   WINNT              F
                  Usage :   X (XmitQ)

     Remote Queue
            Object type :   QREMOTE
                   Name :   WINNT.REMOTEQ      D
  Name on remote system :   WINNT.LOCALQ       E
     Remote system name :   WINNT              C
     Transmission queue :   WINNT              F

     Sender Channel
          Channel name :   MVS.WINNT.SNA      G
         Transport type :   L (LU6.2)
Transmission queue name :   WINNT              F
        Connection name :   M3                13
```

For TCP:

```
     Local Queue
            Object type :   QLOCAL
                   Name :   WINNT              F
                  Usage :   X (XmitQ)

     Remote Queue
            Object type :   QREMOTE
                   Name :   WINNT.REMOTEQ      D
  Name on remote system :   WINNT.LOCALQ       E
     Remote system name :   WINNT              C
     Transmission queue :   WINNT              F

     Sender Channel
          Channel name :   MVS.WINNT.TCP      H
         Transport type :   T (TCP)
Transmission queue name :   WINNT              F
        Connection name :   winnt.tcpip.hostname
```

IBM MQ for z/OS receiver-channel definitions:

This topic details the receiver-channel definitions required to configure IBM MQ for z/OS using LU6.2 or TCP.

For LU 6.2:

```
     Local Queue
            Object type :   QLOCAL
                   Name :   MVS.LOCALQ        B
                  Usage :   N (Normal)

     Receiver Channel
          Channel name :   WINNT.MVS.SNA     I
```

For TCP:

```
     Local Queue
            Object type :   QLOCAL
                   Name :   MVS.LOCALQ        B
```

```
                Usage :   N (Normal)

    Receiver Channel
          Channel name :   WINNT.MVS.TCP     J
```

Example MQ configuration for z/OS using QSGs

z/OS

This section gives an example of how to set up communication links to a queue-sharing group (QSG) from IBM MQ products on Windows and AIX. You can also connect from z/OS to z/OS.

Setting up communication links from a queue-sharing group to a platform other than z/OS is the same as described in "Example MQ configuration for z/OS" on page 56. There are examples to other platforms in that section.

When the connection is established, you must define some channels to complete the configuration. This process is described in "IBM MQ for z/OS shared channel configuration" on page 66.

See "Example IBM MQ configuration for all platforms" on page 1 for background information about this section and how to use it.

Configuration parameters for an LU 6.2 connection: z/OS

The following worksheet lists all the parameters required to set up communication from a z/OS system to one of the other IBM MQ platforms. The worksheet shows examples of the parameters, which have been tested in a working environment, and leaves space for you to enter your own values.

Use the worksheet in this section with the worksheet in the section for the platform to which you are connecting.

The steps required to set up an LU 6.2 connection are described in "Establishing an LU 6.2 connection into a queue-sharing group" on page 63, with numbered cross-references to the parameters on the worksheet.

Numbers in the Reference column indicate that the value must match that in the appropriate worksheet elsewhere in this section. The examples that follow in this section refer to the values in the ID column. The entries in the Parameter Name column are explained in "Explanation of terms" on page 62.

Table 9. Configuration worksheet for z/OS using LU 6.2

ID	Parameter Name	Reference	Example Used	User Value
Definition for local node using generic resources				
1	Command prefix		**/cpf**	
2	Network ID		**NETID**	
3	Node name		**MVSPU**	
6	Modename		**#INTER**	
7	Local Transaction Program name		**MQSERIES**	
8	LAN destination address		**400074511092**	
9	Local LU name		**MVSLU1**	
10	Generic resource name		**MVSGR**	
11	Symbolic destination		**G1**	
12	Symbolic destination for generic resource name		**G2**	

Table 9. Configuration worksheet for z/OS using LU 6.2 (continued)

ID	Parameter Name	Reference	Example Used	User Value
Connection to a Windows system				
13	Symbolic destination		**M3**	
14	Modename	21	**#INTER**	
15	Remote Transaction Program name	7	**MQSERIES**	
16	Partner LU name	5	**WINNTLU**	
21	Remote node ID	4	**05D 30F65**	
Connection to an AIX system				
13	Symbolic Destination		**M4**	
14	Modename	18	**#INTER**	
15	Remote Transaction Program name	6	**MQSERIES**	
16	Partner LU name	4	**AIXLU**	

Explanation of terms: z/OS

An explanation of the terms used in the configuration worksheet.

1 Command prefix
This term is the unique command prefix of your IBM MQ for z/OS queue manager subsystem. The z/OS system programmer defines this value at installation time, in SYS1.PARMLIB(IEFSSNss), and can tell you the value.

2 Network ID
The VTAM startup procedure in your installation is partly customized by the ATCSTRxx member of the data set referenced by the DDNAME VTAMLST. The Network ID is the value specified for the NETID parameter in this member. For Network ID, you must specify the name of the NETID that owns the IBM MQ communications subsystem. Your network administrator can tell you the value.

3 Node name
VTAM, being a low-entry network node, does not have a Control Point name for Advanced Peer-to-Peer Networking (APPN) use. It does however have a system services control point name (SSCPNAME). For node name, you must specify the name of the SSCP that owns the IBM MQ communications subsystem. This value is defined in the same ATCSTRxx member as the Network ID. Your network administrator can tell you the value.

9 Local LU name
A logical unit (LU) is software that serves as an interface or translator between a transaction program and the network. It manages the exchange of data between transaction programs. The local LU name is the unique VTAM APPLID of this IBM MQ subsystem. Your network administrator can tell you this value.

11 12 13 Symbolic destination
This term is the name you give to the CPI-C side information profile. You need a side information entry for each LU 6.2 listener.

6 14 Modename
This term is the name given to the set of parameters that control the LU 6.2 conversation. An entry with this name and similar attributes must be defined at each end of the session. In VTAM, this corresponds to a mode table entry. You network administrator can assign this table entry to you.

7 15 Transaction Program name
IBM MQ applications trying to converse with this queue manager specify a symbolic name for the program to be run at the receiving end. This has been specified in the TPNAME attribute on the channel definition at the sender. For simplicity, wherever possible use a transaction program name of MQSERIES, or in the case of a connection to VSE/ESA, where the length is limited to 4 bytes, use MQTP.

See Defining an LU6.2 connection for z/OS using APPC/MVS for more information.

8 LAN destination address
This term is the LAN destination address that your partner nodes use to communicate with this host. When you are using a 3745 network controller, it is the value specified in the LOCADD parameter for the line definition to which your partner is physically connected. If your partner nodes use other devices such as 317X or 6611 devices, the address is set during the customization of those devices. Your network administrator can tell you this value.

10 Generic resource name
A generic resource name is a unique name assigned to a group of LU names used by the channel initiators in a queue-sharing group.

16 Partner LU name
This term is the LU name of the IBM MQ queue manager on the system with which you are setting up communication. This value is specified in the side information entry for the remote partner.

21 Remote node ID
For a connection to Windows, this ID is the ID of the local node on the Windows system with which you are setting up communication.

Establishing an LU 6.2 connection into a queue-sharing group: z/OS

There are two steps to establish an LU 6.2 connection. Defining yourself to the network and defining a connection to the partner.

Defining yourself to the network using generic resources: z/OS

You can use VTAM Generic Resources to have one connection name to connect to the queue-sharing group.

1. SYS1.PARMLIB(APPCPMxx) contains the start-up parameters for APPC. You must add a line to this file to tell APPC where to locate the sideinfo. This line must be of the form:
   ```
   SIDEINFO
       DATASET(APPC.APPCSI)
   ```
2. Add another line to SYS1.PARMLIB(APPCPMxx) to define the local LU name you intend to use for the IBM MQ LU 6.2 group listener. The line you add must take the form
   ```
   LUADD ACBNAME(mvslu1)
         NOSCHED
         TPDATA(csq.appctp)
         GRNAME(mvsgr)
   ```
   Specify values for `ACBNAME` (9), `TPDATA` and `GRNAME(10)`.

   The NOSCHED parameter tells APPC that our new LU is not using the LU 6.2 scheduler (ASCH), but has one of its own. TPDATA refers to the Transaction Program data set in which LU 6.2 stores information about transaction programs. Again, IBM MQ does not use this parameter, but it is required by the syntax of the LUADD command.
3. Start the APPC subsystem with the command:
   ```
   START APPC,SUB=MSTR,APPC=xx
   ```
   where *xx* is the suffix of the PARMLIB member in which you added the LU in step 1.

Note: If APPC is already running, it can be refreshed with the command:

```
SET APPC=xx
```

The effect of this is cumulative, that is, APPC does not lose its knowledge of objects already defined to it in this member or another PARMLIB member.

4. Add the new LU to a suitable VTAM major node definition. These are typically in SYS1.VTAMLST. The APPL definition will look like the sample shown.

```
MVSLU APPL  ACBNAME=MVSLU1,     9
            APPXC=YES,
            AUTOSES=0,
            DDRAINL=NALLOW,
            DLOGMOD=#INTER,     6
            DMINWML=10,
            DMINWNR=10,
            DRESPL=NALLOW,
            DSESLIM=60,
            LMDENT=19,
            MODETAB=MTCICS,
            PARSESS=YES,
            VERIFY=NONE,
            SECACPT=ALREADYV,
            SRBEXIT=YES
```

5. Activate the major node. This activation can be done with the command:

```
V,NET,ACT,majornode
```

6. Add entries defining your LU and generic resource name to the CPI-C side information data set. Use the APPC utility program ATBSDFMU to do so. Sample JCL is in *thlqual*.SCSQPROC(CSQ4SIDE) (where *thlqual* is the target library high-level qualifier for IBM MQ data sets in your installation.)

   The entries you add will look like this example:

```
SIADD
     DESTNAME(G1)           11
     MODENAME(#INTER)
     TPNAME(MQSERIES)
     PARTNER_LU(MVSLU1)     9
SIADD
     DESTNAME(G2)           12
     MODENAME(#INTER)
     TPNAME(MQSERIES)
     PARTNER_LU(MVSGR)      10
```

7. Alter the queue manager object to use the correct distributed queuing parameters using the following command. You must specify the local LU (9) assigned to your queue manager in the LUGROUP attribute of the queue manager.

```
ALTER QMGR LUGROUP(MVSLU1)
```

Defining a connection to a partner: z/OS

You can define a connection to a partner by adding an entry to the CPI-C side information data set.

Note: This example is for a connection to a Windows system but the task is the same for other platforms.

Add an entry to the CPI-C side information data set to define the connection. Sample JCL to do this definition is in *thlqual*.SCSQPROC(CSQ4SIDE).

The entry you add will look like this:

```
SIADD
     DESTNAME(M3)           13
     MODENAME(#INTER)       14
     TPNAME(MQSERIES)       15
     PARTNER_LU(WINNTLU)    16
```

What next?: z/OS

The connection is now established. You are ready to complete the configuration.

Go to "IBM MQ for z/OS shared channel configuration" on page 66.

Establishing a TCP connection Using Sysplex Distributor: z/OS

You can set up Sysplex distributor to use one connection name to connect to the queue-sharing group.

1. Define a Distributed DVIPA address as follows:
   a. Add a DYNAMICXCF statement to the IPCONFIG. This statement is used for inter-image connectivity using dynamically created XCF TCP/IP links.
   b. Use the VIPADYNAMIC block on each image in the Sysplex.
      1) On the owning image, code a VIPADEFINE statement to create the DVIPA Then code a VIPADISTRIBUTE statement to distribute it to all other or selected images.
      2) On the backup image, code a VIPABACKUP statement for the DVIPA address.
2. If more than one channel initiator will be started on any LPAR in the sysplex then add the SHAREPORT option for the port to be shared in the PORT reservation list in the PROFILE data set.

See *z/OS CS: IP Configuration Guide* and *z/OS CS: IP Configuration Reference* for more information.

Sysplex Distributor balances the inbound connections between each LPAR. If there is more than one channel initiator on an LPAR, then the use of SHAREPORT passes that inbound connection to the listener port with the smallest number of connections.

When you have completed these steps, the TCP connection is established. You are ready to complete the configuration.

Go to "IBM MQ for z/OS shared channel configuration" on page 66.

IBM MQ for z/OS shared channel configuration: z/OS

Configure the shared channel by starting the channel initiator and issuing appropriate commands for your configuration.

1. Start the channel initiator using the command:
   ```
   /cpf   START CHINIT
   ```
2. Start an LU6.2 group listener using the command:
   ```
   /cpf   START LSTR TRPTYPE(LU62) LUNAME( G1 ) INDISP(GROUP)
   ```
   The LUNAME of G1 refers to the symbolic name you gave your LU (11).
3. If you are using Virtual IP Addressing using Sysplex Distributor and want to listen on a specific address, use the command:
   ```
   /cpf   START LSTR TRPTYPE(TCP) PORT(1555) IPADDR( mvsvipa ) INDISP(GROUP)
   ```

There can be only one instance of the shared channel running at a time. If you try to start a second instance of the channel it fails (the error message varies depending on other factors). The shared synchronization queue tracks the channel status.

IBM MQ channels do not initialize successfully if the channel negotiation detects that the message sequence number is different at each end. You might need to reset this manually.

Shared channel configuration example: z/OS

To configure a shared channel, a number of steps must be completed.

The subsequent topics detail the configuration to be performed on the z/OS queue manager to implement the channel described in "Example IBM MQ configuration for all platforms" on page 1.

Examples are given for connecting IBM MQ for z/OS and Windows. To connect to IBM MQ on another platform use the appropriate set of values from the table in place of the values for Windows.

Note: The words in **bold** are user-specified and reflect the names of IBM MQ objects used throughout these examples. If you change the names used here, ensure that you also change the other references made to these objects throughout this section. All others are keywords and must be entered as shown.

Table 10. Configuration worksheet for IBM MQ for z/OS using queue-sharing groups

ID	Parameter Name	Reference	Example Used	User Value
Definition for local node				
A	Queue Manager Name		**QSG**	
B	Local queue name		**QSG.SHAREDQ**	
Connection to IBM MQ for Windows The values in this section of the table must match the values used in "Channel configuration for Windows" on page 50, as indicated.				
C	Remote queue manager name	A	**WINNT**	
D	Remote queue name		**WINNT.REMOTEQ**	
E	Queue name at remote system	B	**WINNT.LOCALQ**	
F	Transmission queue name		**WINNT**	
G	Sender (LU 6.2) channel name		**QSG.WINNT.SNA**	
H	Sender (TCP) channel name		**QSG.WINNT.TCP**	
I	Receiver (LU 6.2) channel name	G	**WINNT.QSG.SNA**	

Table 10. Configuration worksheet for IBM MQ for z/OS using queue-sharing groups (continued)

ID	Parameter Name	Reference	Example Used	User Value
J	Receiver (TCP/IP) channel name	H	**WINNT.QSG.TCP**	
Connection to IBM MQ for AIX The values in this section of the table must match the values used in "Channel configuration for AIX" on page 7, as indicated.				
C	Remote queue manager name		**AIX**	
D	Remote queue name		**AIX.REMOTEQ**	
E	Queue name at remote system	B	**AIX.LOCALQ**	
F	Transmission queue name		**AIX**	
G	Sender (LU 6.2) channel name		**QSG.AIX.SNA**	
H	Sender (TCP/IP) channel name		**QSG.AIX.TCP**	
I	Receiver (LU 6.2) channel name	G	**AIX.QSG.SNA**	
J	Receiver (TCP/IP) channel name	H	**AIX.QSG.TCP**	

IBM MQ for z/OS shared sender-channel definitions: z/OS

An example definition of shared sender-channels for LU 6.2 and TCP.

Using LU 6.2

```
Local Queue
            Object type :   QLOCAL
                   Name :   WINNT              F
                  Usage :   X (XmitQ)
            Disposition :   SHARED

     Remote Queue
            Object type :   QREMOTE
                   Name :   WINNT.REMOTEQ      D
  Name on remote system :   WINNT.LOCALQ       E
     Remote system name :   WINNT              C
     Transmission queue :   WINNT              F
            Disposition :   GROUP

     Sender Channel
          Channel name :   MVS.WINNT.SNA      G
        Transport type :   L (LU6.2)
Transmission queue name :   WINNT              F
        Connection name :   M3                13
            Disposition :   GROUP
```

Using TCP

```
     Local Queue
            Object type :   QLOCAL
                   Name :   WINNT              F
                  Usage :   X (XmitQ)
            Disposition :   SHARED

     Remote Queue
            Object type :   QREMOTE
                   Name :   WINNT.REMOTEQ      D
  Name on remote system :   WINNT.LOCALQ       E
     Remote system name :   WINNT              C
     Transmission queue :   WINNT              F
            Disposition :   GROUP
```

```
     Sender Channel
           Channel name :   QSG.WINNT.TCP      H
         Transport type :   T (TCP)
Transmission queue name :   WINNT              F
        Connection name :   winnt.tcpip.hostname
            Disposition :   GROUP
```

IBM MQ for z/OS shared receiver-channel definitions: z/OS

An example definition of shared receiver-channels for LU 6.2 and TCP.

Using LU 6.2

```
     Local Queue
            Object type :   QLOCAL
                   Name :   QSG.SHAREDQ        B
                  Usage :   N (Normal)
            Disposition :   SHARED

     Receiver Channel
           Channel name :   WINNT.QSG.SNA      I
            Disposition :   GROUP
```

Using TCP

```
     Local Queue
            Object type :   QLOCAL
                   Name :   QSG.SHAREDQ        B
                  Usage :   N (Normal)
            Disposition :   SHARED

     Receiver Channel
           Channel name :   WINNT.QSG.TCP      J
            Disposition :   GROUP
```

Example MQ configuration for z/OS using intra-group queuing

z/OS

This section describes how a typical payroll query application, that currently uses distributed queuing to transfer small messages between queue managers, could be migrated to use queue-sharing groups and shared queues.

Three configurations are described to illustrate the use of distributed queuing, intra-group queuing with shared queues, and shared queues. The associated diagrams show only the flow of data in one direction, that is, from queue manager QMG1 to queue manager QMG3.

Configuration 1: z/OS

Configuration 1 describes how distributed queuing is currently used to transfer messages between queue managers QMG1 and QMG3.

Configuration 1 shows a distributed queuing system that is used to transfer messages received by queue manager QMG1 from the payroll query to queue manager QMG2 and then finally on to queue manager QMG3, to be sent to the payroll server.

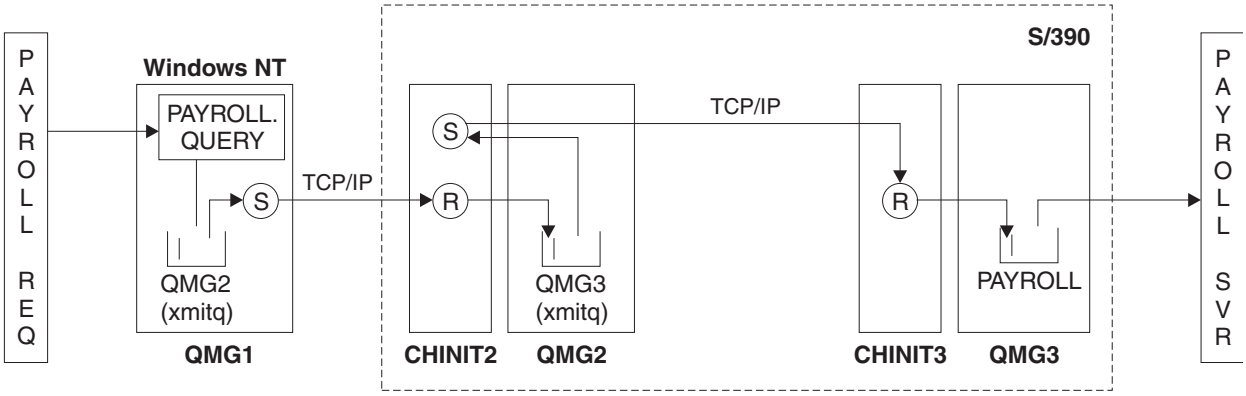


Figure 2. Configuration 1: z/OS using intra-group queuing

The flow of operations is as follows:

1. A query is entered using the payroll request application connected to queue manager QMG1.
2. The payroll request application puts the query on to remote queue PAYROLL.QUERY. As queue PAYROLL.QUERY resolves to transmission queue QMG2, the query is put on to transmission queue QMG2.
3. Sender channel (S) on queue manager QMG1 delivers the query to the partner receiver channel (R) on queue manager QMG2.
4. Receiver channel (R) on queue manager QMG2 puts the query on to queue PAYROLL on queue manager QMG3. As queue PAYROLL on QMG3 resolves to transmission queue QMG3, the query is put on to transmission queue QMG3.
5. Sender channel (S) on queue manager QMG2 delivers the query to the partner receiver channel (R) on queue manager QMG3.
6. Receiver channel (R) on queue manager QMG3 puts the query on to local queue PAYROLL.
7. The payroll server application connected to queue manager QMG3 retrieves the query from local queue PAYROLL, processes it, and generates a suitable reply.

Configuration 1 definitions: z/OS

The definitions required for Configuration 1 are as follows (note that the definitions do not take into account triggering, and that only channel definitions for communication using TCP/IP are provided).

On QMG1

Remote queue definition:

```
DEFINE QREMOTE(PAYROLL.QUERY) DESCR('Remote queue for QMG3') REPLACE +
PUT(ENABLED) RNAME(PAYROLL) RQMNAME(QMG3) XMITQ(QMG2)
```

Transmission queue definition:

```
DEFINE QLOCAL(QMG2) DESCR('Transmission queue to QMG2') REPLACE +
PUT(ENABLED) USAGE(XMITQ) GET(ENABLED)
```

Sender channel definition (for TCP/IP):

```
DEFINE CHANNEL(QMG1.TO.QMG2) CHLTYPE(SDR) TRPTYPE(TCP) REPLACE +
DESCR('Sender channel to QMG2') XMITQ(QMG2) CONNAME('MVSQMG2(1415)')
```

Here you replace MVSQMG2(1415) with your queue manager connection name and port.

Receiver channel definition (for TCP/IP):

```
DEFINE CHANNEL(QMG2.TO.QMG1) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QMG2')
```

Reply-to queue definition:

```
DEFINE QLOCAL(PAYROLL.REPLY) REPLACE PUT(ENABLED) GET(ENABLED) +
DESCR('Reply queue for replies to payroll queries sent to QMG3')
```

On QMG2

Transmission queue definition:

```
DEFINE QLOCAL(QMG1) DESCR('Transmission queue to QMG1') REPLACE +
PUT(ENABLED) USAGE(XMITQ) GET(ENABLED)

DEFINE QLOCAL(QMG3) DESCR('Transmission queue to QMG3') REPLACE +
PUT(ENABLED) USAGE(XMITQ) GET(ENABLED)
```

Sender channel definitions (for TCP/IP):

```
DEFINE CHANNEL(QMG2.TO.QMG1) CHLTYPE(SDR) TRPTYPE(TCP) REPLACE +
DESCR('Sender channel to QMG1') XMITQ(QMG1) CONNAME('WINTQMG1(1414)')
```

Here you replace WINTQMG1(1414) with your queue manager connection name and port.

```
DEFINE CHANNEL(QMG2.TO.QMG3) CHLTYPE(SDR) TRPTYPE(TCP) REPLACE +
DESCR('Sender channel to QMG3') XMITQ(QMG3) CONNAME('MVSQMG3(1416)')
```

Here you replace MVSQMG3(1416) with your queue manager connection name and port.

Receiver channel definition (for TCP/IP):

```
DEFINE CHANNEL(QMG1.TO.QMG2) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QMG1')

DEFINE CHANNEL(QMG3.TO.QMG2) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QMG3')
```

On QMG3

Local queue definition:

```
DEFINE QLOCAL(PAYROLL) DESCR('Payroll query request queue') REPLACE +
PUT(ENABLED) USAGE(NORMAL) GET(ENABLED) SHARE

DEFINE QLOCAL(QMG2) DESCR('Transmission queue to QMG2') REPLACE +
PUT(ENABLED) USAGE(XMITQ) GET(ENABLED)
```

Sender channel definitions (for TCP/IP):

```
DEFINE CHANNEL(QMG3.TO.QMG2) CHLTYPE(SDR) TRPTYPE(TCP) REPLACE +
DESCR('Sender channel to QMG2) XMITQ(QMG2) CONNAME('MVSQMG2(1415)')
```

Here you replace MVSQMG2(1415) with your queue manager connection name and port.

Receiver channel definition (for TCP/IP):

```
DEFINE CHANNEL(QMG2.TO.QMG3) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QMG2)
```

Configuration 2: z/OS

Configuration 2 describes how queue-sharing groups and intra-group queuing can be used, with no effect on the back-end payroll server application, to transfer messages between queue managers QMG1 and QMG3.

Configuration 2 shows a distributed queuing system that uses queue-sharing groups and intra-group queuing to transfer messages from the payroll request application to the payroll server. This configuration removes the need for channel definitions between queue managers QMG2 and QMG3 because intra-group queuing is used to transfer messages between these two queue managers.

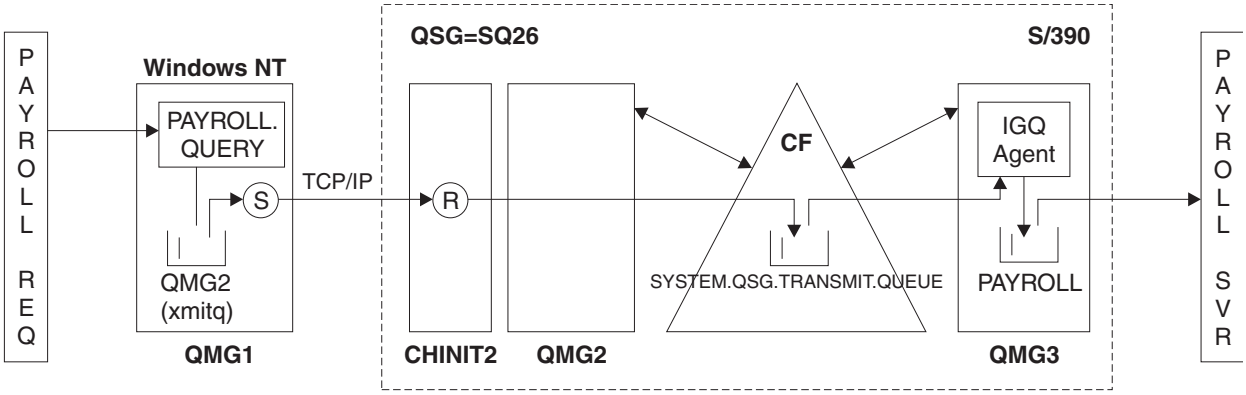


Figure 3. Configuration 2

The flow of operations is as follows:

1. A query is entered using the payroll request application connected to queue manager QMG1.
2. The payroll request application puts the query on to remote queue PAYROLL.QUERY. As queue PAYROLL.QUERY resolves to transmission queue QMG2, the query is put on to transmission queue QMG2.
3. Sender channel (S) on queue manager QMG1 delivers the query to the partner receiver channel (R) on queue manager QMG2.

4. Receiver channel (R) on queue manager QMG2 puts the query on to queue PAYROLL on queue manager QMG3. As queue PAYROLL on QMG3 resolves to shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE, the query is put on to shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
5. IGQ agent on queue manager QMG3 retrieves the query from shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE, and puts it on to local queue PAYROLL on queue manager QMG3.
6. The payroll server application connected to queue manager QMG3 retrieves the query from local queue PAYROLL, processes it, and generates a suitable reply.

Note: The payroll query example transfers small messages only. If you need to transfer both persistent and non-persistent messages, a combination of Configuration 1 and Configuration 2 can be established, so that large messages can be transferred using the distributed queuing route, while small messages can be transferred using the potentially faster intra-group queuing route.

Configuration 2 definitions: z/OS

The definitions required for Configuration 2 are as follows (note that the definitions do not take into account triggering, and that only channel definitions for communication using TCP/IP are provided).

It is assumed that queue managers QMG2 and QMG3 are already configured to be members of the same queue-sharing group.

On QMG1

Remote queue definition:

```
DEFINE QREMOTE(PAYROLL.QUERY) DESCR('Remote queue for QMG3') REPLACE +
PUT(ENABLED) RNAME(PAYROLL) RQMNAME(QMG3) XMITQ(QMG2)
```

Transmission queue definition:

```
DEFINE QLOCAL(QMG2) DESCR('Transmission queue to QMG2') REPLACE +
PUT(ENABLED) USAGE(XMITQ) GET(ENABLED)
```

Sender channel definition (for TCP/IP):

```
DEFINE CHANNEL(QMG1.TO.QMG2) CHLTYPE(SDR) TRPTYPE(TCP) REPLACE +
DESCR('Sender channel to QMG2') XMITQ(QMG2) CONNAME('MVSQMG2(1415)')
```

Here you replace MVSQMG2(1415) with your queue manager connection name and port.

Receiver channel definition (for TCP/IP):

```
DEFINE CHANNEL(QMG2.TO.QMG1) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QMG2')
```

Reply-to queue definition:

```
DEFINE QLOCAL(PAYROLL.REPLY) REPLACE PUT(ENABLED) GET(ENABLED) +
DESCR('Reply queue for replies to payroll queries sent to QMG3')
```

On QMG2

Transmission queue definition:

```
DEFINE QLOCAL(QMG1) DESCR('Transmission queue to QMG1') REPLACE +
PUT(ENABLED) USAGE(XMITQ) GET(ENABLED)

DEFINE QLOCAL(SYSTEM.QSG.TRANSMIT.QUEUE) QSGDISP(SHARED) +
DESCR('IGQ Transmission queue') REPLACE PUT(ENABLED) USAGE(XMITQ) +
GET(ENABLED) INDXTYPE(CORRELID) CFSTRUCT('APPLICATION1') +
DEFSOPT(SHARED) DEFPSIST(NO)
```

Here you replace APPLICATION1 with your defined CF structure name. Also note that this queue, being a shared queue, need only be defined on one of the queue managers in the queue-sharing group.

Sender channel definitions (for TCP/IP):

```
DEFINE CHANNEL(QMG2.TO.QMG1) CHLTYPE(SDR) TRPTYPE(TCP) REPLACE +
DESCR('Sender channel to QMG1') XMITQ(QMG1) CONNAME('WINTQMG1(1414)')
```

Here you replace WINTQMG1(1414) with your queue manager connection name and port.

Receiver channel definition (for TCP/IP):

```
DEFINE CHANNEL(QMG1.TO.QMG2) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QMG1')
```

Queue Manager definition:

```
ALTER QMGR IGQ(ENABLED)
```

On QMG3

Local queue definition:

```
DEFINE QLOCAL(PAYROLL) DESCR('Payroll query request queue') REPLACE +
PUT(ENABLED) USAGE(NORMAL) GET(ENABLED) SHARE
```

Queue Manager definition:

```
ALTER QMGR IGQ(ENABLED)
```

Configuration 3: z/OS

Configuration 3 describes how queue-sharing groups and shared queues can be used, with no effect on the back-end payroll server application, to transfer messages between queue managers QMG1 and QMG3.

Configuration 3 shows a distributed queuing system that uses queue-sharing groups and shared queues to transfer messages between queue manager QMG1 and queue manager QMG3.

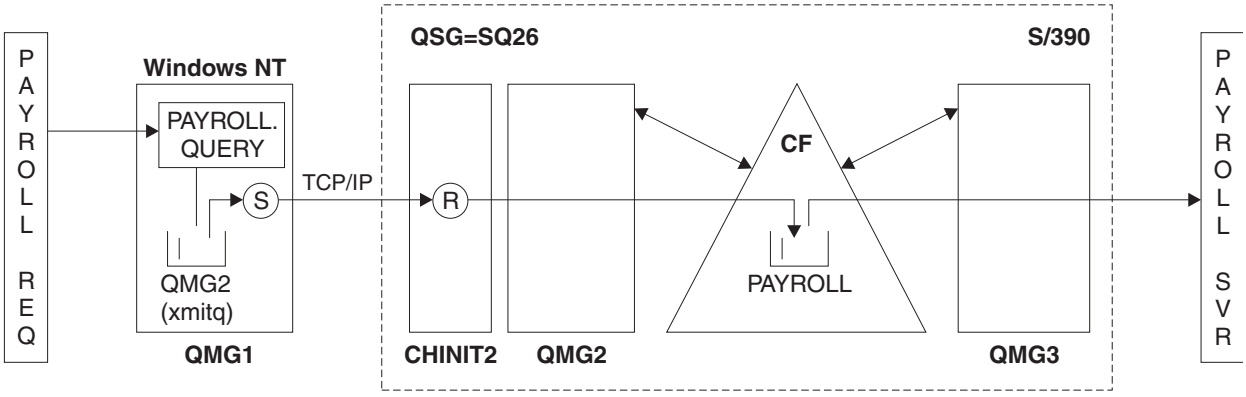


Figure 4. Configuration 3

The flow of operations is:

1. A query is entered using the payroll request application connected to queue manager QMG1.
2. The payroll request application puts the query on to remote queue PAYROLL.QUERY. As queue PAYROLL.QUERY resolves to transmission queue QMG2, the query is put on to transmission queue QMG2.

3. Sender channel (S) on queue manager QMG1 delivers the query to the partner receiver channel (R) on queue manager QMG2.
4. Receiver channel (R) on queue manager QMG2 puts the query on to shared queue PAYROLL.
5. The payroll server application connected to queue manager QMG3 retrieves the query from shared queue PAYROLL, processes it, and generates a suitable reply.

This configuration is certainly the simplest to configure. However, distributed queuing or intra-group queuing would need to be configured to transfer replies (generated by the payroll server application connected to queue manager QMG3) from queue manager QMG3 to queue manager QMG2, and then on to queue manager QMG1. (See "What the queue-sharing group example for z/OS shows" on page 175 for the configuration used to transfer replies back to the payroll request application.)

No definitions are required on QMG3.

Configuration 3 definitions: z/OS

The definitions required for Configuration 3 are as follows (note that the definitions do not take into account triggering, and that only channel definitions for communication using TCP/IP are provided).

It is assumed that queue managers QMG2 and QMG3 are already configured to be members of the same queue-sharing group.

On QMG1

Remote queue definition:

```
DEFINE QREMOTE(PAYROLL.QUERY) DESCR('Remote queue for QMG3') REPLACE +
PUT(ENABLED) RNAME(PAYROLL) RQMNAME(QMG3) XMITQ(QMG2)
```

Transmission queue definition:

```
DEFINE QLOCAL(QMG2) DESCR('Transmission queue to QMG2') REPLACE +
PUT(ENABLED) USAGE(XMITQ) GET(ENABLED)
```

Sender channel definition (for TCP/IP):

```
DEFINE CHANNEL(QMG1.TO.QMG2) CHLTYPE(SDR) TRPTYPE(TCP) +
REPLACE DESCR('Sender channel to QMG2') XMITQ(QMG2) CONNAME('MVSQMG2(1415)')
```

Here you replace MVSQMG2(1415) with your queue manager connection name and port.

Receiver channel definition (for TCP/IP):

```
DEFINE CHANNEL(QMG2.TO.QMG1) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QMG2')
```

Reply-to queue definition:

```
DEFINE QLOCAL(PAYROLL.REPLY) REPLACE PUT(ENABLED) GET(ENABLED) +
DESCR('Reply queue for replies to payroll queries sent to QMG3')
```

On QMG2

Transmission queue definition:

```
DEFINE QLOCAL(QMG1) DESCR('Transmission queue to QMG1') REPLACE +
PUT(ENABLED) USAGE(XMITQ) GET(ENABLED)
```

Sender channel definitions (for TCP/IP):

```
DEFINE CHANNEL(QMG2.TO.QMG1) CHLTYPE(SDR) TRPTYPE(TCP) +
REPLACE DESCR('Sender channel to QMG1') XMITQ(QMG1) CONNAME('WINTQMG1(1414)')
```

Here you replace WINTQMG1(1414) with your queue manager connection name and port.

Receiver channel definition (for TCP/IP):

```
DEFINE CHANNEL(QMG1.TO.QMG2) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QMG1')
```

Local queue definition:

```
DEFINE QLOCAL(PAYROLL) QSGDISP(SHARED) DESCR('Payroll query request queue') +
REPLACE PUT(ENABLED) USAGE(NORMAL) GET(ENABLED) SHARE +
DEFSOPT(SHARED) DEFPSIST(NO) CFSTRUCT(APPLICATION1)
```

Here you replace APPLICATION1 with your defined CF structure name. Also note that this queue, being a shared queue, need only be defined on one of the queue managers in the queue-sharing group.

On QMG3

No definitions are required on QMG3.

Running the example: z/OS

After setting up the sample, you can run the sample.

For Configuration 1:

1. Start queue managers QMG1, QMG2, and QMG3.
2. Start channel initiators for QMG2 and QMG3.
3. Start the listeners on QMG1 to listen to port 1414, QMG2 to listen on port 1415, and QMG3 to listen on port 1416.
4. Start sender channels on QMG1, QMG2, and QMG3.
5. Start the payroll query requesting application connected to QMG1.
6. Start the payroll server application connected to QMG3.
7. Submit a payroll query request to QMG3 and wait for the payroll reply.

For Configuration 2:

1. Start queue managers QMG1, QMG2, and QMG3.
2. Start the channel initiator for QMG2.
3. Start the listeners on QMG1 to listen on port 1414, and QMG2 to listen on port 1415.
4. Start the sender channel on QMG1 and QMG2.
5. Start the payroll query requesting application connected to QMG1.
6. Start the payroll server application connected to QMG3.
7. Submit a payroll query request to QMG3 and wait for the payroll reply.

For Configuration 3:

1. Start queue managers QMG1, QMG2, and QMG3.
2. Start the channel initiator for QMG2.
3. Start the listeners on QMG1 to listen on port 1414, and QMG2 to listen on port 1415.
4. Start sender channels on QMG1 and QMG2.
5. Start the payroll query requesting application connected to QMG1.
6. Start the payroll server application connected to QMG3.
7. Submit a payroll query request to QMG3 and wait for the payroll reply.

Expanding the example: z/OS

The example can be expanded in a number of ways.

The example can be:

- Expanded to use channel triggering as well as application (PAYROLL and PAYROLL.REPLY queue) triggering.
- Configured for communication using LU6.2.
- Expanded to configure more queue managers to the queue-sharing group. Then the server application can be cloned to run on other queue manager instances to provide multiple servers for the PAYROLL query queue.
- Expanded to increase the number of instances of the payroll query requesting application to demonstrate the processing of requests from multiple clients.
- Expanded to use security (IGQAUT and IGQUSER).

IBM MQ file system permissions applied to `/var/mqm`

UNIX Linux IBM i

The following information describes the security applied to the files and directories under `/var/mqm/` and why the file-system permissions are set as they are. In order to ensure the correct operation of IBM MQ you should not alter the file system permissions as set by IBM MQ

IBM MQ file system Security on UNIX, Linux, and IBM i

The files under the IBM MQ data directory (`/var/mqm`) are used to store:

- IBM MQ configuration data
- Application data (IBM MQ objects and the data contained within IBM MQ messages)
- Run-time control information
- Monitoring information (messages and FFST files)

Access to this data is controlled using file system permissions with some of the data being accessible to all users while other data is restricted only to members of the IBM MQ Administrator group 'mqm' (or QMQM on IBM i).

Access is granted in the following three categories:

mqm group only

The files and directories in this category are only accessible to IBM MQ Administrators (members of the 'mqm' group) and the IBM MQ queue manager processes.

The file permissions for these files and directories are:

```
-rwxrwx---    mqm:mqm      (UNIX and Linux)
-rwxrwx---    QMQMADM:QMQM (IBM i)
```

An example of the files and directories in this category is:

```
/var/mqm/qmgrs/QMGR/qm.ini
/var/mqm/qmgrs/QMGR/channel/
/var/mqm/qmgrs/QMGR/channel/SYSTEM!DEF!SCRVONN
/var/mqm/qmgrs/QMGR/queues/
/var/mqm/qmgrs/QMGR/queues/SYSTEM!DEFAULT!LOCAL!QUEUES/
/var/mqm/qmgrs/QMGR/errors/
/var/mqm/qmgrs/QMGR/errors/AMQERR01.LOG
```

```
/var/mqm/qmgrs/QMGR/ssl/
/var/mqm/qmgrs/QMGR/@qmgr/
/var/mqm/qmgrs/QMGR/@qmpersist/
...
```

All users read access - mqm group members read and write access

The files and directories in this category can be read by all users, but only members of the 'mqm' group can modify these files and manipulate these directories.

The file permissions for these files and directories are:

```
-rwxrwxr-x    mqm:mqm      (UNIX and Linux)
-rwxrwxr-x    QMQMADM:QMQM (IBM i)
```

An example of the files and directories in this category is:

```
/var/mqm/mqs.ini
/var/mqm/exits/
/var/mqm/qmgrs/
/var/mqm/qmgrs/QMGR/
/var/mqm/qmgrs/QMGR/@app/
/var/mqm/qmgrs/QMGR/@ipcc/
```

IBM WebSphere® MQ Version 7.1 and Version 7.5, and IBM MQ Version 8.0:

```
/var/mqm/sockets/@SYSTEM
/var/mqm/sockets/QMGR/@app/hostname
/var/mqm/sockets/QMGR/@ipcc/hostname
```

All users read and write access

Files that have read and write access for all users

IBM MQ has no *regular* files that have world writable file permissions (777). However there are a number of *special* files that appear as having world writable file permissions.

These special files provide no security exposure. Although the permissions are shown as 777, they are not *regular* files and you cannot write directly to them.

These special files are:

Symbolic links

Symbolic links are identified by the 'l' character at the start of their permissions. The permissions on the symbolic link have no effect on who is able to access the target file, as access to the command is controlled by the permissions on the target of the symbolic link.

On most UNIX and Linux systems it is not possible to change the permissions on symbolic links, so they always appear as `lrwxrwxrwx`.

Socket files

Socket files are special files created by the operating system, as a result of a process creating a UNIX domain socket. These files can be identified by the 's' at the start of the file permissions, that is `srwxrwxrwx`.

The permissions on the file do not grant access to the file itself, but define who can connect to the UNIX domain socket.

IBM MQ uses a number of these socket files and the permissions are always set according to who is allowed to communicate with the socket.

The following directories contain socket files that have read/write permissions for all users (`srwxrwxrwx`).

IBM WebSphere MQ Version 7.x and IBM MQ Version 8.0:
`/var/mqm/sockets/QMGR/zsocketEC/hostname/Zsocket_*`

Socket files used by applications that connect to IBM MQ using isolated bindings.
`/var/mqm/sockets/QMGR/@ipcc/ssem/hostname/*`

Directories that have read and write access for all users

There are times when IBM MQ applications need to create files under the IBM MQ data directory. To ensure that applications are able to create files when they are required, a number of directories are granted world write access, which means that any user on the system can create files within that directory.

With the exception of the errors logs files, that can be written to by any member of the 'mqm' group, all files created in these directories are created with restricted permissions that allows only the file creator write access. This allows the system administrator to track the user ID of all data written to files in these directories.

/var/mqm/errors/
This directory contains the system error log files and FFST files. The permission of this directory is 'drwxrwsrwt' meaning that all users on the system can create files in this directory.

The SetGroupId bit 's' indicates that all files created in this directory have the group ownership of 'mqm'.

The 't' sticky bit is not set by default on this directory, but an IBM MQ administrator can set this explicitly, to allow users to delete only the files that they create.

Note: IBM i This feature is not available on IBM i.

AMQERR0*.LOG
These error log files can only be written to directly by members of the group but any user can read the messages written to these files (permission: -rw-rw-r--).

AMQnnnnn.*.FDC
These files contain FFST information written when an error occurs in the queue manager or in an application written by a user. These files are created with the permissions -rw-r-----.

/var/mqm/trace/
Trace files are written to this directory when IBM MQ trace is enabled. IBM MQ trace is written by all process associated with a queue manager for which trace is enabled.

The permissions of this directory are 'drwxrwsrwt' meaning that all users on the system can create files in this directory.

The SetGroupId bit 's' indicates that all files created in this directory have the group ownership of 'mqm'.

The 't' sticky bit is not set by default on this directory, but an IBM MQ administrator can set this explicitly, to allow users to delete only the files that they create.

Note: IBM i This feature is not available on IBM i.

AMQnnnnn.*.TRC
These files contain the trace data written by each process which is tracing and are created with permissions -rw-r-----

The permissions on this directory are drwxrwsrwt and the permissions of the socket files created in this directory are srwx------.

IBM WebSphere MQ Version 7.1 and Version 7.5, and IBM MQ Version 8.0:
`/var/mqm/sockets/QMGR/zsocketapp/hostname/`

This directory is used by applications that connect to the IBM MQ queue manager using *isolated* bindings. During connect processing a socket file is created by the connecting application in this directory. The socket file is removed after the connection is made to the queue manager.

The permissions on this directory are `drwxrwsrwt` and the permissions of the socket files created in this directory are `srwx------`.

The `SetGroupId` bit 's' on this directory ensures that all files created in this directory have the group ownership of 'mqm'.

On all platforms except IBM i, this directories also has the 't' sticky bit set which prevents a user from deleting any files except the ones for which they are the owner. This prevents an unauthorized user from deleting files that they do not own.

`/var/mqm/sockets/QMGR/@ipcc/ssem/hostname/`
`/var/mqm/sockets/QMGR/@app/ssem/hostname/`

UNIX For processes that connect to IBM MQ using *shared* bindings then UNIX domain sockets might be used to synchronize between the application and the queue manager. When UNIX domain sockets are being used then the associated socket file is created in these directories.

The permissions on these directories are `drwxrwsrwt` and the permissions of the socket files created in these directories are `srwxrwxrwx`.

The `SetGroupId` bit 's' on these directories ensures that all files created in these directories have the group ownership of 'mqm'.

On all platforms except IBM i, these directories also have the 't' sticky bit set which prevents a user from deleting any files except the ones for which they are the owner. This prevents an unauthorized user from deleting files that they do not own.

Use of System V IPC resources by IBM MQ

IBM MQ uses System V shared memory and semaphores for inter-process communication. These resources are grouped according to how they are used with each group having appropriate ownership and access permissions.

To verify which of the System V IPC resources on a system belong to IBM MQ you can:

- Check the ownership.

  The owning user of IBM MQ System V IPC resources is always the 'mqm' user on UNIX platforms and Linux. On IBM i the owning user is 'QMQM'.
- IBM WebSphere MQ Version 7.5 and later, use the `amqspdbg` utility.

  The `amqspdbg` utility which is shipped with IBM MQ can be used to display the shared memory and semaphore id's for a given queue manager.

  You must issue the command once for the 'system' group of System V resources created by IBM MQ

  ```
  # amqspbg -z -I
  ```

  and then four times for each queue manager on the system to get the complete list of System V resources used by IBM MQ. Assume a queue manager name of `QMGR1` in the following examples:.

  ```
  # amqspdbg -i QMGR1 -I
  # amqspdbg -q QMGR1 -I
  # amqspdbg -p QMGR1 -I
  # amqspdbg -a QMGR1 -I
  ```

The access permissions on the System V resources created by IBM MQ are set to grant only the correct level of access to the permitted users. A number of the System V IPC resources created by IBM MQ are accessible to all users on the machine and have permissions of `-rw-rw-rw-`.

The **-g** *ApplicationGroup* parameter on the crtmqm command can be used to restrict access to a queue manager to membership of a specific operating system group. The use of this restricted group functionality restricts the permissions granted on the System V IPC resources further.

Naming restrictions for queues

There are restrictions on the length of queue names. Some queue names are reserved for queues defined by the queue manager.

Restrictions on name lengths

Queues can have names up to 48 characters long.

Reserved queue names

Names that start with "SYSTEM." are reserved for queues defined by the queue manager. You can use the **ALTER** or **DEFINE REPLACE** commands to change these queue definitions to suit your installation. The following names are defined for IBM MQ:

Queue Name	Description
SYSTEM.ADMIN.ACTIVITY.QUEUE	Queue for activity reports
SYSTEM.ADMIN.CHANNEL.EVENT	Queue for channel events
SYSTEM.ADMIN.COMMAND.EVENT	Queue for command events
SYSTEM.ADMIN.COMMAND.QUEUE	Queue to which PCF command messages are sent
SYSTEM.ADMIN.CONFIG.EVENT	Queue for configuration events
SYSTEM.ADMIN.PERFM.EVENT	Queue for performance events
SYSTEM.ADMIN.PUBSUB.EVENT	System publish/subscribe related event queue
SYSTEM.ADMIN.QMGR.EVENT	Queue for queue manager events
SYSTEM.ADMIN.TRACE.ROUTE.QUEUE	Queue for trace-route reply messages
SYSTEM.AUTH.DATA.QUEUE	The queue that holds access control lists for the queue manager. (Not for z/OS)
SYSTEM.CHANNEL.INITQ	Initiation queue for channels
SYSTEM.CHANNEL.SYNCQ	The queue that holds the synchronization data for channels
SYSTEM.CHLAUTH.DATA.QUEUE	IBM MQ channel authentication data queue
SYSTEM.CICS.INITIATION.QUEUE	Queue used for triggering (not for z/OS)
SYSTEM.CLUSTER.COMMAND.QUEUE	Queue used to communicate repository changes between queue managers
SYSTEM.CLUSTER.HISTORY.QUEUE	The queue is used to store the history of cluster state information for service purposes.
SYSTEM.CLUSTER.REPOSITORY.QUEUE	Queue used to hold information about the repository
SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE	The queue is used to create individual transmit queues for each cluster-sender channel.
SYSTEM.CLUSTER.TRANSMIT.QUEUE	Transmission queue for all destinations managed by cluster support
SYSTEM.COMMAND.INPUT	Queue to which command messages are sent on z/OS
SYSTEM.COMMAND.REPLY.MODEL	Model queue definition for command replies (for z/OS)

Queue Name	Description
SYSTEM.DEAD.LETTER.QUEUE	Dead-letter queue (not for z/OS)
SYSTEM.DEFAULT.ALIAS.QUEUE	Default alias queue definition
SYSTEM.DEFAULT.INITIATION.QUEUE	Queue used to trigger a specified process (not for z/OS)
SYSTEM.DEFAULT.LOCAL.QUEUE	Default local queue definition
SYSTEM.DEFAULT.MODEL.QUEUE	Default model queue definition
SYSTEM.DEFAULT.REMOTE.QUEUE	Default remote queue definition
SYSTEM.DURABLE.SUBSCRIBER.QUEUE	A local queue used to hold a persistent copy of the durable subscriptions in the queue manager
SYSTEM.HIERARCHY.STATE	Queue used to hold information about the state of inter-queue manager relationships in a publish/subscribe hierarchy
SYSTEM.JMS.TEMPQ.MODEL	Model for JMS temporary queues
SYSTEM.INTERNAL.REPLY.QUEUE	IBM MQ internal reply queue (not for z/OS)
SYSTEM.INTER.QMGR.CONTROL	Queue used in a publish/subscribe hierarchy to receive requests from a remote queue manager to create a proxy subscription
SYSTEM.INTER.QMGR.PUBS	Queue used in a publish/subscribe hierarchy to receive publications from a remote queue manager
SYSTEM.INTER.QMGR.FANREQ	Queue used in a publish/subscribe hierarchy to process requests to create a proxy subscription on a remote queue manager
SYSTEM.MQEXPLORER.REPLY.MODEL	Model queue definition for replies for IBM MQ Explorer
SYSTEM.MQSC.REPLY.QUEUE	Model queue definition for MQSC command replies (not for z/OS)
SYSTEM.QSG.CHANNEL.SYNCQ	Shared local queue used for storing messages that contain the synchronization information for shared channels (z/OS only)
SYSTEM.QSG.TRANSMIT.QUEUE	Shared local queue used by the intra-group queuing agent when transmitting messages between queue managers in the same queue-sharing group (z/OS only)
SYSTEM.RETAINED.PUB.QUEUE	A local queue used to hold a copy of each retained publication in the queue manager.
SYSTEM.SELECTION.EVALUATION.QUEUE	IBM MQ internal selection evaluation queue (not for z/OS)
SYSTEM.SELECTION.VALIDATION.QUEUE	IBM MQ internal selection validation queue (not for z/OS)

Naming restrictions for other objects

There are restrictions on the length of object names. Some object names are reserved for objects defined by the queue manager.

Restrictions on name length

Processes, namelists, clusters, topics, services, and authentication information objects can have names up to 48 characters long.

Channels can have names up to 20 characters long.

Storage classes can have names up to 8 characters long.

CF structures can have names up to 12 characters long.

Reserved object names

Names that start with SYSTEM. are reserved for objects defined by the queue manager. You can use the **ALTER** or **DEFINE REPLACE** commands to change these object definitions to suit your installation. The following names are defined for IBM MQ:

Object Name	Description
SYSTEM.ADMIN.SVRCONN	Server-connection channel used for remote administration of a queue manager
SYSTEM.AUTO.RECEIVER	Default receiver channel for auto definition (UNIX, Linux, and Windows systems only)
SYSTEM.AUTO.SVRCONN	Default server-connection channel for auto definition (Multiplatforms only)
SYSTEM.BASE.TOPIC	Base topic for ASPARENT resolution. If a particular administrative topic object has no parent administrative topic objects, any ASPARENT attributes are inherited from this object
SYSTEM.DEF.CLNTCONN	Default client-connection channel definition
SYSTEM.DEF.CLUSRCVR	Default cluster-receiver channel definition
SYSTEM.DEF.CLUSSDR	Default cluster-sender channel definition
SYSTEM.DEF.RECEIVER	Default receiver channel definition
SYSTEM.DEF.REQUESTER	Default requester channel definition
SYSTEM.DEF.SENDER	Default sender channel definition
SYSTEM.DEF.SERVER	Default server channel definition
SYSTEM.DEF.SVRCONN	Default server-connection channel definition
SYSTEM.DEFAULT.AUTHINFO.CRLLDAP	Default authentication information object definition for defining authentication information objects of type CRLLDAP
SYSTEM.DEFAULT.AUTHINFO.OCSP	Default authentication information object definition for defining authentication information objects of type OCSP
SYSTEM.DEFAULT.LISTENER.LU62	Default SNA listener (Windows only)
SYSTEM.DEFAULT.LISTENER.NETBIOS	Default NetBIOS listener (Windows only)
SYSTEM.DEFAULT.LISTENER.SPX	Default SPX listener (Windows only)
SYSTEM.DEFAULT.LISTENER.TCP	Default TCP/IP listener (Multiplatforms only)
SYSTEM.DEFAULT.NAMELIST	Default namelist definition
SYSTEM.DEFAULT.PROCESS	Default process definition

Object Name	Description
SYSTEM.DEFAULT.SEVICE	Default service (Multiplatforms only)
SYSTEM.DEFAULT.TOPIC	Default topic definition
SYSTEM.QPUBSUB.QUEUE.NAMELIST	A list of queues for the Queued Publish/Subscribe interface to monitor
z/OS SYSTEMST	Default storage class definition (z/OS only)

Queue name resolution

This topic contains information about queue name resolution as performed by queue managers at both sending and receiving ends of a channel.

In larger networks, the use of queue managers has a number of advantages over other forms of communication. These advantages derive from the name resolution function in DQM and the main benefits are:

- Applications do not need to make routing decisions
- Applications do not need to know the network structure
- Network links are created by systems administrators
- Network structure is controlled by network planners
- Multiple channels can be used between nodes to partition traffic

The following figure shows an example of queue name resolution. The figure shows two machines in a network, one running a put application, the other running a get application. The applications communicate with each other though the IBM MQ channel, controlled by the MCAs. As far as the application is concerned, the process is the same as putting messages on a local queue.

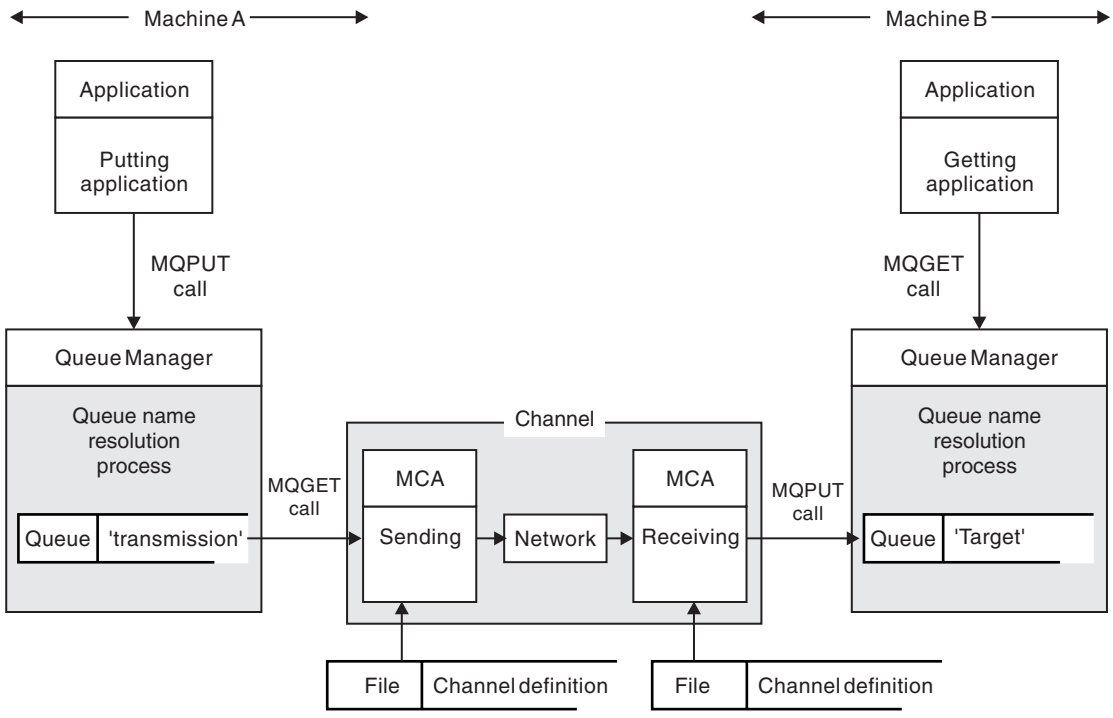


Figure 5. Name resolution

Referring to Figure 5 on page 83, the basic mechanism for putting messages on a remote queue, as far as the application is concerned, is the same as for putting messages on a local queue:

- The application putting the message issues MQOPEN and MQPUT calls to put messages on the target queue.
- The application getting the messages issues MQOPEN and MQGET calls to get the messages from the target queue.

If both applications are connected to the same queue manager then no inter-queue manager communication is required, and the target queue is described as *local* to both applications.

However, if the applications are connected to different queue managers, two MCAs and their associated network connection are involved in the transfer, as shown in the figure. In this case, the target queue is considered to be a *remote queue* to the putting application.

The sequence of events is as follows:

1. The putting application issues MQOPEN and MQPUT calls to put messages to the target queue.
2. During the MQOPEN call, the *name resolution* function detects that the target queue is not local, and decides which transmission queue is appropriate. Thereafter, on the MQPUT calls associated with the MQOPEN call, all messages are placed on this transmission queue.
3. The sending MCA gets the messages from the transmission queue and passes them to the receiving MCA at the remote computer.
4. The receiving MCA puts the messages on the target queue, or queues.
5. The getting application issues MQOPEN and MQGET calls to get the messages from the target queue.

Note: Only step 1 and step 5 involve application code; steps 2 through 4 are performed by the local queue managers and the MCA programs. The putting application is unaware of the location of the target queue, which could be in the same processor, or in another processor on another continent.

The combination of sending MCA, the network connection, and the receiving MCA, is called a *message channel*, and is inherently a unidirectional device. Normally, it is necessary to move messages in both directions, and two channels are set up for this movement, one in each direction.

What is queue name resolution?

Queue name resolution is vital to DQM. It removes the need for applications to be concerned with the physical location of queues, and insulates them against the details of networks.

A systems administrator can move queues from one queue manager to another, and change the routing between queue managers without applications needing to know anything about it.

In order to uncouple from the application design the exact path over which the data travels, it is necessary to introduce a level of indirection between the name used by the application when it refers to the target queue, and the naming of the channel over which the flow occurs. This indirection is achieved using the queue name resolution mechanism.

In essence, when an application refers to a queue name, the name is mapped by the resolution mechanism either to a transmission queue or to a local queue that is not a transmission queue. For mapping to a transmission queue, a second name resolution is needed at the destination, and the received message is placed on the target queue as intended by the application designer. The application remains unaware of the transmission queue and channel used for moving the message.

Note: The definition of the queue and channel is a system management responsibility and can be changed by an operator or a system management utility, without the need to change applications.

An important requirement for the system management of message flows is that alternative paths need to be provided between queue managers. For example, business requirements might dictate that different *classes of service* are sent over different channels to the same destination. This decision is a system management decision and the queue name resolution mechanism provides a flexible way to achieve it. The Application Programming Guide describes this in detail, but the basic idea is to use queue name resolution at the sending queue manager to map the queue name supplied by the application to the appropriate transmission queue for the type of traffic involved. Similarly at the receiving end, queue name resolution maps the name in the message descriptor to a local (not a transmission) queue or again to an appropriate transmission queue.

Not only is it possible for the forward path from one queue manager to another to be partitioned into different types of traffic, but the return message that is sent to the reply-to queue definition in the outbound message can also use the same traffic partitioning. Queue name resolution satisfies this requirement and the application designer need not be involved in these traffic partitioning decisions.

The point that the mapping is carried out at both the sending and receiving queue managers is an important aspect of the way name resolution works. This mapping allows the queue name supplied by the putting application to be mapped to a local queue or a transmission queue at the sending queue manager, and again remapped to a local queue or a transmission queue at the receiving queue manager.

Reply messages from receiving applications or MCAs have the name resolution carried out in the same way, allowing return routing over specific paths with queue definitions at all the queue managers on route.

System and default objects

Lists the system and default objects created by the **crtmqm** command.

When you create a queue manager using the **crtmqm** control command, the system objects and the default objects are created automatically.

- The system objects are those IBM MQ objects needed to operate a queue manager or channel.
- The default objects define all the attributes of an object. When you create an object, such as a local queue, any attributes that you do not specify explicitly are inherited from the default object.

The following tables list the system and default objects created by **crtmqm**:

- Table 11 on page 86 lists the system and default queue objects.
- Table 12 on page 87 lists the system and default topic objects.
- Table 13 on page 87 lists the system and default channel objects.
- Table 14 on page 87 lists the system and default authentication information objects.
- Table 15 on page 87 lists the system and default listener objects.
- Table 16 on page 88 lists the system and default namelist objects.
- Table 17 on page 88 lists the system and default process objects.
- Table 18 on page 88 lists the system and default service objects.

Table 11. System and default objects: queues

Object name	Description
SYSTEM.ADMIN.ACCOUNTING.QUEUE	The queue that holds accounting monitoring data.
SYSTEM.ADMIN.ACTIVITY.QUEUE	The queue that holds returned activity reports.
SYSTEM.ADMIN.CHANNEL.EVENT	Event queue for channels.
SYSTEM.ADMIN.COMMAND.EVENT	Event queue for command events.
SYSTEM.ADMIN.COMMAND.QUEUE	Administration command queue. Used for remote MQSC commands and PCF commands.
SYSTEM.ADMIN.CONFIG.EVENT	Event queue for configuration events.
SYSTEM.ADMIN.PERFM.EVENT	Event queue for performance events.
SYSTEM.ADMIN.PUBSUB.EVENT	System publish/subscribe related event queue
SYSTEM.ADMIN.QMGR.EVENT	Event queue for queue manager events.
SYSTEM.ADMIN.STATISTICS.QUEUE	The queue that holds statistics monitoring data.
SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE	The queue that displays trace activity.
SYSTEM.ADMIN.TRACE.ROUTE.QUEUE	The queue that holds returned trace-route reply messages.
SYSTEM.AUTH.DATA.QUEUE	The queue that holds access control lists for the queue manager.
SYSTEM.CHANNEL.INITQ	Channel initiation queue.
SYSTEM.CHANNEL.SYNCQ	The queue that holds the synchronization data for channels.
SYSTEM.CHLAUTH.DATA.QUEUE	IBM MQ channel authentication data queue
SYSTEM.CICS.INITIATION.QUEUE	Default CICS initiation queue.
SYSTEM.CLUSTER.COMMAND.QUEUE	The queue used to carry messages to the repository queue manager.
SYSTEM.CLUSTER.HISTORY.QUEUE	The queue is used to store the history of cluster state information for service purposes.
SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE	The queue is used to create individual transmit queues for each cluster-sender channel.
SYSTEM.CLUSTER.REPOSITORY.QUEUE	The queue used to store all repository information.
SYSTEM.CLUSTER.TRANSMIT.QUEUE	The transmission queue for all messages to all clusters.
SYSTEM.DEAD.LETTER.QUEUE	Dead-letter (undelivered-message) queue.
SYSTEM.DEFAULT.ALIAS.QUEUE	Default alias queue.
SYSTEM.DEFAULT.INITIATION.QUEUE	Default initiation queue.
SYSTEM.DEFAULT.LOCAL.QUEUE	Default local queue.
SYSTEM.DEFAULT.MODEL.QUEUE	Default model queue.
SYSTEM.DEFAULT.REMOTE.QUEUE	Default remote queue.
SYSTEM.JMS.TEMPQ.MODEL	Model for JMS temporary queues
SYSTEM.MQEXPLORER.REPLY.MODEL	The IBM MQ Explorer reply-to queue. This is a model queue that creates a temporary dynamic queue for replies to the IBM MQ Explorer.
SYSTEM.MQSC.REPLY.QUEUE	MQSC command reply-to queue. This is a model queue that creates a temporary dynamic queue for replies to remote MQSC commands.
SYSTEM.PENDING.DATA.QUEUE	Support deferred messages in JMS.

Table 12. System and default objects: topics

Object name	Description
SYSTEM.BASE.TOPIC	Base topic for ASPARENT resolution. If a particular topic has no parent administrative topic objects, or those parent objects also have ASPARENT, any remaining ASPARENT attributes are inherited from this object.
SYSTEM.DEFAULT.TOPIC	Default topic definition.

Table 13. System and default objects: channels

Object name	Description
SYSTEM.AUTO.RECEIVER	Dynamic receiver channel.
SYSTEM.AUTO.SVRCONN	Dynamic server-connection channel.
SYSTEM.DEF.CLUSRCVR	Default receiver channel for the cluster, used to supply default values for any attributes not specified when a CLUSRCVR channel is created on a queue manager in the cluster.
SYSTEM.DEF.CLUSSDR	Default sender channel for the cluster, used to supply default values for any attributes not specified when a CLUSSDR channel is created on a queue manager in the cluster.
SYSTEM.DEF.RECEIVER	Default receiver channel.
SYSTEM.DEF.REQUESTER	Default requester channel.
SYSTEM.DEF.SENDER	Default sender channel.
SYSTEM.DEF.SERVER	Default server channel.
SYSTEM.DEF.SVRCONN	Default server-connection channel.
SYSTEM.DEF.CLNTCONN	Default client-connection channel.
 SYSTEM.DEF.AMQP	Default AMQP channel.

Table 14. System and default objects: authentication information objects

Object name	Description
SYSTEM.DEFAULT.AUTHINFO.CRLLDAP	Default authentication information object for defining authentication information objects of type CRLLDAP.
SYSTEM.DEFAULT.AUTHINFO.OCSP	Default authentication information object for defining authentication information objects of type OCSP.

Table 15. System and default objects: listeners

Object name	Description
SYSTEM.DEFAULT.LISTENER.TCP	Default TCP listener.
SYSTEM.DEFAULT.LISTENER.LU62 [1]	Default LU62 listener.
SYSTEM.DEFAULT.LISTENER.NETBIOS [1]	Default NETBIOS listener.
SYSTEM.DEFAULT.LISTENER.SPX [1]	Default SPX listener.

1. Windows only

Table 16. System and default objects: namelists

Object name	Description
SYSTEM.DEFAULT.NAMELIST	Default namelist.

Table 17. System and default objects: processes

Object name	Description
SYSTEM.DEFAULT.PROCESS	Default process definition.

Table 18. System and default objects: services

Object name	Description
SYSTEM.DEFAULT.SERVICE	Default service.
SYSTEM.BROKER	Publish/subscribe broker

Windows default configuration objects

Windows

On Windows systems, you can set up a default configuration using the IBM MQ Postcard application.

Note: You cannot set up a default configuration if other queue managers exist on your computer.

Many of the names used for the Windows default configuration objects involve the use of a short TCP/IP name. This is the TCP/IP name of the computer, without the domain part; for example the short TCP/IP name for the computer `mycomputer.hursley.ibm.com` is `mycomputer`. In all cases, where this name has to be truncated, if the last character is a period (.), it is removed.

Any characters within the short TCP/IP name that are not valid for IBM MQ object names (for example, hyphens) are replaced by an underscore character.

Valid characters for IBM MQ object names are: a to z, A to Z, 0 to 9, and the four special characters / % . and _.

The cluster name for the Windows default configuration is DEFAULT_CLUSTER.

If the queue manager is not a repository queue manager, the objects listed in Table 19 are created.

Table 19. Objects created by the Windows default configuration application

Object	Name
Queue manager	The short TCP/IP name prefixed with the characters QM_. The maximum length of the queue manager name is 48 characters. Names exceeding this limit are truncated at 48 characters. If the last character of the name is a period (.), this is replaced by a space ( ). The queue manager has a command server, a channel listener, and channel initiator associated with it. The channel listener listens on the standard IBM MQ port, port number 1414. Any other queue managers created on this machine must not use port 1414 while the default configuration queue manager still exists.
Generic cluster receiver channel	The short TCP/IP name prefixed with the characters TO_QM_. The maximum length of the generic cluster receiver name is 20 characters. Names exceeding this limit are truncated at 20 characters. If the last character of the name is a period (.), this is replaced by a space ( ).

Table 19. Objects created by the Windows default configuration application (continued)

Object	Name
Cluster sender channel	The cluster sender channel is initially created with the name TO_+QMNAME+. When IBM MQ has established a connection to the repository queue manager for the default configuration cluster, this name is replaced with the name of the repository queue manager for the default configuration cluster, prefixed with the characters TO_. The maximum length of the cluster sender channel name is 20 characters. Names exceeding this limit are truncated at 20 characters. If the last character of the name is a period (.), this is replaced by a space ( ).
Local message queue	The local message queue is called `default`.
Local message queue for use by the IBM MQ Postcard application	The local message queue for use by the IBM MQ Postcard application is called `postcard`.
Server connection channel	The server connection channel allows clients to connect to the queue manager. Its name is the short TCP/IP name, prefixed with the characters S_. The maximum length of the server connection channel name is 20 characters. Names exceeding this limit are truncated at 20 characters. If the last character of the name is a period (.), this is replaced by a space ( ).

If the queue manager is a repository queue manager, the default configuration is similar to that described in Table 19 on page 88, but with the following differences:

- The queue manager is defined as a repository queue manager for the default configuration cluster.
- There is no cluster-sender channel defined.
- A local cluster queue that is the short TCP/IP name prefixed with the characters clq_default_ is created. The maximum length of this name is 48 characters. Names exceeding this length are truncated at 48 characters.

If you request remote administration facilities, the server connection channel, SYSTEM.ADMIN.SVRCONN is also created.

SYSTEM.BASE.TOPIC

Base topic for `ASPARENT` resolution. If a particular topic has no parent administrative topic objects, or those parent objects also have `ASPARENT`, any remaining `ASPARENT` attributes are inherited from this object.

Table 20. Default values of `SYSTEM.BASE.TOPIC`

Parameter	Value
`TOPICSTR`	`"`
`CLROUTE`	`DIRECT`
`CLUSTER`	The default value is an empty string.
`COMMINFO`	`SYSTEM.DEFAULT.COMMINFO.MULTICAST`
`DEFPRESP`	`SYNC`
`DEFPRTY`	`0`
`DEFPSIST`	`NO`
`DESCR`	'Base topic for resolving attributes'
`DURSUB`	`YES`
`MCAST`	`DISABLED`
`MDURMDL`	`SYSTEM.DURABLE.MODEL.QUEUE`
`MNDURMDL`	`SYSTEM.NDURABLE.MODEL.QUEUE`
`NPMSGDLV`	`ALLAVAIL`

Table 20. Default values of SYSTEM.BASE.TOPIC (continued)

Parameter	Value
PMSGDLV	ALLDUR
PROXYSUB	FIRSTUSE
PUB	ENABLED
PUBSCOPE	ALL
z/OS QSGDISP (z/OS platform only)	QMGR
SUB	ENABLED
SUBSCOPE	ALL
USEDLQ	YES
WILDCARD	PASSTHRU

If this object does not exist, its default values are still used by IBM MQ for ASPARENT attributes that are not resolved by parent topics further up the topic tree.

Setting the PUB or SUB attributes of SYSTEM.BASE.TOPIC to DISABLED prevents applications publishing or subscribing to topics in the topic tree, with two exceptions:

1. Any topic objects in the topic tree that have PUB or SUB explicitly set to ENABLE. Applications can publish or subscribe to those topics, and their children.
2. Publication and subscription to SYSTEM.BROKER.ADMIN.STREAM is not disabled by the setting the PUB or SUB attributes of SYSTEM.BASE.TOPIC to DISABLED.

See also Special handling for the **PUB** parameter.

System and default objects for IBM i

IBM i

When you create a queue manager using the CRTMQM command, the system objects and the default objects are created automatically.

- The system objects are those IBM MQ objects required for the operation of a queue manager or channel.
- The default objects define all the attributes of an object. When you create an object, such as a local queue, any attributes that you do not specify explicitly are inherited from the default object.

The following tables list the system and default objects created by **CRTMQM**:

- Table 21 on page 91 lists the system and default queue objects.
- Table 22 on page 92 lists the system and default channel objects.
- Table 23 on page 93 gives the system and default authentication information objects.
- Table 24 on page 93 gives the system and default listener object.
- Table 25 on page 93 gives the system and default namelist object.
- Table 26 on page 93 gives the system and default process object.
- Table 27 on page 93 gives the system and default service object.

Table 21. System and default objects: queues

Object name	Description
SYSTEM.ADMIN.ACCOUNTING.QUEUE	Accounting message data generated when an application disconnects from the queue manager.
SYSTEM.ADMIN.ACTIVITY.QUEUE	Activity report message data.
SYSTEM.ADMIN.CHANNEL.EVENT	Event queue for channels.
SYSTEM.ADMIN.COMMAND.QUEUE	Administration command queue. Used for remote MQSC commands and PCF commands.
SYSTEM.ADMIN.LOGGER.EVENT	Logger event (journal receiver) message data.
SYSTEM.ADMIN.PERFM.EVENT	Event queue for performance events.
SYSTEM.ADMIN.PUBSUB.EVENT	System publish/subscribe related event queue
SYSTEM.ADMIN.QMGR.EVENT	Event queue for queue manager events.
SYSTEM.ADMIN.STATISTICS.QUEUE	MQI, queue and channel statistics message data queue.
SYSTEM.ADMIN.TRACE.ROUTE.QUEUE	Trace-route reply message data queue.
SYSTEM.AUTH.DATA.QUEUE	Used by the object authority manager (OAM).
SYSTEM.BROKER.ADMIN.STREAM	Admin stream used by the queued publish/subscribe interface.
SYSTEM.BROKER.CONTROL.QUEUE	Publish/subscribe interface control queue.
SYSTEM.BROKER.DEFAULT.STREAM	The default stream used by the queued publish/subscribe interface.
SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS	Broker to broker communications queue.
SYSTEM.CHANNEL.INITQ	Channel initiation queue.
SYSTEM.CHANNEL.SYNCQ	The queue that holds the synchronization data for channels.
SYSTEM.CHLAUTH.DATA.QUEUE	IBM MQ channel authentication data queue
SYSTEM.DURABLE.MODEL.QUEUE	A queue used as a model for managed durable subscriptions.
SYSTEM.DURABLE.SUBSCRIBER.QUEUE	A queue used to hold a persistent copy of the durable subscriptions in the queue manager.
SYSTEM.CICS.INITIATION.QUEUE	Default CICS initiation queue.
SYSTEM.CLUSTER.COMMAND.QUEUE	The queue used to carry messages to the repository queue manager.
SYSTEM.CLUSTER.HISTORY.QUEUE	The queue is used to store the history of cluster state information for service purposes.
SYSTEM.CLUSTER.REPOSITORY.QUEUE	The queue used to store all repository information.
SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE	The queue is used to create individual transmit queues for each cluster-sender channel.
SYSTEM.CLUSTER.TRANSMIT.QUEUE	The transmission queue for all messages to all clusters.
SYSTEM.DEAD.LETTER.QUEUE	Dead-letter (undelivered message) queue.
SYSTEM.DEFAULT.ALIAS.QUEUE	Default alias queue.
SYSTEM.DEFAULT.AUTHINFO.CRLLDAP	Default authentication information definition.
SYSTEM.DEFAULT.INITIATION.QUEUE	Default initiation queue.
SYSTEM.DEFAULT.LOCAL.QUEUE	Default local queue.
SYSTEM.DEFAULT.MODEL.QUEUE	Default model queue.
SYSTEM.DEFAULT.REMOTE.QUEUE	Default remote queue.

Table 21. System and default objects: queues (continued)

Object name	Description
SYSTEM.JMS.TEMPQ.MODEL	Model for JMS temporary queues
SYSTEM.HIERARCHY.STATE	IBM MQ distributed publish/subscribe hierarchy relationship state.
SYSTEM.INTER.QMGR.CONTROL	IBM MQ distributed publish/subscribe control queue.
SYSTEM.INTER.QMGR.FANREQ	IBM MQ distributed publish/subscribe internal proxy subscription fan-out process input queue.
SYSTEM.INTER.QMGR.PUBS	IBM MQ distributed publish/subscribe publications.
SYSTEM.MQEXPLORER.REPLY.MODEL	IBM MQ Explorer reply-to queue. This is a model queue that creates a temporary dynamic queue for replies to the IBM MQ Explorer.
SYSTEM.MQSC.REPLY.QUEUE	MQSC command reply-to queue. This is a model queue that creates a temporary dynamic queue for replies to remote MQSC commands.
SYSTEM.NDURABLE.MODEL.QUEUE	A queue used as a model for managed non durable subscriptions.
SYSTEM.PENDING.DATA.QUEUE	Support deferred messages in JMS.
SYSTEM.RETAINED.PUB.QUEUE	A queue used to hold a copy of each retained publication in the queue manager.

Table 22. System and default objects: channels

Object name	Description
SYSTEM.AUTO.RECEIVER	Dynamic receiver channel.
SYSTEM.AUTO.SVRCONN	Dynamic server-connection channel.
SYSTEM.DEF.CLNTCONN	Default client connection channel, used to supply default values for any attributes not specified when a CLNTCONN channel is created on a queue manager.
SYSTEM.DEF.CLUSRCVR	Default receiver channel for the cluster used to supply default values for any attributes not specified when a CLUSRCVR channel is created on a queue manager in the cluster.
SYSTEM.DEF.CLUSSDR	Default sender channel for the cluster used to supply default values for any attributes not specified when a CLUSSDR channel is created on a queue manager in the cluster.
SYSTEM.DEF.RECEIVER	Default receiver channel.
SYSTEM.DEF.REQUESTER	Default requester channel.
SYSTEM.DEF.SENDER	Default sender channel.
SYSTEM.DEF.SERVER	Default server channel.
SYSTEM.DEF.SVRCONN	Default server-connection channel.

Table 23. System and default objects: authentication information objects

Object name	Description
SYSTEM.DEFAULT.AUTHINFO.CRLLDAP	Default authentication information object for authentication type CRLLDAP.
SYSTEM.DEFAULT.AUTHINFO.OCSP	Default authentication information object for authentication type OCSP.

Table 24. System and default objects: listeners

Object name	Description
SYSTEM.DEFAULT.LISTENER.TCP	Default listener for TCP transport.

Table 25. System and default objects: namelists

Object name	Description
SYSTEM.DEFAULT.NAMELIST	Default namelist definition.
SYSTEM.QPUBSUB.QUEUE.NAMELIST	A list of queue names monitored by the queued publish/subscribe interface.
SYSTEM.QPUBSUB.SUBPOINT.NAMELIST	A list of topic objects used by the queued publish/subscribe interface to match topic objects to subscription points.

Table 26. System and default objects: processes

Object name	Description
SYSTEM.DEFAULT.PROCESS	Default process definition.

Table 27. System and default objects: services

Object name	Description
SYSTEM.DEFAULT.SERVICE	Default service.

Stanza information

The following information helps you configure the information within stanzas, and lists the contents of the mqs.ini, qm.ini, and mqclient.ini files.

Configuring stanzas

Use the links to help you configure the system, or systems, in your enterprise:

- Changing IBM MQ configuration information helps you configure the:
  - *AllQueueManagers* stanza
  - *DefaultQueueManager* stanza
  - *ExitProperties* stanza
  - *LogDefaults* stanza
  - *Security* stanza in the qm.ini file
- Changing queue manager configuration information helps you configure the:
  - Windows *AccessMode* stanza (Windows only)
  - *Service* stanza - for Installable services
  - *Log* stanza

  - UNIX Linux *RestrictedMode* stanza (UNIX and Linux systems only)
  - *XAResourceManager* stanza
  - *TCP*, *LU62*, and *NETBIOS* stanzas
  - *ExitPath* stanza
  - *QMErrorLog* stanza
  - *SSL* stanza
  - *ExitPropertiesLocal* stanza
- Configuring services and components helps you configure the:
  - *Service* stanza
  - *ServiceComponent* stanza

  and contains links to how they are used for different services on UNIX and Linux, and Windows platforms.
- Configuring API exits helps you configure the:
  - *AllActivityTrace* stanza
  - *AppplicationTrace* stanza
- Configuring activity trace behavior helps you configure the:
  - *ApiExitCommon* stanza
  - *ApiExitTemplate* stanza
  - *ApiExitLocal* stanza
- Configuration information for clients helps you configure the:
  - *CHANNELS* stanza
  - *ClientExitPath* stanza
  - Windows *LU62*, *NETBIOS* and *SPX* stanza (Windows only)
  - *MessageBuffer* stanza
  - *SSL* stanza
  - *TCP* stanza
- "Configuration file stanzas for distributed queuing" on page 96 helps you configure the:
  - *CHANNELS* stanza
  - *TCP* stanza
  - *LU62* stanza
  - *NETBIOS*
  - *ExitPath* stanza
- Setting queued publish/subscribe message attributes helps you configure the:
  - *PersistentPublishRetry* attribute
  - *NonPersistentPublishRetry* attribute
  - *PublishBatchSize* attribute
  - *PublishRetryInterval* attribute

  in the *Broker* stanza.

  Attention: You must create a *Broker* stanza if you need one.

Configuration files

See:

- **mqs.ini** file

- **qm.ini** file
- **mqclient.ini** file

for a list of the possible stanzas in each configuration file. UNIX Linux

mqs.ini file

Example of an IBM MQ configuration file for UNIX and Linux systems shows an example `mqs.ini` file.

An `mqs.ini` file can contain the following stanzas:

- *AllQueueManagers*
- *DefaultQueueManager*
- *ExitProperties*
- *LogDefaults*

In addition, there is one *QueueManager* stanza for each queue manager.

qm.ini file

Example queue manager configuration file for IBM MQ for UNIX and Linux systems shows an example `qm.ini` file.

A `qm.ini` file can contain the following stanzas:

- *ExitPath*
- *Log*
- *QMErrorLog*
- *QueueManager*
- *Security*
- *Service* and *ServiceComponent*

  To configure *InstallableServices* on:

  – UNIX Windows UNIX and Linux platforms, use the *Service* and *ServiceComponent* stanzas.

  – Windows Windows, use **regedit**.
- *Connection* for *DefaultBindType*

  Attention: You must create a *Connection* stanza if you need one.
- *SSL and TLS*
- *TCP, LU62, and NETBIOS*
- XAResourceManager

In addition, you can change the:

- Windows *AccessMode* (Windows only)
- UNIX Linux *RestrictedMode* (UNIX and Linux systems only)

by using the crtmqm command.

mqclient.ini file

An `mqclient.ini` file can contain the following stanzas:

- *CHANNELS*
- *ClientExitPath*
- *LU62*, *NETBIOS*, and *SPX*
- *MessageBuffer*

- *SSL*
- *TCP*

In addition, you might need a PreConnect stanza to configure a preconnect exit.

Configuration file stanzas for distributed queuing

A description of the stanzas of the queue manager configuration file, qm.ini, related to distributed queuing.

This topic shows the stanzas in the queue manager configuration file that relate to distributed queuing. It applies to the queue manager configuration file for IBM MQ for Multiplatforms. The file is called `qm.ini` on all platforms.

The stanzas that relate to distributed queuing are:

- CHANNELS
- TCP
- LU62
- NETBIOS
- EXITPATH

Figure 6 shows the values that you can set using these stanzas. When you are defining one of these stanzas, you do not need to start each item on a new line. You can use either a semicolon (;) or a hash character (#) to indicate a comment.

```
CHANNELS:
  MAXCHANNELS=n        ; Maximum number of channels allowed, the
                       ; default value is 100.
  MAXACTIVECHANNELS=n  ; Maximum number of channels allowed to be active at
                       ; any time, the default is the value of MaxChannels.
  MAXINITIATORS=n      ; Maximum number of initiators allowed, the default
                       ; and maximum value is 3.
  MQIBINDTYPE=type[1]    ; Whether the binding for applications is to be
                       ; "fastpath" or "standard".
                       ; The default is "standard".
  ADOPTNEWMCA=chltype  ; Stops previous process if channel fails to start.
                       ; The default is "NO".
  ADOPTNEWMCATIMEOUT=n ; Specifies the amount of time that the new
                       ; process should wait for the old process to end.
                       ; The default is 60.
  ADOPTNEWMCACHECK=    ; Specifies the type checking required.
           typecheck   ; The default is "NAME","ADDRESS", and "QM".
  TCP:                 ; TCP entries
  PORT=n               ; Port number, the default is 1414
  KEEPALIVE=Yes        ; Switch TCP/IP KeepAlive on
  LIBRARY2=DLLName2    ; Used if code is in two libraries
  EXITPATH:[2]           ; Location of user exits (MQSeries for AIX,
                       ; HP-UX, and Solaris only)
  EXITPATHS=           ; String of directory paths.
```

Figure 6. qm.ini stanzas for distributed queuing

Note:

1. MQIBINDTYPE applies only to the following platforms:
   - AIX
   - HP-UX
   - IBM i
   - Solaris

2. EXITPATH applies only to the following platforms:
   - AIX AIX
   - HP-UX HP-UX
   - Solaris Solaris

Related information:

Configuring

z/OS Configuring z/OS

Changing configuration information on Windows, UNIX, and Linux systems

IBM i Changing configuration information on IBM i

Channel attributes

This section describes the channel attributes held in the channel definitions.

You choose the attributes of a channel to be optimal for a particular set of circumstances for each channel. However, when the channel is running, the actual values might have changed during startup negotiations. See Preparing channels.

Many attributes have default values, and you can use these values for most channels. However, in those circumstances where the defaults are not optimal, see this section for guidance in selecting the correct values.

For cluster channels, you specify the cluster channel attributes on the cluster-receiver channels at the target queue managers. Any attributes you specify on the matching cluster-sender channels are likely to be ignored. See Cluster channels.

Note: In IBM MQ for IBM i, most attributes can be specified as `*SYSDFTCHL`, which means that the value is taken from the system default channel in your system.

Channel attributes and channel types

Different types of channel support different channel attributes.

The channel types for IBM MQ channel attributes are listed in the following table.

Note: For cluster channels (the CLUSSDR and CLUSRCVR columns in the table), if an attribute can be set on both channels, set it on both and ensure that the settings are identical. If there is any discrepancy between the settings, those that you specify on the CLUSRCVR channel are likely to be used. This is explained in Cluster channels.

Table 28. Channel attributes for the channel types

Attribute field	MQSC command parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR-CONN	CLUS-SDR	CLUS-RCVR	AMQP
Alter date	ALTDATE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Alter time	ALTTIME	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Multi V 9.0.0 AMQP keep alive	AMQPKA									Yes
Batch heartbeat interval	BATCHHB	Yes	Yes					Yes	Yes	

Table 28. Channel attributes for the channel types (continued)

Attribute field	MQSC command parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR-CONN	CLUS-SDR	CLUS-RCVR	AMQP V 9.0.0
Batch interval	BATCHINT	Yes	Yes					Yes	Yes	
Batch limit	BATCHLIM	Yes	Yes					Yes	Yes	
Batch size	BATCHSZ	Yes	Yes	Yes	Yes			Yes	Yes	
Certificate label	CERTLABL	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes V 9.0.0
Channel name	CHANNEL	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes V 9.0.0
Channel statistics	STATCHL	Yes	Yes	Yes	Yes			Yes	Yes	
Channel type	CHLTYPE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes V 9.0.0
Client channel weight	CLNTWGHT					Yes				
Cluster	CLUSTER							Yes	Yes	
Cluster namelist	CLUSNL							Yes	Yes	
Cluster workload priority	CLWLPRTY							Yes	Yes	
Cluster workload rank	CLWLRANK							Yes	Yes	
Cluster workload weight	CLWLWGHT							Yes	Yes	
Connection affinity	AFFINITY					Yes				
Connection name	CONNAME	Yes	Yes		Yes	Yes		Yes	Yes	
Convert message	CONVERT	Yes	Yes					Yes	Yes	
Data compression	COMPMSG	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Description	DESCR	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes V 9.0.0
Disconnect interval	DISCINT	Yes	Yes				Yes [1]	Yes	Yes	
Disposition [1]	QSGDISP	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Header compression	COMPHDR	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Heartbeat interval	HBINT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Keepalive interval	KAINT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Local address	LOCLADDR	Yes	Yes		Yes	Yes		Yes	Yes	Yes V 9.0.0
Long retry count	LONGRTY	Yes	Yes					Yes	Yes	
Long retry interval	LONGTMR	Yes	Yes					Yes	Yes	
LU 6.2 mode name	MODENAME	Yes	Yes		Yes	Yes		Yes	Yes	
LU 6.2 transaction program name	TPNAME	Yes	Yes		Yes	Yes		Yes	Yes	
Maximum instances	MAXINST						Yes			Yes V 9.0.0
Maximum instances per client	MAXINSTC						Yes			

Table 28. Channel attributes for the channel types (continued)

Attribute field	MQSC command parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR-CONN	CLUS-SDR	CLUS-RCVR	AMQP
Maximum message length	MAXMSGL	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Message channel agent name	MCANAME	Yes	Yes		Yes			Yes	Yes	
Message channel agent type	MCATYPE	Yes	Yes		Yes			Yes	Yes	
Message channel agent user	MCAUSER	Yes	Yes	Yes	Yes		Yes	Yes	Yes	Yes
Message exit name	MSGEXIT	Yes	Yes	Yes	Yes			Yes	Yes	
Message exit user data	MSGDATA	Yes	Yes	Yes	Yes			Yes	Yes	
Message-retry exit name	MREXIT			Yes	Yes				Yes	
Message-retry exit user data	MRDATA			Yes	Yes				Yes	
Message retry count	MRRTY			Yes	Yes				Yes	
Message retry interval	MRTMR			Yes	Yes				Yes	
Monitoring	MONCHL	Yes	Yes	Yes	Yes		Yes	Yes	Yes	
Network-connection priority	NETPRTY								Yes	
Nonpersistent message speed	NPMSPEED	Yes	Yes	Yes	Yes			Yes	Yes	
Password	PASSWORD	Yes	Yes		Yes	Yes		Yes		
V 9.0.0 Port number	PORT									Yes
Property control	PROPCTL	Yes	Yes					Yes	Yes	
PUT authority	PUTAUT			Yes	Yes		Yes [1]		Yes	
Queue manager name	QMNAME					Yes				
Receive exit name	RCVEXIT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Receive exit user data	RCVDATA	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Security exit name	SCYEXIT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Security exit user data	SCYDATA	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Send exit name	SENDEXIT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Send exit user data	SENDDATA	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Sequence number wrap	SEQWRAP	Yes	Yes	Yes	Yes			Yes	Yes	
Shared connections	SHARECNV					Yes	Yes			
Short retry count	SHORTRTY	Yes	Yes					Yes	Yes	

Table 28. Channel attributes for the channel types (continued)

Attribute field	MQSC command parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR-CONN	CLUS-SDR	CLUS-RCVR	AMQP
Short retry interval	SHORTTMR	Yes	Yes					Yes	Yes	
SSL Cipher Specification	SSLCIPH	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SSL Client Authentication	SSLCAUTH		Yes	Yes	Yes		Yes		Yes	Yes
SSL Peer	SSLPEER	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
V 9.0.0 Topic root	TPROOT									Yes
Transmission queue name	XMITQ	Yes	Yes							
Transport type	TRPTYPE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
V 9.0.0 Use client ID	USECLTID									Yes
Use Dead-Letter Queue	USEDLQ	Yes	Yes	Yes	Yes			Yes	Yes	
User ID	USERID	Yes	Yes		Yes	Yes		Yes		

Note: z/OS

1. Valid on z/OS only.

Related concepts:

"Channel attributes in alphabetical order"
This section describes each attribute of a channel object, with its valid values and notes on its use where appropriate.

Related information:

MQSC reference
Use MQSC commands to manage queue manager objects, including the queue manager itself, queues, process definitions, channels, client connection channels, listeners, services, namelists, clusters, and authentication information objects.

Channel attributes in alphabetical order

This section describes each attribute of a channel object, with its valid values and notes on its use where appropriate.

IBM MQ for some platforms might not implement all the attributes shown in this section. Exceptions and platform differences are mentioned in the individual attribute descriptions, where relevant.

The keyword that you can specify in MQSC is shown in brackets for each attribute.

The attributes are arranged in alphabetical order.

Alter date (ALTDATE):

This attribute is the date on which the definition was last altered, in the form `yyyy-mm-dd`.

This attribute is valid for all channel types.

Alter time (ALTTIME):

This attribute is the time at which the definition was last altered, in the form `hh:mm:ss`.

This attribute is valid for all channel types.

AMQP keep alive (AMQPKA): Multi V 9.0.0

Use the **AMQPKA** attribute to specify a keep alive time for the AMQP client connection. If the AMQP client has not sent any frames within the keep alive interval, then the connection is closed.

The **AMQPKA** attribute determines the value of the idle-timeout attribute sent from IBM MQ to an AMQP client. The attribute is a period of time in milliseconds.

If **AMQPKA** is set to a value > 0, then IBM MQ flows half that value as the idle-timeout attribute. For example, a value of 10000 causes the queue manager to send an idle-timeout value of 5000. The client should ensure that data is sent to IBM MQ at least every 10000 milliseconds. If data is not received by IBM MQ in that time, IBM MQ assumes that the client has lost its connection and forcibly closes the connection with an `amqp:resource-limit-exceeded` error condition.

A value of AUTO or 0 means the IBM MQ does not flow an idle-timeout attribute to the AMQP client.

An AMQP client can still flow an idle-timeout value of its own. If it does, IBM MQ flows data (or an empty AMQP frame) at least that frequently to inform the client that it is available.

Batch Heartbeat Interval (BATCHHB):

This attribute allows a sending channel to verify that the receiving channel is still active just before committing a batch of messages.

The batch heartbeat interval thus allows the batch to be backed out rather than becoming in-doubt if the receiving channel is not active. By backing out the batch, the messages remain available for processing so they could, for example, be redirected to another channel.

If the sending channel has had a communication from the receiving channel within the batch heartbeat interval, the receiving channel is assumed to be still active, otherwise a 'heartbeat' is sent to the receiving channel to check. The sending channel waits for a response from the receiving end of the channel for an interval, based on the number of seconds specified in the channel Heartbeat Interval (HBINT) attribute.

The value is in milliseconds and must be in the range zero through 999999. A value of zero indicates that batch heart beating is not used.

This attribute is valid for channel types of:

- Sender
- Server
- Cluster sender
- Cluster receiver

Batch interval (BATCHINT):

This attribute is a period, in milliseconds, during which the channel keeps a batch open even if there are no messages on the transmission queue.

You can specify any number of milliseconds, from zero through 999 999 999. The default value is zero.

If you do not specify a batch interval, the batch closes when one of the following conditions is met:

- The number of messages specified in BATCHSZ have been sent.
- The number of bytes specified in BATCHLIM have been sent.
- The transmission queue is empty.

On channels with a light load, where the transmission queue frequently becomes empty, the effective batch size might be much smaller than BATCHSZ.

You can use the BATCHINT attribute to make your channels more efficient by reducing the number of short batches. Be aware, however, that you can slow down the response time, because batches last longer and messages remain uncommitted for longer.

If you specify a BATCHINT, batches close only when one of the following conditions is met:

- The number of messages specified in BATCHSZ have been sent.
- The number of bytes specified in BATCHLIM have been sent.
- There are no more messages on the transmission queue and a time interval of BATCHINT has elapsed while waiting for messages (since the first message of the batch was retrieved).

Note: BATCHINT specifies the total amount of time that is spent waiting for messages. It does not include the time spent retrieving messages that are already available on the transmission queue, or the time spent transferring messages.

This attribute is valid for channel types of:

- Sender
- Server
- Cluster sender
- Cluster receiver

Batch limit (BATCHLIM):

This attribute is the limit, in kilobytes, of the amount of data that can be sent through a channel before taking a sync point.

A sync point is taken after the message that caused the limit to be reached has flowed across the channel.

The value must be in the range 0 - 999999. The default value is 5000.

A value of zero in this attribute means that no data limit is applied to batches over this channel.

The batch is terminated when one of the following conditions is met:

- BATCHSZ messages have been sent.
- BATCHLIM bytes have been sent.
- The transmission queue is empty and BATCHINT is exceeded.

This attribute is valid for channel types of:

- Sender

- Server
- Cluster sender
- Cluster receiver

This parameter is supported on all platforms.

Batch size (BATCHSZ):

This attribute is the maximum number of messages to be sent before a sync point is taken.

The batch size does not affect the way the channel transfers messages; messages are always transferred individually, but are committed or backed out as a batch.

To improve performance, you can set a batch size to define the maximum number of messages to be transferred between two *sync points*. The batch size to be used is negotiated when a channel starts, and the lower of the two channel definitions is taken. On some implementations, the batch size is calculated from the lowest of the two channel definitions and the two queue manager MAXUMSGS values. The actual size of a batch can be less; for example, a batch completes when there are no messages left on the transmission queue or the batch interval expires.

A large value for the batch size increases throughput, but recovery times are increased because there are more messages to back out and send again. The default BATCHSZ is 50, and you are advised to try that value first. You might choose a lower value for BATCHSZ if your communications are unreliable, making the need to recover more likely.

Sync point procedure needs a unique logical unit of work identifier to be exchanged across the link every time a sync point is taken, to coordinate batch commit procedures.

If the synchronized batch commit procedure is interrupted, an *in-doubt* situation might arise. In-doubt situations are resolved automatically when a message channel starts. If this resolution is not successful, manual intervention might be necessary, using the RESOLVE command.

Some considerations when choosing the number for batch size:

- If the number is too large, the amount of queue space taken up on both ends of the link becomes excessive. Messages take up queue space when they are not committed, and cannot be removed from queues until they are committed.
- If there is likely to be a steady flow of messages, you can improve the performance of a channel by increasing the batch size because fewer confirm flows are needed to transfer the same quantity of bytes.
- If message flow characteristics indicate that messages arrive intermittently, a batch size of 1 with a relatively large disconnect time interval might provide a better performance.
- The number can be in the range 1 through 9999. However, for data integrity reasons, channels connecting to any of the current platforms must specify a batch size greater than 1. A value of 1 is for use with Version 1 products, apart from IBM MQ for MVS.
- Even though nonpersistent messages on a fast channel do not wait for a sync point, they do contribute to the batch-size count.

This attribute is valid for channel types of:

- Sender
- Server
- Receiver
- Requester
- Cluster sender

- Cluster receiver

Certificate label (CERTLABL):

This attribute specifies the certificate label of the channel definition.

The label identifies which personal certificate in the key repository is sent to the remote peer. The certificate is defined as described in Digital certificate labels.

Inbound channels (including RCVR, CLUSRCVR, unqualified SERVER, and SVRCONN channels) will only send the configured certificate if the IBM MQ version of the remote peer fully supports certificate label configuration and the channel is using a TLS CipherSpec. If that is not the case, the queue manager **CERTLABL** attribute determines the certificate sent. This restriction is because the certificate label selection mechanism for inbound channels depends upon a TLS protocol extension that is not supported in all cases. In particular, Java™ clients, JMS clients, and all versions of IBM MQ prior to Version 8.0 do not support the required protocol extension and will only ever receive the certificate configured by the queue manager **CERTLABL** attribute, regardless of the channel-specific label setting.

None of the administrative interfaces allow this attribute to be inquired or set for CLUSSDR channels. You will receive an MQRCCF_WRONG_CHANNEL_TYPE message. However, the attribute is present in CLUSSDR channel objects (including MQCD structures) and a CHAD exit can set it programmatically if required.

For more information about what the certificate label can contain, see Digital certificate labels, understanding the requirements.

This attribute is valid for all channel types.

Channel name (CHANNEL):

This attribute specifies the name of the channel definition.

The name can contain up to 20 characters, although as both ends of a message channel must have the same name, and other implementations might have restrictions on the size, the actual number of characters might have to be smaller.

Where possible, channel names are unique to one channel between any two queue managers in a network of interconnected queue managers.

The name must contain characters from the following list:

Alphabetic	(A-Z, a-z; note that uppercase and lowercase are significant)
Numerics	(0-9)
Period	(.)
Forward slash	(/)
Underscore	(_)
Percentage sign	(%)

Note:

1. Embedded blanks are not allowed, and leading blanks are ignored.
2. On systems using EBCDIC Katakana, you cannot use lowercase characters.

This attribute is valid for all channel types.

Channel statistics (STATCHL) on Multiplatforms: Multi

On Multiplatforms, this attribute controls the collection of statistics data for channels.

The possible values are:

QMGR
Statistics data collection for this channel is based upon the setting of the queue manager attribute STATCHL. This value is the default value.

OFF Statistics data collection for this channel is disabled.

LOW Statistics data collection for this channel is enabled with a low ratio of data collection.

MEDIUM
Statistics data collection for this channel is enabled with a moderate ratio of data collection.

HIGH Statistics data collection for this channel is enabled with a high ratio of data collection.

For more information about channel statistics, see Monitoring reference.

z/OS On z/OS systems, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying `LOW`, `MEDIUM`, or `HIGH` makes no difference to your results.

This attribute is valid for channel types of:

- Sender
- Server
- Receiver
- Requester
- Cluster sender
- Cluster receiver

Channel type (CHLTYPE):

This attribute specifies the type of the channel being defined.

The possible channel types are:

Message channel types:

- Sender
- Server
- Receiver
- Requester
- Cluster-sender
- Cluster-receiver

MQI channel types:

- Client-connection (Windows and UNIX only)

  Note: Client-connection channels can also be defined on z/OS for use on other platforms.
- Server-connection
- AMQP

The two ends of a channel must have the same name and have compatible types:

- Sender with receiver

- Requester with server
- Requester with sender (for callback)
- Server with receiver (server is used as a sender)
- Client-connection with server-connection
- Cluster-sender with cluster-receiver
- AMQP with AMQP

Client channel weight (CLNTWGHT):

This attribute specifies a weighting to influence which client-connection channel definition is used.

The client channel weighting attribute is used so that client channel definitions can be selected at random based on their weighting when more than one suitable definition is available.

When a client issues an MQCONN requesting connection to a queue manager group, by specifying a queue manager name starting with an asterisk, which enables client weight balancing across several queue managers, and more than one suitable channel definition is available in the client channel definition table (CCDT), the definition to use is randomly selected based on the weighting, with any applicable CLNTWGHT(0) definitions selected first in alphabetical order.

Specify a value in the range 0 - 99. The default is 0.

A value of 0 indicates that no load balancing is performed and applicable definitions are selected in alphabetical order. To enable load balancing choose a value in the range 1 - 99 where 1 is the lowest weighting and 99 is the highest. The distribution of connections between two or more channels with non-zero weightings is proportional to the ratio of those weightings. For example, three channels with CLNTWGHT values of 2, 4, and 14 are selected approximately 10%, 20%, and 70% of the time. This distribution is not guaranteed. If the AFFINITY attribute of the connection is set to PREFERRED, the first connection chooses a channel definition according to client weightings, and then subsequent connections continue to use the same channel definition.

This attribute is valid for the client-connection channel type only.

Cluster (CLUSTER):

This attribute is the name of the cluster to which the channel belongs.

The maximum length is 48 characters conforming to the rules for naming IBM MQ objects.

Up to one of the resultant values of CLUSTER or CLUSNL can be non-blank. If one of the values is non-blank, the other must be blank.

This attribute is valid for channel types of:
- Cluster sender
- Cluster receiver

Cluster namelist (CLUSNL):

This attribute is the name of the namelist that specifies a list of clusters to which the channel belongs.

Up to one of the resultant values of CLUSTER or CLUSNL can be nonblank. If one of the values is nonblank, the other must be blank.

This attribute is valid for channel types of:

- Cluster sender
- Cluster receiver

CLWLPRTY channel attribute:

The CLWLPRTY channel attribute specifies the priority order for channels for cluster workload distribution. The value must be in the range 0-9, where 0 is the lowest priority and 9 is the highest.

Use the CLWLPRTY channel attribute to set a priority order for the available cluster destinations. IBM MQ selects the destinations with the highest priority before selecting destinations with the lowest cluster destination priority. If there are multiple destinations with the same priority, it selects the least recently used destination.

If there are two possible destinations, you can use this attribute to allow failover. Messages go to the queue manager with the highest priority channel. If it becomes unavailable then messages go to the next highest priority queue manager. Lower priority queue managers act as reserves.

IBM MQ checks channel status before prioritizing the channels. Only available queue managers are candidates for selection.

Notes:

- Specify this attribute on the cluster-receiver channel at the target queue manager. Any balancing you specify on the matching cluster-sender channel is likely to be ignored. See Cluster channels.
- The availability of a remote queue manager is based on the status of the channel to that queue manager. When channels start, their state changes several times, with some of the states being less preferential to the cluster workload management algorithm. In practice this means that lower priority (backup) destinations can be chosen while the channels to higher priority (primary) destinations are starting.
- If you need to ensure that no messages go to a backup destination, do not use CLWLPRTY. Consider using separate queues, or CLWLRANK with a manual switch over from the primary to back up.

CLWLRANK channel attribute:

The **CLWLRANK** channel attribute specifies the rank of channels for cluster workload distribution. The value must be in the range 0-9, where 0 is the lowest rank and 9 is the highest.

Use the **CLWLRANK** channel attribute if you want control over the final destination for messages sent to a queue manager in another cluster. Control the choice of final destination by setting the rank of the channels connecting a queue manager to the gateway queue managers at the intersection of the clusters.

When you set **CLWLRANK**, messages take a specified route through the interconnected clusters towards a higher ranked destination. For example, messages arrive at a gateway queue manager that can send them to either of two queue managers using channels ranked 1 and 2. They are automatically sent to the queue manager connected by a channel with the highest rank, in this case the channel to the queue manager ranked 2.

IBM MQ gets the rank of channels before checking channel status. Getting the rank before checking channel status means that even non-accessible channels are available for selection. It allows messages to be routed through the network even if the final destination is unavailable.

Notes:

- Specify this attribute on the cluster-receiver channel at the target queue manager. Any balancing you specify on the matching cluster-sender channel is likely to be ignored. See Cluster channels.
- If you also used the priority attribute **CLWLPRTY**, IBM MQ selects between available destinations. If a channel is not available to the destination with the highest rank, the message is held on the transmission queue. It is released when the channel becomes available. The message does not get sent to the next available destination in the rank order.

CLWLWGHT channel attribute:

The CLWLWGHT channel attribute specifies the weight applied to CLUSSDR and CLUSRCVR channels for cluster workload distribution. The value must be in the range 1-99, where 1 is the lowest weight and 99 is the highest.

Use CLWLWGHT to send servers with more processing power more messages. The higher the channel weight, the more messages are sent over that channel.

Notes:

- Specify this attribute on the cluster-receiver channel at the target queue manager. Any balancing you specify on the matching cluster-sender channel is likely to be ignored. See Cluster channels.
- When CLWLWGHT is modified from the default of 50 on any channel, workload balancing becomes dependent on the total number of times each channel was chosen for a message sent to any clustered queue. For more information, see“The cluster workload management algorithm” on page 151.

Connection affinity (AFFINITY):

This attribute specifies whether client applications that connect multiple times using the same queue manager name, use the same client channel.

Use this attribute when multiple applicable channel definitions are available.

The possible values are:

PREFERRED
The first connection in a process reading a client channel definition table (CCDT) creates a list of applicable definitions based on the client channel weight, with any definitions having a weight of 0 first and in alphabetical order. Each connection in the process attempts to connect using the first definition in the list. If a connection is unsuccessful the next definition is used. Unsuccessful definitions with client channel weight values other than 0 are moved to the end of the list. Definitions with a client channel weight of 0 remain at the start of the list and are selected first for each connection.

Each client process with the same host name always creates the same list.

For client applications written in C, C++, or the .NET programming framework (including fully managed .NET), and for applications that use the IBM MQ classes for Java and IBM MQ classes for JMS, the list is updated if the CCDT has been modified since the list was created.

This value is the default value.

NONE
The first connection in a process reading a CCDT creates a list of applicable definitions. All connections in a process select an applicable definition based on the client channel weight, with any definitions having a weight of 0 selected first in alphabetical order.

For client applications written in C, C++, or the .NET programming framework (including fully managed .NET), and for applications that use the IBM MQ classes for Java and IBM MQ classes for JMS, the list is updated if the CCDT has been modified since the list was created.

This attribute is valid for the client-connection channel type only.

Connection name (CONNAME):

This attribute is the communications connection identifier. It specifies the particular communications links to be used by this channel.

It is optional for server channels, unless the server channel is triggered, in which case it must specify a connection name.

Specify **CONNAME** as a comma-separated list of names of machines for the stated **TRPTYPE**. Typically only one machine name is required. You can provide multiple machine names to configure multiple connections with the same properties. The connections are usually tried in the order they are specified in the connection list until a connection is successfully established. The order is modified for clients if the **CLNTWGHT** attribute is provided. If no connection is successful, the channel attempts the connection again, as determined by the attributes of the channel. With client channels, a connection-list provides an alternative to using queue manager groups to configure multiple connections. With message channels, a connection list is used to configure connections to the alternative addresses of a multi-instance queue manager.

Providing multiple connection names in a list was first supported in IBM WebSphere MQ Version 7.0.1. It changes the syntax of the **CONNAME** parameter. Earlier clients and queue managers connect using the first connection name in the list, and do not read the rest of the connection names in the list. In order for the earlier clients and queue managers to parse the new syntax, you must specify a port number on the first

connection name in the list. Specifying a port number avoids problems when connecting to the channel from a client or queue manager that is running at a level earlier than IBM WebSphere MQ Version 7.0.1.

Multi On Multiplatforms, the TCP/IP connection name parameter of a cluster-receiver channel is optional. If you leave the connection name blank, IBM MQ generates a connection name for you, assuming the default port and using the current IP address of the system. You can override the default port number, but still use the current IP address of the system. For each connection name leave the IP name blank, and provide the port number in parentheses; for example:

```
(1415)
```

The generated **CONNAME** is always in the dotted decimal (IPv4) or hexadecimal (IPv6) form, rather than in the form of an alphanumeric DNS host name.

The maximum name length depends on the platform:

- Multi 264 characters.
- z/OS 48 characters (see note 1).

If the transport type is TCP

CONNAME is either the host name or the network address of the remote machine (or the local machine for cluster-receiver channels). For example, `(ABC.EXAMPLE.COM)`, `(2001:DB8:0:0:0:0:0:0)` or `(127.0.0.1)`. It can include the port number, for example `(MACHINE(123))`.

z/OS It can include the IP_name of a dynamic DNS group or a Network Dispatcher input port.

If you use an IPv6 address in a network that only supports IPv4, the connection name is not resolved. In a network which uses both IPv4 and IPv6, the connection name interacts with the local address to determine which IP stack is used. See “Local Address (LOCLADDR)” on page 116 for further information.

If the transport type is LU 6.2

UNIX Windows IBM i If the TPNAME and MODENAME are specified, give the fully-qualified name of the partner LU.

Multi If the TPNAME and MODENAME are blank, give the CPI-C side information object name for your specific platform.

z/OS There are two forms in which to specify the value:

- Logical unit name

  The logical unit information for the queue manager, comprising the logical unit name, TP name, and optional mode name. This name can be specified in one of three forms:

Form	Example
luname	`IGY12355`
luname/TPname	`IGY12345/APING`
luname/TPname/modename	`IGY12345/APINGD/#INTER`

  For the first form, the TP name and mode name must be specified for the TPNAME and MODENAME attributes; otherwise these attributes must be blank. For client-connection channels, only the first form is allowed.

- Symbolic name

  The symbolic destination name for the logical unit information for the queue manager, as defined in the side information data set. The TPNAME and MODENAME attributes must be blank. Note that, for cluster-receiver channels, the side information is on the other queue

managers in the cluster. In this case it can be a name that a channel auto-definition exit can resolve into the appropriate logical unit information for the local queue manager.

The specified or implied LU name can be that of a VTAM generic resources group.

If the transmission protocol is NetBIOS

CONNAME is the NetBIOS name defined on the remote machine.

If the transmission protocol is SPX

CONNAME is an SPX-style address consisting of a 4 byte network address, a 6 byte node address and a 2 byte socket number. Enter these values in hexadecimal, with the network and node addresses separated by a period and the socket number in brackets. For example:

```
CONNAME('0a0b0c0d.804abcde23a1(5e86)')
```

If the socket number is omitted, the default IBM MQ SPX socket number is used. The default is X'5E86'.

This attribute is valid for channel types of:

- Sender
- Server
- Requester
- Client connection
- Cluster sender
- Cluster receiver

It is optional for server channels, unless the server channel is triggered, in which case it must specify a connection name.

Note:

1. For name lengths, you can work around the 48 character limit in either of the following ways:
   - Set up your DNS servers so that you use, for example, host name of "myserver" instead of "myserver.location.company.com", ensuring you can use the short host name.
   - Use IP addresses.
2. The definition of transmission protocol is contained in “Transport type (TRPTYPE)” on page 136.

Convert message (CONVERT):

This attribute specifies that the message must be converted into the format required by the receiving system before transmission.

Application message data is typically converted by the receiving application. However, if the remote queue manager is on a platform that does not support data conversion, use this channel attribute to specify that the message must be converted into the format required by the receiving system ***before*** transmission.

The possible values are `yes` and `no`. If you specify `yes`, the application data in the message is converted before sending if you have specified one of the built-in format names, or a data conversion exit is available for a user-defined format (See Writing data-conversion exits). If you specify `no`, the application data in the message is not converted before sending.

This attribute is valid for channel types of:

- Sender
- Server
- Cluster sender

- Cluster receiver

Data compression (COMPMSG):

This attribute is a list of message data compression techniques supported by the channel.

For sender, server, cluster-sender, cluster-receiver, and client-connection channels the values specified are in order of preference. The first compression technique supported by the remote end of the channel is used. The channels' mutually supported compression techniques are passed to the sending channel's message exit where the compression technique used can be altered on a per message basis. Compression alters the data passed to send and receive exits. See “Header compression (COMPHDR)” on page 114 for compression of the message header.

The possible values are:

NONE
No message data compression is performed. This value is the default value.

RLE Message data compression is performed using run-length encoding.

ZLIBFAST
Message data compression is performed using the zlib compression technique. A fast compression time is preferred.

ZLIBFAST can optionally be offloaded to the zEnterprise® Data Compression facility. See zEDC Express facility for further information.

ZLIBHIGH
Message data compression is performed using the zlib compression technique. A high level of compression is preferred.

ANY Allows the channel to support any compression technique that the queue manager supports. Only supported for Receiver, Requester and Server-Connection channels.

This attribute is valid for all channel types.

Description (DESCR):

This attribute describes the channel definition and contains up to 64 bytes of text.

Note: The maximum number of characters is reduced if the system is using a double byte character set (DBCS).

Use characters from the character set identified by the coded character set identifier (CCSID) for the queue manager to ensure that the text is translated correctly if it is sent to another queue manager.

This attribute is valid for all channel types.

Disconnect interval (DISCINT):

This attribute is the length of time after which a channel closes down, if no message arrives during that period.

This attribute is a time-out attribute, specified in seconds, for the server, cluster-sender, sender, and cluster-receiver channels. The interval is measured from the point at which a batch ends, that is when the batch size is reached or when the batch interval expires and the transmission queue becomes empty. If no messages arrive on the transmission queue during the specified time interval, the channel closes down. (The time is approximate.)

The close-down exchange of control data between the two ends of the channel includes an indication of the reason for closing. This ensures that the corresponding end of the channel remains available to start again.

You can specify any number of seconds from zero through 999 999 where a value of zero means no disconnect; wait indefinitely.

For server-connection channels using the TCP protocol, the interval represents the client inactivity disconnect value, specified in seconds. If a server-connection has received no communication from its partner client for this duration, it terminates the connection. The server-connection inactivity interval applies under the following circumstances:

- between IBM MQ API calls from a client
- between an MQGET call, only if:
  - client application executes an MQGET with WaitInterval
  - the DISCINT parameter is set on the server-connection channel and is smaller than the MQGET WaitInterval
  - the SHARECNV parameter of the server-connection channel is greater than 0.

This attribute is valid for channel types of:

- Sender
- Server
- Server connection
- Cluster sender
- Cluster receiver

This attribute is not applicable for server-connection channels using protocols other than TCP.

Note: Performance is affected by the value specified for the disconnect interval.

A low value (for example a few seconds) can be detrimental to system performance by constantly starting the channel. A large value (more than an hour) might mean that system resources are needlessly held up. You can also specify a heartbeat interval, so that when there are no messages on the transmission queue, the sending MCA sends a heartbeat flow to the receiving MCA, thus giving the receiving MCA an opportunity to quiesce the channel without waiting for the disconnect interval to expire. For these two values to work together effectively, the heartbeat interval value must be significantly lower than the disconnect interval value.

The default DISCINT value is set to 100 minutes. However, a value of a few minutes is often a reasonable value to use without impacting performance or keeping channels running for unnecessarily long periods of time. If it is appropriate for your environment you can change this value, either on each individual channel or through changing the value in the default channel definitions, for example SYSTEM.DEF.SENDER.

For more information, see Stopping and quiescing channels.

Disposition (QSGDISP):

This attribute specifies the disposition of the channel in a queue-sharing group. It is valid on z/OS only.

Values are:

QMGR
The channel is defined on the page set of the queue manager that executes the command. This value is the default.

GROUP
The channel is defined in the shared repository. This value is allowed only if there is a shared queue manager environment. When a channel is defined with QSGDISP(GROUP), the command DEFINE CHANNEL(name) NOREPLACE QSGDISP(COPY) is generated automatically and sent to all active queue managers to cause them to make local copies on page set 0. For queue managers which are not active, or which join the queue-sharing group at a later date, the command is generated when the queue manager starts.

COPY The channel is defined on the page set of the queue manager that executes the command, copying its definition from the QSGDISP(GROUP) channel of the same name. This value is allowed only if there is a shared queue manager environment.

This attribute is valid for all channel types.

Header compression (COMPHDR):

This attribute is a list of header data compression techniques supported by the channel.

For sender, server, cluster-sender, cluster-receiver, and client-connection channels the values specified are in order of preference with the first compression technique supported by the remote end of the channel being used. The channels' mutually supported compression techniques are passed to the sending channel's message exit where the compression technique used can be altered on a per message basis. Compression alters the data passed to send and receive exits.

Possible values are:

NONE
No header data compression is performed. This value is the default value.

SYSTEM
Header data compression is performed.

This attribute is valid for all channel types.

Heartbeat interval (HBINT):

This attribute specifies the approximate time between heartbeat flows that are to be passed from a sending message channel agent (MCA) when there are no messages on the transmission queue.

Heartbeat flows unblock the receiving MCA, which is waiting for messages to arrive or for the disconnect interval to expire. When the receiving MCA is unblocked, it can disconnect the channel without waiting for the disconnect interval to expire. Heartbeat flows also free any storage buffers that have been allocated for large messages and close any queues that have been left open at the receiving end of the channel.

The value is in seconds and must be in the range 0 - 999 999. A value of zero means that no heartbeat flows are to be sent. The default value is 300. To be most useful, the value must be significantly less than the disconnect interval value.

With applications that use IBM MQ classes for Java, JMS or .NET APIs, the HBINT value is determined in one of the following ways:

- Either by the value on the SVRCONN channel that is used by the application.
- Or by the value on the CLNTCONN channel, if the application has been configured to use a CCDT.

For server-connection and client-connection channels, heartbeats can flow from both the server side as well as the client side independently. If no data has been transferred across the channel for the heartbeat interval, the client-connection MQI agent sends a heartbeat flow and the server-connection MQI agent responds to it with another heartbeat flow. This happens irrespective of the state of the channel, for example, irrespective of whether it is inactive while making an API call, or is inactive waiting for client user input. The server-connection MQI agent is also capable of initiating a heartbeat to the client, again irrespective of the state of the channel. To prevent both server-connection and client-connection MQI agents heart beating to each other at the same time, the server heartbeat is flowed after no data has been transferred across the channel for the heartbeat interval plus 5 seconds.

For server-connection and client-connection channels working in the channel mode before IBM WebSphere MQ Version 7.0, heartbeats flow only when a server MCA is waiting for an MQGET command with the WAIT option specified, which it has issued on behalf of a client application.

For more information about making MQI channels work in the two modes, see SharingConversations (MQLONG).

Related information:

DEFINE CHANNEL
Use the MQSC command **DEFINE CHANNEL** to define a new channel, and set its parameters.

ALTER CHANNEL
Use the MQSC command **ALTER CHANNEL** to alter the parameters of a channel.

Keepalive Interval (KAINT):

This attribute is used to specify a timeout value for a channel.

The Keepalive Interval attribute is a value passed to the communications stack specifying the Keepalive timing for the channel. It allows you to specify a different keepalive value for each channel.

You can set the Keepalive Interval (KAINT) attribute for channels on a per-channel basis.

Multi On Multiplatforms, you can access and modify the parameter, but it is only stored and forwarded; there is no functional implementation of the parameter. If you need the functionality provided by the KAINT parameter, use the Heartbeat Interval (HBINT) parameter, as described in "Heartbeat interval (HBINT)" on page 115.

For this attribute to have any effect, TCP/IP keepalive must be enabled. On z/OS, you do enable keepalive by issuing the ALTER QMGR TCPKEEP(YES) MQSC command. On Multiplatforms, it occurs when the KEEPALIVE=YES parameter is specified in the TCP stanza in the distributed queuing configuration file, qm.ini, or through the IBM MQ Explorer. Keepalive must also be enabled within TCP/IP itself, using the TCP profile configuration data set.

The value indicates a time, in seconds, and must be in the range 0 - 99999. A Keepalive Interval value of 0 indicates that channel-specific Keepalive is not enabled for the channel and only the system-wide Keepalive value set in TCP/IP is used. You can also set KAINT to a value of AUTO (this value is the default). If KAINT is set to AUTO, the Keepalive value is based on the value of the negotiated heartbeat interval (HBINT) as follows:

Table 29. Negotiated HBINT value and the corresponding KAINT value.

The table has two columns. The first column lists the negotiated HBINT values and the second column lists the corresponding KAINT value for each negotiated HBINT.

Negotiated HBINT	KAINT
>0	Negotiated HBINT + 60 seconds
0	0

This attribute is valid for all channel types.

The value is ignored for all channels that have a TransportType (TRPTYPE) other than TCP or SPX

Local Address (LOCLADDR):

This attribute specifies the local communications address for the channel.

Note: AMQP channels do not support the same format of LOCLADDR as other IBM MQ channels. For more information, see "LOCLADDR for AMQP channels" on page 119.

LOCLADDR for all channels except AMQP channels

This attribute only applies if the transport type (TRPTYPE) is TCP/IP. For all other transport types, it is ignored.

When a LOCLADDR value is specified, a channel that is stopped and then restarted continues to use the TCP/IP address specified in LOCLADDR. In recovery scenarios, this attribute might be useful when the channel is communicating through a firewall. It is useful because it removes problems caused by the channel restarting with the IP address of the TCP/IP stack to which it is connected. LOCLADDR can also force a channel to use an IPv4 or IPv6 stack on a dual stack system, or a dual-mode stack on a single stack system.

This attribute is valid for channel types of:

- Sender
- Server
- Requester
- Client connection

- Cluster sender
- Cluster receiver

When LOCLADDR includes a network address, the address must be a network addresses belonging to a network interface on the system where the channel is run. For example, when defining a sender channel on queue manager ALPHA to queue manager BETA with the following MSQC command:

```
DEFINE CHANNEL(TO.BETA) CHLTYPE(SDR) CONNAME(192.0.2.0) XMITQ(BETA) LOCLADDR(192.0.2.1)
```

The LOCLADDR address is the IPv4 address `192.0.2.1`. This sender channel runs on the system of queue manager ALPHA, so the IPv4 address must belong to one of the network interfaces its system.

The value is the optional IP address, and optional port or port range used for outbound TCP/IP communications. The format for this information is as follows:

`LOCLADDR([ip-addr][(low-port[,high-port])]`*`[,[ip-addr][(low-port[,high-port])]]`*`)`

The maximum length of **LOCLADDR**, including multiple addresses, is `MQ_LOCAL_ADDRESS_LENGTH`.

If you omit **LOCLADDR**, a local address is automatically allocated.

Note, that you can set **LOCLADDR** for a C client using the Client Channel Definition Table (CCDT).

All the parameters are optional. Omitting the `ip-addr` part of the address is useful to enable the configuration of a fixed port number for an IP firewall. Omitting the port number is useful to select a particular network adapter without having the identify a unique local port number. The TCP/IP stack generates a unique port number.

Specify *`[,[ip-addr][(low-port[,high-port])]]`* multiple times for each additional local address. Use multiple local addresses if you want to specify a specific subset of local network adapters. You can also use *`[,[ip-addr][(low-port[,high-port])]]`* to represent a particular local network address on different servers that are part of a multi-instance queue manager configuration.

`ip-addr`
`ip-addr` is specified in one of three forms:

IPv4 dotted decimal
For example, `192.0.2.1`

IPv6 hexadecimal notation
For example, `2001:DB8:0:0:0:0:0:0`

Alphanumeric host name form
For example `WWW.EXAMPLE.COM`

`low-port and high-port`
`low-port` and `high-port` are port numbers enclosed in parentheses.

The following table shows how the **LOCLADDR** parameter can be used:

Table 30. Examples of how the **LOCLADDR** *parameter can be used*

LOCLADDR	**Meaning**
9.20.4.98	Channel binds to this address locally
9.20.4.98, 9.20.4.99	Channel binds to either IP address. The address might be two network adapters on one server, or a different network adapter on two different servers in a multi-instance configuration.
9.20.4.98(1000)	Channel binds to this address and port 1000 locally
9.20.4.98(1000,2000)	Channel binds to this address and uses a port in the range 1000 - 2000 locally
(1000)	Channel binds to port 1000 locally
(1000,2000)	Channel binds to port in range 1000 - 2000 locally

When a channel is started the values specified for connection name (CONNAME) and local address (LOCLADDR) determine which IP stack is used for communication. The IP stack used is determined as follows:

- If the system has only an IPv4 stack configured, the IPv4 stack is always used. If a local address (LOCLADDR) or connection name (CONNAME) is specified as an IPv6 network address, an error is generated and the channel fails to start.
- If the system has only an IPv6 stack configured, the IPv6 stack is always used. If a local address (LOCLADDR) is specified as an IPv4 network address, an error is generated and the channel fails to start. On platforms supporting IPv6 mapped addressing, if a connection name (CONNAME) is specified as an IPv4 network address, the address is mapped to an IPv6 address. For example, `xxx.xxx.xxx.xxx` is mapped to `::ffff:xxx.xxx.xxx.xxx`. The use of mapped addresses might require protocol translators. Avoid the use of mapped addresses where possible.
- If a local address (LOCLADDR) is specified as an IP address for a channel, the stack for that IP address is used. If the local address (LOCLADDR) is specified as a host name resolving to both IPv4 and IPv6 addresses, the connection name (CONNAME) determines which of the stacks is used. If both the local address (LOCLADDR) and connection name (CONNAME) are specified as host names resolving to both IPv4 and IPv6 addresses, the stack used is determined by the queue manager attribute IPADDRV.
- If the system has dual IPv4 and IPv6 stacks configured and a local address (LOCLADDR) is not specified for a channel, the connection name (CONNAME) specified for the channel determines which IP stack to use. If the connection name (CONNAME) is specified as a host name resolving to both IPv4 and IPv6 addresses, the stack used is determined by the queue manager attribute IPADDRV.

Multi On Multiplatforms, you can set a default local address value that is used for all sender channels that do not have a local address defined. The default value is defined by setting the MQ_LCLADDR environment variable prior to starting the queue manager. The format of the value matches that of MQSC attribute `LOCLADDR`.

Local addresses with cluster sender channels

Cluster sender channels always inherit the configuration of the corresponding cluster receiver channel as defined on the target queue manager. This is true even if there is a locally defined cluster sender channel of the same name, in which case the manual definition is only used for initial communication.

For this reason, it is not possible to depend on the LOCLADDR defined in the cluster receiver channel as it is likely that the IP address is not owned by the system where the cluster senders are created. For this reason, the LOCLADDR on the cluster receiver should not be used unless there is a reason to restrict only the ports but not the IP address for all potential cluster senders, and it is known that those ports are available on all systems where a cluster sender channel may be created.

If a cluster must use LOCLADDR to get the outbound communication channels to bind to a specific IP address, either use a Channel Auto-Definition Exit, or use the default LOCLADDR for the queue manager when possible. When using a channel exit, it forces the LOCLADDR value from the exit into any of the automatically defined CLUSSDR channels.

If using a non-default LOCLADDR for cluster sender channels through the use of an exit or a default value, any matching manually defined cluster sender channel, for example to a full repository queue manager, must also have the LOCLADDR value set to enable initial communication over the channel.

Note: If the operating system returns a bind error for the port supplied in LOCLADDR (or all ports, if a port range is supplied), the channel does not start; the system issues an error message.

LOCLADDR for AMQP channels

AMQP channels support a different format of LOCLADDR than other IBM MQ channels:

LOCLADDR (*ip-addr*)

LOCLADDR is the local communications address for the channel. Use this parameter if you want to force the client to use a particular IP address. LOCLADDR is also useful to force a channel to use an IPv4 or IPv6 address if a choice is available, or to use a particular network adapter on a system with multiple network adapters.

The maximum length of LOCLADDR is `MQ_LOCAL_ADDRESS_LENGTH`.

If you omit LOCLADDR, a local address is automatically allocated.

ip-addr

`ip-addr` is a single network address, specified in one of three forms:

IPv4 dotted decimal
For example `192.0.2.1`

IPv6 hexadecimal notation
For example `2001:DB8:0:0:0:0:0:0`

Alphanumeric host name form
For example `WWW.EXAMPLE.COM`

If an IP address is entered, only the address format is validated. The IP address itself is not validated.

Related information:

Working with auto-defined cluster-sender channels

Long retry count (LONGRTY):

This attribute specifies the maximum number of times that the channel is to try allocating a session to its partner.

If the initial allocation attempt fails, the *short retry count* number is decremented and the channel retries the remaining number of times. If it still fails, it retries a *long retry count* number of times with an interval of *long retry interval* between each try. If it is still unsuccessful, the channel closes down. The channel must then be restarted with a command (it is not started automatically by the channel initiator).

(Retry is not attempted if the cause of failure is such that a retry is not likely to be successful.)

If the channel initiator (on z/OS) or the channel (on Multiplatforms) is stopped while the channel is retrying, the *short retry count* and *long retry count* are reset when the channel initiator or the channel is restarted, or when a message is successfully put at the sender channel. However, if the channel initiator (on z/OS) or queue manager (on Multiplatforms) is shut down and restarted, the *short retry count* and *long retry count* are not reset. The channel retains the retry count values it had before the queue manager restart or the message being put.

Note: For IBM i, UNIX, and Windows systems:

1. When a channel goes from RETRYING state to RUNNING state, the *short retry count* and *long retry count* are not reset immediately. They are reset only once the first message flows across the channel successfully after the channel went into RUNNING state, that is; once the local channel confirms the number of messages sent to the other end.
2. The *short retry count* and *long retry count* are reset when the channel is restarted.

The *long retry count* attribute can be set from zero through 999 999 999.

This attribute is valid for channel types of:

- Sender
- Server
- Cluster sender
- Cluster receiver

Note: For IBM i IBM i, UNIX, and Windows systems, in order for retry to be attempted a channel initiator must be running. The channel initiator must be monitoring the initiation queue specified in the definition of the transmission queue that the channel is using.

Long retry interval (LONGTMR):

This attribute is the approximate interval in seconds that the channel is to wait before retrying to establish connection, during the long retry mode.

The interval between retries can be extended if the channel has to wait to become active.

The channel tries to connect *long retry count* number of times at this long interval, after trying the *short retry count* number of times at the short retry interval.

This attribute can be set from zero through 999 999.

This attribute is valid for channel types of:

- Sender
- Server
- Cluster sender
- Cluster receiver

LU 6.2 mode name (MODENAME):

This attribute is for use with LU 6.2 connections. It gives extra definition for the session characteristics of the connection when a communication session allocation is performed.

When using side information for SNA communications, the mode name is defined in the CPI-C Communications Side Object or APPC side information, and this attribute must be left blank; otherwise, it must be set to the SNA mode name.

The name must be one to eight alphanumeric characters long.

This attribute is valid for channel types of:

- Sender
- Server
- Requester
- Client connection
- Cluster sender
- Cluster receiver

It is not valid for receiver or server-connection channels.

LU 6.2 transaction program name (TPNAME):

This attribute is for use with LU 6.2 connections. It is the name, or generic name, of the transaction program (MCA) to be run at the far end of the link.

When using side information for SNA communications, the transaction program name is defined in the CPI-C Communications Side Object or APPC side information and this attribute must be left blank. Otherwise, this name is required by sender channels and requester channels.

The name can be up to 64 characters long.

The name must be set to the SNA transaction program name, unless the CONNAME contains a side-object name in which case it must be set to blanks. The actual name is taken instead from the CPI-C Communications Side Object, or the APPC side information data set.

This information is set in different ways on different platforms; see Configuring distributed queuing for more information about setting up communication for your platform.

This attribute is valid for channel types of:

- Sender
- Server
- Requester
- Client connection
- Cluster sender
- Cluster receiver

Maximum instances (MAXINST):

This attribute specifies the maximum number of simultaneous instances of a server-connection channel or AMQP channel that can be started.

See the child topics for information on how the attribute is used for each channel type.

Maximum instances of server-connection channel connections:

This attribute specifies the maximum number of simultaneous instances of a sever-connection channel that can be started.

This attribute can be set from zero through 999 999 999. A value of zero indicates that no client connections are allowed on this channel. The default value is 999 999 999.

If the value is reduced so that it is less than the number of instances of the server-connection channel that are currently running, then the running channels are not affected. However, new instances are not able to start until sufficient existing ones have ceased to run.

Maximum instances of AMQP channel connections: V 9.0.0

This attribute specifies the maximum number of simultaneous instances of an AMQP channel that can be started.

This attribute can be set from zero through 999 999 999. A value of zero indicates that no client connections are allowed on this channel. The default value is 999 999 999.

If a client attempts to connect, and the number of connected clients has reached MAXINST, the channel closes the connection with a close frame. The close frame contains the following message:

```
amqp:resource-limit-exceeded
```

If a client connects with an ID that is already connected (that is, it performs a client-takeover) the takeover will succeed regardless of whether the number of connected clients has reached MAXINST.

Maximum instances per client (MAXINSTC):

This attribute specifies the maximum number of simultaneous instances of a server-connection channel that can be started from a single client.

This attribute can be set from zero through 999 999 999. A value of zero indicates that no client connections are allowed on this channel. The default value is 999 999 999.

If the value is reduced so that it is less than the number of instances of the server-connection channel that are currently running from individual clients, then the running channels are not affected. However, new instances from those clients are not able to start until sufficient existing ones have ceased to run.

This attribute is valid for server-connection channels only.

Maximum message length (MAXMSGL):

This attribute specifies the maximum length of a message that can be transmitted on the channel.

On IBM MQ for IBM i, UNIX, and Windows systems, specify a value greater than or equal to zero, and less than or equal to the maximum message length for the queue manager. See the MAXMSGL parameter of the ALTER QMGR command in ALTER QMGR for more information. On IBM MQ for z/OS, specify a value greater than or equal to zero, and less than or equal to 104 857 600 bytes.

Because various implementations of IBM MQ systems exist on different platforms, the size available for message processing might be limited in some applications. This number must reflect a size that your system can handle without stress. When a channel starts, the lower of the two numbers at each end of the channel is taken.

By adding the digital signature and key to the message, Advanced Message Security increases the length of the message.

Note:

1. You can use a maximum message size of 0 which is taken to mean that the size is to be set to the local queue manager maximum value.

This attribute is valid for all channel types.

Message channel agent name (MCANAME):

This attribute is reserved and if specified must only be set to blanks.

Its maximum length is 20 characters.

Message channel agent type (MCATYPE):

This attribute can specify the message channel agent as a *process* or a *thread*.

On IBM MQ for z/OS, it is supported only for channels with a channel type of cluster-receiver.

Advantages of running as a process include:
- Isolation for each channel providing greater integrity
- Job authority specific for each channel
- Control over job scheduling

Advantages of threads include:
- Much reduced use of storage
- Easier configuration by typing on the command line
- Faster execution - it is quicker to start a thread than to instruct the operating system to start a process

For channel types of sender, server, and requester, the default is `process`. For channel types of cluster-sender and cluster-receiver, the default is `thread`. These defaults can change during your installation.

If you specify `process` on the channel definition, a RUNMQCHL process is started. If you specify `thread`, the MCA runs on a thread of the AMQRMPPA process, or of the RUNMQCHI process if MQNOREMPOOL is specified. On the machine that receives the inbound allocates, the MCA runs as a thread if you use RUNMQLSR. It runs as a process if you use **`inetd`**.

On IBM MQ for z/OS, this attribute is supported only for channels with a channel type of cluster-receiver. On other platforms, it is valid for channel types of:
- Sender
- Server
- Requester
- Cluster sender
- Cluster receiver

Message channel agent user identifier (MCAUSER):

This attribute is the user identifier (a string) to be used by the MCA for authorization to access IBM MQ resources.

Note: An alternative way of providing a user ID for a channel to run under is to use channel authentication records. With channel authentication records, different connections can use the same channel while using different credentials. If both MCAUSER on the channel is set and channel authentication records are used to apply to the same channel, the channel authentication records take precedence. The MCAUSER on the channel definition is only used if the channel authentication record uses USERSRC(CHANNEL).

This authorization includes (if PUT authority is DEF) putting the message to the destination queue for receiver or requester channels.

On IBM MQ for Windows, the user identifier can be domain-qualified by using the format, `user@domain`, where the `domain` must be either the Windows systems domain of the local system, or a trusted domain.

If this attribute is blank, the MCA uses its default user identifier. For more information, see DEFINE CHANNEL.

This attribute is valid for channel types of:

- Receiver
- Requester
- Server connection
- Cluster receiver

Related information:

Channel authentication records

Message exit name (MSGEXIT):

This attribute specifies the name of the user exit program to be run by the channel message exit.

This attribute can be a list of names of programs that are to be run in succession. Leave blank, if no channel message exit is in effect.

The format and maximum length of this attribute depend on the platform, as for “Receive exit name (RCVEXIT)” on page 130.

This attribute is valid for channel types of:

- Sender
- Server
- Receiver
- Requester
- Cluster sender
- Cluster receiver

Message exit user data (MSGDATA):

This attribute specifies user data that is passed to the channel message exits.

You can run a sequence of message exits. The limitations on the user data length and an example of how to specify MSGDATA for more than one exit are as shown for RCVDATA. See “Receive exit user data (RCVDATA)” on page 131.

This attribute is valid for channel types of:

- Sender
- Server
- Receiver
- Requester
- Cluster sender
- Cluster receiver

Message-retry exit name (MREXIT):

This attribute specifies the name of the user exit program to be run by the message-retry user exit.

Leave blank if no message-retry exit program is in effect.

The format and maximum length of the name depend on the platform, as for “Receive exit name (RCVEXIT)” on page 130. However, there can only be one message-retry exit specified

This attribute is valid for channel types of:

- Receiver
- Requester
- Cluster receiver

Message-retry exit user data (MRDATA):

This attribute specifies data passed to the channel message-retry exit when it is called.

This attribute is valid for channel types of:

- Receiver
- Requester
- Cluster receiver

Message retry count (MRRTY):

This attribute specifies the number of times the channel tries to redeliver the message.

This attribute controls the action of the MCA only if the message-retry exit name is blank. If the exit name is not blank, the value of MRRTY is passed to the exit, but the number of attempts made (if any) is controlled by the exit, and not by this attribute.

The value must be in the range 0 - 999 999 999. A value of zero means that no additional attempts are made. The default is 10.

This attribute is valid for channel types of:

- Receiver
- Requester
- Cluster receiver

Message retry interval (MRTMR):

This attribute specifies the minimum interval of time that must pass before the channel can retry the MQPUT operation.

This time interval is in milliseconds.

This attribute controls the action of the MCA only if the message-retry exit name is blank. If the exit name is not blank, the value of MRTMR is passed to the exit for use by the exit, but the retry interval is controlled by the exit, and not by this attribute.

The value must be in the range 0 - 999 999 999. A value of zero means that the retry is performed as soon as possible (if the value of MRRTY is greater than zero). The default is 1000.

This attribute is valid for the following channel types:

- Receiver
- Requester
- Cluster receiver

Monitoring (MONCHL):

This attribute controls the collection of online Monitoring data.

Possible values are:

QMGR
: The collection of Online Monitoring Data is inherited from the setting of the MONCHL attribute in the queue manager object. This value is the default value.

OFF Online Monitoring Data collection for this channel is disabled.

LOW A low ratio of data collection with a minimal effect on performance. However, the monitoring results shown might not be up to date.

MEDIUM
: A moderate ratio of data collection with limited effect on the performance of the system.

HIGH A high ratio of data collection with the possibility of an effect on performance. However, the monitoring results shown are the most current.

This attribute is valid for channel types of:

- Sender
- Server
- Receiver
- Requester
- Server connection
- Cluster sender
- Cluster receiver

For more information about monitoring data, see Displaying queue and channel monitoring data.

NETPRTY channel attribute:

The NETPRTY channel attribute specifies the priority for a CLUSRCVR channel. The value must be in the range 0-9, where 0 is the lowest priority and 9 is the highest.

Use the NETPRTY attribute to make one network the primary network, and another network the backup network. Given a set of equally ranked channels, clustering chooses the path with the highest priority when multiple paths are available.

A typical example of using the NETPRTY channel attribute is to differentiate between networks that have different costs or speeds and connect the same destinations.

Note: Specify this attribute on the cluster-receiver channel at the target queue manager. Any balancing you specify on the matching cluster-sender channel is likely to be ignored. See Cluster channels.

Nonpersistent message speed (NPMSPEED):

This attribute specifies the speed at which nonpersistent messages are to be sent.

Possible values are:

NORMAL
Nonpersistent messages on a channel are transferred within transactions.

FAST Nonpersistent messages on a channel are not transferred within transactions.

The default is FAST. The advantage of this is that nonpersistent messages become available for retrieval far more quickly. The disadvantage is that because they are not part of a transaction, messages might be lost if there is a transmission failure or if the channel stops when the messages are in transit. See Safety of messages.

Notes:

1. If the active recovery logs for IBM MQ for z/OS are switching and archiving more frequently than expected, given that the messages being sent across a channel are non-persistent, setting NPMSPEED(FAST) on both the sending and receiving ends of the channel can minimize the SYSTEM.CHANNEL.SYNCQ updates.
2. If you are seeing high CPU usage relating to updates to the SYSTEM.CHANNEL.SYNCQ, setting NPMSPEED(FAST) can significantly reduce the CPU usage.

This attribute is valid for channel types of:

- Sender
- Server
- Receiver
- Requester

- Cluster sender
- Cluster receiver

Password (PASSWORD):

This attribute specifies a password that can be used by the MCA when attempting to initiate a secure LU 6.2 session with a remote MCA.

You can specify a password of maximum length 12 characters, although only the first 10 characters are used.

It is valid for channel types of sender, server, requester, or client-connection.

On IBM MQ for z/OS, this attribute is valid only for client connection channels. On other platforms, it is valid for channel types of:

- Sender
- Server
- Requester
- Client connection
- Cluster sender

Port number (PORT): V 9.0.0

Specify the port number that is used to connect the AMQP client.

The default port for AMQP 1.0 connections is 5672. If you are already using port 5672, you can specify a different port.

PUT authority (PUTAUT):

This attribute specifies the type of security processing to be carried out by the MCA.

This attribute is valid for channel types of:

- Receiver
- Requester
- Server connection (z/OS only)
- Cluster receiver

Use this attribute to choose the type of security processing to be carried out by the MCA when executing:

- An MQPUT command to the destination queue (for message channels), or
- An MQI call (for MQI channels).

z/OS

On z/OS, the user IDs that are checked, and how many user IDs are checked, depends on the setting of the MQADMIN RACF® class hlq.RESLEVEL profile. Depending on the level of access the user ID of the channel initiator has to hlq.RESLEVEL, zero, one or two user IDs are checked. To see how many user IDs are checked, see RESLEVEL and channel initiator connections. For more information about which user IDs are checked, see User IDs used by the channel initiator.

You can choose one of the following:

Process security, also called default authority (DEF)
The default user ID is used.

On platforms other than z/OS, the user ID used to check open authority on the queue is that of the process or user running the MCA at the receiving end of the message channel.

On z/OS, both the user ID received from the network, and the user ID derived from MCAUSER might be used, depending on the number of user IDs that are to be checked.

The queues are opened with this user ID and the open option MQOO_SET_ALL_CONTEXT.

Context security (CTX)
The user ID from the context information associated with the message is used as an alternate user ID.

The *UserIdentifier* in the message descriptor is moved into the *AlternateUserId* field in the object descriptor. The queue is opened with the open options MQOO_SET_ALL_CONTEXT and MQOO_ALTERNATE_USER_AUTHORITY.

On platforms other than z/OS, the user ID used to check open authority on the queue for MQOO_SET_ALL_CONTEXT and MQOO_ALTERNATE_USER_AUTHORITY is that of the process or user running the MCA at the receiving end of the message channel. The user ID used to check open authority on the queue for MQOO_OUTPUT is the *UserIdentifier* in the message descriptor.

On z/OS, the user ID received from the network or that derived from MCAUSER might be used, as well as the user ID from the context information in the message descriptor, depending on the number of user IDs that are to be checked.

Context security (CTX) is not supported on server-connection channels.

Only Message Channel Agent security (ONLYMCA)
The user ID derived from MCAUSER is used.

Queues are opened with the open option MQOO_SET_ALL_CONTEXT.

This value only applies to z/OS.

Alternate Message Channel Agent security (ALTMCA)
The user ID from the context information (the *UserIdentifier* field) in the message descriptor might be used, as well as the user ID derived from MCAUSER, depending on the number of user IDs that are to be checked.

This value only applies to z/OS.

Further details about context fields and open options can be found in Controlling context information.

More information about security can be found here:

- Securing
- ULW Setting up security on UNIX, Linux, and Windows
- IBM i Setting up security on IBM i
- z/OS Setting up security on z/OS

Queue manager name (QMNAME):

This attribute specifies the name of the queue manager or queue manager group to which an IBM MQ MQI client application can request connection.

This attribute is valid for channel types of:

- Client connection

Receive exit name (RCVEXIT):

This attribute specifies the name of the user exit program to be run by the channel receive user exit.

This attribute can be a list of names of programs that are to be run in succession. Leave blank, if no channel receive user exit is in effect.

The format and maximum length of this attribute depend on the platform:

- On z/OS it is a load module name, maximum length 8 characters, except for client-connection channels where the maximum length is 128 characters.
- On IBM i, it is of the form:
  libname/progname

  when specified in CL commands.

  When specified in IBM MQ Commands (MQSC) it has the form:
  progname libname

  where *progname* occupies the first 10 characters, and *libname* the second 10 characters (both blank-padded to the right if necessary). The maximum length of the string is 20 characters.
- On Windows, it is of the form:
  dllname(*functionname*)

  where *dllname* is specified without the suffix `.DLL`. The maximum length of the string is 40 characters.
- On UNIX, it is of the form:
  libraryname(*functionname*)

  The maximum length of the string is 40 characters.

During cluster sender channel auto-definition on z/OS, channel exit names are converted to z/OS format. If you want to control how exit names are converted, you can write a channel auto-definition exit. For more information, see Channel auto-definition exit program.

You can specify a list of receive, send, or message exit program names. The names must be separated by a comma, a space, or both. For example:

```
RCVEXIT(exit1 exit2)
MSGEXIT(exit1,exit2)
SENDEXIT(exit1, exit2)
```

The total length of the string of exit names and strings of user data for a particular type of exit is limited to 500 characters. In IBM MQ for IBM i, you can list up to 10 exit names. In IBM MQ for z/OS, you can list up to eight exit names.

This attribute is valid for all channel types.

Receive exit user data (RCVDATA):

This attribute specifies user data that is passed to the receive exit.

You can run a sequence of receive exits. The string of user data for a series of exits must be separated by a comma, spaces, or both. For example:

```
RCVDATA(exit1_data exit2_data)
MSGDATA(exit1_data,exit2_data)
SENDDATA(exit1_data, exit2_data)
```

In IBM MQ for UNIX systems, and Windows systems, the length of the string of exit names and strings of user data is limited to 500 characters. In IBM MQ for IBM i, you can specify up to 10 exit names and the length of user data for each is limited to 32 characters. In IBM MQ for z/OS, you can specify up to eight strings of user data each of length 32 characters.

This attribute is valid for all channel types.

Security exit name (SCYEXIT):

This attribute specifies the name of the exit program to be run by the channel security exit.

Leave blank if no channel security exit is in effect.

The format and maximum length of the name depend on the platform, as for "Receive exit name (RCVEXIT)" on page 130. However, you can only specify one security exit.

This attribute is valid for all channel types.

Security exit user data (SCYDATA):

This attribute specifies user data that is passed to the security exit.

The maximum length is 32 characters.

This attribute is valid for all channel types.

Send exit name (SENDEXIT):

This attribute specifies the name of the exit program to be run by the channel send exit.

This attribute can be a list of names of programs that are to be run in sequence. Leave blank if no channel send exit is in effect.

The format and maximum length of this attribute depend on the platform, as for "Receive exit name (RCVEXIT)" on page 130.

This attribute is valid for all channel types.

Send exit user data (SENDDATA):

This attribute specifies user data that is passed to the send exit.

You can run a sequence of send exits. The limitations on the user data length and an example of how to specify SENDDATA for more than one exit, are as shown for RCVDATA. See "Receive exit user data (RCVDATA)" on page 131.

This attribute is valid for all channel types.

Sequence number wrap (SEQWRAP):

This attribute specifies the highest number the message sequence number reaches before it restarts at 1.

The value of the number must be high enough to avoid a number being reissued while it is still being used by an earlier message. The two ends of a channel must have the same sequence number wrap value when a channel starts; otherwise, an error occurs.

The value can be set from 100 through 999 999 999.

This attribute is valid for channel types of:

- Sender
- Server
- Receiver
- Requester
- Cluster sender
- Cluster receiver

Short retry count (SHORTRTY):

This attribute specifies the maximum number of times that the channel is to try allocating a session to its partner.

If the initial allocation attempt fails, the *short retry count* is decremented and the channel retries the remaining number of times with an interval, defined in the **short retry interval** attribute, between each attempt. If it still fails, it retries *long retry count* number of times with an interval of *long retry interval* between each attempt. If it is still unsuccessful, the channel terminates.

(Retry is not attempted if the cause of failure is such that a retry is not likely to be successful.)

If the channel initiator (on z/OS) or the channel (on Multiplatforms) is stopped while the channel is retrying, the *short retry count* and *long retry count* are reset when the channel initiator or the channel is restarted, or when a message is successfully put at the sender channel. However, if the channel initiator (on z/OS) or queue manager (on Multiplatforms) is shut down and restarted, the *short retry count* and *long retry count* are not reset. The channel retains the retry count values it had before the queue manager restart or the message being put.

Note: For IBM i IBM i, UNIX, and Windows systems:

1. When a channel goes from RETRYING state to RUNNING state, the *short retry count* and *long retry count* are not reset immediately. They are reset only once the first message flows across the channel successfully after the channel went into RUNNING state, that is; once the local channel confirms the number of messages sent to the other end.
2. The *short retry count* and *long retry count* are reset when the channel is restarted.

This attribute can be set from zero through 999 999 999.

This attribute is valid for channel types of:
- Sender
- Server
- Cluster sender
- Cluster receiver

Note: On IBM i IBM i, UNIX, and Windows systems, in order for retry to be attempted a channel initiator must be running. The channel initiator must be monitoring the initiation queue specified in the definition of the transmission queue that the channel is using.

Short retry interval (SHORTTMR):

This attribute specifies the approximate interval in seconds that the channel is to wait before retrying to establish connection, during the short retry mode.

The interval between retries might be extended if the channel has to wait to become active.

This attribute can be set from zero through 999 999.

This attribute is valid for channel types of:
- Sender
- Server
- Cluster sender
- Cluster receiver

SSL Cipher Specification (SSLCIPH):

This attribute specifies a single CipherSpec for a TLS connection.

Every IBM MQ channel definition includes the SSLCIPH attribute. The value is a string with a maximum length of 32 characters.

Note the following:
- The SSLCIPH attribute can contain a blank value, meaning that you are not using TLS. If one end of the channel has a blank SSLCIPH attribute, the other end of the channel must also have a blank SSLCIPH attribute.
- Alternatively, if SSLCIPH contains a nonblank value, the channel attempts to use the specified cipher to use TLS. Again, in this case, both ends of the channel must specify the same SSLCIPH value.
- The only exception to the rule that SSLCIPH must be the same at both ends of a channel, is that a fully-managed .NET client can specify the special value **`*NEGOTIATE`**. This option allows the channel to select the most recent protocol version supported by the .NET framework, and negotiate a CipherSpec that the server supports.

It is valid only for channels with a transport type (TRPTYPE) of TCP. If the TRPTYPE is not TCP, the data is ignored and no error message is issued.

For more information about SSLCIPH, see DEFINE CHANNEL and Specifying CipherSpecs.

SSL Client Authentication (SSLCAUTH):

This attribute specifies whether the channel needs to receive and authenticate a TLS certificate from a TLS client.

Possible values are:

OPTIONAL
If the peer TLS client sends a certificate, the certificate is processed as normal but authentication does not fail if no certificate is sent.

REQUIRED
If the TLS client does not send a certificate, authentication fails.

The default value is REQUIRED.

You can specify a value for SSLCAUTH on a non-TLS channel definition. That is, a channel definition on which the SSLCIPH attribute is missing or blank. You can temporarily disable TLS for debugging by setting the value of SSLCAUTH to OPTIONAL. Therefore you do not have to clear and then re-input the TLS parameters.

SSLCAUTH is an optional attribute.

This attribute is valid on all channel types that can ever receive a channel initiation flow, except for sender channels.

This attribute is valid for channel types of:
- Server
- Receiver
- Requester
- Server connection
- Cluster receiver

For more information about SSLCAUTH, see DEFINE CHANNEL (MQTT) and Securing.

SSL Peer (SSLPEER):

This attribute is used to check the Distinguished Name (DN) of the certificate from the peer queue manager or client at the other end of an IBM MQ channel.

Note: An alternative way of restricting connections into channels by matching against the TLS Subject Distinguished Name, is to use channel authentication records. With channel authentication records, different TLS Subject Distinguished Name patterns can be applied to the same channel. If both SSLPEER on the channel and a channel authentication record are used to apply to the same channel, the inbound certificate must match both patterns in order to connect.

If the DN received from the peer does not match the SSLPEER value, the channel does not start.

SSLPEER is an optional attribute. If a value is not specified, the peer DN is not checked when the channel is started.

On z/OS, the maximum length of the attribute is 256 bytes. On all other platforms, it is 1024 bytes. Channel authentication records provide greater flexibility when using SSLPEER and support 1024 bytes on all platforms.

On z/OS, the attribute values used are not checked. If you enter incorrect values, the channel fails at startup, and error messages are written to the error log at both ends of the channel. A Channel SSL Error event is also generated at both ends of the channel. On platforms that support SSLPEER, other than z/OS, the validity of the string is checked when it is first entered.

You can specify a value for SSLPEER on a non-TLS channel definition, one on which SSLCIPH is missing or blank. You can use this to temporarily disable TLS for debugging without having to clear and later re-input the TLS parameters.

For more information about using SSLPEER, see SET CHLAUTH and Securing.

This attribute is valid for all channel types.

Related information:

Channel authentication records

Topic root (TPROOT): V 9.0.0

This attribute specifies the topic root for an AMQP channel.

You can use the TPROOT attribute to specify a topic root for an AMQP channel. Using this attribute ensures that an MQ Light application, when deployed to a queue manager, does not publish or subscribe to messages to or from areas of the topic tree that are being used by other applications.

The default value for TPROOT is SYSTEM.BASE.TOPIC. With this value, the topic string an AMQP client uses to publish or subscribe has no prefix, and the client can exchange messages with other MQ pub/sub applications. To have AMQP clients publish and subscribe under a topic prefix, first create an MQ topic object with a topic string set to the prefix you want, then change the value of the AMQP channel TPROOT attribute to the name of the MQ topic object you created. The following example shows the topic root being set to APPGROUP1.BASE.TOPIC for AMQP channel MYAMQP:

```
DEFINE CHANNEL(MYAMQP) CHLTYPE(AMQP) TPROOT(APPGROUP1.BASE.TOPIC) PORT(5673)
```

Note: If the TPROOT attribute value, or the topic string that underpins it, is changed, existing AMQP topics and their messages might be orphaned.

Transmission queue name (XMITQ):

This attribute specifies the name of the transmission queue from which messages are retrieved.

This attribute is required for channels of type sender or server, it is not valid for other channel types.

Provide the name of the transmission queue to be associated with this sender or server channel, that corresponds to the queue manager at the far side of the channel. You can give the transmission queue the same name as the queue manager at the remote end.

This attribute is valid for channel types of:

- Sender
- Server

Transport type (TRPTYPE):

This attribute specifies the transport type to be used.

The possible values are:

LU62	LU 6.2
TCP	TCP/IP
NETBIOS	NetBIOS (1)
SPX	SPX (1)
Notes: 1. For use on Windows. Can also be used on z/OS for defining client-connection channels for use on Windows.	

This attribute is valid for all channel types.

Use client ID (USECLTID): 

Use client ID for connection to AMQP channel.

Specify whether the client ID is used for connection on an AMQP channel. Set to `Yes` or `No`.

Use Dead-Letter Queue (USEDLQ):

This attribute determines whether the dead-letter queue (or undelivered message queue) is used when messages cannot be delivered by channels.

Possible values are:

NO Messages that cannot be delivered by a channel are treated as a failure. The channel either discards these messages, or the channel ends, in accordance with the setting of NPMSPEED.

YES (default)
If the queue manager DEADQ attribute provides the name of a dead-letter queue, then it is used, otherwise the behavior is as for NO.

User ID (USERID):

This attribute specifies the user ID to be used by the MCA when attempting to initiate a secure SNA session with a remote MCA.

You can specify a task user identifier of 20 characters.

It is valid for channel types of sender, server, requester, or client-connection.

This attribute does not apply to IBM MQ for z/OS except for client-connection channels.

On the receiving end, if passwords are kept in encrypted format and the LU 6.2 software is using a different encryption method, an attempt to start the channel fails with invalid security details. You can avoid this failure by modifying the receiving SNA configuration to either:

- Turn off password substitution, or
- Define a security user ID and password.

On IBM MQ for z/OS, this attribute is valid only for client connection channels. On other platforms, it is valid for channel types of:

- Sender
- Server
- Requester
- Client connection
- Cluster sender

IBM MQ cluster commands

The IBM MQ Script commands **runmqsc** commands have special attributes and parameters that apply to clusters. There are other administrative interfaces you can use to manager clusters.

The MQSC commands are shown as they would be entered by the system administrator at the command console. Remember that you do not have to issue the commands in this way. There are a number of other methods, depending on your platform; for example:

- On IBM MQ for IBM i, you run MQSC commands interactively from option 26 of **WRKMQM**. You can also use CL commands or you can store MQSC commands in a file and use the **STRMQMMQSC** CL command.
- z/OS On z/OS you can use the COMMAND function of the **CSQUTIL** utility, the operations and control panels or you can use the z/OS console.
- On all other platforms, you can store the commands in a file and use **runmqsc**.

In a MQSC command, a cluster name, specified using the CLUSTER attribute, can be up to 48 characters long.

A list of cluster names, specified using the CLUSNL attribute, can contain up to 256 names. To create a cluster namelist, use the DEFINE NAMELIST command.

IBM MQ Explorer

The IBM MQ Explorer GUI can administer a cluster with repository queue managers on IBM MQ for z/OS Version 6 or later. You do not need to nominate an additional repository on a separate system. For earlier versions of IBM MQ for z/OS, the IBM MQ Explorer cannot administer a cluster with repository queue managers. You must therefore nominate an additional repository on a system that the IBM MQ Explorer can administer.

On IBM MQ for Windows and IBM MQ for Linux, you can also use IBM MQ Explorer to work with clusters. You can also use the stand-alone IBM MQ Explorer client.

Using the IBM MQ Explorer, you can view cluster queues and inquire about the status of cluster-sender and cluster-receiver channels. IBM MQ Explorer includes two wizards, which you can use to guide you through the following tasks:

- Create a cluster
- Join an independent queue manager to a cluster

Programmable command formats (PCF)

Table 31. PCF equivalents of MQSC commands specifically to work with clusters

runmqsc command	PCF equivalent
DISPLAY CLUSQMGR	MQCMD_INQUIRE_CLUSTER_Q_MGR
SUSPEND QMGR	MQCMD_SUSPEND_Q_MGR_CLUSTER
RESUME QMGR	MQCMD_RESUME_Q_MGR_CLUSTER
REFRESH CLUSTER	MQCMD_REFRESH_CLUSTER
RESET CLUSTER	MQCMD_RESET_CLUSTER

Related information:
Clustering: Using REFRESH CLUSTER best practices

Queue manager definition commands

Cluster attributes that can be specified on queue manager definition commands.

To specify that a queue manager holds a full repository for a cluster, use the ALTER QMGR command specifying the attribute REPOS(*clustername*). To specify a list of several cluster names, define a cluster namelist and then use the attribute REPOSNL(*namelist*) on the ALTER QMGR command:

```
DEFINE NAMELIST(CLUSTERLIST)
       DESCR('List of clusters whose repositories I host')
       NAMES(CLUS1, CLUS2, CLUS3)
 ALTER QMGR REPOSNL(CLUSTERLIST)
```

You can provide additional cluster attributes on the ALTER QMGR command

CLWLEXIT(*name*)
Specifies the name of a user exit to be called when a message is put to a cluster queue.

CLWLDATA(*data*)
Specifies the data to be passed to the cluster workload user exit.

CLWLLEN(*length*)
Specifies the maximum amount of message data to be passed to the cluster workload user exit

CLWLMRUC(*channels*)
Specifies the maximum number of outbound cluster channels.

CLWLMRUC is a local queue manager attribute that is not propagated around the cluster. It is made available to cluster workload exits and the cluster workload algorithm that chooses the destination for messages.

CLWLUSEQ(LOCAL|ANY)
Specifies the behavior of MQPUT when the target queue has both a local instance and at least one remote cluster instance. If the put originates from a cluster channel, this attribute does not apply. It is possible to specify CLWLUSEQ as both a queue attribute and a queue manager attribute.

If you specify ANY, both the local queue and the remote queues are possible targets of the MQPUT.

If you specify LOCAL, the local queue is the only target of the MQPUT.

The equivalent PCFs are MQCMD_CHANGE_Q_MGR and MQCMD_INQUIRE_Q_MGR.

Channel definition commands

Cluster attributes that can be specified on channel definition commands.

The DEFINE CHANNEL, ALTER CHANNEL, and DISPLAY CHANNEL commands have two specific CHLTYPE parameters for clusters: CLUSRCVR and CLUSSDR. To define a cluster-receiver channel you use the DEFINE CHANNEL command, specifying CHLTYPE(CLUSRCVR). Many attributes on a cluster-receiver channel definition are the same as the attributes on a receiver or sender-channel definition. To define a cluster-sender channel you use the DEFINE CHANNEL command, specifying CHLTYPE(CLUSSDR), and many of the same attributes as you use to define a sender-channel.

It is no longer necessary to specify the name of the full repository queue manager when you define a cluster-sender channel. If you know the naming convention used for channels in your cluster, you can make a CLUSSDR definition using the +QMNAME+ construction. The +QMNAME+ construction is not supported on z/OS. After connection, IBM MQ changes the name of the channel and substitutes the correct full repository queue manager name in place of +QMNAME+. The resulting channel name is truncated to 20 characters.

For more information on naming conventions, see Cluster naming conventions.

The technique works only if your convention for naming channels includes the name of the queue manager. For example, you define a full repository queue manager called QM1 in a cluster called CLUSTER1 with a cluster-receiver channel called CLUSTER1.QM1.ALPHA. Every other queue manager can define a cluster-sender channel to this queue manager using the channel name, CLUSTER1.+QMNAME+.ALPHA.

If you use the same naming convention for all your channels, be aware that only one +QMNAME+ definition can exist at one time.

The following attributes on the DEFINE CHANNEL and ALTER CHANNEL commands are specific to cluster channels:

CLUSTER
The CLUSTER attribute specifies the name of the cluster with which this channel is associated. Alternatively use the CLUSNL attribute.

CLUSNL The CLUSNL attribute specifies a namelist of cluster names.

NETPRTY
Cluster-receivers only.

The NETPRTY attribute specifies a network priority for the channel. NETPRTY helps the workload management routines. If there is more than one possible route to a destination, the workload management routine selects the one with the highest priority.

CLWLPRTY
The CLWLPRTY parameter applies a priority factor to channels to the same destination for workload management purposes. This parameter specifies the priority of the channel for the purposes of cluster workload distribution. The value must be in the range zero through 9, where zero is the lowest priority and 9 is the highest.

CLWLRANK
The CLWLRANK parameter applies a ranking factor to a channel for workload management purposes. This parameter specifies the rank of a channel for the purposes of cluster workload distribution. The value must be in the range zero through 9, where zero is the lowest rank and 9 is the highest.

CLWLWGHT
The CLWLWGHT parameter applies a weighting factor to a channel for workload management purposes. CLWLWGHT weights the channel so that the proportion of messages sent down that channel can be controlled. The cluster workload algorithm uses CLWLWGHT to bias the destination

choice so that more messages can be sent over a particular channel. By default all channel weight attributes are the same default value. The weight attribute allows you to allocate a channel on a powerful UNIX machine a larger weight than another channel on small desktop PC. The greater weight means that the cluster workload algorithm selects the UNIX machine more frequently than the PC as the destination for messages.

CONNAME
The CONNAME specified on a cluster-receiver channel definition is used throughout the cluster to identify the network address of the queue manager. Take care to select a value for the CONNAME parameter that resolves throughout your IBM MQ cluster. Do not use a generic name. Remember that the value specified on the cluster-receiver channel takes precedence over any value specified in a corresponding cluster-sender channel.

These attributes on the DEFINE CHANNEL command and ALTER CHANNEL command also apply to the DISPLAY CHANNEL command.

Note: Auto-defined cluster-sender channels take their attributes from the corresponding cluster-receiver channel definition on the receiving queue manager. Even if there is a manually defined cluster-sender channel, its attributes are automatically modified to ensure that they match the attributes on the corresponding cluster-receiver definition. Beware that you can, for example, define a CLUSRCVR without specifying a port number in the CONNAME parameter, while manually defining a CLUSSDR that does specify a port number. When the auto-defined CLUSSDR replaces the manually defined one, the port number (taken from the CLUSRCVR) becomes blank. The default port number would be used and the channel would fail.

Note: The DISPLAY CHANNEL command does not display auto-defined channels. However, you can use the DISPLAY CLUSQMGR command to examine the attributes of auto-defined cluster-sender channels.

Use the DISPLAY CHSTATUS command to display the status of a cluster-sender or cluster-receiver channel. This command gives the status of both manually defined channels and auto-defined channels.

The equivalent PCFs are MQCMD_CHANGE_CHANNEL, MQCMD_COPY_CHANNEL, MQCMD_CREATE_CHANNEL, and MQCMD_INQUIRE_CHANNEL.

Omitting the CONNAME value on a CLUSRCVR definition

In some circumstances you can omit the CONNAME value on a CLUSRCVR definition. You must not omit the CONNAME value on z/OS.

Multi On Multiplatforms, the TCP/IP connection name parameter of a cluster-receiver channel is optional. If you leave the connection name blank, IBM MQ generates a connection name for you, assuming the default port and using the current IP address of the system. You can override the default port number, but still use the current IP address of the system. For each connection name leave the IP name blank, and provide the port number in parentheses; for example:

```
(1415)
```

The generated **CONNAME** is always in the dotted decimal (IPv4) or hexadecimal (IPv6) form, rather than in the form of an alphanumeric DNS host name.

This facility is useful when you have machines using Dynamic Host Configuration Protocol (DHCP). If you do not supply a value for the CONNAME on a CLUSRCVR channel, you do not need to change the CLUSRCVR definition. DHCP allocates you a new IP address.

If you specify a blank for the CONNAME on the CLUSRCVR definition, IBM MQ generates a CONNAME from the IP address of the system. Only the generated CONNAME is stored in the repositories. Other queue managers in the cluster do not know that the CONNAME was originally blank.

If you issue the DISPLAY CLUSQMGR command you see the generated CONNAME. However, if you issue the DISPLAY CHANNEL command from the local queue manager, you see that the CONNAME is blank.

If the queue manager is stopped and restarted with a different IP address, because of DHCP, IBM MQ regenerates the CONNAME and updates the repositories accordingly.

Queue definition commands

Cluster attributes that can be specified on the queue definition commands.

The DEFINE QLOCAL, DEFINE QREMOTE, and DEFINE QALIAS commands

The cluster attributes on the DEFINE QLOCAL, DEFINE QREMOTE, and DEFINE QALIAS commands, and the three equivalent ALTER commands, are:

CLUSTER
Specifies the name of the cluster to which the queue belongs.

CLUSNL Specifies a namelist of cluster names.

DEFBIND
Specifies the binding to be used when an application specifies MQOO_BIND_AS_Q_DEF on the MQOPEN call. The options for this attribute are:

- Specify DEFBIND(OPEN) to bind the queue handle to a specific instance of the cluster queue when the queue is opened. DEFBIND(OPEN) is the default for this attribute.
- Specify DEFBIND(NOTFIXED) so that the queue handle is not bound to any instance of the cluster queue.
- Specify DEFBIND(GROUP) to allow an application to request that a group of messages are all allocated to the same destination instance.

When multiple queues with the same name are advertised in a Queue Manager Cluster, applications can choose whether to send all messages from this application to a single instance (MQOO_BIND_ON_OPEN), to allow the workload management algorithm to select the most suitable destination on a per message basis (MQOO_BIND_NOT_FIXED), or allow an application to request that a 'group' of messages be all allocated to the same destination instance (MQOO_BIND_ON_GROUP). The workload balancing is re-driven between groups of messages (without requiring an MQCLOSE and MQOPEN of the queue).

When you specify DEFBIND on a queue definition, the queue is defined with one of the attributes, MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXED, or MQBND_BIND_ON_GROUP. Either MQBND_BIND_ON_OPEN or MQBND_BIND_ON_GROUP must be specified when using groups with clusters.

We recommend that you set the DEFBIND attribute to the same value on all instances of the same cluster queue. Because MQOO_BIND_ON_GROUP is new in IBM WebSphere MQ Version 7.1, it must not be used if any of the applications opening this queue are connecting to IBM WebSphere MQ Version 7.0.1 or earlier queue managers.

CLWLRANK
Applies a ranking factor to a queue for workload management purposes. CLWLRANK parameter is not supported on model queues. The cluster workload algorithm selects a destination queue with the highest rank. By default CLWLRANK for all queues is set to zero.

If the final destination is a queue manager on a different cluster, you can set the rank of any intermediate gateway queue managers at the intersection of neighboring clusters. With the intermediate queue managers ranked, the cluster workload algorithm correctly selects a destination queue manager nearer the final destination.

The same logic applies to alias queues. The rank selection is made before the channel status is checked, and therefore even non-accessible queue managers are available for selection. This has the effect of allowing a message to be routed through a network, rather than having it select between two possible destinations (as the priority would). So, if a channel is not started to the

place where the rank has indicated, the message is not routed to the next highest rank, but waits until a channel is available to that destination (the message is held on the transmit queue).

CLWLPRTY
Applies a priority factor to a queue for workload management purposes. The cluster workload algorithm selects a destination queue with the highest priority. By default priority for all queues is set to zero.

If there are two possible destination queues, you can use this attribute to make one destination failover to the other destination. The priority selection is made after the channel status is checked. All messages are sent to the queue with the highest priority unless the status of the channel to that destination is not as favorable as the status of channels to other destinations. This means that only the most accessible destinations are available for selection. This has the effect of prioritizing between multiple destinations that are all available.

CLWLUSEQ
Specifies the behavior of an MQPUT operation for a queue. This parameter specifies the behavior of an MQPUT operation when the target queue has a local instance and at least one remote cluster instance (except where the MQPUT originates from a cluster channel). This parameter is only valid for local queues.

Possible values are: QMGR (the behavior is as specified by the CLWLUSEQ parameter of the queue manager definition), ANY (the queue manager treats the local queue as another instance of the cluster queue, for the purposes of workload distribution), LOCAL (the local queue is the only target of the MQPUT operation, providing the local queue is put enabled). The MQPUT behavior depends upon the cluster workload management algorithm.

The DISPLAY QUEUE and DISPLAY QCLUSTER commands

The attributes on the DEFINE QLOCAL, DEFINE QREMOTE, and DEFINE QALIAS commands also apply to the DISPLAY QUEUE command.

To display information about cluster queues, specify a queue type of QCLUSTER or the keyword CLUSINFO on the DISPLAY QUEUE command, or use the command DISPLAY QCLUSTER.

The DISPLAY QUEUE or DISPLAY QCLUSTER command returns the name of the queue manager that hosts the queue (or the names of all queue managers if there is more than one instance of the queue). It also returns the system name for each queue manager that hosts the queue, the queue type represented, and the date and time at which the definition became available to the local queue manager. This information is returned using the CLUSQMGR, QMID, CLUSQT, CLUSDATE, and CLUSTIME attributes.

The system name for the queue manager (QMID), is a unique, system-generated name for the queue manager.

You can define a cluster queue that is also a shared queue. For example. on z/OS you can define:
```
DEFINE QLOCAL(MYQUEUE) CLUSTER(MYCLUSTER) QSGDISP(SHARED) CFSTRUCT(STRUCTURE)
```

The equivalent PCFs are MQCMD_CHANGE_Q, MQCMD_COPY_Q, MQCMD_CREATE_Q, and MQCMD_INQUIRE_Q.

DISPLAY CLUSQMGR

Use the DISPLAY CLUSQMGR command to display cluster information about queue managers in a cluster.

If you issue this command from a queue manager with a full repository, the information returned applies to every queue manager in the cluster. Otherwise the information returned applies only to the queue managers in which it has an interest. That is, every queue manager to which it has tried to send a message and every queue manager that holds a full repository.

The information includes most channel attributes that apply to cluster-sender and cluster-receiver channels. In addition, the following attributes can be displayed:

CHANNEL
The cluster-receiver channel name for the queue manager.

CLUSDATE
The date at which the definition became available to the local queue manager.

CLUSTER
What clusters the queue manager is in.

CLUSTIME
The time at which the definition became available to the local queue manager.

DEFTYPE
How the queue manager was defined. DEFTYPE can be one of the following values:

CLUSSDR
A cluster sender-channel has been administratively defined on the local queue manager but not yet recognized by the target queue manager. To be in this state the local queue manager has defined a manual cluster-sender channel but the receiving queue manager has not accepted the cluster information. This may be due to the channel never having been established due to availability or to an error in the cluster-sender configuration, for example a mismatch in the CLUSTER property between the sender and receiver definitions. This is a transitory condition or error state and should be investigated.

CLUSSDRA
This value represents an automatically discovered cluster queue manager, no cluster-sender channel is defined locally. This is the DEFTYPE for cluster queue managers for which the local queue manager has no local configuration but has been informed of. For example

- If the local queue manager is a full repository queue manager it should be the DEFTYPE value for all partial repository queue managers in the cluster.
- If the local queue manager is a partial repository, this could be the host of a cluster queue that is being used from this local queue manager or from a second full repository queue manager that this queue manager has been told to work with.

If the DEFTYPE value is CLUSSDRA and the local and remote queue managers are both full repositories for the named cluster, the configuration is not correct as a locally defined cluster-sender channel must be defined to convert this to a DEFTYPE of CLUSSDRB.

CLUSSDRB
A cluster sender-channel has been administratively defined on the local queue manager and accepted as a valid cluster channel by the target queue manager. This is the expected DEFTYPE of a partial repository queue manager's manually configured full repository queue manager. It should also be the DEFTYPE of any CLUSQMGR from one full repository to another full repository in the cluster. Manual cluster-sender channels should not be configured to partial repositories or from a partial repository queue manager to more than one full repository. If a DEFTYPE of CLUSSDRB is seen in either of these situations it should be investigated and corrected.

CLUSRCVR
Administratively defined as a cluster-receiver channel on the local queue manager. This represents the local queue manager in the cluster.

Note: To identify which CLUSQMGRs are full repository queue managers for the cluster, see the QMTYPE property.

For more information on defining cluster channels, see Cluster channels.

QMTYPE Whether it holds a full repository or only a partial repository.

STATUS The status of the cluster-sender channel for this queue manager.

SUSPEND
Whether the queue manager is suspended.

VERSION
The version of the IBM MQ installation that the cluster queue manager is associated with.

The version has the format VVRRMMFF:

- VV: Version
- RR: Release
- MM: Maintenance level
- FF: Fix level

XMITQ The cluster transmission queue used by the queue manager.

See also the DISPLAY QCLUSTER command. This is briefly described in DISPLAY QUEUE and in the DISPLAY QUEUE and DISPLAY QCLUSTER commands section of "Queue definition commands" on page 141. For examples of using DISPLAY QCLUSTER, search the information set for "DISPLAY QCLUSTER" and "DIS QCLUSTER".

Related information:

MQSC command **DISPLAY CLUSQMGR**
Use the MQSC command **DISPLAY CLUSQMGR** to display information about cluster channels for queue managers in a cluster.

SUSPEND QMGR, RESUME QMGR and clusters

Use the SUSPEND QMGR and RESUME QMGR command to temporarily reduce the inbound cluster activity to this queue manager, for example, before you perform maintenance on this queue manager, and then reinstate it.

While a queue manager is suspended from a cluster, it does not receive messages on cluster queues that it hosts if there is an available queue of the same name on an alternative queue manager in the cluster. However, messages that are explicitly targeted at this queue manager, or where the target queue is only available on this queue manager, are still directed to this queue manager.

Receiving further inbound messages while the queue manager is suspended can be prevented by stopping the cluster receiver channels for this cluster. To stop the cluster receiver channels for a cluster, use the FORCE mode of the SUSPEND QMGR command.

Related information:

SUSPEND QMGR
Use the MQSC command SUSPEND QMGR to advise other queue managers in a cluster to avoid sending messages to the local queue manager if possible.

RESUME QMGR
Use the MQSC command RESUME QMGR to inform other queue managers in a cluster that the local queue manager is available again for processing and can be sent messages. It reverses the action of the SUSPEND QMGR command.

Maintaining a queue manager

REFRESH CLUSTER

Issue the REFRESH CLUSTER command from a queue manager to discard all locally held information about a cluster. You are unlikely to need to use this command, except in exceptional circumstances.

There are three forms of this command:

REFRESH CLUSTER(clustername) REPOS(NO)
The default. The queue manager retains knowledge of all locally defined cluster queue manager and cluster queues and all cluster queue managers that are full repositories. In addition, if the queue manager is a full repository for the cluster it also retains knowledge of the other cluster queue managers in the cluster. Everything else is removed from the local copy of the repository and rebuilt from the other full repositories in the cluster. Cluster channels are not stopped if REPOS(NO) is used. A full repository uses its CLUSSDR channels to inform the rest of the cluster that it has completed its refresh.

REFRESH CLUSTER(clustername) REPOS(YES)
In addition to the default behavior, objects representing full repository cluster queue managers are also refreshed. It is not valid to use this option if the queue manager is a full repository, if used the command will fail with an error AMQ9406/CSQX406E logged. If it is a full repository, you must first alter it so that it is not a full repository for the cluster in question. The full repository location is recovered from the manually defined CLUSSDR definitions. After refreshing with REPOS(YES) has been issued the queue manager can be altered so that it is once again a full repository, if required.

REFRESH CLUSTER(*)
Refreshes the queue manager in all the clusters it is a member of. If used with REPOS(YES) REFRESH CLUSTER(*) has the additional effect of forcing the queue manager to restart its search for full repositories from the information in the local CLUSSDR definitions. The search takes place even if the CLUSSDR channel connects the queue manager to several clusters.

Note: For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.

Related information:
Clustering: Using REFRESH CLUSTER best practices

RESET CLUSTER

Use the **RESET CLUSTER** command to forcibly remove a queue manager from a cluster in exceptional circumstances.

You are unlikely to need to use this command, except in exceptional circumstances.

You can issue the **RESET CLUSTER** command only from full repository queue managers. The command takes two forms, depending on whether you reference the queue manager by name or identifier.

1. `RESET CLUSTER( clustername ) QMNAME( qmname ) ACTION(FORCEREMOVE) QUEUES(NO)`
2. `RESET CLUSTER( clustername ) QMID( qmid ) ACTION(FORCEREMOVE) QUEUES(NO)`

You cannot specify both `QMNAME` and `QMID`. If you use `QMNAME`, and there is more than one queue manager in the cluster with that name, the command is not run. Use `QMID` instead of `QMNAME` to ensure the **RESET CLUSTER** command is run.

Specifying `QUEUES(NO)` on a **RESET CLUSTER** command is the default. Specifying `QUEUES(YES)` removes references to cluster queues owned by the queue manager from the cluster. The references are removed in addition to removing the queue manager from the cluster itself.

The references are removed even if the cluster queue manager is not visible in the cluster; perhaps because it was previously forcibly removed, without the `QUEUES` option.

You might use the **RESET CLUSTER** command if, for example, a queue manager has been deleted but still has cluster-receiver channels defined to the cluster. Instead of waiting for IBM MQ to remove these definitions (which it does automatically) you can issue the **RESET CLUSTER** command to tidy up sooner. All other queue managers in the cluster are then informed that the queue manager is no longer available.

If a queue manager is temporarily damaged, you might want to tell the other queue managers in the cluster before they try to send it messages. **RESET CLUSTER** removes the damaged queue manager. Later, when the damaged queue manager is working again, use the **REFRESH CLUSTER** command to reverse the effect of **RESET CLUSTER** and return the queue manager to the cluster.If the queue manager is in a publish/subscribe cluster, you then need to reinstate any required proxy subscriptions. See REFRESH CLUSTER considerations for publish/subscribe clusters.

Note: For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.

Using the **RESET CLUSTER** command is the only way to delete auto-defined cluster-sender channels. You are unlikely to need this command in normal circumstances. The IBM Support Center might advise you to issue the command to tidy up the cluster information held by cluster queue managers. Do not use this command as a short cut to removing a queue manager from a cluster. The correct way to remove a queue manager from a cluster is described in Removing a queue manager from a cluster.

Because repositories retain information for only 90 days, after that time a queue manager that was forcibly removed can reconnect to a cluster. It reconnects automatically, unless it has been deleted. If you want to prevent a queue manager from rejoining a cluster, you need to take appropriate security measures.

All cluster commands, except **DISPLAY CLUSQMGR**, work asynchronously. Commands that change object attributes involving clustering update the object and send a request to the repository processor. Commands for working with clusters are checked for syntax, and a request is sent to the repository processor.

The requests sent to the repository processor are processed asynchronously, along with cluster requests received from other members of the cluster. Processing might take a considerable time if they have to be propagated around the whole cluster to determine if they are successful or not.

Workload balancing in clusters

If a cluster contains more than one instance of the same queue, IBM MQ selects a queue manager to route a message to. It uses the cluster workload management algorithm, and a number of cluster workload-specific attributes, to determine the best queue manager to use.

Suitable destinations are chosen, by the cluster workload management algorithm, based on the availability of the queue manager and queue, and on a number of cluster workload-specific attributes associated with queue managers, queues, and channels. These attributes are described in the subtopics.

Note: Specify the cluster workload channel attributes on the cluster-receiver channels at the target queue managers. Any balancing you specify on the matching cluster-sender channels is likely to be ignored. See Cluster channels.

After you configure the cluster workload-specific attributes, if the configuration does not behave as you expected, explore the details of how the algorithm chooses a queue manager. See “The cluster workload management algorithm” on page 151. If the results of this algorithm do not meet your needs, you can write a cluster workload user exit program, and use this exit to route messages to the queue of your choice in the cluster. See Writing and compiling cluster workload exits.

CLWLPRTY queue attribute:

The CLWLPRTY queue attribute specifies the priority of local, remote, or alias queues for cluster workload distribution. The value must be in the range 0-9, where 0 is the lowest priority and 9 is the highest.

Use the CLWLPRTY queue attribute to set a preference for destination queues. IBM MQ selects the destinations with the highest priority before selecting destinations with the lowest cluster destination priority. If there are multiple destinations with the same priority, it selects the least recently used destination.

If there are two possible destinations, you can use this attribute to allow failover. The highest priority queue manager receives requests, lower priority queue managers act as reserves. If the highest priority queue manager fails, then the next highest priority queue manager that is available, takes over.

IBM MQ obtains the priority of queue managers after checking channel status. Only available queue managers are candidates for selection.

Note:

The availability of a remote queue manager is based on the status of the channel to that queue manager. When channels start, their state changes several times, with some of the states being less preferential to the cluster workload management algorithm. In practice this means that lower priority (backup) destinations can be chosen while the channels to higher priority (primary) destinations are starting.

If you need to ensure that no messages go to a backup destination, do not use CLWLPRTY. Consider using separate queues, or CLWLRANK with a manual switch over from the primary to back up.

CLWLRANK queue attribute:

The CLWLRANK queue attribute specifies the rank of a local, remote, or alias queue for cluster workload distribution. The value must be in the range 0-9, where 0 is the lowest rank and 9 is the highest.

Use the CLWLRANK queue attribute if you want control over the final destination for messages sent to a queue manager in another cluster. When you set CLWLRANK, messages take a specified route through the interconnected clusters towards a higher ranked destination.

For example, you might have defined two identically configured gateway queue managers to improve the availability of a gateway. Suppose you have defined cluster alias queues at the gateways for a local queue defined in the cluster. If the local queue becomes unavailable, you intend the message to be held at one of the gateways pending the queue becoming available again. To hold the queue at a gateway, you must define the local queue with a higher rank than the cluster alias queues at the gateway.

If you define the local queue with the same rank as the queue aliases and the local queue is unavailable, the message travels between the gateways. On finding the local queue unavailable the first gateway queue manager routes the message to the other gateway. The other gateway tries to deliver the message to the target local queue again. If the local queue is still unavailable, it routes the message back to the first gateway. The message keeps being moved back and forth between the gateways until the target local queue became available again. By giving the local queue a higher rank, even if the queue is unavailable, the message is not rerouted to a destination of lower rank.

IBM MQ obtains the rank of queues before checking channel status. Obtaining the rank before checking channel status means that even non-accessible queues are available for selection. It allows messages to be routed through the network even if the final destination is unavailable.

If you used the priority attribute IBM MQ selects between available destinations. If a channel is not available to the destination with the highest rank, the message is held on the transmission queue. It is released when the channel becomes available. The message does not get sent to the next available destination in the rank order.

CLWLUSEQ queue attribute:

The CLWLUSEQ queue attribute specifies whether a local instance of a queue is given preference as a destination over other instances in a cluster.

The CLWLUSEQ queue attribute is valid only for local queues. It only applies if the message is put by an application, or a channel that is not a cluster channel.

LOCAL The local queue is the only target of MQPUT, providing the local queue is put enabled. MQPUT behavior depends upon the cluster workload management.

QMGR The behavior is as specified by the CLWLUSEQ queue manager attribute.

ANY MQPUT treats the local queue the same as any other instance of the queue in the cluster for workload distribution.

CLWLUSEQ queue manager attribute:

The CLWLUSEQ queue manager attribute specifies whether a local instance of a queue is given preference as a destination over other instances of the queue in a cluster. The attribute applies if the CLWLUSEQ queue attribute is set to QMGR.

The CLWLUSEQ queue attribute is valid only for local queues. It only applies if the message is put by an application, or a channel that is not a cluster channel.

LOCAL The local queue is the only target of MQPUT. LOCAL is the default.

ANY MQPUT treats the local queue the same as any other instance of the queue in the cluster for workload distribution.

CLWLMRUC queue manager attribute:

The CLWLMRUC queue manager attribute sets the number of most recently chosen channels. The cluster workload management algorithm uses CLWLMRUC to restrict the number of active outbound cluster channels. The value must be in the range 1 - 999 999 999.

The initial default value is 999 999 999.

CLWLPRTY channel attribute:

The CLWLPRTY channel attribute specifies the priority order for channels for cluster workload distribution. The value must be in the range 0-9, where 0 is the lowest priority and 9 is the highest.

Use the CLWLPRTY channel attribute to set a priority order for the available cluster destinations. IBM MQ selects the destinations with the highest priority before selecting destinations with the lowest cluster destination priority. If there are multiple destinations with the same priority, it selects the least recently used destination.

If there are two possible destinations, you can use this attribute to allow failover. Messages go to the queue manager with the highest priority channel. If it becomes unavailable then messages go to the next highest priority queue manager. Lower priority queue managers act as reserves.

IBM MQ checks channel status before prioritizing the channels. Only available queue managers are candidates for selection.

Notes:

- Specify this attribute on the cluster-receiver channel at the target queue manager. Any balancing you specify on the matching cluster-sender channel is likely to be ignored. See Cluster channels.
- The availability of a remote queue manager is based on the status of the channel to that queue manager. When channels start, their state changes several times, with some of the states being less preferential to the cluster workload management algorithm. In practice this means that lower priority (backup) destinations can be chosen while the channels to higher priority (primary) destinations are starting.
- If you need to ensure that no messages go to a backup destination, do not use CLWLPRTY. Consider using separate queues, or CLWLRANK with a manual switch over from the primary to back up.

CLWLRANK channel attribute:

The **CLWLRANK** channel attribute specifies the rank of channels for cluster workload distribution. The value must be in the range 0-9, where 0 is the lowest rank and 9 is the highest.

Use the **CLWLRANK** channel attribute if you want control over the final destination for messages sent to a queue manager in another cluster. Control the choice of final destination by setting the rank of the channels connecting a queue manager to the gateway queue managers at the intersection of the clusters.

When you set **CLWLRANK**, messages take a specified route through the interconnected clusters towards a higher ranked destination. For example, messages arrive at a gateway queue manager that can send them to either of two queue managers using channels ranked 1 and 2. They are automatically sent to the queue manager connected by a channel with the highest rank, in this case the channel to the queue manager ranked 2.

IBM MQ gets the rank of channels before checking channel status. Getting the rank before checking channel status means that even non-accessible channels are available for selection. It allows messages to be routed through the network even if the final destination is unavailable.

Notes:

- Specify this attribute on the cluster-receiver channel at the target queue manager. Any balancing you specify on the matching cluster-sender channel is likely to be ignored. See Cluster channels.
- If you also used the priority attribute **CLWLPRTY**, IBM MQ selects between available destinations. If a channel is not available to the destination with the highest rank, the message is held on the transmission queue. It is released when the channel becomes available. The message does not get sent to the next available destination in the rank order.

CLWLWGHT channel attribute:

The CLWLWGHT channel attribute specifies the weight applied to CLUSSDR and CLUSRCVR channels for cluster workload distribution. The value must be in the range 1-99, where 1 is the lowest weight and 99 is the highest.

Use CLWLWGHT to send servers with more processing power more messages. The higher the channel weight, the more messages are sent over that channel.

Notes:

- Specify this attribute on the cluster-receiver channel at the target queue manager. Any balancing you specify on the matching cluster-sender channel is likely to be ignored. See Cluster channels.
- When CLWLWGHT is modified from the default of 50 on any channel, workload balancing becomes dependent on the total number of times each channel was chosen for a message sent to any clustered queue. For more information, see"The cluster workload management algorithm" on page 151.

NETPRTY channel attribute:

The NETPRTY channel attribute specifies the priority for a CLUSRCVR channel. The value must be in the range 0-9, where 0 is the lowest priority and 9 is the highest.

Use the NETPRTY attribute to make one network the primary network, and another network the backup network. Given a set of equally ranked channels, clustering chooses the path with the highest priority when multiple paths are available.

A typical example of using the NETPRTY channel attribute is to differentiate between networks that have different costs or speeds and connect the same destinations.

Note: Specify this attribute on the cluster-receiver channel at the target queue manager. Any balancing you specify on the matching cluster-sender channel is likely to be ignored. See Cluster channels.

The cluster workload management algorithm:

The workload management algorithm uses workload balancing attributes and many rules to select the final destination for messages being put onto cluster queues.

The workload management algorithm is exercised every time a choice of destination is required:

- It is used at the point a cluster queue is opened, by using the MQOO_BIND_ON_OPEN option.
- It is used each time a message is put to a cluster queue when it is opened with MQOO_BIND_NOT_FIXED.
- It is used each time a new message group is started when MQOO_BIND_ON_GROUP is used to open a cluster queue.
- For topic host routing, it is used each time a message is published to a clustered topic. If the local queue manager is not a host for this topic, the algorithm is used to choose a host queue manager to route the message through.

The following section describes the workload management algorithm used when determining the final destination for messages being put onto cluster queues. These rules are influenced by the settings applied to the following attributes for queues, queue managers, and channels:

Table 32. Attributes for cluster workload management

Queues	Queue managers	Channels
• CLWLPRTY (This attribute applies only when choosing a clustered queue, not when choosing a topic.) [1] • CLWLRANK [1] • CLWLUSEQ [1] • PUT / PUB	• CLWLUSEQ [1] • CLWLMRUC	• CLWLPRTY • CLWLRANK • CLWLWGHT • NETPRTY

Initially, the queue manager builds a list of possible destinations from two procedures:

- Matching the target ObjectName and ObjectQmgrName with queue manager alias definitions that are shared in the same clusters as the queue manager.
- Finding unique routes (that is, channels) to a queue manager that hosts a queue with the name ObjectName and is in one of the clusters that the queue manager is a member of.

The algorithm steps through the following rules to eliminate destinations from the list of possible destinations.

1. If a queue or topic name is specified:

a. Queues that are not put enabled are eliminated as possible destinations.
b. Administered topic definitions that are not pub enabled are eliminated as possible destinations.
c. Remote instances of queues or topics that do not share a cluster with the local queue manager are eliminated.
d. Remote CLUSRCVR channels that are not in the same cluster as the queue or topic are eliminated.

2. If a queue manager name is specified:
   a. Queue manager aliases that are not put enabled are eliminated.
   b. Remote CLUSRCVR channels that are not in the same cluster as the local queue manager are eliminated.
3. When choosing a queue, if the resulting set of queues contains the local instance of the queue, the local instance is typically used. The local instance of the queue is used if one of these three conditions are true:
   - Either the use-queue attribute of the queue, CLWLUSEQ is set to LOCAL,
   - Or both the following statements are true:
     a. The **use-queue** attribute of the queue, CLWLUSEQ is set to QMGR.
     b. The **use-queue** attribute of the queue manager, CLWLUSEQ is set to LOCAL.
   - Or the message is received over a cluster channel rather than by being put by a local application.
   - For locally defined queues that are defined with CLWLUSEQ(ANY), or which inherit that same setting from the queue manager, the following points are true, within the wider set of conditions that apply:
     a. The local queue is chosen, based on the *status* of the locally-defined CLUSRCVR channels in the same cluster as the queue. This status is compared to the status of the CLUSSDR channels that would take the message to remotely defined queues of the same name.

        For example, there is one CLUSRCVR in the same cluster as the queue. That CLUSRCVR has STOPPING status, whereas the other queues of the same name in the cluster have RUNNING or INACTIVE status.

        In this case the remote channels will be chosen, and the local queue is not used.
     b. The local queue is chosen based on the *number* of CLUSRCVR channels, in any comparison with CLUSSDR channels of the same status, that would take the message to remotely defined queues of the same name.

        For example, there are four CLUSRCVR channels in the same cluster as the queue, and one CLUSSDR channel. All the channels have the same status of either INACTIVE or RUNNING.

        Therefore there are five channels to choose from, and two instances of the queue. Four fifths (80 percent) of the messages go to the local queue.
4. All channels to queue managers or queue manager aliases that have a CLWLRANK less than the maximum rank of all remaining channels or queue manager aliases are eliminated.
5. All queues (not queue manager aliases) with a CLWLRANK less than the maximum rank of all remaining queues are eliminated.
6. If only remote instances of a queue or topic remain, resumed queue managers are chosen in preference to suspended ones.
7. If more than one remote instance of a queue or topic remains, all channels that are inactive or running are included. The state constants are listed:
   - MQCHS_INACTIVE
   - MQCHS_RUNNING
8. If no remote instance of a queue or topic remains, all channels that are in binding, initializing, starting, or stopping state are included. The state constants are listed:
   - MQCHS_BINDING
   - MQCHS_INITIALIZING
   - MQCHS_STARTING
   - MQCHS_STOPPING

9. If no remote instance of a queue or topic remains, all channels that are being tried again are included. The state constant is listed:
   - MQCHS_RETRYING
10. If no remote instance of a queue or topic remains, all channels in requesting, paused, or stopped state are included. The state constants are listed:
    - MQCHS_REQUESTING
    - MQCHS_PAUSED
    - MQCHS_STOPPED
11. If more than one remote instance of a queue or topic on any queue manager remains, channels with the highest NETPRTY value for each queue manager are chosen.
12. If a queue manager is being chosen:
    - All remaining channels and queue manager aliases other than channels and aliases with the highest priority, CLWLPRTY, are eliminated. If any queue manager aliases remain, channels to the queue manager are kept.
13. If a queue is being chosen:
    - All queues other than queues with the highest priority, CLWLPRTY, are eliminated, and channels are kept.
14. The remaining channels are then reduced to no more than the maximum allowed number of most recently-used channels, CLWLMRUC, by eliminating the channels with the lowest values of MQWDR.DestSeqNumber.

Note: Internal cluster control messages are sent using the same cluster workload algorithm where appropriate.

After the list of valid destinations has been calculated, messages are workload balanced across them, using the following logic:

- When more than one remote instance of a destination remains and all channels to those destination have CLWLWGHT set to the default setting of 50, then the least recently used channel is chosen. This approximately equates to a round-robin style of workload balancing when multiple remote instances exist.
- When more than one remote instance of a destination remains and one or more of the channels to those queues has CLWLWGHT set to a non-default setting (even if they all have a matching non-default value), then routing becomes dependent on the relative weightings of each channel and the total number of times each channel has previously been chosen when sending messages.
- When observing the distribution of messages for a single clustered queue with multiple instances, this can appear to lead to an unbalanced distribution across a subset of queue instances. This is because it is the historic use of each cluster sender channel from this queue manager that is being balanced, not just the message traffic for that queue. If this behavior is not desirable, do one of the following steps.
  - Either set CLWLWGHT to 50 on all cluster receiver channels if even distribution is required,
  - Or, if certain queue instances need to be weighted differently from others, define those queues in a dedicated cluster, with defined dedicated cluster receiver channels. This action isolates the workload balancing of these queues from others in the cluster.
- The historic data that is used to balance the channels is reset if any cluster workload attributes of available cluster receiver channels are altered or the status of a cluster receiver channel becomes available. Modification to the workload attributes of manually defined cluster sender channels does not reset the historic data.
- When you are considering cluster workload exit logic, the chosen channel is the one with the lowest MQWDR.DestSeqFactor. Each time a channel is chosen, this value is increased by approximately 1000/CLWLWGHT. If there is more than one channel with the lowest value, one of the channels with the lowest MQWDR.DestSeqNumber value is chosen.

The distribution of user messages is not always exact because administration and maintenance of the cluster causes messages to flow across channels. The result is an uneven distribution of user messages that can take some time to stabilize. Because of the admixture of administration and user messages, place no reliance on the exact distribution of messages during workload balancing.

Asynchronous behavior of CLUSTER commands on z/OS

The command issuer of a cluster command on z/OS receives confirmation a command has been sent, but not that it has completed successfully.

For both `REFRESH CLUSTER` and `RESET CLUSTER`, message CSQM130I is sent to the command issuer indicating that a request has been sent. This message is followed by message CSQ9022I to indicate that the command has completed successfully, in that a request has been sent. It does not indicate that the cluster request has been completed successfully.

Any errors are reported to the z/OS console on the system where the channel initiator is running, they are not sent to the command issuer.

The asynchronous behavior is in contrast to `CHANNEL` commands. A message indicating that a channel command has been accepted is issued immediately. At some later time, when the command has been completed, a message indicating either normal or abnormal completion is sent to the command issuer.

Related information:

Checking that async commands for distributed networks have finished

Channel programs

This section looks at the different types of channel programs (MCAs) available for use at the channels.

The names of the MCAs are shown in the following tables.

Table 33. Channel programs for Windows, UNIX and Linux systems

Program name	Direction of connection	Communication
amqrmppa		Any
runmqlsr	Inbound	Any
amqcrs6a	Inbound	LU 6.2
amqcrsta	Inbound	TCP
runmqchl	Outbound	Any
runmqchi	Outbound	Any

runmqlsr (Run IBM MQ listener), runmqchl (Run IBM MQ channel), and runmqchi (Run IBM MQ channel initiator) are control commands that you can enter at the command line.

amqcrsta is invoked for TCP channels on UNIX and Linux systems using inetd, where no listener is started.

amqcrs6a is invoked as a transaction program when using LU6.2

Environment variables

A list of all the server and client environment variables.

Examples of use

- UNIX Linux On UNIX and Linux systems use: `export [environment variable]=value`.
- Windows On Windows Systems, use: `Set [environment variable]=value`.
- IBM i On IBM i systems use: `ADDENVVAR ENVVAR(environment variable) VALUE(xx)`

AMQ_MQS_INI_LOCATION

On UNIX and Linux systems, you can alter the location that is used for the `mqs.ini` file by setting the location of the `mqs.ini` file in this variable. This variable must be set at the system level.

AMQ_REVERSE_COMMIT_ORDER

This variable configures a queue manager so that in an XA transaction the IBM MQ queue manager change is committed after the corresponding database update is completed.

Do not set AMQ_REVERSE_COMMIT_ORDER without reading and understanding the scenario that is described in the following topic: Isolation level.

V 9.0.0.1 V 9.0.2

AMQ_SSL_ALLOW_DEFAULT_CERT

FromIBM MQ Version 9.0.0, Fix Pack 1 and IBM MQ Version 9.0.2, when the `AMQ_SSL_ALLOW_DEFAULT_CERT` environment variable is not set, an application can connect to a queue manager with a personal certificate in the client keystore only when the certificate includes the label name of `ibmwebspheremq`*`userid`*. When the `AMQ_SSL_ALLOW_DEFAULT_CERT` environment variable is set, the certificate does not require the label name of `ibmwebspheremq`*`userid`*. That is, the certificate that is used to connect to a queue manager can be a default certificate, provided that a default certificate is present in the queue repository, and the key repository does not contain a personal certificate with the prefix `ibmwebspheremq`*`userid`*. For more information, see the technote Specifying the userid in the SSL certificate label for an MQ client.

A value of 1 enables the use of a default certificate.

From IBM MQ Version 8.0, instead of using the AMQ_SSL_ALLOW_DEFAULT_CERT environment variable, an application can use the **`CertificateLabel`** setting of the SSL stanza in the `mqclient.ini` file. For more information, see Digital certificate labels, understanding the requirements and SSL stanza of the client configuration file.

V 9.0.0.2 V 9.0.4

AMQ_SSL_LDAP_SERVER_VERSION

From IBM MQ Version 9.0.0, Fix Pack 2 and IBM MQ Version 9.0.4, this variable can be used to ensure that either LDAP v2 or LDAP v3 is used by IBM MQ cryptographic components in cases where CRL servers require that a specific version of the LDAP protocol be used.

Set the variable to appropriate value in the environment that is used to start the queue manager or channel. To request that LDAP v2 is used, set `AMQ_SSL_LDAP_SERVER_VERSION=2`. To request that LDAP v3 is used, set `AMQ_SSL_LDAP_SERVER_VERSION=3`.

This variable does not affect LDAP connections established by the IBM MQ queue manager for user authentication or user authorization.

GMQ_MQ_LIB

When both the IBM MQ MQI client and IBM MQ server are installed on your system, MQAX applications run against the server by default. To run MQAX against the client, the client bindings library must be specified in the GMQ_MQ_LIB environment variable, for example, set `GMQ_MQ_LIB=mqic.dll`. For a client only installation, it is not necessary to set the GMQ_MQ_LIB

environment variable. When this variable is not set, IBM MQ attempts to load `amqzst.dll`. If this DLL is not present (as is the case in a client only installation), IBM MQ attempts to load `mqic.dll`.

HOME
This variable contains the name of the directory that is searched for the `mqclient.ini` file. This file contains configuration information that is used by IBM MQ MQI clients on the following platforms:

- 

IBM i
- 

UNIX
- 

Linux

HOMEDRIVE and HOMEPATH
To be used both of these variables must be set. They are used to contain the name of the directory that is searched for the `mqclient.ini` file. This file contains configuration information that is used by IBM MQ MQI clients on Windows systems.

LDAP_BASEDN
The required environment variable for running an LDAP sample program. It specifies the base Distinguished Name for the directory search.

LDAP_HOST
An optional variable for running an LDAP sample program. It specifies the name of the host where the LDAP server is running; it defaults to the local host if it is not specified.

LDAP_VERSION
An optional variable for running an LDAP sample program. It specifies the version of the LDAP protocol to be used, and can be either 2 or 3. Most LDAP servers now support version 3 of the protocol; they all support the older version 2. This sample works equally well with either version of the protocol, and if it is not specified it defaults to version 2.

MQAPI_TRACE_LOGFILE
The sample API exit program generates an MQI trace to a user-specified file with a prefix that is defined in the MQAPI_TRACE_LOGFILE environment variable.

MQCCSID
Specifies the coded character set number to be used and overrides the native CCSID of the application.

MQCERTLABL
Defines the certificate label.

MQCERTVPOL
Determines the type of certificate validation used:

ANY Use any certificate validation policy that is supported by the underlying secure sockets library. This setting is the default setting.

RFC5280
Use only certificate validation that complies with the RFC 5280 standard.

MQCHLLIB
Specifies the directory path to the file that contains the client channel definition table (CCDT). The file is created on the server, but can be copied across to the IBM MQ MQI client workstation.

MQCHLTAB
MQCHLTAB specifies the name of the file that contains the client channel definition table (ccdt). The default file name is AMQCLCHL.TAB.

MQC_IPC_HOST
When sharing IBM MQ files and the generated value of `myHostName` creates a problem set `myHostName` using the environment variable MQC_IPC_HOST.

MQCLNTCF
Use this environment variable to modify the `mqclient.ini` file path.

MQ_CONNECT_TYPE
On the following platforms, use this environment variable in combination with the type of binding specified in the Options field of the MQCNO structure that is used on an MQCONNX call. See MQCONNX environment variable.

- Windows Windows
- UNIX Linux UNIX and Linux
- IBM i IBM i

V 9.0.0.1

MQ_CROSS_QUEUE_ORDER_ALL
When you set the MQ_CROSS_QUEUE_ORDER_ALL environment variable to a non-zero value, the message put order is maintained in a unit of work. This means that, if messages in a Unit of Work (UoW) are put onto multiple queues (for example, Q1, then Q2), when an MQCMIT is issued, the messages are delivered and made available in the same queue order in which they were PUT.

In a multi-queue manager environment, MQ_CROSS_QUEUE_ORDER_ALL must exist and have a non-empty value on both the sending and receiving side before each queue manager is started.

MQ_FILE_PATH
During the installation of the runtime package on the Windows platform, a new environment variable that is called MQ_FILE_PATH is configured. This environment variable contains the same data as the following key in the Windows Registry:

`HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere MQ\Installation\`*`InstallationName`*`\FilePath`

MQIPADDRV
MQIPADDRV specifies which IP protocol to use for a channel connection. It has the possible string values of "MQIPADDR_IPv4" or "MQIPADDR_IPv6". These values have the same meanings as IPv4 and IPv6 in ALTER QMGR IPADDRV. If it is not set, "MQIPADDR_IPv4" is assumed.

MQ_JAVA_DATA_PATH
Specifies the directory for log and trace output.

MQ_JAVA_INSTALL_PATH
Specifies the directory where IBM MQ classes for Java are installed, as shown in IBM MQ classes for Java installation directories.

MQ_JAVA_LIB_PATH
Specifies the directory where the IBM MQ classes for Java libraries are stored. Some scripts that are supplied with IBM MQ classes for Java, such as IVTRun, use this environment variable.

MQNAME
MQNAME specifies the local NetBIOS name that the IBM MQ processes can use.

MQNOREMPOOL
When you set this variable, it switches off channel pooling and causes channels to run as threads of the listener.

MQPSE_TRACE_LOGFILE
Use when you Publish the Exit Sample Program. In the application process to be traced, this environment variable describes where the trace files must be written to. See The Publish Exit sample program.

MQSERVER
MQSERVER environment variable is used to define a minimal channel. You cannot use MQSERVER to define a TLS channel or a channel with channel exits. MQSERVER specifies the location of the IBM MQ server and the communication method to be used.

MQ_SET_NODELAYACK
When you set this variable, it switches off TCP delayed acknowledgment

AIX When you set this variable on AIX, the setting switches off TCP delayed acknowledgment by calling the operating system's setsockopt call with the TCP_NODELAYACK option. Only AIX supports this function, so the MQ_SET_NODELAYACK environment variable only has an effect on AIX.

MQSNOAUT
MQSNOAUT disables the object authority manager (OAM) and prevents any security checking. The MQSNOAUT variable only takes effect when a queue manager is created.

MQSPREFIX
As an alternative to changing the default prefix, you can use the environment variable MQSPREFIX to override the `DefaultPrefix` for the **crtmqm** command.

MQSSLCRYP
MQSSLCRYP holds a parameter string that you can use to configure the cryptographic hardware present on the system. The permitted values are the same as for the SSLCRYP parameter of the ALTER QMGR command.

MQSSLFIPS
MQSSLFIPS specifies whether only FIPS-certified algorithms are to be used if cryptography is carried out in IBM MQ. The values are the same as for the SSLFIPS parameter of the ALTER QMGR command.

MQSSLKEYR
MQSSLKEYR specifies the location of the key repository that holds the digital certificate belonging to the user, in stem format. Stem format means that it includes the full path and the file name without an extension. For full details, see the SSLKEYR parameter of the ALTER QMGR command.

MQSSLPROXY
MQSSLPROXY specifies the host name and port number of the HTTP proxy server to be used by GSKit for OCSP checks.

MQSSLRESET
MQSSLRESET represents the number of unencrypted bytes sent and received on a TLS channel before the TLS secret key is renegotiated.

MQS_TRACE_OPTIONS
Use the environment variable MQS_TRACE_OPTIONS to activate the high detail and parameter tracing functions individually.

MQTCPTIMEOUT
This variable specifies how long IBM MQ waits for a TCP connect call.

MQSUITEB
This variable specifies whether Suite B compliant cryptography is to be used. In the instance that Suite B cryptography is used you can specify the strength of the cryptography by setting MQSUITEB to one of the following:

- NONE
- 128_BIT, 192_BIT
- 128_BIT
- 192_BIT

ODQ_MSG
If you use a dead-letter queue handler that is different from RUNMQDLQ the source of the sample is available for you to use as your base. The sample is like the dead-letter handler provided within the product but trace and error reporting are different. Use the ODQ_MSG environment variable to set the name of the file containing error and information messages. The file provided is called amqsdlq.msg.

ODQ_TRACE
If you use a dead-letter queue handler that is different from RUNMQDLQ the source of the sample is available for you to use as your base. The sample is like the dead-letter handler provided within the product but trace and error reporting are different. Set the ODQ_TRACE environment variable to YES or yes to enable tracing.

OMQ_PATH
This environment variable is where you can find the First Failure Symptom report if your IBM MQ automation classes for ActiveX script fails.

OMQ_TRACE
MQAX includes a trace facility to help the service organization identify what is happening when you have a problem. It shows the paths taken when you run your MQAX script. Unless you have a problem, run with tracing set off to avoid any unnecessary use of system resources. OMQ_TRACE is one of the three environment variables set to control trace. Specifying any value for OMQ_TRACE switches the trace facility on. Even if you set OMQ_TRACE to OFF, trace is still active. See Using trace.

OMQ_TRACE_PATH
One of the three environment variables set to control trace. See Using trace.

OMQ_TRACE_LEVEL
One of the three environment variables set to control trace. See Using trace.

ONCONFIG
The name of the Informix® server configuration file. For example, on UNIX and Linux systems, use:

```
export ONCONFIG=onconfig.hostname_1
```

On Windows systems, use:

```
set ONCONFIG=onconfig.hostname_1
```

WCF_TRACE_ON
Two different trace methods are available for the WCF custom channel, the two trace methods are activated independently or together. Each method produces its own trace file, so when both trace methods have been activated, two trace output files are generated. There are four combinations for enabling and disabling the two different trace methods. As well as these combinations to enable WCF trace, the XMS .NET trace can also be enabled using the WCF_TRACE_ON environment variable. See WCF trace configuration and trace file names.

WMQSOAP_HOME
Use when making additional configuration steps after the .NET SOAP over JMS service hosting environment is correctly installed and configured in IBM MQ. It is accessible from a local queue manager. See WCF client to a .NET service hosted by IBM MQ sample and WCF client to an Axis Java service hosted by IBM MQ sample.

Also use when you install IBM MQ web transport for SOAP. See Installing IBM MQ Web transport for SOAP.

Related information:
Using IBM MQ environment variables

Intercommunication jobs

IBM i

The following jobs are associated with Intercommunication on IBM i. The names are contained in the following table.

Table 34. Job names

Job name	Description
AMQCLMAA	Non-threaded Listener
AMQCRSTA	Non-threaded Responder Job
AMQRMPPA	Channel Pool Job
RUNMQCHI	Channel Initiator
RUNMQCHL	Channel Job
RUNMQLSR	Threaded Listener

Channel states on IBM i

IBM i

Channel states are displayed on the Work with Channels panel

Table 35. Channel states on IBM i

State name	Meaning
STARTING	Channel is ready to begin negotiation with target MCA
BINDING	Establishing a session and initial data exchange
REQUESTING	Requester channel initiating a connection
RUNNING	Transferring or ready to transfer
PAUSED	Waiting for message-retry interval
STOPPING	Establishing whether to retry or stop
RETRYING	Waiting until next retry attempt
STOPPED	Channel stopped because of an error or because an end-channel command is issued
INACTIVE	Channel ended processing normally or channel never started
*None	No state (for server-connection channels only)

Message channel planning example for UNIX, Linux, and Windows

ULW

This section provides a detailed example of how to connect two queue managers together so that messages can be sent between them.

The example illustrates the preparations required to enable an application using queue manager QM1 to put messages on a queue at queue manager QM2. An application running on QM2 can retrieve these messages, and send responses to a reply queue on QM1.

The example illustrates the use of TCP/IP connections. The example assumes that channels are to be triggered to start when the first message arrives on the transmission queue they are servicing. You must start the channel initiator in order for triggering to work.

This example uses SYSTEM.CHANNEL.INITQ as the initiation queue. This queue is already defined by IBM MQ. You can use a different initiation queue, but you must define it yourself and specify the name of the queue when you start the channel initiator.

What the example for UNIX, Linux, and Windows shows

ULW

The example shows the IBM MQ commands (MQSC) that you can use.

In all the examples, the MQSC commands are shown as they would appear in a file of commands, and as they would be typed at the command line. The two methods look identical, but, to issue a command at the command line, you must first type `runmqsc`, for the default queue manager, or `runmqsc` *qmname* where *qmname* is the name of the required queue manager. Then type any number of commands, as shown in the examples.

An alternative method is to create a file containing these commands. Any errors in the commands are then easy to correct. If you called your file mqsc.in then to run it on queue manager QMNAME use:

```
runmqsc QMNAME < mqsc.in > mqsc.out
```

You could verify the commands in your file before running it using:

```
runmqsc -v QMNAME < mqsc.in > mqsc.out
```

For portability, you should restrict the line length of your commands to 72 characters. Use a concatenation character to continue over more than one line. On Windows use Ctrl-z to end the input at the command line. On UNIX and Linux systems use Ctrl-d. Alternatively, use the **end** command.

Figure 7 on page 162 shows the example scenario.

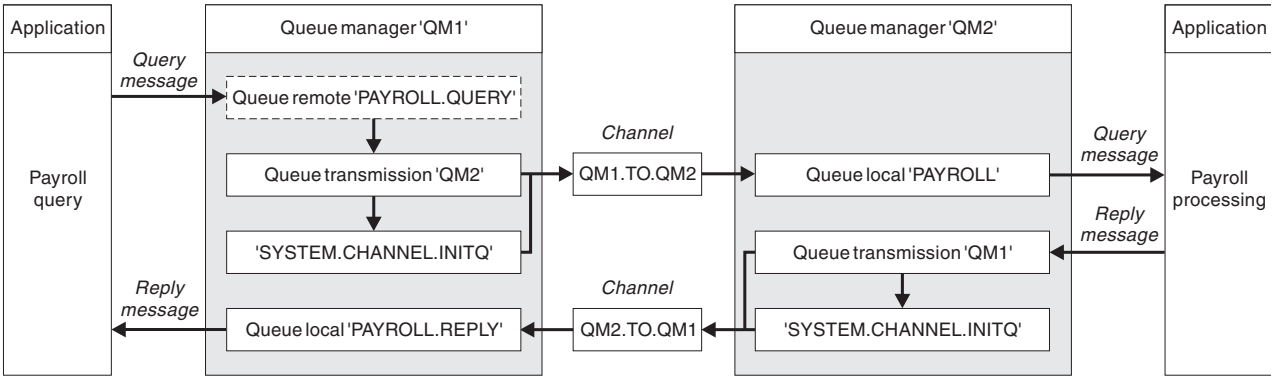


Figure 7. The message channel example for UNIX, Linux, and Windows systems

The example involves a payroll query application connected to queue manager QM1 that sends payroll query messages to a payroll processing application running on queue manager QM2. The payroll query application needs the replies to its queries sent back to QM1. The payroll query messages are sent from QM1 to QM2 on a sender-receiver channel called QM1.TO.QM2, and the reply messages are sent back from QM2 to QM1 on another sender-receiver channel called QM2.TO.QM1. Both of these channels are triggered to start as soon as they have a message to send to the other queue manager.

The payroll query application puts a query message to the remote queue “PAYROLL.QUERY” defined on QM1. This remote queue definition resolves to the local queue “PAYROLL” on QM2. In addition, the payroll query application specifies that the reply to the query is sent to the local queue “PAYROLL.REPLY” on QM1. The payroll processing application gets messages from the local queue “PAYROLL” on QM2, and sends the replies to wherever they are required; in this case, local queue “PAYROLL.REPLY” on QM1.

In the example definitions for TCP/IP, QM1 has a host address of 192.0.2.0 and is listening on port 1411, and QM2 has a host address of 192.0.2.1 and is listening on port 1412. The example assumes that these are already defined on your system and available for use.

The object definitions that need to be created on QM1 are:

- Remote queue definition, PAYROLL.QUERY
- Transmission queue definition, QM2 (default=remote queue manager name)
- Sender channel definition, QM1.TO.QM2
- Receiver channel definition, QM2.TO.QM1
- Reply-to queue definition, PAYROLL.REPLY

The object definitions that need to be created on QM2 are:

- Local queue definition, PAYROLL
- Transmission queue definition, QM1 (default=remote queue manager name)
- Sender channel definition, QM2.TO.QM1
- Receiver channel definition, QM1.TO.QM2

The connection details are supplied in the CONNAME attribute of the sender channel definitions.

You can see a diagram of the arrangement in Figure 7.

Queue manager QM1 example for UNIX, Linux, and Windows: ULW

These object definitions allow applications connected to queue manager QM1 to send request messages to a queue called PAYROLL on QM2, and to receive replies on a queue called PAYROLL.REPLY on QM1.

All the object definitions have been provided with the DESCR and REPLACE attributes. The other attributes supplied are the minimum required to make the example work. The attributes that are not supplied take the default values for queue manager QM1.

Run the following commands on queue manager QM1.

Remote queue definition

```
DEFINE QREMOTE(PAYROLL.QUERY) DESCR('Remote queue for QM2') REPLACE +
PUT(ENABLED) XMITQ(QM2) RNAME(PAYROLL) RQMNAME(QM2)
```

Note: The remote queue definition is not a physical queue, but a means of directing messages to the transmission queue, QM2, so that they can be sent to queue manager QM2.

Transmission queue definition

```
DEFINE QLOCAL(QM2) DESCR('Transmission queue to QM2') REPLACE +
USAGE(XMITQ) PUT(ENABLED) GET(ENABLED) TRIGGER TRIGTYPE(FIRST) +
INITQ(SYSTEM.CHANNEL.INITQ) PROCESS(QM1.TO.QM2.PROCESS)
```

When the first message is put on this transmission queue, a trigger message is sent to the initiation queue, SYSTEM.CHANNEL.INITQ. The channel initiator gets the message from the initiation queue and starts the channel identified in the named process.

Sender channel definition

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) TRPTYPE(TCP) +
REPLACE DESCR('Sender channel to QM2') XMITQ(QM2) +
CONNAME('192.0.2.1(1412)')
```

Receiver channel definition

```
DEFINE CHANNEL(QM2.TO.QM1) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QM2')
```

Reply-to queue definition

```
DEFINE QLOCAL(PAYROLL.REPLY) REPLACE PUT(ENABLED) GET(ENABLED) +
DESCR('Reply queue for replies to query messages sent to QM2')
```

The reply-to queue is defined as PUT(ENABLED). This ensures that reply messages can be put to the queue. If the replies cannot be put to the reply-to queue, they are sent to the dead-letter queue on QM1 or, if this queue is not available, remain on transmission queue QM1 on queue manager QM2. The queue has been defined as GET(ENABLED) to allow the reply messages to be retrieved.

Queue manager QM2 example for UNIX, Linux, and Windows: 

The following object definitions allow applications connected to queue manager QM2 to retrieve request messages from a local queue called PAYROLL, and to put replies to these request messages to a queue called PAYROLL.REPLY on queue manager QM1.

You do not need to provide a remote queue definition to enable the replies to be returned to QM1. The message descriptor of the message retrieved from local queue PAYROLL contains both the reply-to queue and the reply-to queue manager names. Therefore, as long as QM2 can resolve the reply-to queue manager name to that of a transmission queue on queue manager QM2, the reply message can be sent. In this example, the reply-to queue manager name is QM1 and so queue manager QM2 requires a transmission queue of the same name.

All the object definitions have been provided with the DESCR and REPLACE attributes and are the minimum required to make the example work. The attributes that are not supplied take the default values for queue manager QM2.

Run the following commands on queue manager QM2.

Local queue definition

```
DEFINE QLOCAL(PAYROLL) REPLACE PUT(ENABLED) GET(ENABLED) +
DESCR('Local queue for QM1 payroll details')
```

This queue is defined as PUT(ENABLED) and GET(ENABLED) for the same reason as the reply-to queue definition on queue manager QM1.

Transmission queue definition

```
DEFINE QLOCAL(QM1) DESCR('Transmission queue to QM1') REPLACE +
USAGE(XMITQ) PUT(ENABLED) GET(ENABLED) TRIGGER TRIGTYPE(FIRST) +
INITQ(SYSTEM.CHANNEL.INITQ) PROCESS(QM2.TO.QM1.PROCESS)
```

When the first message is put on this transmission queue, a trigger message is sent to the initiation queue, SYSTEM.CHANNEL.INITQ. The channel initiator gets the message from the initiation queue and starts the channel identified in the named process.

Sender channel definition

```
DEFINE CHANNEL(QM2.TO.QM1) CHLTYPE(SDR) TRPTYPE(TCP) +
REPLACE DESCR('Sender channel to QM1') XMITQ(QM1) +
CONNAME('192.0.2.0(1411)')
```

Receiver channel definition

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QM1')
```

Running the example for UNIX, Linux, and Windows

ULW

Information about starting the channel initiator and listener and suggestions for expanding on this scenario.

Once these definitions have been created, you need to:

- Start the channel initiator on each queue manager.
- Start the listener for each queue manager.

For information about starting the channel initiator and listener, see Setting up communication for Windows and Setting up communication on UNIX and Linux systems.

Expanding this example

This simple example could be expanded with:

- The use of LU 6.2 communications for interconnection with CICS systems, and transaction processing.
- Adding more queue, process, and channel definitions to allow other applications to send messages between the two queue managers.
- Adding user-exit programs on the channels to allow for link encryption, security checking, or additional message processing.
- Using queue manager aliases and reply-to queue aliases to understand more about how these can be used in the organization of your queue manager network.

Message channel planning example for IBM i

IBM i

This section provides a detailed example of how to connect two IBM i queue managers together so that messages can be sent between them.

The example illustrates the preparations needed to allow an application using queue manager QM1 to put messages on a queue at queue manager QM2. An application running on QM2 can retrieve these messages, and send responses to a reply queue on QM1.

The example illustrates the use of TCP/IP connections. The example assumes that channels are to be triggered to start when the first message arrives on the transmission queue they are servicing.

This example uses SYSTEM.CHANNEL.INITQ as the initiation queue. This queue is already defined by IBM MQ. You can use a different initiation queue, but you have to define it yourself, start a new instance of the channel initiator using the STRMQMCHLI command, and provide it with the name of your initiation queue. For more information about triggering channels, see Triggering channels.

What the example for IBM i shows

IBM i

This example involves a payroll query application connected to queue manager QM1 that sends payroll query messages to a payroll processing application running on queue manager QM2. The payroll query application needs the replies to its queries sent back to QM1.

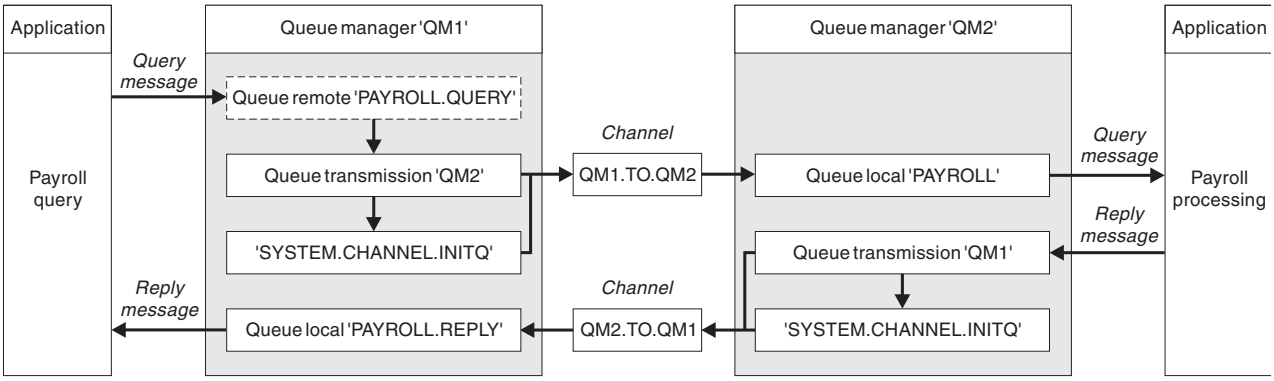


Figure 8. The message channel example for IBM MQ for IBM i

The payroll query messages are sent from QM1 to QM2 on a sender-receiver channel called QM1.TO.QM2, and the reply messages are sent back from QM2 to QM1 on another sender-receiver channel called QM2.TO.QM1. Both of these channels are triggered to start as soon as they have a message to send to the other queue manager.

The payroll query application puts a query message to the remote queue “PAYROLL.QUERY” defined on QM1. This remote queue definition resolves to the local queue “PAYROLL” on QM2. In addition, the payroll query application specifies that the reply to the query is sent to the local queue “PAYROLL.REPLY” on QM1. The payroll processing application gets messages from the local queue “PAYROLL” on QM2, and sends the replies to wherever they are required; in this case, local queue “PAYROLL.REPLY” on QM1.

Both queue managers are assumed to be running on IBM i. In the example definitions, QM1 has a host address of 192.0.2.0 and is listening on port 1411. QM2 has a host address of 192.0.2.1 and is listening on port 1412. The example assumes that these queue managers are already defined on your IBM i system, and are available for use.

The object definitions that need to be created on QM1 are:

- Remote queue definition, PAYROLL.QUERY
- Transmission queue definition, QM2 (default=remote queue manager name)
- Sender channel definition, QM1.TO.QM2
- Receiver channel definition, QM2.TO.QM1
- Reply-to queue definition, PAYROLL.REPLY

The object definitions that need to be created on QM2 are:

- Local queue definition, PAYROLL
- Transmission queue definition, QM1 (default=remote queue manager name)
- Sender channel definition, QM2.TO.QM1
- Receiver channel definition, QM1.TO.QM2

The connection details are supplied in the CONNAME attribute of the sender channel definitions.

You can see a diagram of the arrangement in Figure 8 on page 166.

Queue manager QM1 example for IBM i: IBM i

The following object definitions allow applications connected to queue manager QM1 to send request messages to a queue called PAYROLL on QM2, and to receive replies on a queue called PAYROLL.REPLY on QM1.

All the object definitions have been provided with the TEXT attributes. The other attributes supplied are the minimum required to make the example work. The attributes that are not supplied take the default values for queue manager QM1.

Run the following commands on queue manager QM1:

Remote queue definition
The CRTMQMQ command with the following attributes:

QNAME	'PAYROLL.QUERY'
QTYPE	*RMT
TEXT	'Remote queue for QM2'
PUTENBL	*YES
TMQNAME	'QM2' (default = remote queue manager name)
RMTQNAME	'PAYROLL'
RMTMQMNAME	'QM2'

Note: The remote queue definition is not a physical queue, but a means of directing messages to the transmission queue, QM2, so that they can be sent to queue manager QM2.

Transmission queue definition
The CRTMQMQ command with the following attributes:

QNAME	QM2
QTYPE	*LCL
TEXT	'Transmission queue to QM2'
USAGE	*TMQ
PUTENBL	*YES
GETENBL	*YES
TRGENBL	*YES
TRGTYPE	*FIRST
INITQNAME	SYSTEM.CHANNEL.INITQ
TRIGDATA	QM1.TO.QM2

When the first message is put on this transmission queue, a trigger message is sent to the initiation queue, SYSTEM.CHANNEL.INITQ. The channel initiator gets the message from the initiation queue and starts the channel identified in the named process.

Sender channel definition
The CRTMQMCHL command with the following attributes:

```
CHLNAME     QM1.TO.QM2
CHLTYPE     *SDR
TRPTYPE     *TCP
TEXT        'Sender channel to QM2'
TMQNAME     QM2
CONNAME     '192.0.2.1(1412)'
```

Receiver channel definition
The CRTMQMCHL command with the following attributes:

```
CHLNAME     QM2.TO.QM1
CHLTYPE     *RCVR
TRPTYPE     *TCP
TEXT        'Receiver channel from QM2'
```

Reply-to queue definition
The CRTMQMQ command with the following attributes:

```
QNAME       PAYROLL.REPLY
QTYPE       *LCL
TEXT        'Reply queue for replies to query messages sent to QM2'
PUTENBL     *YES
GETENBL     *YES
```

The reply-to queue is defined as PUT(ENABLED). This definition ensures that reply messages can be put to the queue. If the replies cannot be put to the reply-to queue, they are sent to the dead-letter queue on QM1 or, if this queue is not available, remain on transmission queue QM1 on queue manager QM2. The queue has been defined as GET(ENABLED) to allow the reply messages to be retrieved.

Queue manager QM2 example for IBM i: IBM i

The following object definitions allow applications connected to queue manager QM2 to retrieve request messages from a local queue called PAYROLL, and to put replies to these request messages to a queue called PAYROLL.REPLY on queue manager QM1.

You do not need to provide a remote queue definition to enable the replies to be returned to QM1. The message descriptor of the message retrieved from local queue PAYROLL contains both the reply-to queue and the reply-to queue manager names. Therefore, as long as QM2 can resolve the reply-to queue manager name to that of a transmission queue on queue manager QM2, the reply message can be sent. In this example, the reply-to queue manager name is QM1 and so queue manager QM2 requires a transmission queue of the same name.

All the object definitions have been provided with the TEXT attribute and are the minimum required to make the example work. The attributes that are not supplied take the default values for queue manager QM2.

Run the following commands on queue manager QM2:

Local queue definition
The CRTMQMQ command with the following attributes:

QNAME	PAYROLL
QTYPE	*LCL
TEXT	'Local queue for QM1 payroll details'
PUTENBL	*YES
GETENBL	*YES

This queue is defined as PUT(ENABLED) and GET(ENABLED) for the same reason as the reply-to queue definition on queue manager QM1.

Transmission queue definition
The CRTMQMQ command with the following attributes:

QNAME	QM1
QTYPE	*LCL
TEXT	'Transmission queue to QM1'
USAGE	*TMQ
PUTENBL	*YES
GETENBL	*YES
TRGENBL	*YES
TRGTYPE	*FIRST
INITQNAME	SYSTEM.CHANNEL.INITQ
TRIGDATA	QM2.TO.QM1

When the first message is put on this transmission queue, a trigger message is sent to the initiation queue, SYSTEM.CHANNEL.INITQ. The channel initiator gets the message from the initiation queue and starts the channel identified in the trigger data.

Sender channel definition
The CRTMQMCHL command with the following attributes:

CHLNAME	QM2.TO.QM1
CHLTYPE	*SDR
TRPTYPE	*TCP
TEXT	'Sender channel to QM1'
TMQNAME	QM1
CONNAME	'192.0.2.0(1411)'

Receiver channel definition
The CRTMQMCHL command with the following attributes:

CHLNAME	QM1.TO.QM2
CHLTYPE	*RCVR
TRPTYPE	*TCP
TEXT	'Receiver channel from QM1'

Running the example for IBM i

IBM i

When you have created the required objects you must start the channel initiators and listeners for both queue managers.

The applications can then send messages to each other. The channels are triggered to start by the first message arriving on each transmission queue, so you do not need to issue the STRMQMCHL command.

For details about starting a channel initiator and a listener, see Monitoring and controlling channels on IBM i.

Expanding the example for IBM i

IBM i

The example can be expanded in a number of ways.

This example can be expanded by:

- Adding more queue and channel definitions to allow other applications to send messages between the two queue managers.
- Adding user exit programs on the channels to allow for link encryption, security checking, or additional message processing.
- Using queue manager aliases and reply-to queue aliases to understand more about how these objects can be used in the organization of your queue manager network.

For a version of this example that uses MQSC commands, see "Message channel planning example for z/OS."

Message channel planning example for z/OS

z/OS

This section provides a detailed example of how to connect z/OS or MVS queue managers together so that messages can be sent between them.

The example illustrates the preparations needed to allow an application using queue manager QM1 to put messages on a queue at queue manager QM2. An application running on QM2 can retrieve these messages, and send responses to a reply queue on QM1.

The example illustrates the use of both TCP/IP and LU 6.2 connections. The example assumes that channels are to be triggered to start when the first message arrives on the transmission queue they are servicing.

What the example for z/OS shows

z/OS

This example involves a payroll query application connected to queue manager QM1 that sends payroll query messages to a payroll processing application running on queue manager QM2. The payroll query application needs the replies to its queries sent back to QM1.

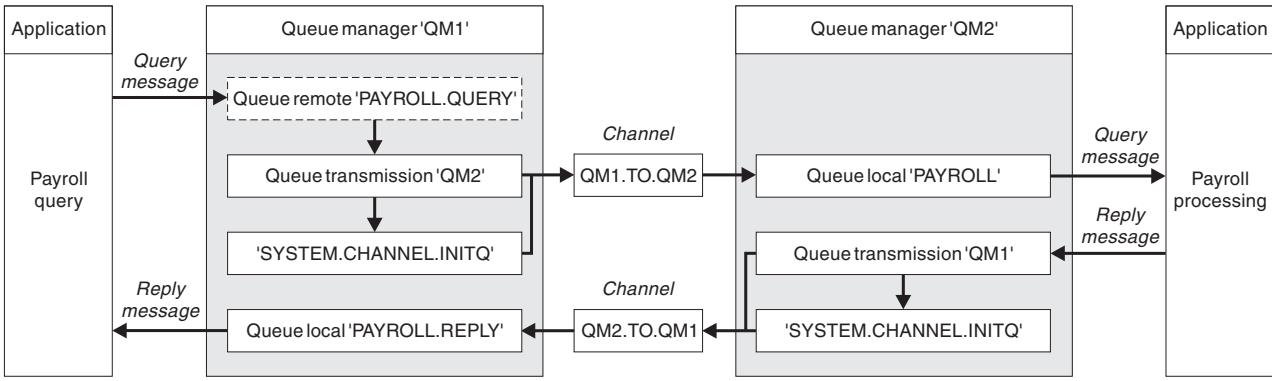


Figure 9. The first example for IBM MQ for z/OS

The payroll query messages are sent from QM1 to QM2 on a sender-receiver channel called QM1.TO.QM2, and the reply messages are sent back from QM2 to QM1 on another sender-receiver channel called QM2.TO.QM1. Both of these channels are triggered to start as soon as they have a message to send to the other queue manager.

The payroll query application puts a query message to the remote queue "PAYROLL.QUERY" defined on QM1. This remote queue definition resolves to the local queue "PAYROLL" on QM2. In addition, the payroll query application specifies that the reply to the query is sent to the local queue "PAYROLL.REPLY" on QM1. The payroll processing application gets messages from the local queue "PAYROLL" on QM2, and sends the replies to wherever they are required; in this case, local queue "PAYROLL.REPLY" on QM1.

Both queue managers are assumed to be running on z/OS. In the example definitions for TCP/IP, QM1 has a host address of 192.0.2.0 and is listening on port 1411, and QM2 has a host address of 192.0.2.1 and is listening on port 1412. In the definitions for LU 6.2, QM1 is listening on a symbolic luname called LUNAME1 and QM2 is listening on a symbolic luname called LUNAME2. The example assumes that these lunames are already defined on your z/OS system and available for use. To define them, see "Example MQ configuration for z/OS" on page 56.

The object definitions that need to be created on QM1 are:

- Remote queue definition, PAYROLL.QUERY
- Transmission queue definition, QM2 (default=remote queue manager name)
- Sender channel definition, QM1.TO.QM2
- Receiver channel definition, QM2.TO.QM1
- Reply-to queue definition, PAYROLL.REPLY

The object definitions that need to be created on QM2 are:

- Local queue definition, PAYROLL
- Transmission queue definition, QM1 (default=remote queue manager name)
- Sender channel definition, QM2.TO.QM1

- Receiver channel definition, QM1.TO.QM2

The example assumes that all the SYSTEM.COMMAND.* and SYSTEM.CHANNEL.* queues required to run DQM have been defined as shown in the supplied sample definitions, **CSQ4INSG** and **CSQ4INSX**.

The connection details are supplied in the CONNAME attribute of the sender channel definitions.

You can see a diagram of the arrangement in Figure 9 on page 171.

Queue manager QM1 example for z/OS: z/OS

The following object definitions allow applications connected to queue manager QM1 to send request messages to a queue called PAYROLL on QM2. It also allows applications to receive replies on a queue called PAYROLL.REPLY on QM1.

All the object definitions have been provided with the DESCR and REPLACE attributes. The other attributes supplied are the minimum required to make the example work. The attributes that are not supplied take the default values for queue manager QM1.

Run the following commands on queue manager QM1.

Remote queue definition

```
DEFINE QREMOTE(PAYROLL.QUERY) DESCR('Remote queue for QM2') REPLACE +
PUT(ENABLED) XMITQ(QM2) RNAME(PAYROLL) RQMNAME(QM2)
```

Note: The remote queue definition is not a physical queue, but a means of directing messages to the transmission queue, QM2, so that they can be sent to queue manager QM2.

Transmission queue definition

```
DEFINE QLOCAL(QM2) DESCR('Transmission queue to QM2') REPLACE +
USAGE(XMITQ) PUT(ENABLED) GET(ENABLED) TRIGGER TRIGTYPE(FIRST) +
TRIGDATA(QM1.TO.QM2) INITQ(SYSTEM.CHANNEL.INITQ)
```

When the first message is put on this transmission queue, a trigger message is sent to the initiation queue, SYSTEM.CHANNEL.INITQ. The channel initiator gets the message from the initiation queue and starts the channel identified in the trigger data. The channel initiator can only get trigger messages from the SYSTEM.CHANNEL.INITQ queue, so do not use any other queue as the initiation queue.

Sender channel definition

For a TCP/IP connection:

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) TRPTYPE(TCP) +
REPLACE DESCR('Sender channel to QM2') XMITQ(QM2) +
CONNAME('192.0.2.1(1412)')
```

For an LU 6.2 connection:

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) TRPTYPE(LU62) +
REPLACE DESCR('Sender channel to QM2') XMITQ(QM2) +
CONNAME('LUNAME2')
```

Receiver channel definition

For a TCP/IP connection:

```
DEFINE CHANNEL(QM2.TO.QM1) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QM2')
```

For an LU 6.2 connection:

```
DEFINE CHANNEL(QM2.TO.QM1) CHLTYPE(RCVR) TRPTYPE(LU62) +
REPLACE DESCR('Receiver channel from QM2')
```

Reply-to queue definition

```
DEFINE QLOCAL(PAYROLL.REPLY) REPLACE PUT(ENABLED) GET(ENABLED) +
DESCR('Reply queue for replies to query messages sent to QM2')
```

The reply-to queue is defined as PUT(ENABLED) which ensures that reply messages can be put to the queue. If the replies cannot be put to the reply-to queue, they are sent to the dead-letter queue on QM1 or, if this queue is not available, remain on transmission queue QM1 on queue manager QM2. The queue has been defined as GET(ENABLED) to allow the reply messages to be retrieved.

Queue manager QM2 example for z/OS: z/OS

The following object definitions allow applications connected to queue manager QM2 to retrieve request messages from a local queue called PAYROLL and to put replies to these request messages to a queue called PAYROLL.REPLY on queue manager QM1.

You do not need to provide a remote queue definition to enable the replies to be returned to QM1. The message descriptor of the message retrieved from local queue PAYROLL contains both the reply-to queue and the reply-to queue manager names. Therefore, as long as QM2 can resolve the reply-to queue manager name to that of a transmission queue on queue manager QM2, the reply message can be sent. In this example, the reply-to queue manager name is QM1 and so queue manager QM2 requires a transmission queue of the same name.

All the object definitions have been provided with the DESCR and REPLACE attributes and are the minimum required to make the example work. The attributes that are not supplied take the default values for queue manager QM2.

Run the following commands on queue manager QM2.

Local queue definition

```
DEFINE QLOCAL(PAYROLL) REPLACE PUT(ENABLED) GET(ENABLED) +
DESCR('Local queue for QM1 payroll details')
```

This queue is defined as PUT(ENABLED) and GET(ENABLED) for the same reason as the reply-to queue definition on queue manager QM1.

Transmission queue definition

```
DEFINE QLOCAL(QM1) DESCR('Transmission queue to QM1') REPLACE +
USAGE(XMITQ) PUT(ENABLED) GET(ENABLED) TRIGGER TRIGTYPE(FIRST) +
TRIGDATA(QM2.TO.QM1) INITQ(SYSTEM.CHANNEL.INITQ)
```

When the first message is put on this transmission queue, a trigger message is sent to the initiation queue, SYSTEM.CHANNEL.INITQ. The channel initiator gets the message from the initiation queue and starts the channel identified in the trigger data. The channel initiator can only get trigger messages from SYSTEM.CHANNEL.INITQ so do not use any other queue as the initiation queue.

Sender channel definition

For a TCP/IP connection:

```
DEFINE CHANNEL(QM2.TO.QM1) CHLTYPE(SDR) TRPTYPE(TCP) +
REPLACE DESCR('Sender channel to QM1') XMITQ(QM1) +
CONNAME('192.0.2.0(1411)')
```

For an LU 6.2 connection:

```
DEFINE CHANNEL(QM2.TO.QM1) CHLTYPE(SDR) TRPTYPE(LU62) +
REPLACE DESCR('Sender channel to QM1') XMITQ(QM1) +
CONNAME('LUNAME1')
```

Receiver channel definition

For a TCP/IP connection:

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QM1')
```

For an LU 6.2 connection:

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(RCVR) TRPTYPE(LU62) +
REPLACE DESCR('Receiver channel from QM1')
```

Running the example for z/OS

z/OS

When you have created the required objects, you must start the channel initiators and listeners for both queue managers.

The applications can then send messages to each other. Because the channels are triggered to start by the arrival of the first message on each transmission queue, you do not need to issue the START CHANNEL MQSC command.

For details about starting a channel initiator see Starting a channel initiator, and for details about starting a listener see Starting a channel listener.

Expanding the example for z/OS

z/OS

The example can be expanded in a number of ways.

The example can be expanded by:

- Adding more queue, and channel definitions to allow other applications to send messages between the two queue managers.
- Adding user exit programs on the channels to allow for link encryption, security checking, or additional message processing.
- Using queue manager aliases and reply-to queue aliases to understand more about how these aliases can be used in the organization of your queue manager network.

Message channel planning example for z/OS using queue-sharing groups

z/OS

This example illustrates the preparations needed to allow an application using queue manager QM3 to put a message on a queue in a queue-sharing group that has queue members QM4 and QM5.

Ensure you are familiar with the example in “Message channel planning example for z/OS” on page 170 before trying this example.

What the queue-sharing group example for z/OS shows

z/OS

This example shows the IBM MQ commands (MQSC) that you can use in IBM MQ for z/OS for distributed queuing with queue-sharing groups.

This example expands the payroll query scenario of the example in “Message channel planning example for z/OS” on page 170 to show how to add higher availability of query processing by adding more serving applications to serve a shared queue.

The payroll query application is now connected to queue manager QM3 and puts a query to the remote queue 'PAYROLL QUERY' defined on QM3. This remote queue definition resolves to the shared queue 'PAYROLL' hosted by the queue managers in the queue-sharing group QSG1. The payroll processing application now has two instances running, one connected to QM4 and one connected to QM5.

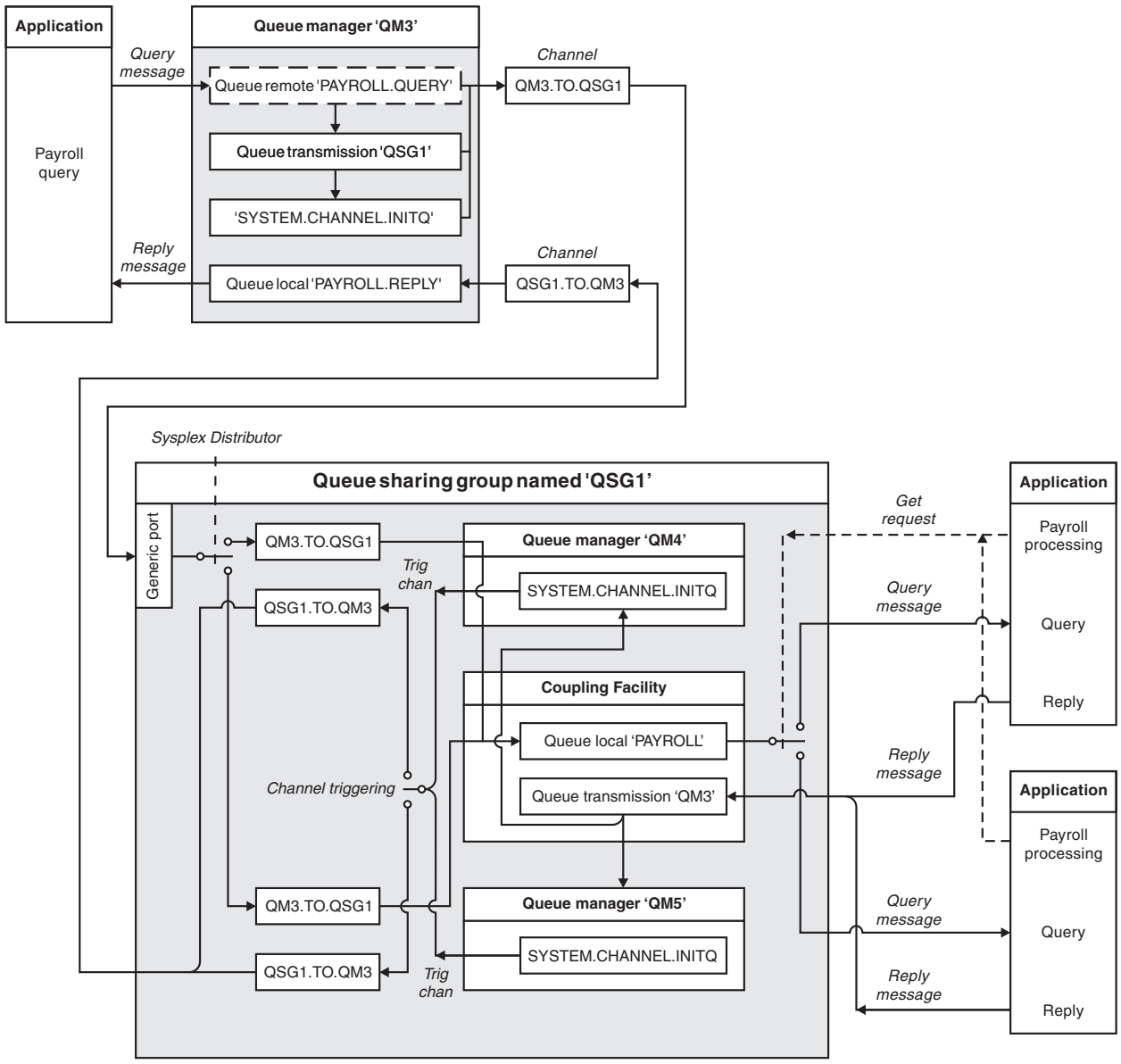


Figure 10. Message channel planning example for IBM MQ for z/OS using queue-sharing groups

All three queue managers are assumed to be running on z/OS. In the example definitions for TCP/IP, QM4 has a VIPA address of MVSIP01 and QM5 has a VIPA address of MVSIP02. Both queue managers are listening on port 1414. The generic address that Sysplex Distributor provides for this group is QSG1.MVSIP. QM3 has a host address of 192.0.2.0 and is listening on port 1411.

In the example definitions for LU6.2, QM3 is listening on a symbolic luname called LUNAME1. The name of the generic resource defined for VTAM for the lunames listened on by QM4 and QM5 is LUQSG1. The example assumes that they are already defined on your z/OS system and are available for use. To define them see "Defining yourself to the network using generic resources" on page 63.

In this example QSG1 is the name of a queue-sharing group, and queue managers QM4 and QM5 are the names of members of the group.

Queue-sharing group definitions

z/OS

Producing the following object definitions for one member of the queue-sharing group makes them available to all the other members.

Queue managers QM4 and QM5 are members of the queue-sharing group. The definitions produced for QM4 are also available for QM5.

It is assumed that the coupling facility list structure is called 'APPLICATION1'. If it is not called 'APPLICATION1', you must use your own coupling facility list structure name for the example.

Shared objects

The shared object definitions are stored in DB2® and their associated messages are stored within the coupling facility.

```
DEFINE QLOCAL(PAYROLL) QSGDISP(SHARED) REPLACE PUT(ENABLED) GET(ENABLED) +
CFSTRUCT(APPLICATION1) +
DESCR('Shared queue for payroll details')

DEFINE QLOCAL(QM3) QSGDISP(SHARED) REPLACE USAGE(XMITQ) PUT(ENABLED) +
CFSTRUCT(APPLICATION1) +
DESCR('Transmission queue to QM3') TRIGGER TRIGTYPE(FIRST) +
TRIGDATA(QSG1.TO.QM3) GET(ENABLED) INITQ(SYSTEM.CHANNEL.INITQ)
```

Group objects

The group object definitions are stored in Db2, and each queue manager in the queue-sharing group creates a local copy of the defined object.

Sender channel definition for a TCP/IP connection:

```
DEFINE CHANNEL(QSG1.TO.QM3) CHLTYPE(SDR) QSGDISP(GROUP) TRPTYPE(TCP) +
REPLACE DESCR('Sender channel to QM3') XMITQ(QM3) +
CONNAME('192.0.2.0(1411)')
```

Sender channel definition for an LU 6.2 connection:

```
DEFINE CHANNEL(QSG1.TO.QM3) CHLTYPE(SDR) QSGDISP(GROUP) TRPTYPE(LU62) +
REPLACE DESCR('Sender channel to QM3') XMITQ(QM3) +
CONNAME('LUNAME1')
```

Receiver channel definition for a TCP/IP connection:

```
DEFINE CHANNEL(QM3.TO.QSG1) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QM3') QSGDISP(GROUP)
```

Receiver channel definition for an LU 6.2 connection:

```
DEFINE CHANNEL(QM3.TO.QSG1) CHLTYPE(RCVR) TRPTYPE(LU62) +
REPLACE DESCR('Receiver channel from QM3') QSGDISP(GROUP)
```

Related reference:

"Disposition (QSGDISP)" on page 114
This attribute specifies the disposition of the channel in a queue-sharing group. It is valid on z/OS only.

Queue manager QM3 example for z/OS

z/OS

QM3 is not a member of the queue-sharing group. The following object definitions allow it to put messages to a queue in the queue-sharing group.

The CONNAME for this channel is the generic address of the queue-sharing group, which varies according to transport type.

For a TCP/IP connection:

```
DEFINE CHANNEL(QM3.TO.QSG1) CHLTYPE(SDR) TRPTYPE(TCP) +
REPLACE DESCR('Sender channel to QSG1') XMITQ(QSG1) +
CONNAME('QSG1.MVSIP(1414)')
```

For an LU 6.2 connection:

```
DEFINE CHANNEL(QM3.TO.QSG1) CHLTYPE(SDR) TRPTYPE(LU62) +
REPLACE DESCR('Sender channel to QSG1') XMITQ(QSG1) +
CONNAME('LUQSG1') TPNAME('MQSERIES') MODENAME('#INTER')
```

Other definitions

These definitions are required for the same purposes as the definitions in the first example.

```
DEFINE QREMOTE(PAYROLL.QUERY) DESCR('Remote queue for QSG1') REPLACE +
PUT(ENABLED) XMITQ(QSG1) RNAME(APPL) RQMNAME(QSG1)

DEFINE QLOCAL(QSG1) DESCR('Transmission queue to QSG1') REPLACE +
USAGE(XMITQ) PUT(ENABLED) GET(ENABLED) TRIGGER TRIGTYPE(FIRST) +
TRIGDATA(QM3.TO.QSG1) INITQ(SYSTEM.CHANNEL.INITQ)

DEFINE CHANNEL(QSG1.TO.QM3) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QSG1')

DEFINE CHANNEL(QSG1.TO.QM3) CHLTYPE(RCVR) TRPTYPE(LU62) +
REPLACE DESCR('Receiver channel from QSG1')

DEFINE QLOCAL(PAYROLL.REPLY) REPLACE PUT(ENABLED) GET(ENABLED) +
DESCR('Reply queue for replies to query messages sent to QSG1')
```

Running the queue-sharing group example for z/OS

z/OS

When you have created the required objects you need to start the channel initiators for all three queue managers. You also need to start the listeners for both queue managers in the queue-sharing group.

For a TCP/IP connection, each member of the group must have a group listener started that is listening on port 1414.

```
STA LSTR PORT(1414) IPADDR(MVSIP01) INDISP(GROUP)
```

The previous entry starts the listener on QM4, for example.

For an LU6.2 connection, each member of the group must have a group listener started that is listening on a symbolic luname. This luname must correspond to the generic resource LUQSG1.

- Start the listener on QM3

```
STA LSTR PORT(1411)
```

Using an alias to refer to an MQ library

You can define an alias to refer to an MQ library in your JCL, rather than use the name of the MQ library directly. Then, if the name of the MQ library changes, you have only to delete and redefine the alias.

Example

The following example defines an alias MQM.SCSQANLE to refer to the MQ library MQM.V600.SCSQANLE:

```
//STEP1 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DELETE (MQM.SCSQANLE)
DEFINE ALIAS (NAME(MQM.SCSQANLE) RELATE(MQM.V600.SCSQANLE))
/*
```

Then, to refer to the MQM.V600.SCSQANLE library in your JCL, use the alias MQM.SCSQANLE.

Note: The library and alias names must be in the same catalog, so use the same high level qualifier for both; in this example, the high level qualifier is MQM.

`mqzOSConnectService` element

z/OS V 9.0.1

The MQ Service Provider is provided as a standard Liberty feature and so is configured using server.xml. Each one or two-way service is defined in an `mqzOSConnectService` element. This element and all of its attributes apply to both z/OS Connect V1 and z/OS Connect EE.

Important: An `mqzOSConnectService` element needs to be referenced by a `zOSConnectService` element before it can be used.

An example `mqzOSConnectService` element with some attributes specified is shown below.

```
<mqzOSConnectService id="twoWay "
                     connectionFactory="jms/cf1"
                     destination="jms/requestQueue"
                     replyDestination="jms/replyQueue"
                     expiry="-1"
```

```
waitInterval="10000"
replySelection="msgIDToCorrelID"
selector=""
persistence="false"/>
```

Attention: Depending on how the MQ Service Provider has been installed, the `mqzOSConnectService` element might be prefixed with a string followed by an underscore, for example `usr_mqzOSConnectService`.

This is described in Installing the MQ Service Provider into WLP for z/OS Connect V1 and Installing the MQ Service Provider into IBM z/OS Connect EE for z/OS Connect EE.

The format shown in the following example is where the MQ Service Provider has been installed into the WLP kernel (as described in option 1 of Installing the MQ Service Provider into WLP

Table 36. Attributes of an `mqzOSConnectService` element

Attribute name	Type	Default value	Description
id	string		"id"
connectionFactory	A JNDI name (string).		"connectionFactory"
destination	A JNDI name (string).		"destination" on page 181
replyDestination	A JNDI name (string).		"replyDestination" on page 181
expiry	integer	-1	"expiry" on page 181
waitInterval	integer		"waitInterval" on page 181
replySelection	string	`msgIDToCorrelID`	"replySelection" on page 182
selector	string		"selector" on page 182
persistence	boolean	false	"persistence" on page 183
mqmdFormat	string		"mqmdFormat" on page 183
userName	string		"userName" on page 183
password	string		"password" on page 183
useCallerPrincipal	boolean	false	"useCallerPrincipal" on page 184
receiveTextCCSID	integer	37	"receiveTextCCSID" on page 184

id

`id` is a required attribute and must be unique across all elements in server.xml. **`id`** is used by the `zosConnectService` element to refer to a target service provider instance.

connectionFactory

`connectionFactory` specifies the JNDI name of an IBM MQ messaging provider connection factory. The MQ Service Provider uses the connection factory to connect to IBM MQ.

`connectionFactory` is a required attribute. For more information on connection factories, see JMS Connection Factory.

You should specify **`transportType`**="*BINDINGS*" for the connection factory.

destination

destination specifies the JNDI name of an IBM MQ messaging provider destination.

destination is a required attribute.

For more information on configuring a:
- Queue in WLP, see JMS Queue.
- Topic in WLP, see JMS Topic.

For a one-way service, **destination** is used as the target for HTTP POST, HTTP GET, and HTTP DELETE requests.

Note that queue destinations are supported for all three request types whereas topic destinations are supported only with HTTP POST requests.

For a two-way service, **destination** must be a queue destination which represents the request queue used by the back end service.

Two-way services support only HTTP POST requests.

replyDestination

replyDestination specifies the JNDI name of an IBM MQ messaging provider queue.

replyDestination is an optional attribute.

For more information on configuring a queue in WLP, see JMS Queue.

If **replyDestination** is not specified, the service is a one-way service. If **replyDestination** is specified, the service is a two-way service.

This queue is the reply destination where the back end service sends reply messages to.

expiry

expiry specifies how long messages sent by the MQ Service Provider are valid for, in thousandths of a second, from the time they were sent. The message becomes eligible to be discarded if it has not been removed from the destination queue before this period of time elapses.

expiry is an optional attribute, and is equivalent to setting the MQMD Expiry field.

Negative values means that messages never expire. The default value of **expiry** is -1.

REST clients can override **expiry** by specifying an `ibm-mq-md-expiry` HTTP header with a valid 64-bit integer.

waitInterval

For HTTP DELETE requests to one-way services, **waitInterval** specifies the number of milliseconds that the service waits for a matching message on the queue, specified by the **destination** attribute.

For HTTP POST requests to two-way services, **waitInterval** specifies the number of milliseconds that the service waits for a matching message on the queue, specified by the **replydestination** attribute.

waitInterval is an optional attribute for one-way services, a required attribute for two-way services, and is equivalent to setting the MQMD WaitInterval field.

waitInterval is not supported with HTTP GET requests.

If **waitInterval** is:

- Zero, the service does not wait.

  A **waitInterval** of zero is not supported with two-way services.
- Negative, the service waits for ever until a message is available.

REST clients can override this value by specifying an `ibm-mq-gmo-waitInterval` HTTP header with a valid 64 bit integer.

Note: Specifying a large, or negative **waitInterval**, is likely to result in transaction timeouts and asynchronous service request timeouts. If either, or both, of these events occur, increase the timeout, reduce the wait interval, or do both.

replySelection

`replySelection` describes the mechanism used to match reply messages with request messages.

`replySelection` is optional and used only used with two-way services. If `replySelection` is used with a one-way service, it is ignored.

The value is one of the following:

msgIDToCorrelID
: Reply messages are assumed to be generated with the correlation ID set to the value of the message ID from the request message. The service generates a suitable message selector based on this information. This is the default value.

none
: No mechanism is used to correlate reply messages with request messages. The service gets the first available message on the reply queue.

correlIDToCorrelID
: Reply messages are assumed to be generated with the correlation ID set to the value of the correlation ID from the request message. The service generates a suitable message selector based on this information. If the request message does not have a correlation ID specified (see "ibm-mq-md-correlID" on page 184) the service generates a random correlation ID for the request message.

selector

`selector` must be a valid JMS message selector as described by the JMS specification.

`selector` is only used with one-way services and is optional. If `selector` is specified on a two-way service it is ignored. For more information on selectors, see Message selectors in JMS.

`selector` is used on HTTP GET and HTTP DELETE requests to select which message is returned. If the "ibm-mq-md-msgID" on page 184 or "ibm-mq-md-correlID" on page 184 headers are specified, `selector` is ignored.

Some selector characters need to be encoded in order to be embedded in `server.xml`. You can do this using standard mechanisms as follows:

```
" becomes &quot;
' becomes &apos;
< becomes &lt;
> becomes &gt;
```

persistence

`persistence` specifies the persistence of messages sent by a service.

`persistence` is optional, and is equivalent to setting the MQMD Persistence field.

The value is one of the following:

false Means messages are non-persistent. This is the default value.

true Means messages are persistent.

You can override `persistence` by using an `ibm-mq-md-persistence` HTTP header which takes the same values.

mqmdFormat

This attribute is used to set the value of the MQMD format field in messages that are sent by the MQ Service Provider. However, it is only used when the MQ Service Provider has been configured to use z/OS Connect data transformations, otherwise it is ignored.

If you do not specify this attribute, and data transformations are used, messages are sent with the MQMD format field set to blanks. The value of this attribute must be less than, or equal to, eight characters in length.

userName

The user name that the MQ Service Provider presents to IBM MQ for authentication and authorization purposes.

If you do not specify this attribute, the **`userName`** attribute in the connection factory referred to by the **`connectionFactory`** attribute is used.

If a **`userName`** attribute is specified, both on the referenced connection factory and on the MQ Service Provider, the MQ Service Provider value is used.

If you specify this attribute, you must specify the **`password`** attribute.

password

The password that the MQ Service Provider presents to IBM MQ for authentication and authorization purposes.

You can specify the password in plain text, although you should not do so. Instead, you should encode the password using the **`securityUtility`** tool provided with z/OS Connect, using the encode option.For more information see Liberty: securityUtility command.

If you do not specify this attribute, the password attribute in the connection factory referred to by the **`connectionFactory`** attribute is used.

If a password attribute is specified both on the referenced connection factory and on the MQ Service Provider the MQ Service Provider value is used.

If you specify this attribute, you must also specify the **userName** attribute.

useCallerPrincipal

When a request is made to z/OS Connect the caller authenticates with z/OS Connect. The name of the authenticated principle can be passed onto IBM MQ for authentication and authorization purposes.

To do this, set the value of **useCallerPrincipal** to true.

The name of the principal, but no password, is used when connecting to IBM MQ. Any values specified in the **password** and **userName** attributes are ignored.

receiveTextCCSID

The CCSID that is used when a data transformation is received and a `javax.jms.TextMessage` is being consumed (that is, an HTTP GET or HTTP DELETE with a one-way service, or on retrieving a response message for a two-way service).

The text in the message is converted into the CCSID specified by **receiveTextCCSID**.

HTTP headers that can be used with the MQ Service Provider

The only time the MQ Service Provider expects specific HTTP headers, is when an HTTP POST is issued.

In that case the Content-Type header must be set to "`application/json`". If you specify a character set as part of this header, its value must be utf-8.

For example `Content-Type=application/json;charset=utf-8`.

Other HTTP headers can be specified on the HTTP request to change the behavior of the MQ Service Provider; these are detailed in the following sections. Any other HTTP headers are ignored.

ibm-mq-md-msgID

This header can be specified when issuing HTTP GET or HTTP DELETE requests to one-way services.

The value of this header is used to generate a message selector to select a message with the specified message ID. If an “ibm-mq-md-correlID” header is also specified, a message selector that matches both IDs will be generated.

See `msgId`: HTTP `x-msg-msgId` entity-header for details of the value of the format of this header.

ibm-mq-md-correlID

This header can be specified when issuing an HTTP POST, in which case it is used to set the MQMD CorrelID field of the message that gets sent.

This header can also be specified when issuing HTTP GET or DELETE requests to one-way services. The value of this header is used to generate a message selector to select a message with the specified correlation ID. If an “ibm-mq-md-msgID” header is also specified, a message selector that matches both will be generated.

See `correlId`: HTTP `x-msg-correlId` entity-header for details of the value of the format of this header.

ibm-mq-pmo-retain

You can specify this header with a value of TRUE when issuing an HTTP POST request to a one-way service backed by a topic. This results in a retained publication being generated. For more information, see Retained publications.

ibm-mq-usr

You can use this header to provide message properties on the IBM MQ messages sent as a result of HTTP POST requests to both one-way and two-way services.

For details of the value of the format of this header, see usr: HTTP x-msg-usr entity-header.

Although the name used by the MQ Service Provider is different, see `require-headers`: HTTP `x-msg-require-headers` request-header for details of the value of the format of this header.

Administration reference

Use the links to reference information in this section to help you operate and administer IBM MQ.

- "Syntax diagrams"
- "IBM MQ Control commands reference" on page 195
- IBM i "CL commands reference for IBM i" on page 1065
- "MQSC reference" on page 363
- "Programmable command formats reference" on page 1068
- z/OS "Using the IBM MQ utilities on z/OS" on page 1895
- "IBM MQ Administration Interface" on page 1811

Related information:

Queue names
There are restrictions on the length of queue names. Some queue names are reserved for queues defined by the queue manager.

IBM i IBM MQ for IBM i system and default objects
When you create a queue manager using the `CRTMQM` command, the system objects and the default objects are created automatically.

Syntax diagrams

z/OS

The syntax for a command and its options is presented in the form of a syntax diagram called a railroad diagram. Railroad diagrams are a visual format suitable for sighted users. It tells you what options you can supply with the command, how to enter them, indicates relationships between different options, and sometimes different values of an option.

Each railroad diagram begins with a double right arrow and ends with a right and left arrow pair. Lines beginning with a single right arrow are continuation lines. You read a railroad diagram from left to right and from top to bottom, following the direction of the arrows.

Other conventions used in railroad diagrams are:

Table 37. How to read railroad diagrams

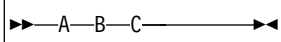
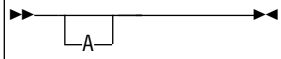
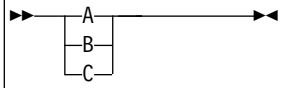
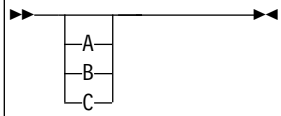
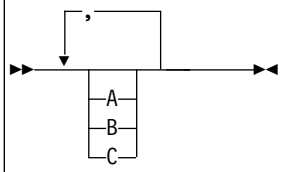
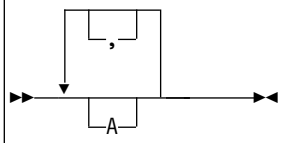
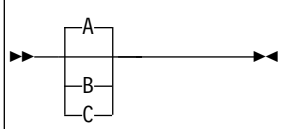
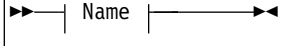
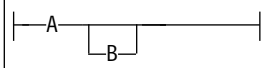
Convention	Meaning
	You must specify values `A`, `B`, and `C`. Required values are shown on the main line of a railroad diagram.
	You may specify value `A`. Optional values are shown below the main line of a railroad diagram.
	Values `A`, `B`, and `C` are alternatives, one of which you must specify.
	Values `A`, `B`, and `C` are alternatives, one of which you might specify.
	You might specify one or more of the values `A`, `B`, and `C`. Any required separator for multiple or repeated values (in this example, the comma (,)) is shown on the arrow.
	You might specify value `A` multiple times. The separator in this example is optional.
	Values `A`, `B`, and `C` are alternatives, one of which you might specify. If you specify none of the values shown, the default `A` (the value shown above the main line) is used.
 **Name:** 	The railroad fragment `Name` is shown separately from the main railroad diagram.

Table 37. How to read railroad diagrams (continued)

Convention	Meaning
Punctuation and uppercase values	Specify exactly as shown.

How to read railroad diagrams

Each railroad diagram begins with a double right arrow and ends with a right and left arrow pair. Lines beginning with a single right arrow are continuation lines. You read a railroad diagram from left to right and from top to bottom, following the direction of the arrows.

This information has moved. See “Syntax diagrams” on page 185.

Command sets comparison

ULW

The tables in this section compare the facilities available for UNIX, Linux, and Windows from the different administration command sets, and also show whether you can perform each function by using the IBM MQ Explorer or REST API.

Note: z/OS These comparison tables do not apply to IBM MQ for z/OS. For information on how to use MQSC commands and PCF commands on z/OS, see Issuing commands to IBM MQ for z/OS.

IBM i These comparison tables do not apply to IBM MQ for IBM i. For information on how to use MQSC commands and PCF commands on IBM i, see Alternative ways of administering IBM MQ for IBM i.

Related information:

Administering IBM MQ

Script (MQSC) Commands

Introduction to Programmable Command Formats

Administering using the REST API

Introduction to MQ Explorer

Queue manager commands

ULW

A table of queue manager commands, showing the PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

Table 38. Queue manager commands

Description	PCF command	MQSC command	Control command	V 9.0.1 REST API resource and HTTP method	IBM MQ Explorer equivalent?
Change Queue Manager	Change Queue Manager	ALTER QMGR	No equivalent		Yes
Create Queue Manager	No equivalent	No equivalent	**crtmqm**		Yes
Delete Queue Manager	No equivalent	No equivalent	**dltmqm**		Yes

Table 38. Queue manager commands (continued)

Description	PCF command	MQSC command	Control command	V 9.0.1 REST API resource and HTTP method	IBM MQ Explorer equivalent?
Inquire Queue Manager	Inquire Queue Manager	DISPLAY QMGR	No equivalent		Yes
Inquire Queue Manager Status	Inquire Queue Manager Status	DISPLAY QMSTATUS	**dspmq**	V 9.0.1 GET `/admin/installation` V 9.0.3 GET `/admin/qmgr`	Yes
Ping Queue Manager	Ping Queue Manager	PING QMGR	No equivalent		No
Refresh Queue Manager	Refresh Queue Manager	REFRESH QMGR	No equivalent		Yes
Reset Queue Manager	Reset Queue Manager	RESET QMGR	No equivalent		No
Start Queue Manager	No equivalent	No equivalent	**strmqm**		Yes
Stop Queue Manager	No equivalent	No equivalent	**endmqm**		Yes

Related information:

Creating and managing queue managers on Multiplatforms

Command server commands

ULW

A table of command server commands, showing the PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

Table 39. Commands for command server administration

Description	PCF command	MQSC command	Control command	V 9.0.1 REST API resource and HTTP method	IBM MQ Explorer equivalent?
Display command server	Inquire Queue Manager Status	DISPLAY QMSTATUS	**dspmqcsv**		Yes
Start command server	Change Queue Manager	ALTER QMGR	**strmqcsv**		Yes
Stop command server	No equivalent	No equivalent	**endmqcsv**		Yes

Authority commands

ULW

A table of authority commands, showing the PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

Table 40. Commands for authority administration

PCF command	MQSC command	Control command	V 9.0.1 REST API resource and HTTP method	IBM MQ Explorer equivalent?
Delete authority record	DELETE AUTHREC	**setmqaut**		Yes
Inquire authority records	DISPLAY AUTHREC	**dmpmqaut**		Yes
Inquire entity authority	DISPLAY ENTAUTH	**dspmqaut**		Yes
Refresh Security	REFRESH SECURITY	No equivalent		Yes
Set authority record	SET AUTHREC	**setmqaut**		Yes

Cluster commands

ULW

A table of cluster commands, showing the PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

Table 41. Cluster commands

PCF command	MQSC command	Control command	V 9.0.1 REST API resource and HTTP method	IBM MQ Explorer equivalent?
Inquire Cluster Queue Manager	DISPLAY CLUSQMGR	No equivalent		Yes
Refresh Cluster	REFRESH CLUSTER	No equivalent		Yes
Reset Cluster	RESET CLUSTER	No equivalent		No
Resume Queue Manager Cluster	RESUME QMGR	No equivalent		Yes
Suspend Queue Manager Cluster	SUSPEND QMGR	No equivalent		Yes

Authentication information commands

ULW

A table of authentication information commands, showing the PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

Table 42. Authentication information commands

PCF command	MQSC command	Control command	V 9.0.1 REST API resource and HTTP method	IBM MQ Explorer equivalent?
Change Authentication Information Object	ALTER AUTHINFO	No equivalent		Yes
Copy Authentication Information Object	DEFINE AUTHINFO(x) LIKE(y)	No equivalent		Yes
Create Authentication Information Object	DEFINE AUTHINFO	No equivalent		Yes
Delete Authentication Information Object	DELETE AUTHINFO	No equivalent		Yes
Inquire Authentication Information Object	DISPLAY AUTHINFO	No equivalent		Yes

Channel commands

ULW

A table of channel commands, showing the PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

Table 43. Channel commands

PCF command	MQSC command	Control command	V 9.0.1 REST API resource and HTTP method	IBM MQ Explorer equivalent?
Change Channel	ALTER CHANNEL	No equivalent		Yes
Copy Channel	DEFINE CHANNEL(x) LIKE(y)	No equivalent		Yes
Create Channel	DEFINE CHANNEL	No equivalent		Yes
Delete Channel	DELETE CHANNEL	No equivalent		Yes
Inquire Channel	DISPLAY CHANNEL	No equivalent		Yes
Inquire Channel Names	DISPLAY CHANNEL	No equivalent		Yes
Inquire Channel Status	DISPLAY CHSTATUS	No equivalent		Yes
Ping Channel	PING CHANNEL	No equivalent		Yes
Purge Channel	PURGE CHANNEL	No equivalent		Yes

Table 43. Channel commands (continued)

PCF command	MQSC command	Control command	V 9.0.1 REST API resource and HTTP method	IBM MQ Explorer equivalent?
Reset Channel	RESET CHANNEL	No equivalent		Yes
Resolve Channel	RESOLVE CHANNEL	No equivalent		Yes
Start Channel	START CHANNEL	**runmqchl**		Yes
Start Channel Initiator	START CHINIT	**runmqchi**		No
Stop Channel	STOP CHANNEL	No equivalent		Yes

Listener commands

ULW

A table of listener commands, showing the PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

Table 44. Listener commands

PCF command	MQSC command	Control command	V 9.0.1 REST API resource and HTTP method	IBM MQ Explorer equivalent?
Change Listener	ALTER LISTENER	No equivalent		Yes
Copy Listener	DEFINE LISTENER(x) LIKE(y)	No equivalent		Yes
Create Listener	DEFINE LISTENER	No equivalent		Yes
Delete Listener	DELETE LISTENER	No equivalent		Yes
Inquire Listener	DISPLAY LISTENER	No equivalent		Yes
Inquire Listener Status	DISPLAY LSSTATUS	No equivalent		Yes
Start Channel Listener	START LISTENER [1]	**runmqlsr**		Yes
Stop Listener	STOP LISTENER	**endmqlsr** [2]		Yes

Notes:

1. Used with listener objects only
2. Stops all active listeners

Namelist commands

ULW

A table of namelist commands, showing the PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

Table 45. Namelist commands

PCF command	MQSC command	Control command	V 9.0.1 REST API resource and HTTP method	IBM MQ Explorer equivalent?
Change Namelist	ALTER NAMELIST	No equivalent		Yes
Copy Namelist	DEFINE NAMELIST(x) LIKE(y)	No equivalent		Yes
Create Namelist	DEFINE NAMELIST	No equivalent		Yes
Delete Namelist	DELETE NAMELIST	No equivalent		Yes
Inquire Namelist	DISPLAY NAMELIST	No equivalent		Yes
Inquire Namelist Names	DISPLAY NAMELIST	No equivalent		Yes

Process commands

ULW

A table of process commands, showing the PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

Table 46. Process commands

PCF command	MQSC command	Control command	V 9.0.1 REST API resource and HTTP method	IBM MQ Explorer equivalent?
Change Process	ALTER PROCESS	No equivalent		Yes
Copy Process	DEFINE PROCESS(x) LIKE(y)	No equivalent		Yes
Create Process	DEFINE PROCESS	No equivalent		Yes
Delete Process	DELETE PROCESS	No equivalent		Yes
Inquire Process	DISPLAY PROCESS	No equivalent		Yes
Inquire Process Names	DISPLAY PROCESS	No equivalent		Yes

Queue commands

ULW

A table of queue commands, showing the PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

Table 47. Queue commands

PCF command	MQSC command	Control command	V 9.0.1 REST API resource and HTTP method	IBM MQ Explorer equivalent?
Change Queue	ALTER QLOCAL ALTER QALIAS ALTER QMODEL ALTER QREMOTE	No equivalent	V 9.0.2 PATCH /admin/qmgr/{qmgrName}/queue	Yes
Clear Queue	CLEAR QLOCAL	No equivalent		Yes
Copy Queue	DEFINE QLOCAL(x) LIKE(y) DEFINE QALIAS(x) LIKE(y) DEFINE QMODEL(x) LIKE(y) DEFINE QREMOTE(x) LIKE(y)	No equivalent		Yes
Create Queue	DEFINE QLOCAL DEFINE QALIAS DEFINE QMODEL DEFINE QREMOTE	No equivalent	V 9.0.2 POST /admin/qmgr/{qmgrName}/queue	Yes
Delete Queue	DELETE QLOCAL DELETE QALIAS DELETE QMODEL DELETE QREMOTE	No equivalent	V 9.0.2 DELETE /admin/qmgr/{qmgrName}/queue	Yes
Inquire Queue	DISPLAY QUEUE	No equivalent	V 9.0.2 GET /admin/qmgr/{qmgrName}/queue	Yes
Inquire Queue Names	DISPLAY QUEUE	No equivalent		Yes
Inquire Queue Status	DISPLAY QSTATUS	No equivalent	V 9.0.2 GET /admin/qmgr/{qmgrName}/queue	Yes
Reset Queue Statistics	No equivalent	No equivalent		No

Service commands

ULW

A table of service commands, showing the PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

Table 48. Service commands

PCF command	MQSC command	Control command	V 9.0.1 REST API resource and HTTP method	IBM MQ Explorer equivalent?
Change Service	ALTER SERVICE	No equivalent		Yes
Copy Service	DEFINE SERVICE(x) LIKE(y)	No equivalent		Yes
Create Service	DEFINE SERVICE	No equivalent		Yes
Delete Service	DELETE SERVICE	No equivalent		Yes
Inquire Service	DISPLAY SERVICE	No equivalent		Yes
Inquire Service Status	DISPLAY SVSTATUS	No equivalent		Yes
Start Service	START SERVICE	No equivalent		Yes
Stop Service	STOP SERVICE	No equivalent		Yes

Other commands

ULW

A table of other commands, showing the command description, and its PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

Table 49. Other commands

Description	PCF command	MQSC command	Control command	V 9.0.1 REST API resource and HTTP method	IBM MQ Explorer equivalent?
Create conversion exit	No equivalent	No equivalent	**crtmqcvx**		No
Display files used by objects	No equivalent	No equivalent	**dspmqfls**		No
Display formatted trace	No equivalent	No equivalent	**dspmqtrc** [1]		No
Display version information	No equivalent	No equivalent	**dspmqver**		No
Display transactions	No equivalent	No equivalent	**dspmqtrn**		No
Dump log	No equivalent	No equivalent	**dmpmqlog**		No
Dump MQ Configuration	No equivalent	No equivalent	**dmpmqcfg**		No
End trace	No equivalent	No equivalent	**endmqtrc**		Yes

Table 49. Other commands (continued)

Description	PCF command	MQSC command	Control command	V 9.0.1 REST API resource and HTTP method	IBM MQ Explorer equivalent?
Escape	Escape	No equivalent	No equivalent	V 9.0.4 `POST /admin/action/qmgr/{qmgrName}/mqsc`	No
Record media image	No equivalent	No equivalent	**rcdmqimg**		No
Re-create media object	No equivalent	No equivalent	**rcrmqobj**		No
Resolve transactions	No equivalent	No equivalent	**rsvmqtrn**		No
Run client trigger monitor	No equivalent	No equivalent	**runmqtmc**		No
Run dead-letter queue handler	No equivalent	No equivalent	**runmqdlq**		No
Run MQSC commands	No equivalent	No equivalent	**runmqsc**		No
Run trigger monitor	No equivalent	No equivalent	**runmqtrm**		No
Set service connection points	No equivalent	No equivalent	**setmqscp** [2]		No
Start IBM MQ trace	No equivalent	No equivalent	**strmqtrc**		Yes
IBM MQ Services control	No equivalent	No equivalent	**amqmdain** [2]		No

Notes:

1. Not supported on IBM MQ for Windows.
2. Supported by IBM MQ for Windows only.

IBM MQ Control commands reference

Reference information about the IBM MQ control commands.

For information about running these commands, see Administration using the control commands.

Using control commands

There are three categories of control commands: queue manager commands, channel commands, and utility commands.

This information has moved. See Administration using the control commands.

Using control commands on Windows systems:

In IBM MQ for Windows, you enter control commands at a command prompt.

This information has moved. See Administration using the control commands.

Using control commands on UNIX and Linux systems:

In IBM MQ for UNIX and Linux systems, you enter control commands in a shell window.

This information has moved. See Administration using the control commands.

The control commands

This collection of topics provides reference information for each of the IBM MQ control commands.

This information has moved. See “IBM MQ Control commands reference” on page 195.

addmqinf

UNIX Windows

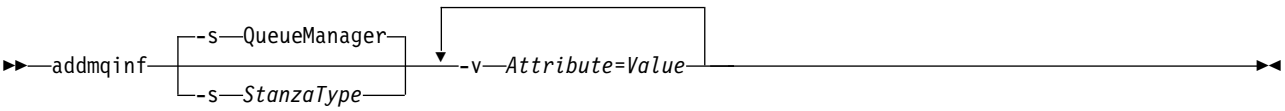
Add IBM MQ configuration information (UNIX and Windows only).

Purpose

Use the **addmqinf** command to add information to the IBM MQ configuration data.

For example, use **dspmqinf** and **addmqinf** to copy configuration data from the system where a queue manager was created, to other systems where the same multi-instance queue manager is also to be started.

Syntax



Required parameters

-v *Attribute* = *Value*

The name and value of the stanza attributes to be placed in the stanza specified in the command.

Table 50 on page 197 lists the QueueManager stanza attribute values. The queue manager stanza is the only stanza that is currently supported.

Table 50. QueueManager stanza attributes

Attribute	*Value*	**Required or optional**
Name	The name of the queue manager. You must provide a different name from any other queue manager stanza on the system.	Required
Prefix	The directory path under which this queue manager data directory is stored by default. You can use **Prefix** to modify the location of the queue manager data directories. The value of **Directory** is automatically appended to this path.	Required
Directory	The name of the queue manager data directory. Sometimes the name must be provided (as in "Example"), because it is different from the queue manager name. Copy the directory name from the value returned by **dspmqinf**. The rules for transforming queue manager names into directory names are described in Understanding IBM MQ file names.	Required
DataPath	The directory path where the queue manager data files are placed. The value of **Directory** is not automatically appended to this path and you must provide the transformed queue manager name as part of **DataPath**. UNIX If the **DataPath** attribute is omitted on UNIX, the queue manager data directory path is defined as ***Prefix*** / ***Directory***.	UNIX On UNIX: Optional Windows On Windows: Required

Optional parameters

-s *StanzaType*

A stanza of the type *StanzaType* is added to the IBM MQ configuration.

The default value of *StanzaType* is `QueueManager`.

The only supported value of *StanzaType* is `QueueManager`.

Return codes

Return code	Description
0	Successful operation
1	Queue manager location is invalid (either **Prefix** or **DataPath**)
39	Bad command-line parameters
45	Stanza already exists
46	Required configuration attribute is missing
58	Inconsistent use of installations detected
69	Storage is not available
71	Unexpected error
72	Queue manager name error
100	Log location is invalid

Example

```
addmqinf -v DataPath=/MQHA/qmgrs/QM!NAME +
  -v Prefix=/var/mqm  +
  -v Directory=QM!NAME  +
  -v Name=QM.NAME
```

Creates the following stanza in mqs.ini:

```
QueueManager:
   Name=QM.NAME
   Prefix=/var/mqm
   Directory=QM!NAME
   DataPath=/MQHA/qmgrs/QM!NAME
```

Usage notes

Use **dspmqinf** with **addmqinf** to create an instance of a multi-instance queue manager on a different server.

To use this command you must be an IBM MQ administrator and a member of the mqm group.

Related commands

Command	Description
"dspmqinf (display configuration information)" on page 245	Display IBM MQ configuration information
"rmvmqinf (remove configuration information)" on page 291	Remove IBM MQ configuration information

amqmdain (services control)

Windows

amqmdain is used to configure or control some Windows specific administrative tasks.

Purpose

The **amqmdain** command applies to IBM MQ for Windows only.

Use **amqmdain** to perform some Windows specific administrative tasks.

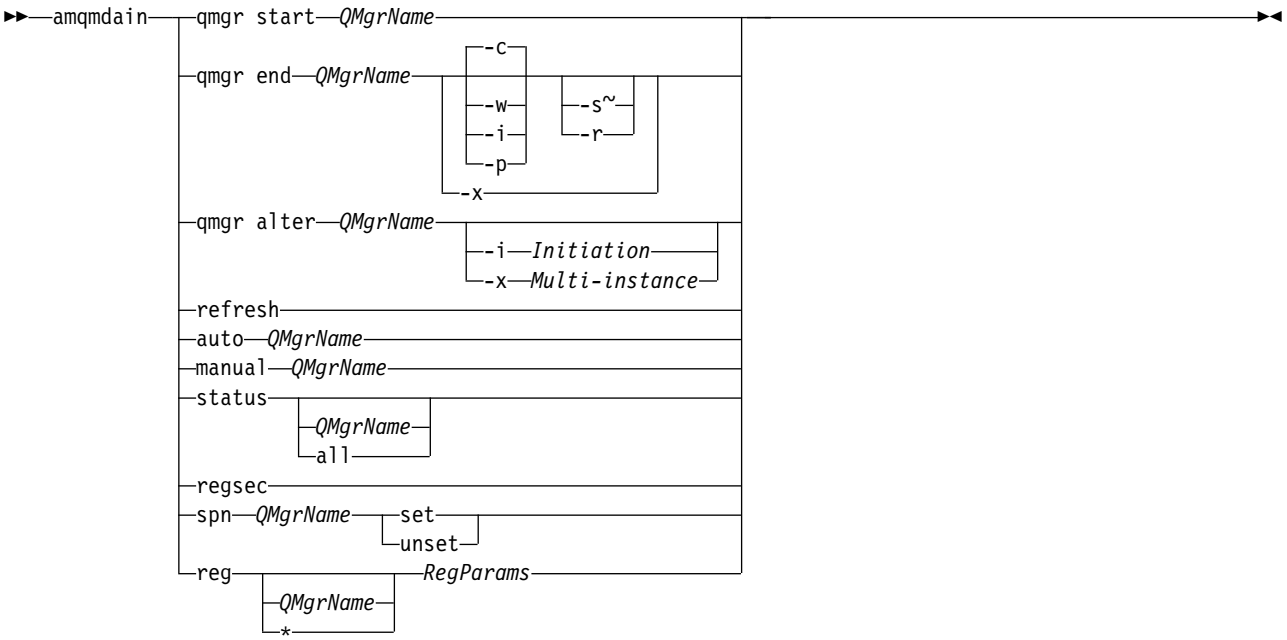
Starting a queue manager with **amqmdain** is equivalent to using the **strmqm** command with the option **-ss**. **amqmdain** makes the queue manager run in a non-interactive session under a different user account. However, to ensure that all queue manager startup feedback is returned to the command line, use the strmqm -ss command rather than **amqmdain**.

You must use the **amqmdain** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the dspmq -o installation command.

To administer and define IBM MQ service and listener objects, use MQSC commands, PCF commands, or the IBM MQ Explorer.

The **amqmdain** command has been updated to modify either the .ini files or the registry as appropriate.

Syntax



Keywords and parameters

All parameters are required unless the description states they are optional.

In every case, *QMgrName* is the name of the queue manager to which the command applies.

qmgr start *QMgrName*
Starts a queue manager.

This parameter can also be written in the form *start QMgrName*.

If you start your queue manager as a service and need the queue manager to continue to run after logoff, use `strmqm -ss qmgr` instead of `amqmdain start qmgr`.

qmgr end *QMgrName*
Ends a queue manager.

This parameter can also be written in the form **end QMgrName**.

For consistency across platforms, use `endmqm qmgr` instead of `amqmdain end qmgr`.

For fuller descriptions of the options, see "endmqm (end queue manager)" on page 269.

-c Controlled (or quiesced) shutdown.

-w Wait shutdown.

-i Immediate shut down.

-p Pre-emptive shut down.

-r Reconnect clients.

-s Switch over to a standby queue manager instance.

-x End the standby instance of the queue manager without ending the active instance.

qmgr alter *QMgrName*
Alters a queue manager.

-i *Initiation*
Specifies the initiation type. Possible values are:

Table 51. Initiation command parameters.

Value	Description
`auto`	Sets the queue manager to automatic startup (when the machine starts, or more precisely when the IBM MQ service starts). The syntax is: `amqmdain qmgr alter QmgrName -i auto`
`interactive`	Sets the queue manager to manual startup that then runs under the logged on (interactive) user. The syntax is: `amqmdain qmgr alter QmgrName -i interactive`
`service`	Sets the queue manager to manual startup that then runs as a service. The syntax is: `amqmdain qmgr alter QmgrName -i service`

-x *Multi-instance*
Specifies if `auto` queue manager start by the IBM MQ service permits multiple instances. Equivalent to the `-sax` option on the **crtmqm** command. Also specifies if the **amqmdain start** `qmgr` command permits standby instances. Possible values are:

Table 52. Multi-instance command parameters.

Value	Description
`set`	Sets automatic queue manager startup to permit multiple instances. Issues **strmqm -x** . The `set` option is ignored for queue managers that are initiated interactively or as a manual service startup. The syntax of the command is: `amqmdain qmgr alter QmgrName -x set`
`unset`	Sets automatic queue manager startup to single instance. Issues **strmqm**. The `unset` option is ignored for queue managers that are initiated interactively or as a manual service startup. The syntax of the command is: `amqmdain qmgr alter QmgrName -x unset`

refresh
Refreshes or checks the status of a queue manager. You will not see anything returned on the screen after executing this command.

auto *QMgrName*
Sets a queue manager to automatic startup.

manual *QMgrName*
Sets a queue manager to manual startup.

status *QMgrName*| all
These parameters are optional.

Table 53. Status command parameters.

Header	Header
If no parameter is supplied:	Displays the status of the IBM MQ services.
If a *QMgrName* is supplied:	Displays the status of the named queue manager.
If the parameter *all* is supplied:	Displays the status of the IBM MQ services and all queue managers.

regsec
Ensures that the security permissions assigned to the Registry keys containing installation information are correct.

spn *QMgrName* set | unset
You can set or unset the service principal name for a queue manager.

reg *QMgrName*| * *RegParams*
Parameters *QMgrName*, and * are optional.

Table 54. Reg command parameters.

Value	Description
If *RegParams* is specified alone:	Modifies queue manager configuration information related to the default queue manager.
If *QMgrName* and *RegParams* are specified:	Modifies queue manager configuration information related to the queue manager specified by *QMgrName*.
If * and *RegParams* are specified:	Modifies IBM MQ configuration information.

The parameter, *RegParams*, specifies the stanzas to change, and the changes that are to be made. *RegParams* takes one of the following forms:

- `-c add -s stanza -v attribute= value`
- `-c remove -s stanza -v [attribute|*]`
- `-c display -s stanza -v [attribute|*]`

If you are specifying queue manager configuration information, the valid values for *stanza* are:

```
XAResourceManager\name
ApiExitLocal\name
Channels
ExitPath
InstanceData
Log
QueueManagerStartup
TCP
LU62
SPX
NetBios
Connection
QMErrorLog
Broker

ExitPropertiesLocal
SSL
```

If you are modifying IBM MQ configuration information, the valid values for *stanza* are:

```
ApiExitCommon\name
ApiExitTemplate\name
ACPI
AllQueueManagers
Channels
DefaultQueueManager
LogDefaults
ExitProperties
```

Note the following usage considerations:

- **amqmdain** does not validate the values you specify for *name*, *attribute*, or *value*.
- When you specify add, and an attribute exists, it is modified.
- If a stanza does not exist, **amqmdain** creates it.
- When you specify remove, you can use the value * to remove all attributes.

- When you specify `display`, you can use the value `*` to display all attributes which have been defined. This value only displays the attributes which have been defined and not the complete list of valid attributes.
- If you use `remove` to delete the only attribute in a stanza, the stanza itself is deleted.
- Any modification you make to the Registry re-secures all IBM MQ Registry entries.

Examples

The following example adds an XAResourceManager to queue manager TEST. The commands issued are:

```
amqmdain reg TEST -c add -s XAResourceManager\Sample -v SwitchFile=sf1
amqmdain reg TEST -c add -s XAResourceManager\Sample -v ThreadOfControl=THREAD
amqmdain reg TEST -c add -s XAResourceManager\Sample -v XAOpenString=openit
amqmdain reg TEST -c add -s XAResourceManager\Sample -v XACloseString=closeit
```

To display the values set by the commands, use:

```
amqmdain reg TEST -c display -s XAResourceManager\Sample -v *
```

The display would look something like the following:

```
0784726, 5639-B43 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.
Displaying registry value for Queue Manager 'TEST'
        Attribute = Name, Value = Sample
        Attribute = SwitchFile, Value = sf1
        Attribute = ThreadOfControl, Value = THREAD
        Attribute = XAOpenString, Value = openit
        Attribute = XACloseString, Value = closeit
```

To remove the XAResourceManager from queue manager TEST, use:

```
amqmdain reg TEST -c remove -s XAResourceManager\Sample -v *
```

Return codes

Return code	Description
0	Command completed normally
-2	Syntax error
-3	Failed to initialize MFC
-6	Feature no longer supported
-7	Configuration failed
-9	Unexpected Registry error
-16	Failed to configure service principal name
-29	Inconsistent use of installations detected
62	The queue manager is associated with a different installation
71	Unexpected error
119 Windows	Permission denied (Windows only)

Note:

1. If the **qmgr start QMgrName** command is issued, all return codes that can be returned with **strmqm**, can be returned here also. For a list of these return codes, see “strmqm (start queue manager)” on page 351.
2. If the **qmgr end QMgrName** command is issued, all return codes that can be returned with **endmqm**, can be returned here also. For a list of these return codes, see “endmqm (end queue manager)” on page 269.

amqmfsck (file system check)

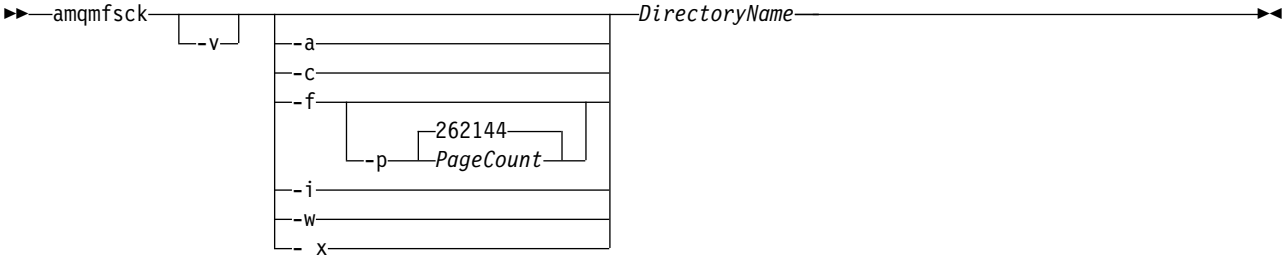
UNIX IBM i

amqmfsck checks whether a shared file system on UNIX and IBM i systems meets the requirements for storing the queue manager data of a multi-instance queue manager.

Purpose

The **amqmfsck** command applies only to UNIX and IBM i systems. You do not need to check the network drive on Windows. **amqmfsck** tests that a file system correctly handles concurrent writes to a file and the waiting for and releasing of locks.

Syntax



Required parameters

DirectoryName
The name of the directory to check.

Optional parameters

-a Perform the second phase of the data integrity test.

Run this on two machines at the same time. You must have formatted the test file using the **-f** option previously

-c Test writing to a file in the directory concurrently.

-f Perform the first phase of the data integrity test.

Formats a file in the directory in preparation for data integrity testing.

-i Perform the third phase of the data integrity test.

Checks the integrity of the file after the failure to discover whether the test worked.

-p Specifies the size of the test file used in the data integrity test in pages. .

The size is rounded up to the nearest multiple of 16 pages. The file is formatted with *PageCount* pages of 4 KB.

The optimum size of the file depends on the speed of the filesystem and the nature of the test you perform. If this parameter is omitted, the test file is 262144 pages, or 1 GB.

The size is automatically reduced so that the formatting completes in about 60 seconds even on a very slow filesystem.

-v Verbose output.

-w Test waiting for and releasing locks.

-x Deletes any files created by **amqmfsck** during the testing of the directory.

Do not use this option until you have completed the testing, or if you need to change the number of pages used in the integrity test.

Usage

You must be an IBM MQ Administrator to run the command. You must have read/write access to the directory being checked.

On IBM i, use QSH to run the program. There is no CL command.

The command returns an exit code of zero if the tests complete successfully.

The task, Verifying shared file system behavior, describes how to use **amqmfsck** to check the whether of a file system is suitable for multi-instance queue managers.

Interpreting your results

If the check fails, the file system is not capable of being used by IBM MQ queue managers. If the tests fail, choose verbose mode to help you to interpret the errors. The output from the `verbose` option helps you understand why the command failed, and if the problem can be solved by reconfiguring the file system.

Sometimes the failure might be an access control problem that can be fixed by changing directory ownership or permissions. Sometimes the failure can be fixed by reconfiguring the file system to behave in a different way. For example, some file systems have performance options that might need to be changed. It is also possible that the file system protocol does not support concurrency sufficiently robustly, and you must use a different file system. For example, you must use NFSv4 rather than NFSv3.

If the check succeeds, the command reports `The tests on the directory completed successfully`. If your environment is not listed as supported in the Testing and support statement for IBM MQ multi-instance queue managers, this result does not necessarily mean that you can run IBM MQ multi-instance queue managers successfully.

You must plan and run a variety of tests to satisfy yourself that you have covered all foreseeable circumstances. Some failures are intermittent, and there is a better chance of discovering them if you run the tests more than once.

Related information:

Verifying shared file system behavior

crtmqcvx (create data conversion code)

Create data conversion code from data type structures.

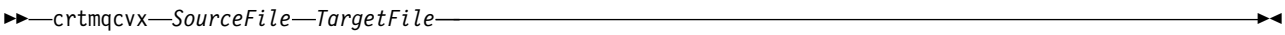
Purpose

Use the **crtmqcvx** command to create a fragment of code that performs data conversion on data type structures. The command generates a C function that can be used in an exit to convert C structures.

The command reads an input file containing structures to be converted, and writes an output file containing code fragments to convert those structures.

For information about using this command, see Utility for creating conversion-exit code.

Syntax



Required parameters

SourceFile
The input file containing the C structures to convert.

TargetFile
The output file containing the code fragments generated to convert the structures.

Return codes

Return code	Description
0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

The following example shows the results of using the data conversion command against a source C structure. The command issued is:

```
crtmqcvx source.tmp target.c
```

The input file, `source.tmp`, looks like this:

```
/* This is a test C structure which can be converted by the */
/* crtmqcvx utility                                         */

struct my_structure
{
    int    code;
    MQLONG value;
};
```

The output file, `target.c`, produced by the command, looks like this:

```
MQLONG Convertmy_structure(
              PMQDXP  pExitParms,
              PMQBYTE *in_cursor,
              PMQBYTE *out_cursor,
              PMQBYTE in_lastbyte,
              PMQBYTE out_lastbyte,
              MQHCONN hConn,
              MQLONG  opts,
              MQLONG  MsgEncoding,
              MQLONG  ReqEncoding,
              MQLONG  MsgCCSID,
              MQLONG  ReqCCSID,
              MQLONG  CompCode,
              MQLONG  Reason)
{
    MQLONG ReturnCode = MQRC_NONE;

    ConvertLong(1); /* code */

    AlignLong();
    ConvertLong(1); /* value */

Fail:
    return(ReturnCode);
}
```

You can use these code fragments in your applications to convert data structures. However, if you do so, the fragment uses macros supplied in the header file `amqsvmha.h`.

crtmqdir (create IBM MQ directories)

ULW V 9.0.3

Create, check, and correct IBM MQ directories and files.

Purpose

Use the **crtmqdir** command to check that the necessary directories and files used by IBM MQ exist and have the appropriate ownership and permissions. The command can optionally create any missing directories or files, and correct any inconsistent ownership or permissions.

The system-wide directories and files are created as part of the IBM MQ installation procedure. The tool can subsequently be run to check or ensure that the necessary IBM MQ directories and files continue to have appropriate ownership and permissions.

Important: You must have sufficient permission to determine whether the configuration is correct and, optionally, correct that configuration.

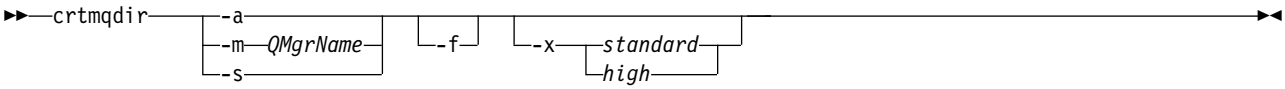
Windows On Windows, this typically means that you are a member of the IBM MQ administration group. This is necessary when using the **-a** or **-m** parameters.

UNIX On UNIX, this typically means that you are the `mqm` user. This is necessary when using the **-a** or **-m** parameters, together with the **-f** parameter.

Depending on the configuration, the **crtmqdir** command might require you to be an operating system administrator, or superuser.

Note: UNIX The security of *data path*`/log/`*qm*, on UNIX, is set to 2770.

Syntax



Required parameters

Specify one of the following parameters only:

-a Check all directories; that is, system-wide directories and all the queue managers.

Attention: The queue manager must be associated with the current installation.

-m Check directories for the specified queue manager name.

Attention: The queue manager must be associated with the current installation.

-s Check the system-wide directories; that is, directories that are not queue manager specific.

Optional parameters

-f

This option causes directories or files to be created if they are missing, and on UNIX only, ownership or permissions to be corrected if they are inappropriately set.

If **-a** or **-m** is specified on UNIX, as a minimum, the program attempts to correct ownership or permissions on files that were created at the time of queue manager creation.

-x *level of permissions*

Specify one of the following values only:

standard
By default, directories and files get a standard set of permissions, but a high level of permissions can be requested.

high This option ensures that files in the following directories on AIX, HP-UX, Linux, or Solaris, can be deleted only by the owner:

- errors
- trace
- webui

Return codes

Return code	Description
0	Successful completion
10	A warning occurred
20	An error occurred

Examples

- The following command checks and fixes the system-wide directories:
  ```
  crtmqdir -s -f
  ```
- The following command checks (but does not fix) queue manager QM1:
  ```
  crtmqdir -m Qm1
  ```

crtmqenv (create IBM MQ environment)

ULW

Create a list of environment variables for an installation of IBM MQ, on UNIX, Linux, and Windows.

Purpose

You can use the **crtmqenv** command to create a list of environment variables with the appropriate values for an installation of IBM MQ. The list of environment variables is displayed on the command line, and any variables that exist on the system have the IBM MQ values added to them. This command does not set the environment variables for you, but gives you the appropriate strings to set the variables yourself, for example, within your own scripts.

If you want the environment variables set for you in a shell environment, you can use the **setmqenv** command instead of using the **crtmqenv** command.

You can specify which installation the environment is created for by specifying a queue manager name, an installation name, or an installation path. You can also create the environment for the installation that issues the **crtmqenv** command by issuing the command with the **-s** parameter.

This command lists the following environment variables, and their values, appropriate to your system:

- CLASSPATH
- INCLUDE
- LIB
- MANPATH
- MQ_DATA_PATH
- MQ_ENV_MODE
- MQ_FILE_PATH
- MQ_JAVA_INSTALL_PATH
- MQ_JAVA_DATA_PATH
- MQ_JAVA_LIB_PATH
- MQ_JAVA_JVM_FLAG
- MQ_JRE_PATH
- PATH

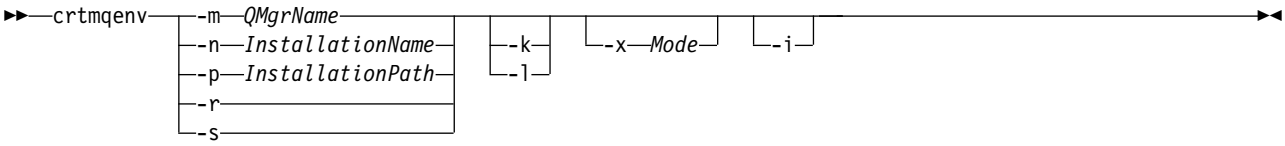
UNIX Linux On UNIX and Linux systems, if the **-l** or **-k** flag is specified, an environment variable is set as follows:

- The *LIBPATH* environment variable is set on AIX.
- The *LD_LIBRARY_PATH* environment variable is set on the following platforms:
  - HP-UX
  - Linux
  - Solaris

Usage notes

The **crtmqenv** command removes all directories for all IBM MQ installations from the environment variables before adding new references to the installation for which you are setting up the environment. Therefore, if you want to set any additional environment variables that reference IBM MQ, set the variables after issuing the **crtmqenv** command. For example, if you want to add *MQ_INSTALLATION_PATH*`/java/lib` to *LD_LIBRARY_PATH*, you must do so after running **crtmqenv**.

Syntax



Required Parameters

-m *QMgrName*
Create the environment for the installation associated with the queue manager *QMgrName*.

-n *InstallationName*
Create the environment for the installation named *InstallationName*.

-p *InstallationPath*
Create the environment for the installation in the path *InstallationPath*.

-r Remove all installations from the environment.

-s Create the environment for the installation that issued the command.

Optional Parameters

-k UNIX and Linux only.

Include the *LD_LIBRARY_PATH*, or *LIBPATH*, environment variable in the environment, adding the path to the IBM MQ libraries at the start of the current *LD_LIBRARY_PATH*, or *LIBPATH*, variable.

-l UNIX and Linux only.

Include the *LD_LIBRARY_PATH*, or *LIBPATH*, environment variable in the environment, adding the path to the IBM MQ libraries at the end of the current *LD_LIBRARY_PATH*, or *LIBPATH*, variable.

-x *Mode*
Mode can take the value 32, or 64.

Create a 32-bit or 64-bit environment. If this parameter is not specified, the environment matches that of the queue manager or installation specified in the command.

Any attempt to display a 64-bit environment with a 32-bit installation fails.

-i List only the additions to the environment.

When this parameter is specified, the environment variables set for previous installations remain in the environment variable path and must be manually removed.

Return codes

Return code	Description
0	Command completed normally.
10	Command completed with unexpected results.
20	An error occurred during processing.

Examples

The following examples assume that a copy of IBM MQ is installed in `/opt/mqm` on a UNIX or Linux system.

1. This command creates a list of environment variables for an installation installed in `/opt/mqm`:
   ```
   /opt/mqm/bin/crtmqenv -s
   ```

2. This command creates a list of environment variables for an installation installed in `/opt/mqm2`, and includes the path to the installation at the end of the current value of the *LD_LIBRARY_PATH* variable:

   ```
   /opt/mqm/bin/crtmqenv -p /opt/mqm2 -l
   ```

3. This command creates a list of environment variables for the queue manager `QM1`, in a 32-bit environment:

   ```
   /opt/mqm/bin/crtmqenv -m QM1 -x 32
   ```

The following example assumes that a copy of IBM MQ is installed in `C:\Program Files\IBM\MQ` on a Windows system.

1. This command creates a list of environment variables for an installation called `installation1`:

   ```
   "C:\Program Files\IBM\MQ\crtmqenv" -n installation1
   ```

Related reference:

"setmqenv (set IBM MQ environment)" on page 332
Use the **setmqenv** command to set up the IBM MQ environment on UNIX, Linux, and Windows.

Related information:

Choosing a primary installation

Multiple installations

crtmqinst (create IBM MQ installation)

UNIX Linux

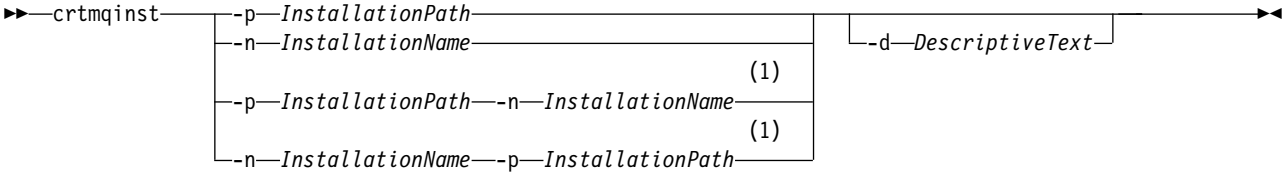
Create installation entries in `mqinst.ini` on UNIX and Linux systems.

Purpose

File `mqinst.ini` contains information about all IBM MQ installations on a system. For more information about `mqinst.ini`, see Installation configuration file, mqinst.ini.

The first IBM MQ installation is automatically given an installation name of `Installation1` because the **crtmqinst** command is not available until an installation of IBM MQ is on the system. Subsequent installations can have an installation name set before installation occurs, by using the **crtmqinst** command. The installation name cannot be changed after installation. For more information about installation names, see Choosing an installation name.

Syntax



Notes:

1 When specified together, the installation name and installation path must refer to the same installation.

Parameters

-d Text that describes the installation.

The text can be up to 64 single-byte characters, or 32 double-byte characters. The default value is all blanks. You must use quotation marks around the text if it contains spaces.

-n *InstallationName*
The name of the installation.

The name can contain up to 16 single-byte characters and must be a combination of alphabetic and numeric characters in the ranges a-z, A-Z, and 0-9. The installation name must be unique, regardless of whether uppercase or lowercase characters are used. For example, the names `INSTALLATIONNAME` and `InstallationName` are not unique. If you do not supply the installation name, the next available name in the series Installation1, Installation2... is used.

-p *InstallationPath*
The installation path. If you do not supply the installation path, `/opt/mqm` is used on UNIX and Linux systems, and `/usr/mqm` is used on AIX.

Return codes

Return code	Description
0	Entry created without error
10	Invalid installation level
36	Invalid arguments supplied
37	Descriptive text was in error
45	Entry already exists
59	Invalid installation specified
71	Unexpected error
89	.ini file error
96	Could not lock .ini file
98	Insufficient authority to access .ini file
131	Resource problem

Example

1. This command creates an entry with an installation name of `myInstallation`, an installation path of `/opt/myInstallation`, and a description "My IBM MQ installation":

```
crtmqinst -n MyInstallation -p /opt/myInstallation -d "My IBM MQ installation"
```

Quotation marks are needed because the descriptive text contains spaces.

Note: On UNIX, the **crtmqinst** command must be run by the root user because full access permissions are required to write to the mqinst.ini configuration file.

`crtmqm` (create queue manager)

Create a queue manager.

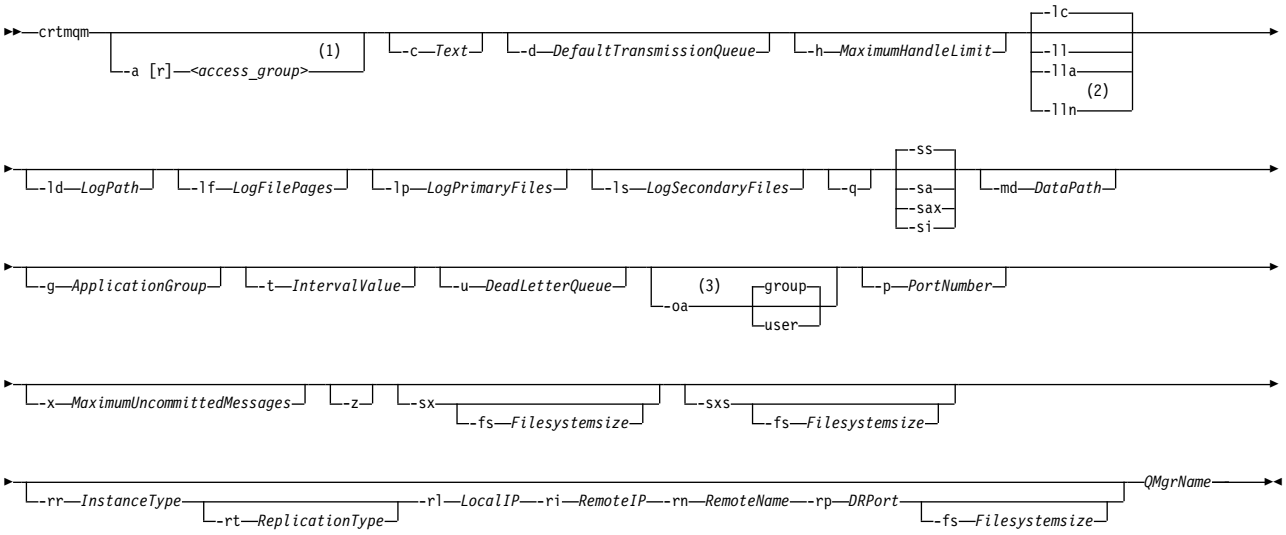
Purpose

Use the **crtmqm** command to create a queue manager and define the default and system objects. The objects created by the **crtmqm** command are listed in System and default objects. When you have created a queue manager, use the **strmqm** command to start it.

The queue manager is automatically associated with the installation from which the **crtmqm** command was issued. To change the associated installation, use the **setmqm** command.

Windows Note that the Windows installer does not automatically add the user that performs the installation to the `mqm` group. For more details, see Authority to administer IBM MQ on UNIX, Linux and Windows systems.

Syntax



Notes:

1 Windows only

2 Specify one only of lc, ll, lla, or lln

3 UNIX and Linux only

Required parameters

QMgrName
The name of the queue manager that you want to create. The name can contain up to 48 characters. This parameter must be the last item in the command.

Note: The *QMgrName* is used by IBM MQ applications, other IBM MQ queue managers, and IBM MQ control commands to identify this queue manager.

No other queue manager with the same name can exist on this machine. Where this queue manager is going to connect to other queue managers you must ensure that queue manager names are unique within that group of queue managers.

The *QMgrName* is also used to name the directories created on disk for the queue manager. Due to filesystem limitations the name of the directories created might not be identical to the *QMgrName* supplied on the **crtmqm** command.

In these cases the directories created will be based upon the supplied *QMgrName*, but might be modified, or have a suffix such as `.000` or `.001`, and so on, added to the queue manager name.

Optional parameters

Windows **-a[r]** ***access_group***
Use the access group parameter to specify a Windows security group, members of which will be granted full access to all queue manager data files. The group can either be a local or global group, depending on the syntax used.

Valid syntax for the group name is as follows:

LocalGroup

Domain name\GlobalGroup name

GlobalGroup name @ Domain name

You must define the additional access group before running the **crtmqm** command with the **-a [r]** option.

If you specify the group using **-ar** instead of **-a**, the local mqm group is not granted access to the queue manager data files. Use this option if the file system hosting the queue manager data files does not support access control entries for locally defined groups.

The group is typically a global security group, which is used to provide multi-instance queue managers with access to a shared queue manager data and logs folder. Use the additional security access group to set read and write permissions on the folder or to share containing queue manager data and log files.

The additional security access group is an alternative to using the local group named mqm to set permissions on the folder containing queue manager data and logs. Unlike the local group mqm, you can make the additional security access group a local or a global group. It must be a global group to set permissions on the shared folders that contain the data and log files used by multi-instance queue managers.

The Windows operating system checks the access permissions to read and write queue manager data and log files. It checks the permissions of the user ID that is running queue manager processes. The user ID that is checked depends on whether you started the queue manager as a service or you started it interactively. If you started the queue manager as a service, the user ID checked by the Windows system is the user ID you configured with the Prepare IBM MQ wizard. If you started the queue manager interactively, the user ID checked by the Windows system is the user ID that ran the **strmqm** command.

The user ID must be a member of the local mqm group to start the queue manager. If the user ID is a member of the additional security access group, the queue manager can read and write files that are given permissions by using the group.

Restriction: You can specify an additional security access group only on Windows operating system. If you specify an additional security access group on other operating systems, the **crtmqm** command returns an error.

-c ***Text***

Descriptive text for this queue manager. You can use up to 64 characters; the default is all blanks.

If you include special characters, enclose the description in single quotation marks. The maximum number of characters is reduced if the system is using a double-byte character set (DBCS).

-d ***DefaultTransmissionQueue***

The name of the local transmission queue where remote messages are put if a transmission queue is not explicitly defined for their destination. There is no default.

Linux UNIX **-g** ***ApplicationGroup***

On UNIX and Linux, the name of the group that contains members that are allowed to perform the following actions:

- Run MQI applications
- Update all IPCC resources
- Change the contents of some queue manager directories

The default value is **-g all**, which allows unrestricted access.

The **-g** *ApplicationGroup* value is recorded in the queue manager configuration file, qm.ini.

The mqm user ID and the user running the command must belong to the specified Application Group. For further details of the operation of restricted mode, see Restricted mode.

-h ***MaximumHandleLimit***

The maximum number of handles that an application can open at the same time.

Specify a value in the range 1 - 999999999. The default value is 256.

The next set of parameter descriptions relate to logging, which is described in Using the log for recovery.

Note: Choose the logging arrangements with care, because some cannot be changed after they are committed. The defaults for the logging options to **crtmqm** can be overridden by attributes in the mqs.ini file.

If you specify the logging attributes in the mqs.ini file, those attributes override the default values of the logging command line parameters to **crtmqm**.

-lc
Use circular logging. This method is the default logging method.

-ld ***LogPath***
The directory used to store log files. The default directory to store log paths is defined when you install IBM MQ.

If the volume containing the log file directory supports file security, the log file directory must have access permissions. The permissions allow the user IDs, under whose authority the queue manager runs, read and write access to the directory and its subdirectories. When you install IBM MQ, you grant permissions to the user IDs and to the `mqm` group on the default log directory. If you set the *LogPath* parameter to write the log file to a different directory, you must grant the user IDs permission to read and write to the directory. The user ID and permissions for UNIX and Linux are different from those for the Windows system:

Linux UNIX **UNIX and Linux**

The directory and its subdirectories must be owned by the user `mqm` in the group `mqm`.

If the log file is shared between different instances of the queue manager, the security identifiers (sid) that are used must be the same for the different instances. You must have set the user `mqm` to the same `sid` on the different servers running instances of the queue manager. Likewise for the group `mqm`.

Windows **Windows**

If the directory is accessed by only one instance of the queue manager, you must give read and write access permission to the directory for the following groups and users:

- The local group `mqm`
- The local group `Administrators`
- The `SYSTEM` user ID

To give different instances of a queue manager access to the shared log directory, the queue manager must access the log directory using a global user. Give the global group, which contains the global user, read and write access permission to the log directory. The global group is the additional security access group specified in the **-a** parameter.

Windows In IBM MQ for Windows systems, the default directory is `C:\ProgramData\IBM\MQ\log` (assuming that `C:` is your data drive). If the volume supports file security, the SYSTEM ID, Administrators, and `mqm` group must be granted read/write access to the directory.

Linux UNIX In IBM MQ for UNIX and Linux systems, the default directory is `/var/mqm/log`. User ID `mqm` and group `mqm` must have full authorities to the log files.

If you change the locations of these files, you must give these authorities yourself. If these authorities are set automatically, then the log files are in their default locations.

-lf ***LogFilePages***
The log data is held in a series of files called log files. The log file size is specified in units of 4 KB pages.

Linux UNIX In IBM MQ for UNIX and Linux systems, the default number of log file pages is 4096, giving a log file size of 16 MB. The minimum number of log file pages is 64 and the maximum is 65535.

Windows In IBM MQ for Windows systems, the default number of log file pages is 4096, giving a log file size of 16 MB. The minimum number of log file pages is 32 and the maximum is 65535.

Note: The size of the log files for a queue manager specified during creation of that queue manager cannot be changed.

-ll ***LinearLogging***
Use linear logging.

Multi V 9.0.2 On Multiplatforms, from IBM MQ Version 9.0.2, if you create a queue manager using the existing **-ll** option, you need to carry out manual management of log extents as previously (**LogManagement**=Manual).

Multi V 9.0.2 **-lla**
Use linear logging with automatic management of log extents (**LogManagement**=Automatic).

Multi V 9.0.2 **-lln**
Use linear logging with archive management of log extents (**LogManagement**=Archive).

-lp ***LogPrimaryFiles***
The log files allocated when the queue manager is created.

Windows On a Windows system:
- The minimum number of primary log files that you can have is 2 and the maximum is 254.
- The total number of primary and secondary log files must not exceed 255 and must not be less than 3.

Linux UNIX On UNIX and Linux systems:
- The minimum number of primary log files you can have is 2 and the maximum is 510. The default is 3.
- The total number of primary and secondary log files must not exceed 511 and must not be less than 3.

Operating system limits can reduce the maximum log size.

The value is examined when the queue manager is created or started. You can change it after the queue manager has been created. However, a change in the value is not effective until the queue manager is restarted, and the effect might not be immediate.

For more information about primary log files, see What logs look like.

To calculate the size of the primary log files, see Calculating the size of the log.

-ls ***LogSecondaryFiles***
The log files allocated when the primary files are exhausted.

Windows On a Windows system:
- The minimum number of secondary log files that you can have is 1 and the maximum is 253.
- The total number of primary and secondary log files must not exceed 255 and must not be less than 3.

Linux UNIX On UNIX and Linux systems:
- The minimum number of secondary log files that you can have is 2 and the maximum is 509. The default is 2.
- The total number of primary and secondary log files must not exceed 511 and must not be less than 3.

Operating system limits can reduce the maximum log size.

The value is examined when the queue manager is started. You can change this value, but changes do not become effective until the queue manager is restarted, and even then the effect might not be immediate.

For more information about the use of secondary log files, see What logs look like.

To calculate the size of the secondary log files, see Calculating the size of the log.

-md *DataPath*
The directory used to hold the data files for a queue manager.

Windows In IBM MQ for Windows systems, the default is `C:\ProgramData\IBM\MQ\qmgrs` (assuming that `C:` is your data drive). If the volume supports file security, the SYSTEM ID, Administrators, and `mqm` group must be granted read/write access to the directory.

Linux UNIX In IBM MQ for UNIX and Linux systems, the default is `/var/mqm/qmgrs`. User ID `mqm` and group `mqm` must have full authorities to the log files.

The **DataPath** parameter is provided to assist in the configuration of multi-instance queue managers. For example, on UNIX and Linux systems: if the `/var/mqm` directory is located on a local file system, use the **DataPath** parameter and the **LogPath** parameter to point to the shared file systems accessible to multiple queue managers.

Note: A queue manager created using **DataPath** parameter runs on versions of the product earlier than IBM WebSphere MQ Version 7.0.1, but the queue manager must be reconfigured to remove the **DataPath** parameter. You have two options to restore the queue manager to a pre-Version 7.0.1 configuration and run without the **DataPath** parameter: If you are confident about editing queue manager configurations, you can manually configure the queue manager using the `Prefix` queue manager configuration parameter. Alternatively, complete the following steps to edit the queue manager:

1. Stop the queue manager.
2. Save the queue manager data and log directories.
3. Delete the queue manager.
4. Backout IBM WebSphere MQ to the pre-Version 7.0.1 fix level.
5. Create the queue manager with the same name.
6. Replace the new queue manager data and log directories with the ones you saved.

-oa group|user
Linux UNIX On UNIX and Linux systems, you can specify whether group or user authorization is to be used. If you do not set this parameter, group authorization is used. You can change the authorization model later by setting the **SecurityPolicy** parameter in the Service stanza of the `qm.ini` file (see Service stanza format).

For further information, see Object authority manager (OAM).

-p *PortNumber*
Create a managed TCP listener on the specified port.

Specify a valid port value in the range 1-65535, to create a TCP listener object that uses the specified port. The new listener is called SYSTEM.LISTENER.TCP.1. This listener is under queue manager control, and is started and stopped along with the queue manager.

-q Makes this queue manager the default queue manager. The new queue manager replaces any existing default queue manager.

If you accidentally use this flag and you want to revert to an existing queue manager as the default queue manager, change the default queue manager as described in Making an existing queue manager the default.

Linux V 9.0.5 **-rr *InstanceType***

Create a disaster recover replicated data queue manager (DR RDQM). Specify **-rr p** to create the primary instance of the queue manager or specify **-rr s** to create the secondary instance.

Linux V 9.0.5 **-rt *ReplicationType***

Optionally specify whether your DR RDQM configuration uses synchronous or asynchronous replication. Specify **-rt s** for synchronous and **-rt a** for asynchronous. Asynchronous is the default.

Linux V 9.0.5 **-rl *LocalIP***

Specify the local IP address used for replication of data between primary and secondary instances of a DR RDQM.

Linux V 9.0.5 **-ri *RemoteIP***

Specify the remote IP address used for replication of data between primary and secondary instances of a DR RDQM.

Linux V 9.0.5 **-rn *RemoteName***

Specifies the name of the system that is hosting the other instance of the queue manager. The name is that value that is returned if you run `uname -n` on that server.

Linux V 9.0.5 **-rp *DRPort***

Specifies the port to use for DR replication.

Windows **-sa**

Automatic queue manager startup. For Windows systems only.

The queue manager is configured to start automatically when the IBM MQ Service starts.

This is the default option if you create a queue manager from IBM MQ Explorer.

Queue managers created in IBM WebSphere MQ releases earlier than Version 7 retain their existing startup type.

Windows **-sax**

Automatic queue manager startup, permitting multiple instances. For Windows systems only.

The queue manager is configured to start automatically when the IBM MQ Service starts.

If an instance of the queue manager is not already running the queue manager starts, the instance becomes active, and standby instances are permitted elsewhere. If a queue manager instance that permits standbys is already active on a different server, the new instance becomes a standby instance.

Only one instance of a queue manager can run on a server.

Queue managers created in versions of the product earlier than IBM WebSphere MQ Version 7.0.1 retain their existing startup type.

-si

Interactive (manual) queue manager startup.

The queue manager is configured to start only when you manually request startup by using the **strmqm** command. The queue manager runs under the (interactive) user when that user is logged-on. Queue managers configured with interactive startup end when the user who started them logs off.

-ss

Service (manual) queue manager startup.

A queue manager configured to start only when manually requested by using the **strmqm** command. The queue manager then runs as a child process of the service when the IBM MQ Service starts. Queue managers configured with service startup continue to run even after the interactive user has logged off.

This is the default option if you create a queue manager from the command line.

Linux V 9.0.4 **-sx [-fs *FilesystemSize*]**

Create a high availability replicated data queue manager (HA RDQM) on the primary node for that queue manager. RDQM is a high availability solution that is available on Linux only. See Creating an HA RDQM for more details about creating an RDQM.

Linux V 9.0.4 **-sxs [-fs *FilesystemSize*]**

Create a replicated data queue manager (RDQM) on a secondary node. RDQM is a high availability solution that is available on Linux only. See Creating an HA RDQM for more details about creating an RDQM.

-t *IntervalValue*

The trigger time interval in milliseconds for all queues controlled by this queue manager. This value specifies the length of time triggering is suspended, after the queue manager receives a trigger-generating message. That is, if the arrival of a message on a queue causes a trigger message to be put on the initiation queue, any message arriving on the same queue within the specified interval does not generate another trigger message.

You can use the trigger time interval to ensure that your application is allowed sufficient time to deal with a trigger condition before it is alerted to deal with another trigger condition on the same queue. You might choose to see all trigger events that happen; if so, set a low or zero value in this field.

Specify a value in the range 0 - 999999999. The default is 999999999 milliseconds; a time of more than 11 days. Allowing the default to be used effectively means that triggering is disabled after the first trigger message. However, an application can enable triggering again by servicing the queue using a command to alter the queue to reset the trigger attribute.

-u *DeadLetterQueue*

The name of the local queue that is to be used as the dead-letter (undelivered-message) queue. Messages are put on this queue if they cannot be routed to their correct destination.

The default is no dead-letter queue.

-x *MaximumUncommittedMessages*

The maximum number of uncommitted messages under any one sync point. The uncommitted messages are the sum of:

- The number of messages that can be retrieved from queues
- The number of messages that can be put on queues
- Any trigger messages generated within this unit of work

This limit does not apply to messages that are retrieved or put outside a sync point.

Specify a value in the range 1 - 999999999. The default value is 10000 uncommitted messages.

-z Suppresses error messages.

This flag is used within IBM MQ to suppress unwanted error messages. Do not use this flag when using a command line. Using this flag can result in a loss of information.

Return codes

Return code	Description
0	Queue manager created
8	Queue manager exists
39	Invalid parameter specified
49	Queue manager stopping
58	Inconsistent use of installations detected
69	Storage unavailable
70	Queue space unavailable
71	Unexpected error
72	Queue manager name error
74	The IBM MQ service is not started.
100	Log location invalid
111	Queue manager created. However, there was a problem processing the default queue manager definition in the product configuration file. The default queue manager specification might be incorrect.
115	Invalid log size
119	Windows Permission denied (Windows only)

Examples

- The following command creates a default queue manager called `Paint.queue.manager`, with a description of `Paint shop`, and creates the system and default objects. It also specifies that linear logging is to be used:

  ```
  crtmqm -c "Paint shop" -ll -q Paint.queue.manager
  ```

- The following command creates a default queue manager called `Paint.queue.manager`, creates the system and default objects, and requests two primary and three secondary log files:

  ```
  crtmqm -c "Paint shop" -ll -lp 2 -ls 3 -q Paint.queue.manager
  ```

- The following command creates a queue manager called `travel`, creates the system and default objects, sets the trigger interval to 5000 milliseconds (5 seconds), and specifies `SYSTEM.DEAD.LETTER.QUEUE` as its dead-letter queue.

  ```
  crtmqm -t 5000 -u SYSTEM.DEAD.LETTER.QUEUE travel
  ```

- Linux UNIX The following command creates a queue manager called `QM1` on UNIX and Linux systems, which has log and queue manager data folders in a common parent directory. The parent directory is to be shared on highly available networked storage to create a multi-instance queue manager. Before issuing the command, create other parameters `/MQHA`, `/MQHA/logs` and `/MQHA/qmgrs` owned by the user and group `mqm`, and with permissions `rwxrwxr-x`.

  ```
  crtmqm -ld /MQHA/logs -md /MQHA/qmgrs QM1
  ```

Related reference:

strmqm (start queue manager)
Start a queue manager or ready it for standby operation.

endmqm (end queue manager)
Stop a queue manager or switch to a standby queue manager.

dltmqm (delete queue manager)
Delete a queue manager.

setmqm (set the associated installation of a queue manager)
Set the associated installation of a queue manager.

Related information:

Working with dead-letter queues

dltmqinst (delete MQ installation)

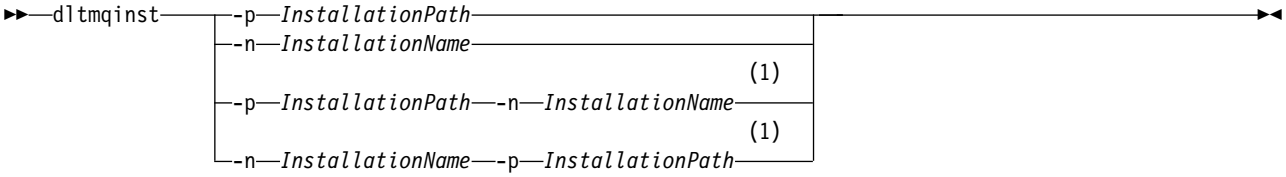
UNIX Linux

Delete installation entries from `mqinst.ini` on UNIX and Linux systems.

Purpose

File `mqinst.ini` contains information about all IBM MQ installations on a system. For more information about `mqinst.ini`, see Installation configuration file, mqinst.ini.

Syntax



Notes:

1 When specified together, the installation name and installation path must refer to the same installation.

Parameters

-n *InstallationName*
The name of the installation.

-p *InstallationPath*
The installation path is the location where IBM MQ is installed.

Return codes

Return code	Description
0	Entry deleted without error
5	Entry still active
36	Invalid arguments supplied
44	Entry does not exist
59	Invalid installation specified
71	Unexpected error
89	ini file error
96	Could not lock ini file

Return code	Description
98	Insufficient authority to access ini file
131	Resource problem

Example

1. This command deletes an entry with an installation name of `myInstallation`, and an installation path of `/opt/myInstallation`:

   ```
   dltmqinst -n MyInstallation -p /opt/myInstallation
   ```

Note: You can only use the **dltmqinst** command on another installation from the one it runs from. If you only have one IBM MQ installation, the command will not work.

Note: Solaris On a Solaris 10 MQ Client installation, only the root user has permissions to edit the `mqinst.ini` file.

dltmqm (delete queue manager)

Delete a queue manager.

Purpose

Use the **dltmqm** command to delete a specified queue manager and all objects associated with it. Before you can delete a queue manager, you must end it using the **endmqm** command.

You must use the **dltmqm** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

Windows On Windows, it is an error to delete a queue manager when queue manager files are open. If you get this error, close the files and reissue the command.

Syntax

Required parameters

QMgrName
The name of the queue manager to delete.

Optional parameters

-z Suppresses error messages.

Return codes

Return code	Description
0	Queue manager deleted
3	Queue manager being created
5	Queue manager running
16	Queue manager does not exist
24	A process that was using the previous instance of the queue manager has not yet disconnected.
25	An error occurred while creating or checking the directory structure for the queue manager.
26	Queue manager running as a standby instance.
27	Queue manager could not obtain data lock.
29	Queue manager deleted, however there was a problem removing it from Active Directory.
33	An error occurred while deleting the directory structure for the queue manager.
49	Queue manager stopping
58	Inconsistent use of installations detected
62	The queue manager is associated with a different installation
69	Storage not available
71	Unexpected error
72	Queue manager name error
74	The IBM MQ service is not started.
100	Log location invalid.
112	Queue manager deleted. However, there was a problem processing the default queue manager definition in the product configuration file. The default queue manager specification might be incorrect.
119	Windows Permission denied (Windows only).

Examples

1. The following command deletes the queue manager `saturn.queue.manager`.

   ```
   dltmqm saturn.queue.manager
   ```

2. The following command deletes the queue manager `travel` and also suppresses any messages caused by the command.

   ```
   dltmqm -z travel
   ```

Usage notes

Windows On Windows, it is an error to delete a queue manager when queue manager files are open. If you get this error, close the files and reissue the command.

Deleting a cluster queue manager does not remove it from the cluster. To check whether the queue manager you want to delete is part of a cluster, issue the command **DIS CLUSQMGR(*)**. Then check whether this queue manager is listed in the output. If it is listed as a cluster queue manager you must remove the queue manager from the cluster before deleting it. See the related link for instructions.

If you do delete a cluster queue manager without first removing it from the cluster, the cluster continues to regard the deleted queue manager as a member of the cluster for at least 30 days. You can remove it from the cluster using the command **RESET CLUSTER** on a full repository queue manager. Re-creating a queue manager with an identical name and then trying to remove that queue manager from the cluster does not result in the cluster queue manager being removed from the cluster. This is because the newly created queue manager, although having the same name, does not have the same queue manager ID (QMID). Therefore it is treated as a different queue manager by the cluster.

Related reference:

crtmqm (create queue manager)
Create a queue manager.

strmqm (start queue manager)
Start a queue manager or ready it for standby operation.

endmqm (end queue manager)
Stop a queue manager or switch to a standby queue manager.

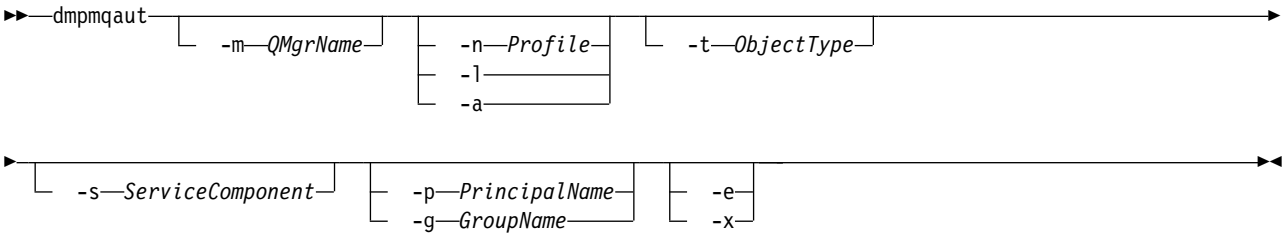
dmpmqaut (dump MQ authorizations)

Dump a list of current authorizations for a range of IBM MQ object types and profiles.

Purpose

Use the **dmpmqaut** command to dump the current authorizations to a specified object.

Syntax



Optional parameters

-m *QMgrName*
Dump authority records only for the queue manager specified. If you omit this parameter, only authority records for the default queue manager are dumped.

-n *Profile*
The name of the profile for which to dump authorizations. The profile name can be generic, using wildcard characters to specify a range of names as explained in Using OAM generic profiles on UNIX, Linux, and Windows systems.

-l Dump only the profile name and type. Use this option to generate a *terse* list of all defined profile names and types.

-a Generate set authority commands.

-t *ObjectType*
The type of object for which to dump authorizations. Possible values are:

Value	Description
authinfo	An authentication information object, for use with TLS channel security
channel or **chl**	A channel
clntconn or **clcn**	A client connection channel
listener or **lstr**	A listener
namelist or **nl**	A namelist
process or **prcs**	A process
queue or **q**	A queue or queues matching the object name parameter
qmgr	A queue manager

Value	Description
rqmname or **rqmn**	A remote queue manager name
service or **srvc**	A service
topic or **top**	A topic

-s *ServiceComponent*
If installable authorization services are supported, specifies the name of the authorization service for which to dump authorizations. This parameter is optional; if you omit it, the authorization inquiry is made to the first installable component for the service.

Windows **-p** *PrincipalName*
This parameter applies to Windows only; UNIX systems keep only group authority records.

The name of a user for whom to dump authorizations to the specified object. The name of the principal can optionally include a domain name, specified in the following format:

```
userid@domain
```

For more information about including domain names on the name of a principal, see Principals and groups.

-g *GroupName*
The name of the user group for which to dump authorizations. You can specify only one name, which must be the name of an existing user group.

Windows For IBM MQ for Windows only, the group name can optionally include a domain name, specified in the following formats:

```
GroupName@domain
domain\GroupName
```

-e Display all profiles used to calculate the cumulative authority that the entity has to the object specified in -n *Profile*. The variable *Profile* must not contain any wildcard characters.

The following parameters must also be specified:

- -m *QMgrName*
- -n *Profile*
- -t *ObjectType*

and either -p *PrincipalName*, or -g *GroupName*.

-x Display all profiles with the same name as specified in **-n** *Profile*. This option does not apply to the QMGR object, so a dump request of the form dmpmqaut -m QM -t QMGR ... -x is not valid.

Examples

The following examples show the use of **dmpmqaut** to dump authority records for generic profiles:

1. This example dumps all authority records with a profile that matches queue a.b.c for principal user1.

```
dmpmqaut -m qm1 -n a.b.c -t q -p user1
```

The resulting dump would look something like this:

```
profile:     a.b.*
object type: queue
entity:      user1
type:        principal
authority:   get, browse, put, inq
```

Note: UNIX On UNIX, you cannot use the -p option. You must use -g *groupname* instead.

2. This example dumps all authority records with a profile that matches queue a.b.c.

```
dmpmqaut -m qmgr1 -n a.b.c -t q
```

   The resulting dump would look something like this:

```
profile:     a.b.c
object type: queue
entity:      Administrator
type:        principal
authority:   all
- - - - - - - - - - - - - - - - -
profile:     a.b.*
object type: queue
entity:      user1
type:        principal
authority:   get, browse, put, inq
- - - - - - - - - - - - - - - - -
profile:     a.**
object type: queue
entity:      group1
type:        group
authority:   get
```

3. This example dumps all authority records for profile a.b.*, of type queue.

```
dmpmqaut -m qmgr1 -n a.b.* -t q
```

   The resulting dump would look something like this:

```
profile:     a.b.*
object type: queue
entity:      user1
type:        principal
authority:   get, browse, put, inq
```

4. This example dumps all authority records for queue manager qmX.

```
dmpmqaut -m qmX
```

   The resulting dump would look something like this:

```
profile:     q1
object type: queue
entity:      Administrator
type:        principal
authority:   all
- - - - - - - - - - - - - - - - -
profile:     q*
object type: queue
entity:      user1
type:        principal
authority:   get, browse
- - - - - - - - - - - - - - - - -
profile:     name.*
object type: namelist
entity:      user2
type:        principal
authority:   get
- - - - - - - - - - - - - - - - -
profile:     pr1
object type: process
entity:      group1
type:        group
authority:   get
```

5. This example dumps all profile names and object types for queue manager qmX.

```
dmpmqaut -m qmX -l
```

   The resulting dump would look something like this:

```
profile: q1, type: queue
profile: q*, type: queue
profile: name.*, type: namelist
profile: pr1, type: process
```

Note:

1. Windows For Windows only, all principals displayed include domain information, for example:

   ```
   profile:     a.b.*
   object type: queue
   entity:      user1@domain1
   type:        principal
   authority:   get, browse, put, inq
   ```

2. Each class of object has authority records for each group or principal. These records have the profile name `@CLASS` and track the `crt` (create) authority common to all objects of that class. If the `crt` authority for any object of that class is changed then this record is updated. For example:

   ```
   profile:     @class
   object type: queue
   entity:      test
   entity type: principal
   authority:   crt
   ```

   This shows that members of the group `test` have `crt` authority to the class `queue`.

3. Windows For Windows only, members of the "Administrators" group are by default given full authority. This authority, however, is given automatically by the OAM, and is not defined by the authority records. The **dmpmqaut** command displays authority defined only by the authority records. Unless an authority record has been explicitly defined, therefore, running the **dmpmqaut**command against the "Administrators" group displays no authority record for that group.

dmpmqcfg (dump queue manager configuration)

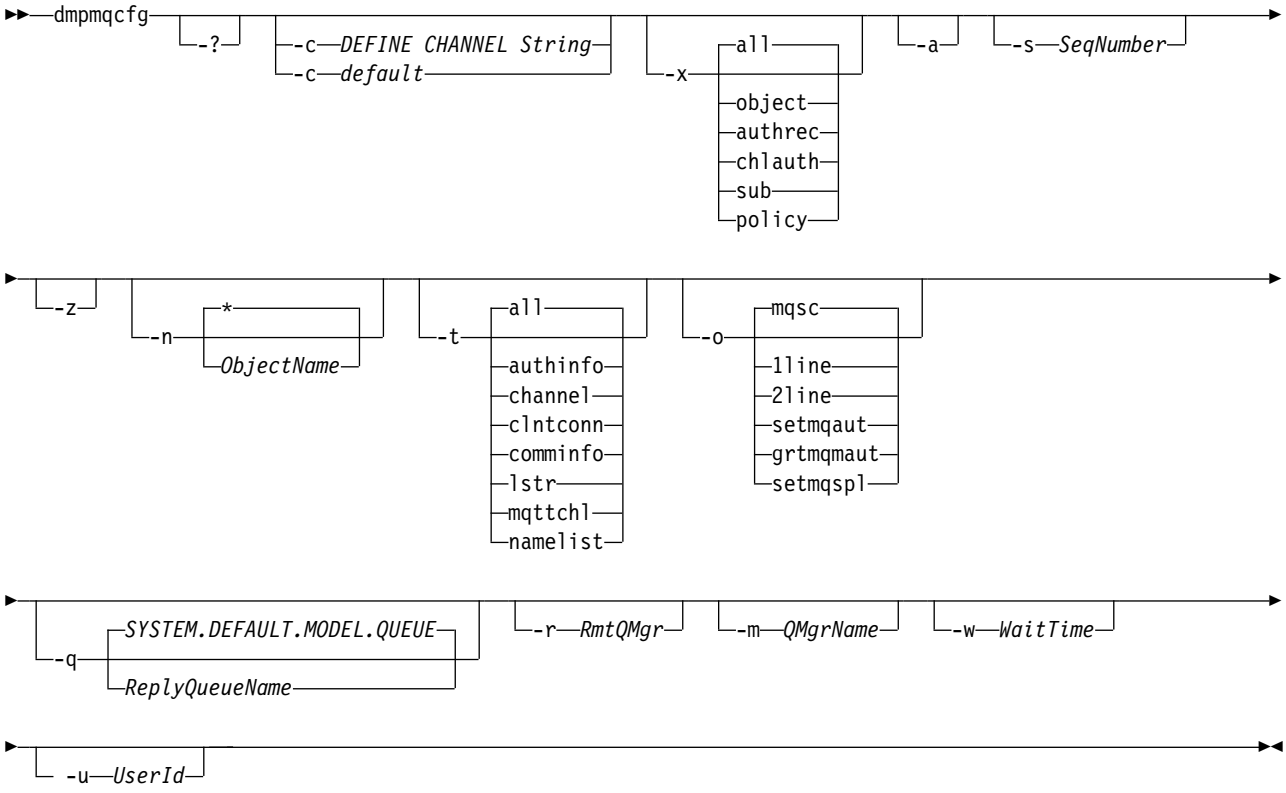
Use the **dmpmqcfg** command to dump the configuration of an IBM MQ queue manager.

Purpose

Use the dmpmqcfg command to dump the configuration of IBM MQ queue managers. If any default object has been edited, the -a option must be used if the dumped configuration will be used to restore the configuration.

The **dmpmqcfg** utility dumps only subscriptions of type MQSUBTYPE_ADMIN, that is, only subscriptions that are created using the MQSC command **DEFINE SUB** or its PCF equivalent. The output from **dmpmqcfg** is a **runmqsc** command to enable the administration subscription to be re-created. Subscriptions that are created by applications using the MQSUB MQI call of type MQSUBTYPE_API are not part of the queue manager configuration, even if durable, and so are not dumped by **dmpmqcfg**.MQTT channels will only be returned for types -t all and -t mqttchl if the telemetry (MQXR) service is running. For instructions on how to start the telemetry service, see Administering MQ Telemetry.

Note: The **dmpmqcfg** command does not make a backup of the Advanced Message Security policies. If you want to export the Advanced Message Security policies, ensure that you run **dspmqspl** with the ¬export flag. This command exports the policies for Advanced Message Security into a text file, which can be used for restoration purposes. For more information see "dspmqspl (display security policy)" on page 256.



Optional Parameters

-? Inquire the usage message for `dmpmqcfg`.

-c Force a client mode connection. If the **-c** parameter is qualified with the option `default`, the default client connection process is used. If **-c** is omitted, the default is to attempt to connect to the queue manager first by using server bindings and then if this fails by using client bindings.

If the option is qualified with an MQSC DEFINE CHANNEL CHLTYPE(CLNTCONN) string then this is parsed and if successful, used to create a temporary connection to the queue manager.

-x [all|object|authrec|chlauth|sub|policy]
Filter the definition procedure to show `object` definitions, `authority records`, `channel authentication` records, durable `subscriptions` or `policy`. The default value `all` is that all types are returned.

Note that when you specify an export type of `policy`, the security policies for the queue manager are reported in the configuration information dumped.

-a Return object definitions to show all attributes. The default is to return only attributes which differ from the defaults for the object type.

-s *SeqNumber*
Reset channel sequence number for sender, server and cluster sender channel types to the numeric value specified. The value `SeqNumber` must be in the range 1 - 999999999.

-z Activate silent mode in which warnings, such as those which appear when inquiring attributes from a queue manager of a higher command level are suppressed.

-n [*|ObjectName]
Filter the definitions produced by object or profile name, the object/profile name can contain a single asterisk. The `*` option can be placed only at the end of the entered filter string.

-t Choose a single type of object to export. Possible values are:

Value	Description
`all`	All object types
`authinfo`	An authentication information object
`channel` or `chl`	A channel
`comminfo`	A communications information object
`lstr` or `listener`	A listener
`mqttchl`	An MQTT channel
`namelist` or `nl`	A namelist
`process` or `prcs`	A process
`queue` or `q`	A queue
`qmgr`	A queue manager
`srvc` or `service`	A service
`topic` or `top`	A topic

-o [mqsc|1line|2line|setmqaut|grtmqmaut|setmqspl]
Possible values are:

Value	Description
`mqsc`	Multi-line MQSC that can be used as direct input to **runmqsc**
`1line`	MQSC with all attributes on a single line for line diffing
`2line`	MQSC with output on two lines. The first line is an MQSC command string and the second is a commented version with immutable values.
`setmqaut`	setmqaut statements for UNIX and Windows queue managers; valid only when `-x authrec` is specified
`grtmqmaut`	Linux only; generates iSeries syntax for granting access to the objects.
`setmqspl`	The security policies for the queue manager are reported in the format of **setmqspl** command lines. This format can be used to generate scripts to restore policy configuration to a queue manager. Note that the **setmqspl** command lines produced by this format includes parameters (-m) that specify the queue manager from which the definition was backed up. This implies that the definitions need to be replayed against the same queue manager. If you need to back up policy definitions from one queue manager, and restore them to a different queue manager, consider using the default MQSC format where the queue manager name is not explicitly specified.

-q The name of the reply-to queue used when getting configuration information.

-r The name of the remote queue manager/transmit queue when using queued mode. If this parameter is omitted the configuration for the directly connected queue manager (specified with the **-m** parameter) is dumped.

-m The name of the queue manager to connect to. If omitted the default queue manager name is used.

V 9.0.4 V 9.0.0.2 **-w *WaitTime***
The time, in seconds, that **dmpmqcfg** waits for replies to its commands.

Any replies received after a timeout are discarded, but the MQSC commands still run.

The check for timeout is performed once for each command reply.

Specify a time in the range 1 through 999999; the default value is 60 seconds.

Timed-out failure is indicated by:

- Nonzero return code to the calling shell or environment.
- Error message to `stdout` or `stderr`.

-u *UserId*
The ID of the user authorized to dump the configuration of queue managers.

Authorizations

The user must have MQZAO_OUTPUT (+put) authority to access the command input queue (SYSTEM.ADMIN.COMMAND.QUEUE) and MQZAO_DISPLAY (+dsp) authority to access the default model queue (SYSTEM.DEFAULT.MODEL.QUEUE), to be able to create a temporary dynamic queue if using the default reply queue.

The user must also have MQZAO_CONNECT (+connect) and MQZAO_INQUIRE (+inq) authority for the queue manager, and MQZAO_DISPLAY (+dsp) authority for every object that is requested.

Return code

If a failure occurs **dmpmqcfg** returns an error code. Otherwise, the command outputs a footer, an example of which follows:

```
********************************************************************************
* Script ended on 2016-01-05   at 05.10.09
* Number of Inquiry commands issued: 14
* Number of Inquiry commands completed: 14
* Number of Inquiry responses processed: 273
* QueueManager count: 1
* Queue count: 55
* NameList count: 3
* Process count: 1
* Channel count: 10
* AuthInfo count: 4
* Listener count: 1
* Service count: 1
* CommInfo count: 1
* Topic count: 5
* Subscription count: 1
* ChlAuthRec count: 3
* Policy count: 1
* AuthRec count: 186
* Number of objects/records: 273
********************************************************************************
```

Examples

To make these examples work you need to ensure that your system is set up for remote MQSC operation. See Preparing queue managers for remote administration and Preparing channels and transmission queues for remote administration.

```
dmpmqcfg -m MYQMGR -c "DEFINE CHANNEL(SYSTEM.ADMIN.SVRCONN) CHLTYPE(CLNTCONN)
CONNAME('myhost.mycorp.com(1414)')"
```

dumps all the configuration information from remote queue manager *MYQMGR* in MQSC format and creates an ad-hoc client connection to the queue manager using a client channel called *SYSTEM.ADMIN.SVRCONN*.

Note: You need to ensure that a server-connection channel with the same name exists.

```
dmpmqcfg -m LOCALQM -r MYQMGR
```

dumps all configuration information from remote queue manager *MYQMGR*, in MQSC format, connects initially to local queue manager *LOCALQM*, and sends inquiry messages through this local queue manager.

Note: You need to ensure that the local queue manager has a transmission queue named *MYQMGR*, with channel pairings defined in both directions, to send and receive replies between queue managers.

Related information:

Multi Backing up queue manager configuration

Multi Restoring queue manager configuration

dmpmqlog (dump MQ formatted log)

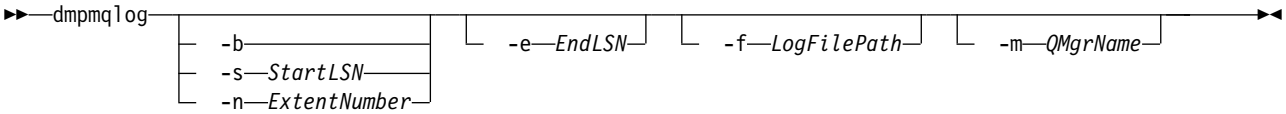
Display and format a portion of the IBM MQ system log.

Purpose

Use the `dmpmqlog` command to dump a formatted version of the IBM MQ system log to standard out.

The log to be dumped must have been created on the same type of operating system as that being used to issue the command.

Syntax



Optional parameters

Dump start point

Use one of the following parameters to specify the log sequence number (LSN) at which the dump should start. If you omit this, dumping starts by default from the LSN of the first record in the active portion of the log.

-b Start dumping from the base LSN. The base LSN identifies the start of the log extent that contains the start of the active portion of the log.

-s *StartLSN*
Start dumping from the specified LSN. The LSN is specified in the format `nnnn:nnnn:nnnn:nnnn`.

If you are using a circular log, the LSN value must be equal to or greater than the base LSN value of the log.

-n *ExtentNumber*
Start dumping from the specified extent number. The extent number must be in the range 0 - 9999999.

This parameter is valid only for queue managers using linear logging.

-e *EndLSN*
End dumping at the specified LSN. The LSN is specified in the format `nnnn:nnnn:nnnn:nnnn`.

-f *LogFilePath*
The absolute (rather than relative) directory path name to the log files. The specified directory must contain the log header file (`amqhlctl.lfh`) and a subdirectory called `active`. The active subdirectory must contain the log files. By default, log files are assumed to be in the directories specified in the IBM MQ configuration information. If you use this option, queue names associated with queue identifiers are shown in the dump only if you use the -m option to name a queue manager name that has the object catalog file in its directory path.

On a system that supports long file names this file is called qmqmobjcat and, to map the queue identifiers to queue names, it must be the file used when the log files were created. For example, for

a queue manager named qm1, the object catalog file is located in the directory `..\qmgrs\qm1\qmanager\`. To achieve this mapping, you might need to create a temporary queue manager, for example named tmpq, replace its object catalog with the one associated with the specific log files, and then start dmpmqlog, specifying `-m tmpq` and `-f` with the absolute directory path name to the log files.

-m *QMgrName*
The name of the queue manager. If you omit this parameter, the name of the default queue manager is used.

Note: Do not dump the log while the queue manager is running, and do not start the queue manager while `dmpmqlog` is running.

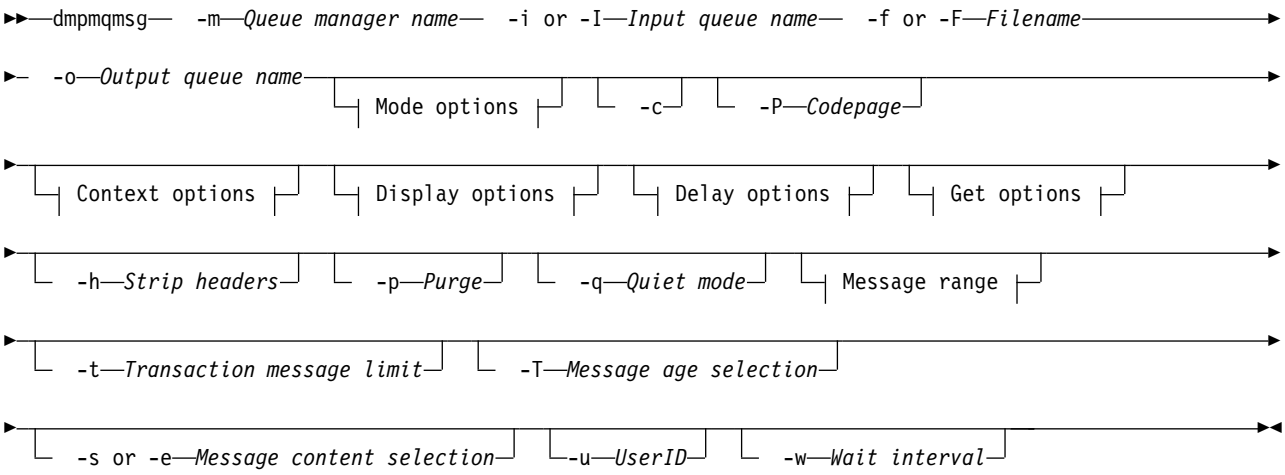
dmpmqmsg (queue load and unload)

Formerly the IBM MQ queue load and unload utility.

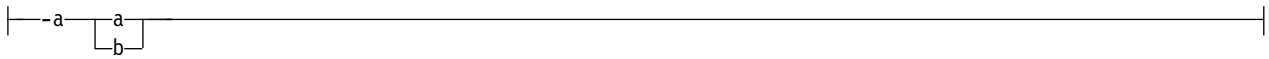
Purpose

Use the **dmpmqmsg** utility to copy or move the contents of a queue, or its messages, to a file.

Syntax



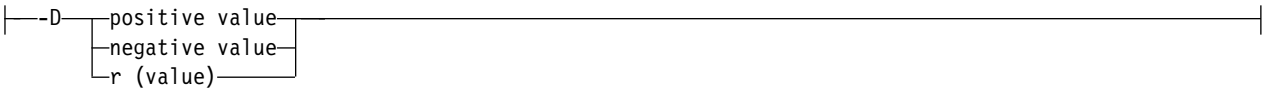
Mode options:



Context options:

Display options:

Delay options:



Get options:

Message range options:

Required parameters

-m *QueueManagerName*
The name of the queue manager on which the queue, or queues, exist.

-i or -I *Input queue name*
The name of the input queue.

Note: Using *-i* browses the queue, whereas using *-I* gets messages from the queue.

Optional parameters

-f or -F *Filename*
Specifies either the source or target file name.

Note: Using *-F* on a target file forces output to a file if it already exists. The program does not ask you if the file should be overwritten.

-o *Output queue name*
Specifies the name of the output queue.

-a Controls whether the file is opened in append or binary mode, by adding one of the following values to the keyword:

a Append mode

b Binary mode

-c Connect in client mode.

If you do not select this flag, the utility runs in local mode, which is the default.

z/OS This option is not available on z/OS.

-P Controls whether messages taken from a queue are converted.

Use the command
`-P CCSID [ : X 'Encoding' ]`

For example `-P850:111`

-C Controls the context option, by adding one of the following values to the keyword:

A Set all context. This is the default value.

I Set identity context.

a Pass all context.

p Pass identity context.

Use of the *pass* options is not applicable if the source messages are browsed on a queue.

d Default context.

n No context.

-d Controls the display option or options, by adding one or more of the following values to the keyword. For example `-dsCM`:

a Add ASCII columns to the hexadecimal output in the file to aid readability.

A Write ASCII lines of data wherever possible.

c Output *ApplicationOriginData* and *ApplicationIdentityData* as characters

C Display the *Correlation Identifier* in the queue summary.

H Do not write the file header.

Files created with this option are not loadable by the program as the program does not recognize the file format. However, if necessary you can use an editor to add an appropriate header manually, to make the file loadable.

i Include the message index in the output.

p Printable character output format.

This format is not code page safe. Loading a file written in this format, while running in a new code page does not guarantee to produce the same message.

s Write a simple summary of the messages found on input.

M Display the *Message Identifier* in the queue summary.

N Do not write out the message descriptor content, only the message payload.

t Text line output format.

This format is not code page safe. Loading a file written in this format, while running in a new code page does not guarantee to produce the same message.

T Display the time the message has been on the queue.

w *Length*
Set the data width for the output.

-D Add a delay, expressed in milliseconds, before writing a message to the output destination, by adding one of the following values to the keyword. For example:

-D*positive_value*
Add a fixed delay before putting a message. For example, `-D500` puts each message half a second apart.

-D*negative_value*
Add a random delay, up to the specified value before putting a message. For example, `-D-10000` adds a random delay of up to 10 seconds before putting a message.

r*value* Replays the messages at a percentage of their original put speed. For example:

r Replays messages at their original speed.

r50 Replays messages at half their original speed.

r200 Replays messages at twice their original speed.

-g Filter by Message identifier, Correlation identifier, or Group identifier, by adding one of the following values to the keyword.

c*value* Get by character Correlation identifier.

m*value*
Get by character Message identifier.

g*value* Get by character Group identifier.

xc*value*
Get by hexadecimal Correlation identifier.

xm*value*
Get by hexadecimal Message identifier.

xg*value*
Get by hexadecimal Group identifier.

-h Strip headers.

Any Dead Letter Queue header (MQDLH) or Transmission Queue header (MQXQH) is removed from the message before the message is written.

-o Output queue name.

-p Causes the source queue to be purged of messages as they are copied to the target destination.

-q Sets quiet mode. When set, the program does not output its usual summary of activity.

-r Sets the applicable message range by adding one of the following values to the keyword.

x Only message *x*. For example, `-r 10`.

x..y From message *x* to message *y*. For example, `-r 10..20`.

x#y Output *y* messages, starting at message *x*. For example, `-r 100#10`.

#x Output the first *x* messages. For example, `-r #100`.

-t Set the transaction message limit. If the optional **n** flag is not set, all the messages are done in a single transaction.

n The message operations are split into groups of n messages. For example `-t1000` deals with 1000 messages in a single transaction.

-T Allows message selection based on message age.

See "Using message age" for information on selection using message age.

-s or -e
Allows message selection based on message content.

ULW On ASCII platforms (UNIX, Linux, and Windows) use the **-s** option to search for a natively encoded string.

z/OS On EBCDIC platforms (z/OS) use the **-e** option to search for a natively encoded string.

See "Using message content" on page 236 for information on selection using message content.

V 9.0.5 **-u**
If you use the -u parameter to supply a user ID, you are prompted for a matching password.

If you have configured the CONNAUTH AUTHINFO record with CHCKLOCL(REQUIRED) or CHCKLOCL(REQDADM), you must use the -u parameter otherwise you will not be able to copy or move the contents of a queue.

If you specify this parameter and redirect stdin, a prompt will not be displayed and the first line of redirected input should contain the password.

-w Wait interval, in seconds, for consuming messages. If specified the program waits for messages to arrive, for the specified period, before ending.

See Examples of using the **dmpmqmsg** utility for examples on using the utility.

Message selection:

Message selection can be based on message age or message content.

Using message age

You can choose to process only messages older than a certain time interval using the `-T` flag.

Time interval can be specified in Days, Hours and Minutes. The general format being `[days:]hours:]minutes`.

The parameter can take one or two times, `-T [OlderThanTime][,YoungerThanTime]`.

For example:

- Display messages older than five minutes
  ```
  dmpmqmsg -m QM1 -i Q1 -fstdout -T5
  ```
- Display messages younger than five minutes
  ```
  dmpmqmsg -m QM1 -i Q1 -fstdout -T,5
  ```
- Display messages older than one day but younger than two days.
  ```
  dmpmqmsg -m QM1 -i Q1 -fstdout -T1440,2880
  ```
- The following command copies messages older than one hour from Q1 to Q2.
  ```
  dmpmqmsg -m QM1 -i Q1 -o Q2 -T1:0
  ```
- The following command moves messages older than one week from Q1 to Q2
  ```
  dmpmqmsg -m QM1 -I Q1 -o Q2 -T7:0:0
  ```

Using message content

You can specify a maximum of three of each search string. If multiple strings are used, they are treated as follows:

Positive search strings
When multiple positive strings are used, then all the strings must be present for the search to match. For example, the command

```
dmpmqmsg -iMATCH -s LIVERPOOL -s CHELSEA
```

only returns messages that contain both strings.

Negative search strings
When multiple negative strings are used, then none of the strings must be present for the search to match. For example, the command

```
dmpmqmsg -IMATCH -S HOME -S DRAW
```

only returns messages that contain neither string.

dspmq (display queue managers)

Multi

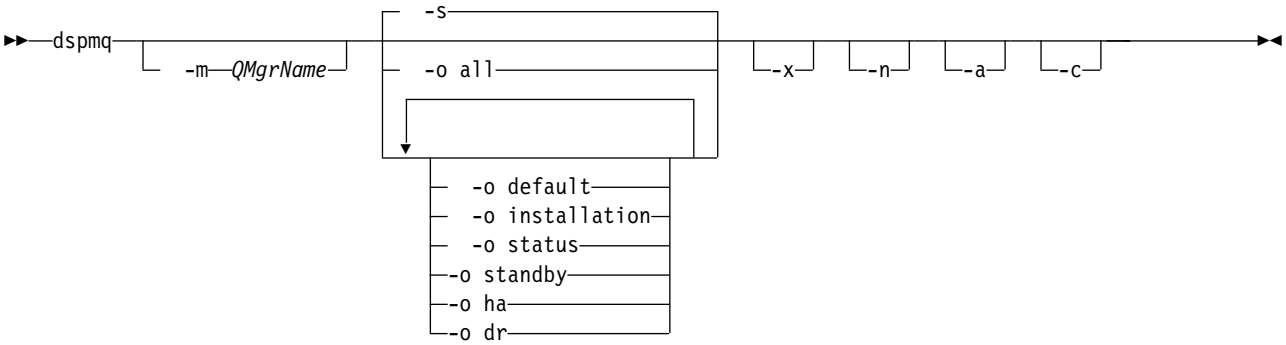
Display information about queue managers on Multiplatforms.

Purpose

Use the dspmq command to display names and details of the queue managers on a system.

z/OS V 9.0.1 The equivalent utility to dspmq on z/OS is CSQUDSPM.

Syntax



Required parameters

None

Optional parameters

-a Displays information about the active queue managers only.

A queue manager is active if it is associated with the installation from which the **dspmq** command was issued and one or more of the following statements are true:

- The queue manager is running

- A listener for the queue manager is running
- A process is connected to the queue manager

-m ***QMgrName***
The queue manager for which to display details. If you give no name, all queue manager names are displayed.

-n Suppresses translation of output strings.

-s The operational status of the queue managers is displayed. This parameter is the default status setting.

The parameter *-o status* is equivalent to *-s*.

-o all
The operational status of the queue managers is displayed, and whether any are the default queue manager.

ULW On UNIX, Linux, and Windows, the installation name (INSTNAME), installation path (INSTPATH), and installation version (INSTVER) of the installation that the queue manager is associated with is also displayed.

-o default
Displays whether any of the queue managers are the default queue manager.

ULW **-o installation**
UNIX, Linux, and Windows only.

Displays the installation name (INSTNAME), installation path (INSTPATH), and installation version (INSTVER) of the installation that the queue manager is associated with.

-o status
The operational status of the queue managers is displayed.

-o standby
Displays whether a queue manager currently permits starting a standby instance. The possible values are shown in Table 55.

Table 55. Standby values

Value	Description
Permitted	The queue manager is running and is permitting standby instances.
Not permitted	The queue manager is running and is not permitting standby instances.
Not applicable	The queue manager is not running. You can start the queue manager and this instance becomes active if it starts successfully.

V 9.0.4 **-o ha | HA**
Indicates whether a queue manager is an HA RDQM (high availability replicated data queue manager) or not. If the queue manager is an HA RDQM, `HA(Replicated)` is displayed. Otherwise, `HA()` is displayed. For example:

```
dspmq ha

QMNAME(RDQM8)                                 HA(Replicated)
QMNAME(RDQM9)                                 HA(Replicated)
QMNAME(RDQM7)                                 HA(Replicated)
QMNAME(QM7)                                   HA()
```

V 9.0.5 **-o dr | DR**
Indicates whether a queue manager is a DR RDQM (disaster recovery replicated data queue manager) or not. One of the following responses is displayed: One of the following responses is displayed:

dr() Indicates that the queue manager is not configured for disaster recovery.

DR(Primary)
Indicates that the queue manager is configured as the DR primary.

DR(Secondary)
Indicates that the queue manager is configured as the DR secondary.

-x Information about queue manager instances are displayed. The possible values are shown in Table 56.

Table 56. Instance values

Value	Description
Active	The instance is the active instance.
Standby	The instance is a standby instance.

-c Shows the list of processes currently connected to the IPCC, QMGR, and PERSISTENT sub pools for a queue manager.

For example, this list typically includes:

- Queue manager processes
- Applications, including those that are inhibiting shutdown
- Listeners

Queue manager states

The different states that a queue manager can be in are as follows:

- Starting
- Running
- Running as standby
- Running elsewhere
- Quiescing
- Ending immediately
- Ending pre-emptively
- Ended normally
- Ended immediately
- Ended unexpectedly
- Ended pre-emptively
- Status not available

Return codes

Return code	Description
0	Command completed normally
36	Invalid arguments supplied
58	Inconsistent use of installations detected
71	Unexpected error
72	Queue manager name error

Examples

1. The following command displays queue managers on this server:

   ```
   dspmq -o all
   ```

2. The following command displays standby information for queue managers on this server that have ended immediately:

   ```
   dspmq -o standby
   ```

3. The following command displays standby information and instance information for queue managers on this server:

```
dspmq -o standby -x
```

dspmqaut (display object authorization)

dspmqaut displays the authorizations of a specific IBM MQ object.

Purpose

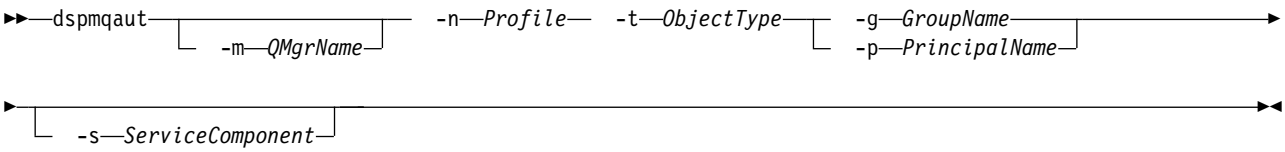
Use the **dspmqaut** command to display the current authorizations to a specified object.

If a user ID is a member of more than one group, this command displays the combined authorizations of all the groups.

Only one group or principal can be specified.

For more information about authorization service components, see Installable services, Service components, and Authorization service interface.

Syntax



Required parameters

-n *Profile*
The name of the profile for which to display authorizations. The authorizations apply to all IBM MQ objects with names that match the profile name specified.

This parameter is required, unless you are displaying the authorizations of a queue manager. In this case you must not include it and instead specify the queue manager name using the **-m** parameter.

-t *ObjectType*
The type of object on which to make the inquiry. Possible values are:

Table 57. The object type on which to make the inquiry.

Object Type	Description
authinfo	An authentication information object, for use with TLS channel security
channel or **chl**	A channel
clntconn or **clcn**	A client connection channel
listener or **lstr**	A Listener
namelist or **nl**	A namelist
process or **prcs**	A process
queue or **q**	A queue or queues matching the object name parameter
qmgr	A queue manager
rqmname or **rqmn**	A remote queue manager name
service or **srvc**	A service
topic or **top**	A topic

Optional parameters

-m *QMgrName*
The name of the queue manager on which to make the inquiry. This parameter is optional if you are displaying the authorizations of your default queue manager.

-g *GroupName*
The name of the user group on which to make the inquiry. You can specify only one name, which must be the name of an existing user group.

Windows For IBM MQ for Windows only, the group name can optionally include a domain name, specified in the following formats:

```
GroupName@domain
domain\GroupName
```

-p *PrincipalName*
The name of a user for whom to display authorizations to the specified object.

Windows For IBM MQ for Windows only, the name of the principal can optionally include a domain name, specified in the following format:

```
userid@domain
```

For more information about including domain names on the name of a principal, see Principals and groups.

-s *ServiceComponent*
If installable authorization services are supported, specifies the name of the authorization service to which the authorizations apply. This parameter is optional; if you omit it, the authorization inquiry is made to the first installable component for the service.

Returned parameters

Returns an authorization list, which can contain none, one, or more authorization values. Each authorization value returned means that any user ID in the specified group or principal has the authority to perform the operation defined by that value.

Table 58 shows the authorities that can be given to the different object types.

Table 58. Specifying authorities for different object types

Authority	Queue	Process	Queue manager	Remote queue manager name	Namelist	Topic	Auth info	Clntconn	Channel	Listener	Service
all	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
alladm	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
allmqi	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No
none	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
altusr	No	No	Yes	No	No	No	No	No	No	No	No
browse	Yes	No	No	No	No	No	No	No	No	No	No
chg	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
clr	Yes	No	No	No	No	Yes	No	No	No	No	No
connect	No	No	Yes	No	No	No	No	No	No	No	No
crt	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 58. Specifying authorities for different object types (continued)

Authority	Queue	Process	Queue manager	Remote queue manager name	Namelist	Topic	Auth info	Clntconn	Channel	Listener	Service
ctrl	No	No	No	No	No	Yes	No	No	Yes	Yes	Yes
ctrlx	No	No	No	No	No	No	No	No	Yes	No	No
dlt	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
dsp	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
get	Yes	No	No	No	No	No	No	No	No	No	No
pub	No	No	No	No	No	Yes	No	No	No	No	No
put	Yes	No	No	Yes	No	Yes	No	No	No	No	No
inq	Yes	Yes	Yes	No	Yes	No	Yes	No	No	No	No
passall	Yes	No	No	No	No	Yes	No	No	No	No	No
passid	Yes	No	No	No	No	Yes	No	No	No	No	No
resume	No	No	No	No	No	Yes	No	No	No	No	No
set	Yes	Yes	Yes	No	No	No	No	No	No	No	No
setall	Yes	No	Yes	No	No	Yes	No	No	No	No	No
setid	Yes	No	Yes	No	No	Yes	No	No	No	No	No
sub	No	No	No	No	No	Yes	No	No	No	No	No
system	No	No	Yes	No	No	No	No	No	No	No	No

The following list defines the authorizations associated with each value:

Table 59. Authorizations associated with each value.

Value	Description
all	Use all operations relevant to the object. `all` authority is equivalent to the union of the authorities `alladm`, `allmqi`, and `system` appropriate to the object type.
alladm	Perform all administration operations relevant to the object
allmqi	Use all MQI calls relevant to the object
altusr	Specify an alternative user ID on an MQI call
browse	Retrieve a message from a queue by issuing an `MQGET` call with the BROWSE option
chg	Change the attributes of the specified object, using the appropriate command set
clr	Clear a queue (PCF command Clear queue only) or a topic
ctrl	Start, and stop the specified channel, listener, or service, and ping the specified channel.
ctrlx	Reset or resolve the specified channel
connect	Connect the application to the specified queue manager by issuing an `MQCONN` call
crt	Create objects of the specified type using the appropriate command set

Table 59. Authorizations associated with each value. (continued)

Value	Description
dlt	Delete the specified object using the appropriate command set
dsp	Display the attributes of the specified object using the appropriate command set
get	Retrieve a message from a queue by issuing an MQGET call
inq	Make an inquiry on a specific queue by issuing an MQINQ call
passall	Pass all context
passid	Pass the identity context
pub	Publish a message on a topic using the MQPUT call.
put	Put a message on a specific queue by issuing an MQPUT call
resume	Resume a subscription using the MQSUB call.
set	Set attributes on a queue from the MQI by issuing an MQSET call
setall	Set all context
setid	Set the identity context
sub	Create, alter, or resume a subscription to a topic using the MQSUB call.
system	Use queue manager for internal system operations

The authorizations for administration operations, where supported, apply to these command sets:

- Control commands
- MQSC commands
- PCF commands

Return codes

Return code	Description
0	Successful operation
26	Queue manager running as a standby instance.
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
58	Inconsistent use of installations detected
69	Storage not available
71	Unexpected error
72	Queue manager name error
133	Unknown object name
145	Unexpected object name
146	Object name missing
147	Object type missing
148	Invalid object type
149	Entity name missing

Examples

- The following example shows a command to display the authorizations on queue manager `saturn.queue.manager` associated with user group `staff`:

  ```
  dspmqaut -m saturn.queue.manager -t qmgr -g staff
  ```

  The results from this command are:

  ```
  Entity staff has the following authorizations for object:
          get
          browse
          put
          inq
          set
          connect
          altusr
          passid
          passall
          setid
  ```

- The following example displays the authorities user1 has for queue a.b.c:

  ```
  dspmqaut -m qmgr1 -n a.b.c -t q -p user1
  ```

  The results from this command are:

  ```
  Entity user1 has the following authorizations for object:
          get
          put
  ```

dspmqcsv (display command server)

The status of a command server is displayed

Purpose

Use the **dspmqcsv** command to display the status of the command server for the specified queue manager.

The status can be one of the following:

- Starting
- Running
- Running with SYSTEM.ADMIN.COMMAND.QUEUE not enabled for gets
- Ending
- Stopped

You must use the **dspmqcsv** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

Syntax

Required parameters

None

Optional parameters

QMgrName
The name of the local queue manager for which the command server status is being requested.

Return codes

Return code	Description
0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

The following command displays the status of the command server associated with `venus.q.mgr`:

```
dspmqcsv venus.q.mgr
```

Related commands

Command	Description
strmqcsv	Start a command server
endmqcsv	End a command server

Related reference:

"Command server commands" on page 188
A table of command server commands, showing the PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

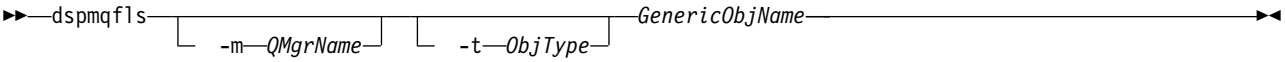
dspmqfls (display file names)

Display the file names corresponding to IBM MQ objects.

Purpose

Use the `dspmqfls` command to display the real file system name for all IBM MQ objects that match a specified criterion. You can use this command to identify the files associated with a particular object. This command is useful for backing up specific objects. See Understanding IBM MQ file names for information about name transformation.

Syntax



Required parameters

GenericObjName
The name of the object. The name is a string with no flag and is a required parameter. Omitting the name returns an error.

This parameter supports an asterisk (*) as a wildcard at the end of the string.

Optional parameters

-m *QMgrName*
The name of the queue manager for which to examine files. If you omit this name, the command operates on the default queue manager.

-t *ObjType*
The object type. The following list shows the valid object types. The abbreviated name is shown first followed by the full name.

Table 60. Valid object types.

Object Type	Description
* or **all**	All object types; this parameter is the default
authinfo	Authentication information object, for use with TLS channel security
channel or **chl**	A channel
clntconn or **clcn**	A client connection channel
catalog or **ctlg**	An object catalog
namelist or **nl**	A namelist
listener or **lstr**	A listener
process or **prcs**	A process
queue or **q**	A queue or queues matching the object name parameter
qalias or **qa**	An alias queue
qlocal or **ql**	A local queue
qmodel or **qm**	A model queue
qremote or **qr**	A remote queue
qmgr	A queue manager object
service or **srvc**	A service

Note:

1. The **dspmqfls** command displays the name of the directory containing the queue, not the name of the queue itself.
2. UNIX On UNIX, you must prevent the shell from interpreting the meaning of special characters, for example, an asterisk (*). The way you do this depends on the shell you are using. It may involve the use of single quotation marks, double quotation marks, or a backslash.

Return codes

Return code	Description
0	Command completed normally
10	Command completed but not entirely as expected
20	An error occurred during processing

Examples

1. The following command displays the details of all objects with names beginning SYSTEM.ADMIN defined on the default queue manager.

   ```
   dspmqfls SYSTEM.ADMIN*
   ```

2. The following command displays file details for all processes with names beginning PROC defined on queue manager RADIUS.

   ```
   dspmqfls -m RADIUS -t prcs PROC*
   ```

dspmqinf (display configuration information)

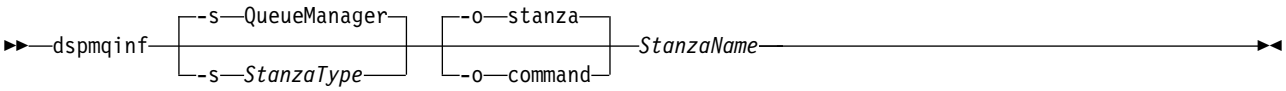
UNIX Windows

Display IBM MQ configuration information (UNIX and Windows only).

Purpose

Use the **dspmqinf** command to display IBM MQ configuration information.

Syntax



Required parameters

StanzaName
The name of the stanza. That is, the value of the key attribute that distinguishes between multiple stanzas of the same type.

Optional parameters

-s *StanzaType*
The type of stanza to display. If omitted, the QueueManager stanza is displayed.

The only supported value of *StanzaType* is `QueueManager`.

-o stanza
Displays the configuration information in stanza format as it is shown in the `.ini` files. This format is the default output format.

Use this format to display stanza information in a format that is easy to read.

-o command
Displays the configuration information as an **addmqinf** command.

Information about the installation associated with the queue manager is not displayed using this parameter. The **addmqinf** command does not require information about the installation.

Use this format to paste into a command shell.

Return codes

Return code	Description
0	Successful operation
39	Bad command-line parameters
44	Stanza does not exist
58	Inconsistent use of installations detected
69	Storage not available
71	Unexpected error
72	Queue manager name error

Examples

```
dspmqinf QM.NAME
```

The command defaults to searching for a QueueManager stanza named `QM.NAME` and displays it in stanza format.

```
QueueManager:
   Name=QM.NAME
   Prefix=/var/mqm
   Directory=QM!NAME
   DataPath=/MQHA/qmgrs/QM!NAME
   InstallationName=Installation1
```

The following command gives the same result:

```
dspmqinf -s QueueManager -o stanza QM.NAME
```

The next example displays the output in **addmqinf** format.

```
dspmqinf -o command QM.NAME
```

The output is on one line:

```
addmqinf -s QueueManager -v Name=QM.NAME -v Prefix=/var/mqm -v Directory=QM!NAME
         -v DataPath=/MQHA/qmgrs/QM!NAME
```

Usage notes

Use **dspmqinf** with **addmqinf** to create an instance of a multi-instance queue manager on a different server.

To use this command you must be an IBM MQ administrator and a member of the mqm group.

Related commands

Command	Description
"addmqinf" on page 196	Add queue manager configuration information
"rmvmqinf (remove configuration information)" on page 291	Remove queue manager configuration information

dspmqinst (display IBM MQ installation)

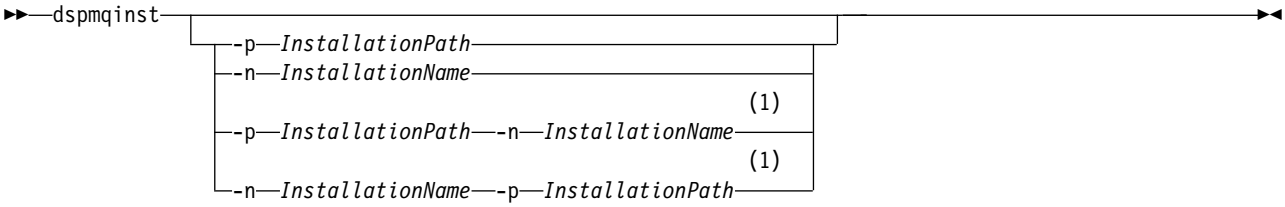
ULW

Display installation entries from mqinst.ini on UNIX, Linux, and Windows.

Purpose

File mqinst.ini contains information about all IBM MQ installations on a system. For more information about mqinst.ini, see Installation configuration file, mqinst.ini.

Syntax



Notes:

1 When specified together, the installation name and installation path must refer to the same installation.

Required parameters

None

Optional parameters

-n ***InstallationName***
The name of the installation.

-p ***InstallationPath***
The installation path.

? Display usage information.

Return codes

Return code	Description
0	Entry displayed without error
36	Invalid arguments supplied
44	Entry does not exist
59	Invalid installation specified
71	Unexpected error
89	.ini file error
96	Could not lock .ini file
131	Resource problem

Examples

1. Display details of all IBM MQ installations on the system:
   ```
   dspmqinst
   ```
2. Query the entry for the installation named *Installation3*:
   ```
   dspmqinst -n Installation3
   ```
3. Query the entry with an installation path of `/opt/mqm`:
   ```
   dspmqinst -p /opt/mqm
   ```
4. Query the entry for the installation named *Installation3*. Its expected installation path is `/opt/mqm`:
   ```
   dspmqinst -n Installation3 -p /opt/mqm
   ```

dspmqrte (display route information)

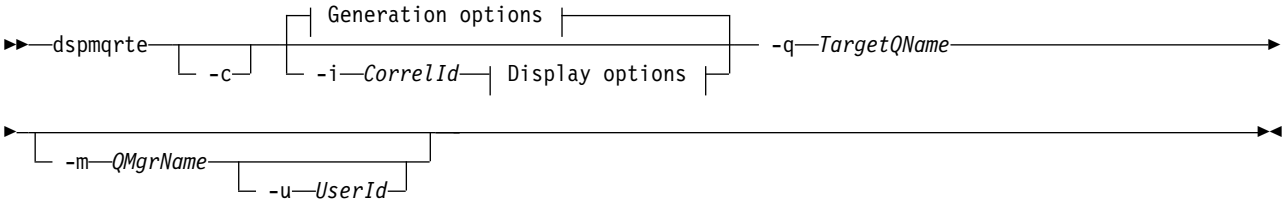
Determine the route that a message has taken through a queue manager network.

Purpose

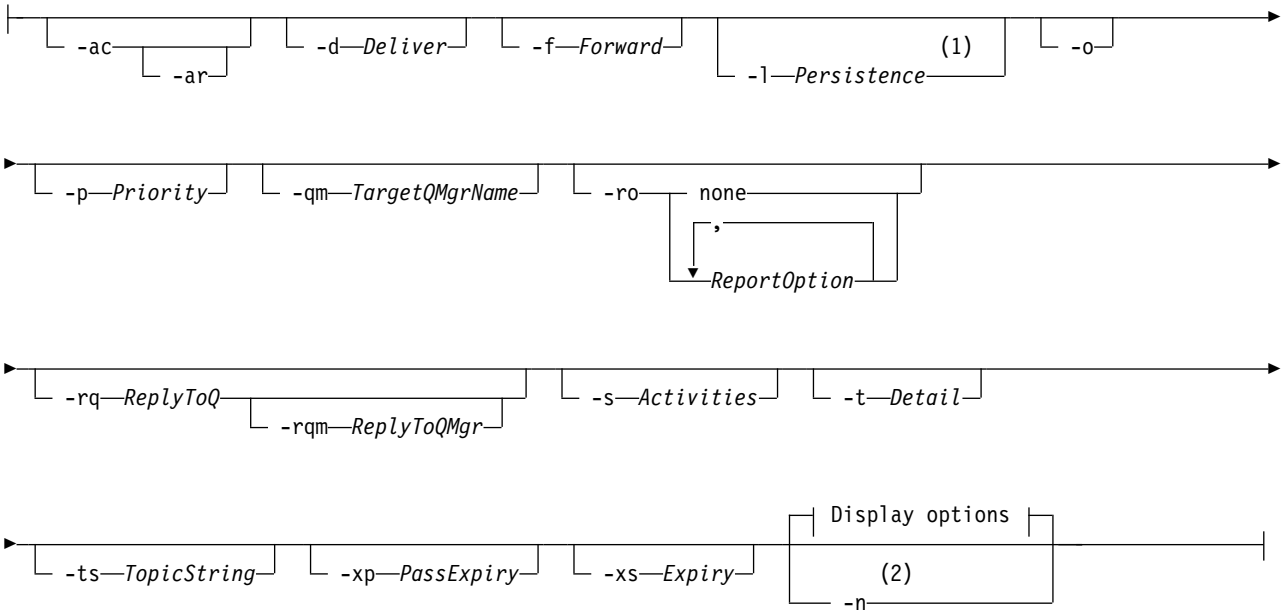
z/OS The IBM MQ display route application (**dspmqrte**) command can be run on all platforms except z/OS. You can run the IBM MQ display route application as a client to an IBM MQ for z/OS queue manager by specifying the **-c** parameter when issuing the **dspmqrte** command.

The IBM MQ display route application generates and puts a trace-route message into a queue manager network. As the trace-route message travels through the queue manager network, activity information is recorded. When the trace-route message reaches its target queue, the activity information is collected by the IBM MQ display route application and displayed. For more information, and examples of using the IBM MQ display route application, see IBM MQ display route application.

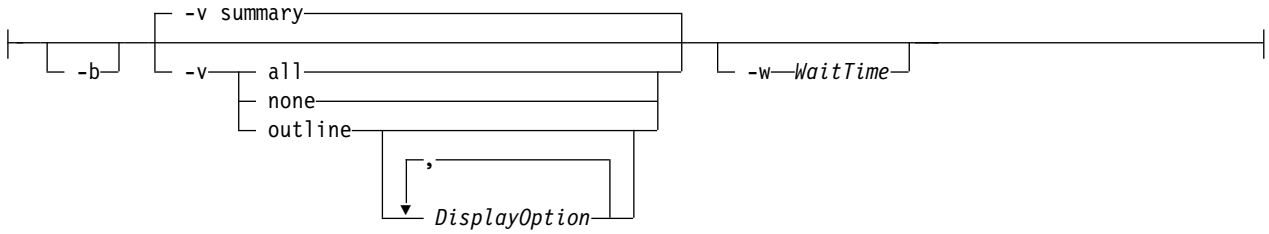
Syntax



Generation options:



Display options:



Notes:

1 If *Persistence* is specified as yes, and is accompanied by a request for a trace-route reply message (*-ar*), or any report generating options (*-ro ReportOption*), then you must specify the parameter *-rq ReplyToQ*. The reply-to queue must not resolve to a temporary dynamic queue.

2 If this parameter is accompanied by a request for a trace-route reply message (*-ar*), or any of the report generating options (*-ro ReportOption*), then a specific (non-model) reply-to queue must be specified using *-rq ReplyToQ*. By default, activity report messages are requested.

Required parameters

-q *TargetQName*
If the IBM MQ display route application is being used to send a trace-route message into a queue manager network, *TargetQName* specifies the name of the target queue.

If the IBM MQ display route application is being used to view previously gathered activity information, *TargetQName* specifies the name of the queue where the activity information is stored.

Optional parameters

-c Specifies that the IBM MQ display route application connects as a client application. For more information about how to set up client machines, see:

- 

Installing an IBM MQ client on an AIX workstation
- 

Installing an IBM MQ client on an HP-UX workstation
- 

Installing an IBM MQ client on a Linux workstation
- 

Installing an IBM MQ client on a Solaris workstation
- 

Installing an IBM MQ client on a Windows workstation
- 

Installing an IBM MQ client on an IBM i workstation

This parameter can be used only if the client component is installed.

-i *CorrelId*
This parameter is used when the IBM MQ display route application is used to display previously accumulated activity information only. There can be many activity reports and trace-route reply messages on the queue specified by **-q** *TargetQName*. *CorrelId* is used to identify the activity reports, or a trace-route reply message, related to a trace-route message. Specify the message identifier of the original trace-route message in *CorrelId*.

The format of *CorrelId* is a 48 character hexadecimal string.

-m *QMgrName*
The name of the queue manager to which the IBM MQ display route application connects. The name can contain up to 48 characters.

If you do not specify this parameter, the default queue manager is used.

Generation options

The following parameters are used when the IBM MQ display route application is used to put a trace-route message into a queue manager network.

-ac
Specifies that activity information is to be accumulated within the trace-route message.

If you do not specify this parameter, activity information is not accumulated within the trace-route message.

-ar
Requests that a trace-route reply message containing all accumulated activity information is generated in the following circumstances:

- The trace-route message is discarded by a IBM WebSphere MQ Version 7.0 queue manager.
- The trace-route message is put to a local queue (target queue or dead-letter queue) by a IBM WebSphere MQ Version 7.0 queue manager.
- The number of activities performed on the trace-route message exceeds the value of specified in **-s** *Activities*.

For more information about trace-route reply messages, see Trace-route reply message reference.

If you do not specify this parameter, a trace-route reply message is not requested.

-d *Deliver*

Specifies whether the trace-route message is to be delivered to the target queue on arrival. Possible values for *Deliver* are:

Table 61. Delivery parameter values.

Value	Description
yes	On arrival, the trace-route message is put to the target queue, even if the queue manager does not support trace-route messaging.
no	On arrival, the trace-route message is not put to the target queue.

If you do not specify this parameter, the trace-route message is not put to the target queue.

-f *Forward*

Specifies the type of queue manager that the trace-route message can be forwarded to. Queue managers use an algorithm when determining whether to forward a message to a remote queue manager. For details of this algorithm, see The cluster workload management algorithm. The possible values for *Forward* are:

Table 62. Forward parameter values.

Value	Description
all	The trace-route message is forwarded to any queue manager. **Note:** If forwarded to an IBM WebSphere MQ queue manager before Version 6.0, the trace-route message is not recognized and can be delivered to a local queue despite the value of the **-d Deliver** parameter.
supported	The trace-route message is only forwarded to a queue manager that honors the *Deliver* parameter from the *TraceRoute* PCF group.

If you do not specify this parameter, the trace-route message is only forwarded to a queue manager that honors the *Deliver* parameter.

-l *Persistence*

Specifies the persistence of the generated trace-route message. Possible values for *Persistence* are:

Table 63. Persistence parameter values.

Value	Description
yes	The generated trace-route message is persistent. (MQPER_PERSISTENT).
no	The generated trace-route message is not persistent. (MQPER_NOT_PERSISTENT).
q	The generated trace-route message inherits its persistence value from the queue specified by -q *TargetQName*. (MQPER_PERSISTENCE_AS_Q_DEF).

A trace-route reply message, or any report messages, returned shares the same persistence value as the original trace-route message.

If *Persistence* is specified as yes, you must specify the parameter -rq *ReplyToQ*. The reply-to queue must not resolve to a temporary dynamic queue.

If you do not specify this parameter, the generated trace-route message is not persistent.

-o Specifies that the target queue is not bound to a specific destination. Typically this parameter is used when the trace-route message is to be put across a cluster. The target queue is opened with option MQOO_BIND_NOT_FIXED.

If you do not specify this parameter, the target queue is bound to a specific destination.

-p *Priority*
Specifies the priority of the trace-route message. The value of *Priority* is either greater than or equal to 0, or MQPRI_PRIORITY_AS_Q_DEF. MQPRI_PRIORITY_AS_Q_DEF specifies that the priority value is taken from the queue specified by `-q TargetQName`.

If you do not specify this parameter, the priority value is taken from the queue specified by `-q TargetQName`.

-qm *TargetQMgrName*
Qualifies the target queue name; normal queue manager name resolution applies. The target queue is specified with `-q TargetQName`.

If you do not specify this parameter, the queue manager to which the IBM MQ display route application is connected is used as the reply-to queue manager.

-ro none|ReportOption

Table 64. ReportOption parameter values.

Value	Description
`none`	Specifies no report options are set.
ReportOption	Specifies report options for the trace-route message. Multiple report options can be specified using a comma as a separator. Possible values for *ReportOption* are: **activity** The report option MQRO_ACTIVITY is set. **coa** The report option MQRO_COA_WITH_FULL_DATA is set. **cod** The report option MQRO_COD_WITH_FULL_DATA is set. **exception** The report option MQRO_EXCEPTION_WITH_FULL_DATA is set. **expiration** The report option MQRO_EXPIRATION_WITH_FULL_DATA is set. **discard** The report option MQRO_DISCARD_MSG is set.

If `-ro ReportOption` or `-ro none` are not specified, then the MQRO_ACTIVITY and MQRO_DISCARD_MSG report options are specified.

-rq *ReplyToQ*
Specifies the name of the reply-to queue that all responses to the trace-route message are sent to. If the trace-route message is persistent, or if the **-n** parameter is specified, a reply-to queue must be specified that is not a temporary dynamic queue.

If you do not specify this parameter, the system default model queue, SYSTEM.DEFAULT.MODEL.QUEUE is used as the reply-to queue. Using this model queue causes a temporary dynamic queue, for the IBM MQ display route application, to be created.

-rqm *ReplyToQMgr*
Specifies the name of the queue manager where the reply-to queue is located. The name can contain up to 48 characters.

If you do not specify this parameter, the queue manager to which the IBM MQ display route application is connected is used as the reply-to queue manager.

-s *Activities*
Specifies the maximum number of recorded activities that can be performed on behalf of the trace-route message before it is discarded. This parameter prevents the trace-route message from being forwarded indefinitely if caught in an infinite loop. The value of *Activities* is either greater than or equal to 1, or MQROUTE_UNLIMITED_ACTIVITIES. MQROUTE_UNLIMITED_ACTIVITIES specifies that an unlimited number of activities can be performed on behalf of the trace-route message.

If you do not specify this parameter, an unlimited number of activities can be performed on behalf of the trace-route message.

-t *Detail*
Specifies the activities that are recorded. The possible values for *Detail* are:

Table 65. Detail parameter values.

Value	Description
`low`	Activities performed by user-defined application are recorded only.
`medium`	Activities specified in `low` are recorded. Additionally, activities performed by MCAs are recorded.
`high`	Activities specified in `low`, and `medium` are recorded. MCAs do not expose any further activity information at this level of detail. This option is available to user-defined applications that are to expose further activity information only. For example, if a user-defined application determines the route a message takes by considering certain message characteristics, the routing logic can be included with this level of detail.

If you do not specify this parameter, medium level activities are recorded.

-ts *TopicString*
Specifies a topic string to which the IBM MQ display route application is to publish a trace-route message, and puts this application into topic mode. In this mode, the application traces all of the messages that result from the publish request.

-xp *PassExpiry*
Specifies whether the report option MQRO_DISCARD_MSG and the remaining expiry time from the trace-route message is passed on to the trace-route reply message. Possible values for *PassExpiry* are:

Table 66. PassExpiry parameter values.

Value	Description
`yes`	The report option MQRO_PASS_DISCARD_AND_EXPIRY is specified in the message descriptor of the trace-route message. If a trace-route reply message, or activity reports, are generated for the trace-route message, the MQRO_DISCARD_MSG report option (if specified), and the remaining expiry time are passed on. This parameter is the default value.

Table 66. PassExpiry parameter values. (continued)

Value	Description
`no`	The report option MQRO_PASS_DISCARD_AND_EXPIRY is not specified. If a trace-route reply message is generated for the trace-route message, the discard option and remaining expiry time from the trace-route message are not passed on.

If you do not specify this parameter, the MQRO_PASS_DISCARD_AND_EXPIRY report option is not specified in the trace-route message.

-xs ***Expiry***
Specifies the expiry time for the trace-route message, in seconds.

If you do not specify this parameter, the expiry time is specified as 60 seconds.

-n Specifies that activity information returned for the trace-route message is not to be displayed.

If this parameter is accompanied by a request for a trace-route reply message (`-ar`), or any of the report generating options from (`-ro` *ReportOption*), then a specific (non-model) reply-to queue must be specified using `-rq` *ReplyToQ*. By default, activity report messages are requested.

After the trace-route message is put to the specified target queue, a 48 character hexadecimal string is returned containing the message identifier of the trace-route message. The message identifier can be used by the IBM MQ display route application to display the activity information for the trace-route message at a later time. This can be done using the **-i** *CorrelId* parameter.

If you do not specify this parameter, activity information returned for the trace-route message is displayed in the form specified by the **-v** parameter.

Display options

The following parameters are used when the IBM MQ display route application is used to display collected activity information.

-b Specifies that the IBM MQ display route application only browses activity reports or a trace-route reply message related to a message. This parameter allows activity information to be displayed again at a later time.

If you do not specify this parameter, the IBM MQ display route application gets activity reports and deletes them, or a trace-route reply message related to a message.

-v summary | all | none | outline ***DisplayOption***

Table 67. DisplayOption parameter values.

Value	Description
`summary`	The queues that the trace-route message was routed through are displayed.
`all`	All available information is displayed.
`none`	No information is displayed.

Table 67. DisplayOption parameter values. (continued)

Value	Description
`outline` *DisplayOption*	Specifies display options for the trace-route message. Multiple display options can be specified using a comma as a separator. If no values are supplied the subsequent information is displayed: • The application name • The type of each operation • Any operation-specific parameters Possible values for *DisplayOption* are: **activity** All non-PCF group parameters in *Activity* PCF groups are displayed. **identifiers** Values with parameter identifiers MQBACF_MSG_ID or MQBACF_CORREL_ID are displayed. This overrides *msgdelta*. **message** All non-PCF group parameters in *Message* PCF groups are displayed. When this value is specified, you cannot specify *msgdelta*. **msgdelta** All non-PCF group parameters in *Message* PCF groups, that have changed since the last operation, are displayed. When this value is specified, you cannot specify *message*. **operation** All non-PCF group parameters in *Operation* PCF groups are displayed. **traceroute** All non-PCF group parameters in *TraceRoute* PCF groups are displayed.

If you do not specify this parameter, a summary of the message route is displayed.

-w *WaitTime*
Specifies the time, in seconds, that the IBM MQ display route application waits for activity reports, or a trace-route reply message, to return to the specified reply-to queue.

If you do not specify this parameter, the wait time is specified as the expiry time of the trace-route message, plus 60 seconds.

-u *UserId*
The ID of the user authorized to determine the route that a message has taken through a queue manager network.

Return codes

Return code	Description
0	Command completed normally
10	Invalid arguments supplied
20	An error occurred during processing

Examples

1. The following command puts a trace-route message into a queue manager network with the target queue specified as TARGET.Q. Providing queue managers on route are enabled for activity recording, activity reports are generated. Depending on the queue manager attribute, ACTIVREC, activity reports are either delivered to the reply-to queue ACT.REPORT.REPLY.Q, or are delivered to a system queue. The trace-route message is discarded on arrival at the target queue.

   ```
   dspmqrte -q TARGET.Q -rq ACT.REPORT.REPLY.Q
   ```

   Providing one or more activity reports are delivered to the reply-to queue, ACT.REPORT.REPLY.Q, the IBM MQ display route application orders and displays the activity information.
2. The following command puts a trace-route message into a queue manager network with the target queue specified as TARGET.Q. Activity information is accumulated within the trace-route message, but activity reports are not generated. On arrival at the target queue, the trace-route message is discarded. Depending on the value of the target queue manager attribute, ROUTEREC, a trace-route reply message can be generated and delivered to either the reply-to queue, TRR.REPLY.TO.Q, or to a system queue.

   ```
   dspmqrte -ac -ar -ro discard -rq TRR.REPLY.TO.Q -q TARGET.Q
   ```

   Providing a trace-route reply message is generated, and delivered to the reply-to queue TRR.REPLY.TO.Q, the IBM MQ display route application orders and displays the activity information that was accumulated in the trace-route message.

For more examples of using the IBM MQ display route application and its output, see IBM MQ display route application examples.

dspmqspl (display security policy)

Use the **dspmqspl** command to display a list of all policies and details of a named policy.

Syntax

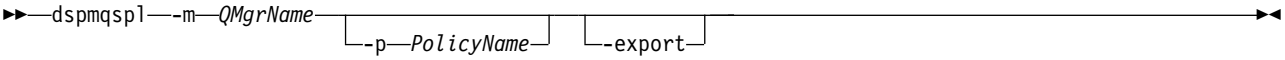


Table 68. dspmqspl command flags

Command flag	Explanation
-m	Queue manager name (mandatory).
-p	Policy name.
-export The output is written to a DD named EXPORT	Adding this flag generates output which can easily be applied to a different queue manager.

V 9.0.0

Examples

The **dspmqspl** command shows the key reuse count for all policies. The following example is the output you receive on Multiplatforms:

```
Policy Details:
Policy name: PROT
Quality of protection: PRIVACY
Signature algorithm: SHA256
Encryption algorithm: AES256
Signer DNs: -
Recipient DNs:
  CN=Name, O=Organization, C=Country
Toleration: 0
Key Reuse Count: 0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Policy Details:
Policy name: PROT2
Quality of protection: CONFIDENTIALITY
Signature algorithm: NONE
Encryption algorithm: AES256
Signer DNs: -
Recipient DNs:
  CN=Name, O=Organization, C=Country
Toleration: 0
Key Reuse Count: 100
```

z/OS On z/OS, you can use the **dspmqspl** command with the CSQ0UTIL utility. For more information, see The message security policy utility (CSQ0UTIL).

dspmqtrc (display formatted trace)

UNIX NSS Client

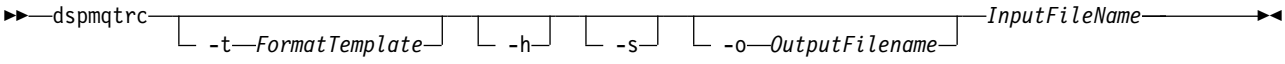
Format and display IBM MQ trace.

Purpose

The **dspmqtrc** command is supported on UNIX and HP Integrity NonStop Server systems only. Use the **dspmqtrc** command to display IBM MQ formatted trace output.

The runtime TLS trace files have the names AMQ.SSL.TRC and AMQ.SSL.TRC.1. You cannot format any of the TLS trace files. The TLS trace files are binary files and, if they are transferred to IBM support by FTP, they must be transferred in binary transfer mode.

Syntax



Required parameters

InputFileName
The name of the file containing the unformatted trace, for example:

```
/var/mqm/trace/AMQ12345.01.TRC
```

If you provide one input file, **dspmqtrc** formats it to the output file you name. If you provide more than one input file, any output file you name is ignored, and formatted files are named `AMQ yyyyy.zz.FMT`, based on the PID of the trace file.

Optional parameters

-t *FormatTemplate*
The name of the template file containing details of how to display the trace. If this parameter is not supplied, the default template file location is used:

AIX For AIX systems, the default value is as follows:

MQ_INSTALLATION_PATH/lib/amqtrc2.fmt

UNIX NSS Client For all UNIX platforms other than AIX and HP Integrity NonStop Server, the default value is as follows:

MQ_INSTALLATION_PATH/lib/amqtrc.fmt

MQ_INSTALLATION_PATH represents the high-level directory in which IBM MQ is installed.

-h Omit header information from the report.

-s Extract trace header and put to stdout.

-o *output_filename*
The name of the file into which to write formatted data.

Related commands

Command	Description
endmqtrc	End trace
"strmqtrc (Start trace)" on page 355	Start trace

Related reference:

Command sets comparison: Other commands
A table of other commands, showing the command description, and its PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

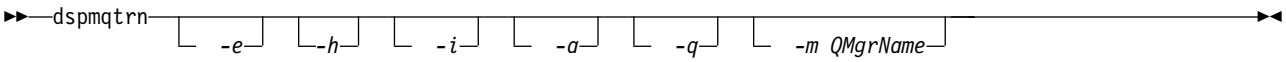
dspmqtrn (display incomplete transactions)

Display in-doubt and heuristically completed transactions.

Purpose

Use the **dspmqtrn** command to display details of transactions. This command includes transactions coordinated by IBM MQ and by an external transaction manager.

Syntax



Optional parameters

-e Requests details of externally coordinated, in-doubt transactions. Such transactions are those for which IBM MQ has been asked to prepare to commit, but has not yet been informed of the transaction outcome.

-h Requests details of externally coordinated transactions that were resolved by the **rsvmqtrn** command, and the external transaction coordinator has yet to acknowledge with an `xa-forget` command. This transaction state is termed *heuristically completed* by X/Open.

Note: If you do not specify **-e**, **-h**, or **-i**, details of both internally and externally coordinated in-doubt transactions are displayed, but details of externally coordinated, heuristically completed transactions are not displayed.

-i Requests details of internally coordinated, in-doubt transactions. Such transactions are those for which each resource manager has been asked to prepare to commit, but IBM MQ has yet to inform the resource managers of the transaction outcome.

Information about the state of the transaction in each of its participating resource managers is displayed. This information can help you assess the affects of failure in a particular resource manager.

Note: If you do not specify **-e** or **-i**, details of both internally and externally coordinated in-doubt transactions are displayed.

-a Requests a list of all transactions known to the queue manager . The returned data includes transaction details for all transactions known to the queue manager. If a transaction is currently associated with an IBM MQ application connection, information related to that IBM MQ application connection is also returned. The data returned by this command might typically be correlated with the output of a runmqsc "DISPLAY CONN" on page 822 command, and the output fields have the same meaning as in that command.

Not all of the fields are appropriate for all transactions. When the fields are not meaningful, they are displayed as blank. For example: The UOWLOG value when the command is issued against a circular logging queue manager.

-q Specifying this parameter on its own is the same as specifying **-a -q**.

Displays all the data from the **-a** parameter and a list of up to 100 unique objects updated within the transaction. If more than 100 objects are updated in the same transaction, only the first 100 distinct objects are listed for each transaction.

-m *QMgrName*
The name of the queue manager for which to display transactions. If you omit the name, the transaction of the default queue manager are displayed.

Return codes

Return code	Description
0	Successful operation
26	Queue manager running as a standby instance.
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
58	Inconsistent use of installations detected
69	Storage not available
71	Unexpected error
72	Queue manager name error
102	No transactions found

Related commands

Command	Description
rsvmqtrn	Resolve transaction

dspmqver (display version information)

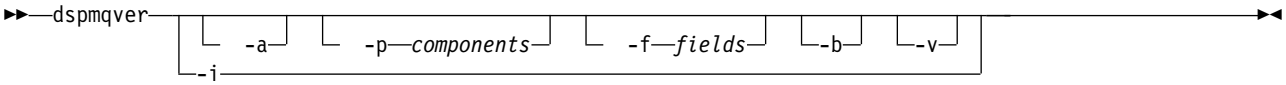
Display IBM MQ version and build information.

Purpose

Use the **dspmqver** command to display IBM MQ version and build information.

By default, the **dspmqver** command displays details of the installation from which it was invoked. A note is displayed if other installations exist; use the **-i** parameter to display their details.

Syntax



Optional parameters

-a Display information about all fields and components.

-p *Components*
Display information for the components specified by *component*. Either a single component or multiple components can be specified. Enter either the value of a single component or the sum of the values of all the required components. Available components and related values are as follows:

Value	Description
1	IBM MQ server, or client.
2	IBM MQ classes for Java.
4	IBM MQ classes for Java Message Service.
8	WebScale Distribution Hub
16 [1]	Windows IBM MQ custom channel for Windows Communication Foundation
32	Windows IBM Message Service Client for .NET (XMS .NET) - this component is only available on Windows
64	GSKit NSS Client For HP Integrity NonStop Server, this is TLS
128	Advanced Message Security
256	IBM MQ AMQP Service
512	IBM MQ Telemetry Service
1024	Other bundled components that are used by IBM MQ
V 9.0.1 2048	WebSphere Application Server Liberty profile
V 9.0.2 4096	IBM MQ Java Runtime Environment
V 9.0.4 8192	IBM MQ Replicated Data Queue Managers

Notes:

1. Windows Supported by IBM MQ for Windows only. If you have not installed Microsoft.NET 3 or later, the following error message is displayed:

   ```
   Title: WMQWCFCustomChannelLevel.exe - Application Error The application failed to
   initialize properly (0x0000135).
   ```

The default value is 1.

-f *Fields*
Display information for the fields specified by *field*. Specify either a single field or multiple fields. Enter either the value of a single field or the sum of the values of all the required fields. Available fields and related values are as follows:

Value	Description
1	Name
2	Version, in the form `V.R.M.F`: Where `V` =Version, `R` =Release, `M` =Modification, and `F` =Fix pack
4	Level
8	Build type
16	Platform
32	Addressing mode
64	Operating system
128	Installation path
256 NSS Client [1]	Installation description
512 NSS Client [1]	Installation name
1024 NSS Client [1]	Maximum command level
2048 NSS Client [1]	Primary installation
4096	Data Path
8192	License type

Note: NSS Client

1. Not applicable to HP Integrity NonStop Server.

Information for each selected field is displayed on a separate line when the **dspmqver** command is run.

The default value is 8191. This displays information for all fields.

-b Omit header information from the report.

-v Display verbose output.

-i Display information about all installations. You cannot use this option with other options. The installation from which the **dspmqver** command was issued is displayed first. For any other installations, only the following fields are displayed: Name, Version, Installation name, Installation description, Installation path, and Primary installation.

NSS Client Not applicable to HP Integrity NonStop Server.

Return codes

Return code	Description
0	Command completed normally.
10	Command completed with unexpected results.
20	An error occurred during processing.

Examples

The following command displays IBM MQ version and build information, using the default settings for **-p** and **-f**:

```
dspmqver
```

The following command displays information about all fields and components and is the equivalent of specifying `dspmqver -p 63 -f 4095`:

```
dspmqver -a
```

The following command displays version and build information for the IBM MQ classes for Java:

```
dspmqver -p 2
```

The following command displays the Common Services for Java Platform Standard Edition, IBM MQ, Java Message Service Client, and IBM MQ classes for Java Message Service:

```
dspmqver -p 4
```

The following command displays the build level of the WebScale Distribution Hub:

```
dspmqver -p 8 -f 4
```

Windows The following command displays the name and build type for IBM MQ custom channel for Windows Communication Foundation:

```
dspmqver -p 16 -f 9
```

The following command displays information about installations of IBM MQ.

```
dspmqver -i
```

Command failure

V 9.0.2 The **dspmqver** command can fail if you try to view version or build information for the IBM MQ classes for Java, and you have not correctly configured your environment, or if the IBM MQ JRE component is not installed, and an alternative JRE could not be located.

V 9.0.2 For example, you might see the following message:

```
[root@blade883 ~]# dspmqver -p 2
AMQ8351: IBM MQ Java environment has not been configured
correctly, or the IBM MQ JRE feature has not been installed.
```

To resolve this problem, consider installing the IBM MQ JRE component if it is not already installed, or ensure that the path is configured to include the JRE, and that the correct environment variables are set; for example, by using `setjmsenv` or `setjmsenv64`.

For example:

```
export PATH=$PATH:/opt/mqm/java/jre/bin
cd /opt/mqm/java/bin/
. ./setjmsenv64

[root@blade883 bin]# dspmqver -p 2
Name:       IBM MQ classes for Java
Version:    8.0.0.0
Level:      k000-L110908
Build Type: Production
```

UNIX Note that the **setjmsenv** and **setjmsenv64** commands apply to UNIX only.

Windows V 9.0.2 On Windows, if theIBM MQ JRE component is installed, you need to issue the **setmqenv** command to resolve error AMQ8351.

dspmqweb (display mqweb server configuration)

V 9.0.1

Display information about the status of the mqweb server, or about the configuration of the mqweb server. The mqweb server is used to support the IBM MQ Console and administrative REST API.

z/OS

Using the command on z/OS

Before issuing either the **setmqweb** or **dspmqweb** commands on z/OS, you must set the WLP_USER_DIR environment variable, so that the variable points to your mqweb server configuration.

To do this, issue the following command:

```
export WLP_USER_DIR=WLP_user_directory
```

where *WLP_user_directory* is the name of the directory passed to `crtmqweb.sh`. For example:

```
export WLP_USER_DIR=/var/mqm/web/installation1
```

See Create the Liberty server definition for more information.

Purpose - dspmqweb status

Use the **dspmqweb** command to view information about the status of the mqweb server.

The mqweb server must be running to use the IBM MQ Console or the administrative REST API. If the server is running then the available root context URLs and associated ports that are used by the IBM MQ Console and administrative REST API are displayed by the **dspmqweb** command. Alternatively, from Version 9.0.4, use the **dspmqweb status** command.

V 9.0.4

Purpose - dspmqweb properties

Use the **dspmqweb properties** command to view details of the configuration of the mqweb server. It is not necessary for the mqweb server to be running.

The following list outlines the available configuration properties on all platforms, including the IBM MQ Appliance:

The following properties can be returned by the **dspmqweb properties** command on all platforms, including the IBM MQ Appliance:

ltpaExpiration
This configuration property is used to specify the time, in seconds, before the LTPA token expires.

The value for this property is an integer value.

maxTraceFiles
This configuration property is used to specify the maximum number of trace files that are generated by the mqweb server.

The value for this property is an integer value.

maxTraceFileSize
This configuration property is used to specify the maximum size, in MB, that each log file can reach.

The value for this property is an integer value.

mqRestCorsAllowedOrigins
This configuration property is used to specify the origins that are allowed to access the REST API. For more information about CORS, see Configuring CORS for the REST API.

The value for this property is a string value.

mqRestCorsMaxAgeInSeconds
This configuration property is used to specify the time, in seconds, that a web browser can cache the results of any CORS pre-flight checks.

The value for this property is an integer value.

V 9.0.5 **mqRestCsrfExpirationInMinutes**
This configuration property no longer exists at Version 9.0.5.

Applicable to Version 9.0.4 only, and used to specify the time, in minutes, before the CSRF token expires.

The value for this property is an integer value.

mqRestCsrfValidation
This configuration property is used to specify whether CSRF validation checks are performed. A value of `false` removes the CSRF token validation checks.

The value for this property is a boolean value.

mqRestGatewayEnabled
This configuration property is used to specify whether the administrative REST API gateway is enabled.

The value for this property is a boolean value.

mqRestGatewayQmgr
This configuration property is used to specify the name of the queue manager to use as the gateway queue manager. This queue manager must be in the same installation as the mqweb server. A blank value indicates that no queue manager is configured as the gateway queue manager.

The value for this property is a string value.

mqRestMessagingEnabled
This configuration property is used to specify whether the messaging REST API is enabled.

The value for this property is a boolean value.

mqRestRequestTimeout
This configuration property is used to specify the time, in seconds, before a REST request times out.

The value for this property is an integer value.

traceSpec
This configuration property is used to specify the level of trace that is generated by the mqweb server. For a list of possible values, see Configuring logging for the IBM MQ Console and REST API.

The value for this property is a string value.

UNIX Linux Windows z/OS

The following properties are the additional properties that can be returned by the **dspmqweb properties** command on z/OS, UNIX, Linux, and Windows:

httpHost
This configuration property is used to specify the HTTP host name as an IP address, domain name server (DNS) host name with domain name suffix, or the DNS host name of the server where IBM MQ is installed.

You can use an asterisk in double quotation marks to specify all available network interfaces.

You can use the value `localhost` to allow only local connections.

The value for this property is a string value.

httpPort
This configuration property is used to specify the HTTP port number that is used for HTTP connections.

You can use a value of `-1` to disable the port.

The value for this property is an integer value.

httpsPort
This configuration property is used to specify the HTTPS port number that is used for HTTPS connections.

You can use a value of `-1` to disable the port.

The value for this property is an integer value.

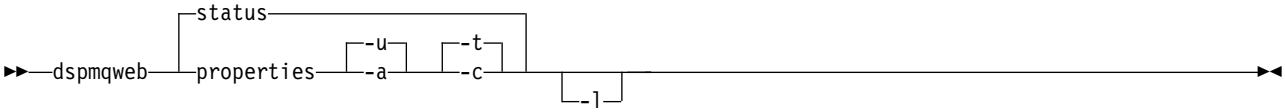
mqConsoleAutostart
This configuration property is used to specify whether the IBM MQ Console automatically starts when the mqweb server starts.

The value for this property is a boolean value.

mqRestAutostart
This configuration property is used to specify whether the REST API automatically starts when the mqweb server starts.

The value for this property is a boolean value.

Syntax



Optional parameters

V 9.0.4 **status**
Displays information about the status of the mqweb server. That is, whether the mqweb server is running. If the mqweb server is running, information about the available root context URLs and associated ports that are used by the IBM MQ Console and administrative REST API are displayed.

For example:

```
Server mqweb is running.
URLs:
  https://localhost:9443/ibmmq/console/
  https://localhost:9443/ibmmq/rest/v1/
```

V 9.0.4 **properties**
Displays information about the configurable properties of the mqweb server. That is, which properties are configurable by the user and those that have been modified. It is not necessary for the mqweb server to be running.

-u Displays only the configurable properties that have been modified by the user.

-a Displays all available configurable properties, including those which have been modified by the user.

-t Formats the output as text name-value pairs.

-c Formats the output as command text which can be used as input to the corresponding **setmqweb properties** command.

-l Enable verbose logging. Diagnostic information is written to a mqweb server log-file.

Return codes

Return code	Description
0	Command successful
>0	Command not successful.

For a full list of server command exit codes, see Liberty:server command options in the WebSphere Application Server documentation.

Related commands

Command	Description
strmqweb	Start the mqweb server.
endmqweb	Stop the mqweb server.
V 9.0.4 setmqweb	Configure the mqweb server.

endmqcsv (end command server)

Stop the command server for a queue manager.

Purpose

Use the **endmqscv** command to stop the command server on the specified queue manager.

You must use the **endmqscv** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

If the queue manager attribute, SCMDSERV, is specified as QMGR then changing the state of the command server using **endmqscv** does not effect how the queue manager acts upon the SCMDSERV attribute at the next restart.

Syntax

Required parameters

QMgrName
The name of the queue manager for which to end the command server.

Optional parameters

-c Stops the command server in a controlled manner. The command server can complete the processing of any command message that it has already started. No new message is read from the command queue.

This parameter is the default.

-i Stops the command server immediately. Actions associated with a command message currently being processed might not complete.

Return codes

Return code	Description
0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

1. The following command stops the command server on queue manager `saturn.queue.manager`:

   ```
   endmqcsv -c saturn.queue.manager
   ```

   The command server can complete processing any command it has already started before it stops. Any new commands received remain unprocessed in the command queue until the command server is restarted.
2. The following command stops the command server on queue manager `pluto` immediately:

   ```
   endmqcsv -i pluto
   ```

Related commands

Command	Description
strmqcsv	Start a command server
dspmqcsv	Display the status of a command server

Related reference:

"Command server commands" on page 188
A table of command server commands, showing the PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

endmqlsr (end listener)

End all listener process for a queue manager.

Purpose

The **endmqlsr** command ends all listener processes for the specified queue manager.

You must use the **endmqlsr** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

You do not have to stop the queue manager before issuing the **endmqlsr** command. If any of the listeners are configured to have inbound channels running within the **runmqlsr** listener process, rather than within a pool process, the request to end that listener might fail if channels are still active. In this case a message is written indicating how many listeners were successfully ended and how many listeners are still running.

If the listener attribute, CONTROL, is specified as QMGR then changing the state of the listener using **endmqlsr** does not effect how the queue manager acts upon the CONTROL attribute at the next restart.

Syntax

Optional parameters

-m *QMgrName*
The name of the queue manager. If you omit this parameter, the command operates on the default queue manager.

-w Wait before returning control.

Control is returned to you only after all listeners for the specified queue manager have stopped.

Return codes

Return code	Description
0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Related reference:

"Listener commands" on page 191
A table of listener commands, showing the PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

Related information:

Applying maintenance level updates to multi-instance queue managers on Windows

Applying maintenance level updates to multi-instance queue managers on UNIX and Linux

endmqdnm (stop .NET monitor)

Windows

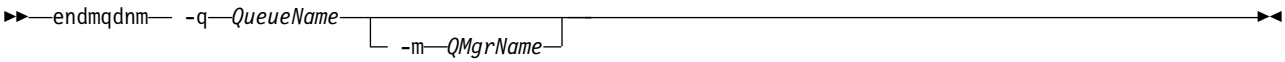
Stop the .NET monitor for a queue (Windows only).

Purpose

Note: The endmqdnm command applies to IBM MQ for Windows only.

Use the **endmqdnm** control command to stop a .NET monitor.

Syntax



Required parameters

-q *QueueName*
The name of the application queue that the .NET monitor is monitoring.

Optional parameters

-m *QMgrName*
The name of the queue manager that hosts the application queue.

If omitted, the default queue manager is used.

Return codes

Return code	Description
0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
58	Inconsistent use of installations detected
71	Unexpected error
72	Queue manager name error
133	Unknown object name error

Related information:
Using the .NET monitor

endmqm (end queue manager)

Stop a queue manager or switch to a standby queue manager.

Purpose

Use the **endmqm** command to end (stop) a specified queue manager. This command stops a queue manager in one of three modes:

- Controlled or quiesced shutdown
- Immediate shutdown
- Pre-emptive shut down

The **endmqm** command stops all instances of a multi-instance queue manager in the same way as it stops a single instance queue manager. You can issue the **endmqm** on either the active instance, or one of the standby instances of a multi-instance queue manager. You must issue **endmqm** on the active instance to end the queue manager.

If you issue the **endmqm** command on the active instance of a multi-instance queue manager, you can permit a standby instance to switch over to being the new active instance when the current active instance completes its shutdown.

If you issue the **endmqm** command on a standby instance of a multi-instance queue manager, you can end the standby instance by adding the **-x** option, and leave the active instance running. The queue manager reports an error if you issue **endmqm** on the standby instance without the **-x** option.

Issuing the **endmqm** command will affect any client application connected through a server-connection channel. The effect varies depending on the parameter used, but it is as though a STOP CHANNEL command was issued in one of the three possible modes. See Stopping MQI channels, for information

about the effects of STOP CHANNEL modes on server-connection channels. The **endmqm** optional parameter descriptions state which STOP CHANNEL mode they will be equivalent to.

If you issue **endmqm** to stop a queue manager, reconnectable clients do not try to reconnect. To override this behavior, specify either the `-r` or `-s` option to enable clients to start trying to reconnect.

Note: If a queue manager or a channel ends unexpectedly, reconnectable clients start trying to reconnect.

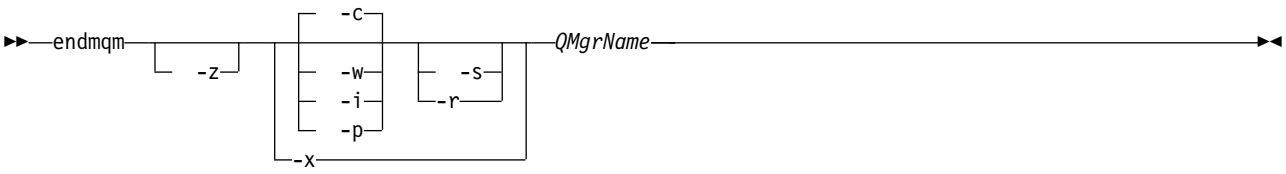
Note: The client might not reconnect to this queue manager. Depending on the MQCONNX reconnect option the client has used, and the definition of the queue manager group in the client connection table, the client might reconnect to a different queue manager. You can configure the client to force it to reconnect to the same queue manager.

You must use the **endmqm** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

The attributes of the queue manager and the objects associated with it are not affected by the **endmqm** command. You can restart the queue manager using the **strmqm** (Start queue manager) command.

To delete a queue manager, stop it and then use the **dltmqm** (Delete queue manager) command.

Syntax



Required parameters

QMgrName
The name of the message queue manager to be stopped.

Optional parameters

-c Controlled (or quiesced) shutdown. This parameter is the default.

The queue manager stops, but only after all applications have disconnected. Any MQI calls currently being processed are completed. In the unlikely event that a "dspmq (display queue managers)" on page 236 command is issued in the small timeframe between the applications disconnecting and the queue manager actually stopping, the "dspmq (display queue managers)" on page 236 command might transiently report the status as `Ending immediately`, even though a controlled shutdown was requested.

Control is returned to you immediately and you are not notified of when the queue manager has stopped.

The effect on any client applications connected through a server-connection channel is equivalent to a STOP CHANNEL command issued in QUIESCE mode.

-i Immediate shutdown. The queue manager stops after it has completed all the MQI calls currently being processed. Any MQI requests issued after the command has been issued fail. Any incomplete units of work are rolled back when the queue manager is next started.

Control is returned after the queue manager has ended.

The effect on any client applications connected through a server-connection channel is equivalent to a STOP CHANNEL command issued in FORCE mode.

-p Pre-emptive shutdown.

Important: Use this type of shutdown only in exceptional circumstances, for example, when a queue manager does not stop as a result of a normal **endmqm** command.

The queue manager might stop without waiting for applications to disconnect or for MQI calls to complete. This can give unpredictable results for IBM MQ applications. The shutdown mode is set to *immediate shutdown*. If the queue manager has not stopped after a few seconds, the shutdown mode is escalated, and all remaining queue manager processes are stopped.

The effect on any client applications connected through a server-connection channel is equivalent to a STOP CHANNEL command issued in TERMINATE mode.

-r Start trying to reconnect reconnectable clients. This parameter has the effect of reestablishing the connectivity of clients to other queue managers in their queue manager group.

-s Switch over to a standby queue manager instance after shutting down. The command checks that there is a standby instance running before ending the active instance. It does not wait for the standby instance to start before ending.

Connections to the queue manager are broken by the active instance shutting down. Reconnectable clients start trying to reconnect.

You can configure the reconnection options of a client to reconnect only to another instance of the same queue manager, or to reconnect to other queue managers in the queue manager group.

-w Wait shutdown.

This type of shutdown is equivalent to a controlled shutdown except that control is returned to you only after the queue manager has stopped. You receive the message `Waiting for queue manager` *qmName* `to end` while shutdown progresses. In the unlikely event that a "dspmq (display queue managers)" on page 236 command is issued in the small timeframe between the applications disconnecting and the queue manager actually stopping, the "dspmq (display queue managers)" on page 236 command might transiently report the status as `Ending immediately`, even though a controlled shutdown was requested.

The effect on any client applications connected through a server-connection channel is equivalent to a STOP CHANNEL command issued in QUIESCE mode.

-x End a standby instance of the queue manager, without ending the active instance of the queue manager.

-z Suppresses error messages on the command.

Return codes

Return code	Description
0	Queue manager ended
3	Queue manager being created
16	Queue manager does not exist
40	Queue manager not available
49	Queue manager stopping
58	Inconsistent use of installations detected
62	The queue manager is associated with a different installation
69	Storage not available
71	Unexpected error
72	Queue manager name error
77	IBM MQ queue manager cannot switch over
79	Active instance of IBM MQ queue manager *QmgrName* not ended

Return code	Description
90	Standby instance of IBM MQ queue manager *QmgrName* not ended
119	Permission denied

Examples

The following examples show commands that stop the specified queue managers.

1. This command ends the queue manager named `mercury.queue.manager` in a controlled way. All applications currently connected are allowed to disconnect.

   ```
   endmqm mercury.queue.manager
   ```

2. This command ends the queue manager named `saturn.queue.manager` immediately. All current MQI calls complete, but no new ones are allowed.

   ```
   endmqm -i saturn.queue.manager
   ```

The results of issuing **endmqm** to the local instance of a multi-instance queue manager are shown in Table 69. The results of the command depend on whether the -s or -x switch is used, and the running status of local and remote instances of the queue manager.

Table 69. endmqm actions.

endmqm option	Local machine	Remote machine	RC	Message	Result
	Active	None	0	-	Queue manager ended.
		Standby			Queue manager ended, including the standby instance.
	Standby	Active	90	AMQ8368	Standby instance of IBM MQ queue manager *QmgrName* not ended.
-s	Active	None	77	AMQ7276	IBM MQ queue manager cannot switch over.
		Standby	0	-	Queue manager QMNAME ended, permitting switchover to a standby instance.
	Standby	Active	90	AMQ8368	Standby instance of IBM MQ queue manager *QmgrName* not ended.
-x	Active	None	79	AMQ8367	Active instance of IBM MQ queue manager *QmgrName* not ended.
		Standby			
	Standby	Active	0	-	Standby instance of queue manager QMNAME ended.

Related reference:

crtmqm (create queue manager)
Create a queue manager.

endmqm (end queue manager)
Stop a queue manager or switch to a standby queue manager.

dltmqm (delete queue manager)
Delete a queue manager.

Related information:

Stopping a queue manager

Stopping a queue manager manually

Applying maintenance level updates to multi-instance queue managers on Windows

Applying maintenance level updates to multi-instance queue managers on UNIX and Linux

endmqsvc

End the IBM MQ service on Windows.

Purpose

The command ends the IBM MQ service on Windows. Run the command on Windows only.

If you are running IBM MQ on Windows systems with User Account Control (UAC) enabled, you must invoke **endmqsvc** with elevated privileges.

Run the command to end the service, if the service is running.

Restart the service for IBM MQ processes to pick up a new environment, including new security definitions.

Syntax

endmqsvc

Parameters

The **endmqsvc** command has no parameters.

You must set the path to the installation that contains the service. Either make the installation primary, run the **setmqenv** command, or run the command from the directory containing the **endmqsvc** binary file.

Related reference:
"strmqsvc (start IBM MQ service)" on page 350
Start the IBM MQ service on Windows.

endmqtrc (end trace)

End trace for some or all of the entities that are being traced.

Purpose

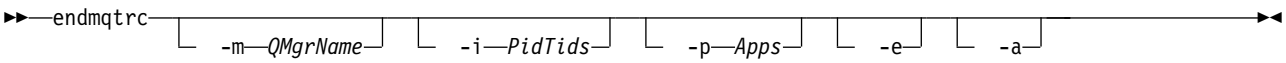
Use the **endmqtrc** command to end tracing for the specified entity or all entities. The **endmqtrc** command ends only the trace that is described by its parameters. Using **endmqtrc** with no parameters ends early tracing of all processes.

All **endmqtrc** commands set the type of output to *mqm* on strmqtrc.

Attention: There can be a slight delay between the **endmqtrc** command ending, and all trace operations actually completing. This is because IBM MQ processes are accessing their own trace files. As each process becomes active at different times, their trace files close independently of one another.

Syntax

The syntax of this command is as follows:



Optional parameters

-m ***QMgrName***

The name of the queue manager for which to end tracing. NSS Client This parameter applies to server products only.

The *QMgrName* supplied must match exactly the *QMgrName* supplied on the **strmqtrc** command. If the **strmqtrc** command used wildcards, the **endmqtrc** command must use the same wildcard specification including the escaping of any wildcard characters to prevent them being processed by the command environment.

A maximum of one -m flag and associated queue manager name can be supplied on the command.

-i ***PidTids***

Process identifier (PID) and thread identifier (TID) for which to end tracing. You cannot use the **-i** flag with the **-e** flag. If you try to use the **-i** flag with the **-e** flag, then an error message is issued. This parameter must only be used under the guidance of IBM Service personnel.

-p ***Apps***

The named processes for which to end tracing. *Apps* is a comma-separated list. You must specify each name in the list exactly as the program name would be displayed in the "Program Name" FDC header. Asterisk (*) or question mark (?) wildcards are allowed. You cannot use the **-p** flag with the **-e** flag. If you try to use the **-p** flag with the **-e** flag, then an error message is issued.

-e Ends early tracing of all processes.

Using **endmqtrc** with no parameters has the same effect as **endmqtrc -e**. You cannot specify the **-e** flag with the **-m** flag, the **-i** flag, or the **-p** flag.

-a Ends all tracing.

Important: This flag must be specified alone.

Return codes

Return code	Description
AMQ5611	This message is issued if you supply invalid arguments to the command.
58	Inconsistent use of installations detected

Examples

This command ends tracing of data for a queue manager called QM1.

```
endmqtrc -m QM1
```

The following examples are a sequence that shows how the endmqtrc command ends only the trace that is described by its parameters.

1. The following command enables tracing for queue manager QM1 and process amqxxx.exe:
   ```
   strmqtrc -m QM1 -p amqxxx.exe
   ```
2. The following command enables tracing for queue manager QM2:
   ```
   strmqtrc -m QM2
   ```
3. The following command ends tracing for queue manager QM2 only. Tracing of queue manager QM1 and process amqxxx.exe continues:
   ```
   endmqtrc -m QM2
   ```

Related commands

Command	Description
dspmqtrc	Display formatted trace output
"strmqtrc (Start trace)" on page 355	Start trace

Related reference:

Command sets comparison: Other commands
A table of other commands, showing the command description, and its PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

endmqweb (end mqweb server)

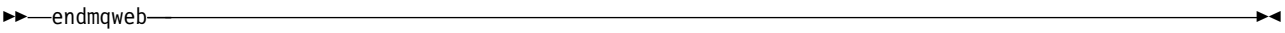
V 9.0.1

Stop the mqweb server that is used to support the IBM MQ Console and REST API.

Purpose

Use the **endmqweb** command to stop the mqweb server. If you stop the mqweb server, you cannot use the IBM MQ Console or the REST API.

Syntax



Optional parameters

None.

Return codes

Return code	Description
0	Command successful
>0	Command not successful.

For a full list of server command exit codes, see Liberty:server command options in the WebSphere Application Server documentation.

Related commands

Command	Description
dspmqweb	Display the status of the mqweb server.
strmqweb	Start the mqweb server.

migmqlog (migrate IBM MQ logs)

ULW V 9.0.4

The **migmqlog** command migrates logs, and can also change the type of your queue manager logs from linear to circular or from circular to linear.

IBM i z/OS **migmqlog** is not supported on IBM i or z/OS.

Usage notes

Windows On Windows, running **migmqlog** enables you to move your queue manager logs to an Advanced Format disk

migmqlog can only run when the queue manager is inactive.

If the running of **migmqlog** is interrupted by, for example, a power failure, you should be rerun the same command until it completes normally.

A partially migrated log can not be used to start a queue manager, and the result of attempting to do so is not well defined.

migmqlog migrates logs 'in place', or migrates logs to a new location. When logs are migrated to a new log location, no change is made to any existing log files and all valid recovery log files in the old location are migrated to the new location.

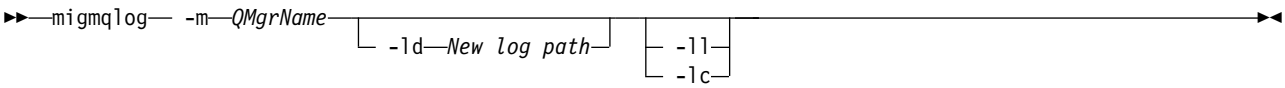
migmqlog updates the `qm.ini` file to reflect the new log configuration, that is, **LogType** and **LogPath**, as needed.

Following any log migration, the log is configured such that all future log writes occur with 4096 byte alignment, at minimum.

Windows For further information on migrating logs on Windows to be Advanced Format, see Migrating logs to an Advanced Format disk.

See Types of logging for more information about linear and circular logging.

Syntax



Required parameters

-m *QMgrName*
The name of the queue manager on which to migrate logs.

Optional parameters

-ld *New log path*
If you specify **-ld** and do not point to the existing log location, migration will be to a new log location.

If you do not specify **-ld**, or you specify **-ld** and point to the existing log location, migration will be 'in place'.

-ll
If you pass **-ll** to the command, and the queue manager is currently defined to be using circular logging, the queue manager will be reconfigured to use linear logging.

-lc
If you pass **-lc** to the command, and the queue manager is currently defined to be using linear logging, the queue manager will be reconfigured to use circular logging.

Related information:

Migrating the log of your queue manager from linear to circular

Migrating the log of your queue manager from circular to linear

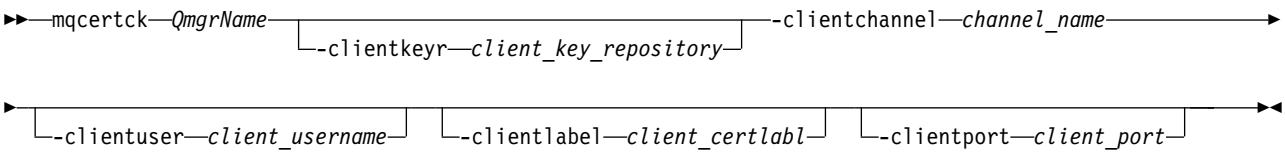
mqcertck (certify TLS setup)

Use the **mqcertck** command to diagnose potential TLS problems with your queue managers.

Purpose

The command can be used as a first check to determine why a connection using TLS has been unable to successfully connect on queue managers within your enterprise, and works with multiple certificates.

Syntax



Required parameters

QmgrName
Name of the queue manager to check for TLS errors.

-clientchannel ***channel_name***
Name of the channel on the referenced queue manager to check for TLS errors.

Optional parameters

-clientkeyr ***client_key_repository***
Required if you supplied the **-clientuser**, **-clientlabel**, or **-clientport** parameters.

Location of the client key repository that would be used by a client application connecting to the referenced queue manager.

Important: You must supply the name without the `.kdb` extension.

-clientuser ***client_username***
Cannot be used if you supplied the **-clientlabel** parameter.

User running the client application, connecting to the referenced queue manager, if the client application is not using the client CERTLABL attribute to supply a certificate label.

-clientlabel ***client_certlabl***
Cannot be used if you supplied the **-clientuser** parameter.

Certificate label that is given to the client, connecting to the referenced queue manager, using one of the IBM MQ MQI client CERTLABL methods.

-clientport ***client_port***
Specify a specific port to use when testing the client.

The value must be:

- An integer value between 1 and 65535 inclusive.
- A port number, which must be a free port that **mqcertck** can use during its client checks.
- Not be a port that is in use by the queue manager, or any other process on the machine running **mqcertck**.

If you do not specify a value, port 5857 is used.

Examples

Example 1

After configuring an IBM MQ queue manager for TLS connections, you can use **mqcertck** to verify that no mistakes have been made, prior to attempting to start your channels.

To do this, run the command:

```
mqcertck QmgrName
```

where *Qmgrname* is the name of your queue manager, and check the output for any problems identified with your configuration.

Example 2

After creating a key repository, certificate, and exchanging certificates for a client application, you can use **mqcertck** to verify that a client application is able to connect to a queue manager.

To do this, you need to run **mqcertck** on the machine where the IBM MQ queue manager is running, and have access to the client key repository.

You can do this in a variety of ways, for example, a file system mount. After you have set up your machine, run the following command:

```
mqcertck QMGR Name -clientkeyr Location of Client Key Repository
                   -clientlabel Client certificate label
```

Check the output for any problems identified with your configuration.

Note, that if you are planning on having your clients connect anonymously, you can run the preceding command without the **-clientlabel** parameter.

mqconfig (check system configuration)

UNIX Linux

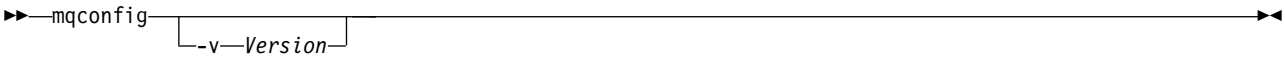
Checks that the system configuration meets the requirements to run IBM MQ (UNIX and Linux platforms only).

Purpose

The **mqconfig** command is run to verify the system configuration matches or exceeds that which is required by IBM MQ. The configuration values are minimum values, and large installations might require values greater than those checked by this command.

For further information about configuring your system for IBM MQ, see the *Operating system configuration and tuning information for IBM MQ* on the platform, or platforms, that your enterprise uses.

Syntax



Optional parameters

-v *Version*
The system requirements vary between different versions of IBM MQ. Specify the version of IBM MQ for which you need to verify the current system configuration.

The default value, if **-v** is not specified, is the current version.

Example

The following output is an example of what the command produces on a Linux system:

```
# mqconfig -v 8.0
mqconfig: V3.7 analyzing Red Hat Enterprise Linux Server release 6.5
(Santiago) settings for IBM MQ V8.0

System V Semaphores
semmsl   (sem:1) 500 semaphores           IBM>=32      PASS
semmns   (sem:2) 35 of 256000 semaphores  (0%)  IBM>=4096     PASS
semopm   (sem:3) 250 operations           IBM>=32      PASS
semmni   (sem:4) 3 of 1024 sets       (0%)  IBM>=128     PASS

System V Shared Memory
shmmax       68719476736 bytes         IBM>=268435456  PASS
shmmni       1549 of 4096 sets     (37%)  IBM>=4096     PASS
shmall       7464 of 2097152 pages   (0%)  IBM>=2097152   PASS

System Settings
file-max      4416 of 524288 files    (1%)  IBM>=524288    PASS

Current User Limits (root)
nofile    (-Hn) 10240 files            IBM>=10240    PASS
```

```
nofile    (-Sn) 10240 files            IBM>=10240    PASS
nproc    (-Hu) 11 of 30501 processes   (0%)  IBM>=4096     PASS
nproc    (-Su) 11 of 4096 processes    (1%)  IBM>=4096     PASS
```

Note: Any values listed in the `Current User Limits` section are resource limits for the user who ran **`mqconfig`**. If you normally start your queue managers as the `mqm` user, you should switch to `mqm` and run **`mqconfig`** there.

If other members of the `mqm` group (and perhaps root) also start queue managers, all those members should all run **`mqconfig`**, to ensure that their limits are suitable for IBM MQ.

Related information:

Configuring and tuning the operating system on Linux

MQExplorer (launch IBM MQ Explorer)

Windows Linux

Start IBM MQ Explorer (Windows and Linux x86-64 platforms only).

Purpose

To launch IBM MQ Explorer by using the system menu on Linux, or the start menu on Windows, you must left-click on the installation that you want to launch.

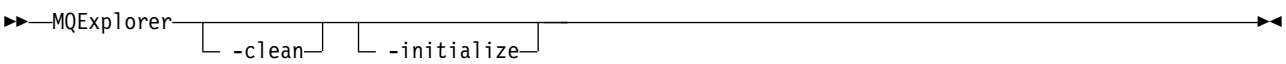
On Windows, open the start menu, and select the IBM MQ Explorer installation entry under the **IBM MQ** folder that corresponds to the installation that you want to launch. Each instance of IBM MQ Explorer listed is identified by the name that you chose for its installation.

On Linux, the system menu entry for IBM MQ Explorer is added to the **Development** category. Where it appears within the system menu is dependent on your Linux distribution (SUSE or Red Hat), and your desktop environment (GNOME or KDE).

- On SUSE
  - Left-click **Computer > More Applications...,** and find the installation of IBM MQ Explorer that you want to launch under the **Development** category.
- On Red Hat
  - The installation of IBM MQ Explorer that you want to launch can be found under **Applications> Programming**.

Syntax

The **MQExplorer** command is stored in `MQ_INSTALLATION_PATH/bin`. **MQExplorer.exe** (the MQExplorer command) supports standard Eclipse runtime options. The syntax of this command is as follows:



Optional parameters

`-clean`
Is passed to Eclipse. This parameter causes Eclipse to delete any cached data used by the Eclipse runtime.

`-initialize`
Is passed to Eclipse. This parameter causes Eclipse to discard configuration information used by the Eclipse runtime.

The graphical user interface (GUI) does not start.

mqrc (MQ return code)

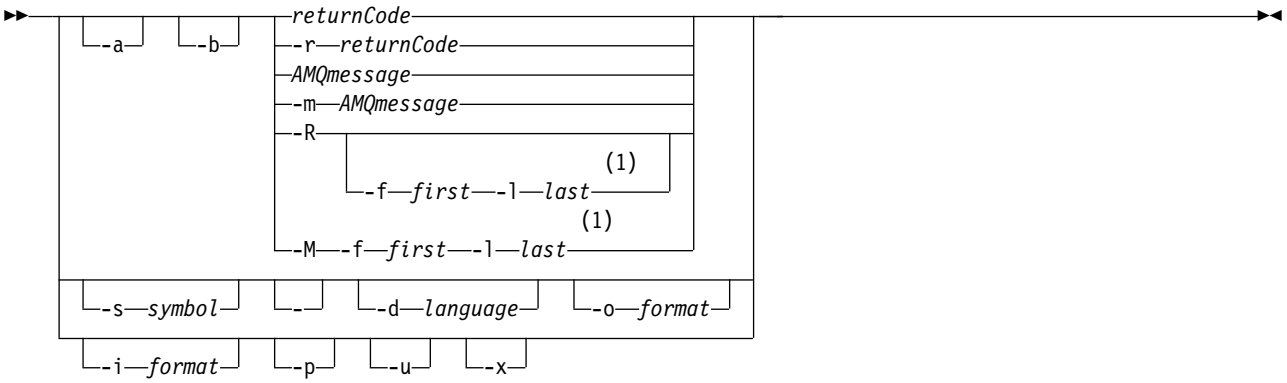
Display information about return codes.

Purpose

You can use the **mqrc** command to display information about symbols, return codes, and AMQ messages. You can specify a range of return codes or AMQ messages, as well as specifying specific return codes or AMQ messages.

Numeric arguments are interpreted as decimal if they start with a digit 1 - 9, or hex if prefixed with 0x.

Syntax



Notes:

1 If there is a problem with a message within a range, an indication is displayed before the message text. **?** is displayed if there are no matching return codes for the message. **!** is displayed if the message severity is different to the return code severity.

Parameters

returnCode
The return code to display

AMQmessage
The AMQ message to display

symbol
The symbol to display

-a Try all severities to find message text

-b Display messages without extended information

-f *first*
First number in a range

-l *last*
Last number in a range

-m *AMQmessage*
The AMQ message to list

-M Display AMQ messages in a range

-r *returnCode*
The return code to display

-R Display all return codes. If used with the **-f** and **-l** parameters, **-R** displays the return codes within a range.

-s *symbol*
The symbol to display

V 9.0.5 **-**
If a - is given as a trailing parameter, it indicates that further input will come from `stdin`.

V 9.0.4 **-d *language***
Display the message in the specified language, for example, `Fr_FR`.

V 9.0.4 **-i *format***
Determine the message to display from a message in the specified format, which must be one of the following:

text The textual format of the **QMErrorLog** service, including the Insert attributes.

V 9.0.5

json JSON format diagnostic messages, specified in UTF-8.

V 9.0.4 **-o *format***
Display the message in the specified format, which must be one of the following:

mqrc The format used by **mqrc** in previous versions of the product.

text The textual format of the **QMErrorLog** service.

V 9.0.5

json The JSON format, described in JSON format diagnostic messages.

V 9.0.4 **-p**
Display the message explanation only. For example:

```
mqrc -p AMQ8118
```

displays

```
The queue manager insert_5 does not exist.
```

V 9.0.4 **-u**
Display the user response only. For example:

```
mqrc -u AMQ8118
```

displays

```
Either create the queue manager (crtmqm command) or correct the queue manager name used in the command
and then try the command again.
```

V 9.0.4 **-x**
Display extended message information, including the message severity. For example, the following message has an error (**E**) severity of 30:

```
mqrc -x AMQ8118
536903960  0x20008118  E  30  urcMS_MQCONN_FAILED
536903960  0x20008118  E  30  zrc_CSPRC_Q_MGR_DOES_NOT_EXIST


MESSAGE:
IBM MQ queue manager does not exist.
```

```
EXPLANATION:
The queue manager <insert three> does not exist.

ACTION:
Either create the queue manager (crtmqm command) or correct the queue manager name used in the command
and then try the command again.
```

Examples

1. This command displays AMQ message 5005:

   ```
   mqrc AMQ5005
   ```

2. This command displays return codes in the range 2505 - 2530:

   ```
   mqrc -R -f 2505 -l 2530
   ```

3. V 9.0.5 Running the following command, where `AMQERR01.json` contains JSON formatted messages in any language, converts all messages into US English in the original textual **QMErrorLog** format:

   ```
   cat AMQERR01.json | mqrc -d En_US -i json -o text -
   ```

   Alternatively, you could take `AMQERR01.LOG` and convert it to JSON:

   ```
   cat AMQERR01.LOG | mqrc -i text -o json -
   ```

4. V 9.0.4 Running the following command, where `AMQERR01.LOG` contains text formatted messages in any language, converts messages into US English:

   ```
   cat AMQERR01.LOG | mqrc -d En_US -i text -o text -
   ```

rcdmqimg (record media image)

Write the image of an object or group of objects to the log for media recovery.

Purpose

Use the **rcdmqimg** command to write an image of an object, or group of objects, to the log for use in media recovery. This command can be used only when using linear logging. See Types of logging for more information about linear logging. Use the associated command **rcrmqobj** to re-create the object from the image.

V 9.0.2 Before IBM MQ Version 9.0.2, or when using **LogManagement**=*Manual*, the command does not run automatically as it must be run in accordance with, and as determined by, the usage of each individual customer of IBM MQ.

V 9.0.2 After IBM MQ Version 9.0.2, when using **LogManagement**=*Automatic* or *Archive*, the queue manager automatically records media images, however **rcdmqimg** can also be run manually as well, if required.

Running **rcdmqimg** moves the log sequence number (LSN) forwards and frees up old log files for archival or deletion.

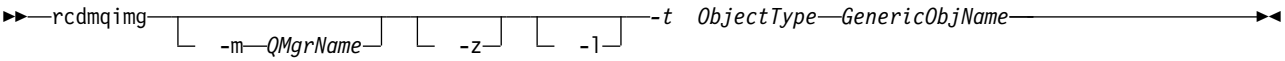
When determining when and how often to run **rcdmqimg**, consider these factors:

Disk space
: If disk space is limited, regular running of **rcdmqimg** releases log files for archive or deletion.

Impact on normal system performance
: **rcdmqimg** activity can take a long time if the queues on the system are deep. At this time, other system usage is slower and disk utilization increases because data is being copied from the queue files to the logs. Therefore, the ideal time to run **rcdmqimg** is when the queues are empty and the system is not being heavily used.

You use this command with an active queue manager. Further activity on the queue manager is logged so that, although the image becomes out of date, the log records reflect any changes to the object.

Syntax



Required parameters

GenericObjName
The name of the object to record. This parameter can have a trailing asterisk to record that any objects with names matching the portion of the name before the asterisk.

This parameter is required unless you are recording a queue manager object or the channel synchronization file. Any object name you specify for the channel synchronization file is ignored.

-t *ObjectType*
The types of object for which to record images. Valid object types are:

Table 70. Valid object types

Object Type	Description
all and *	All the object types; **ALL** for objtype and * for GenericObjName
authinfo	Authentication information object, for use with TLS channel security
channel or **chl**	Channels
clntconn or **clcn**	Client connection channels
catalog or **ctlg**	An object catalog
listener or **lstr**	Listeners
namelist or **nl**	Namelists
process or **prcs**	Processes
queue or **q**	All types of queue
qalias or **qa**	Alias queues
qlocal or **ql**	Local queues
qmodel or **qm**	Model queues
qremote or **qr**	Remote queues
qmgr	Queue manager object
service or **srvc**	Service
syncfile	Channel synchronization file.
topic or **top**	Topics

Note: UNIX When using IBM MQ for UNIX systems, you must prevent the shell from interpreting the meaning of special characters, for example, an asterisk (*). How you do this depends on the shell you are using, but might involve the use of single quotation marks ('), double quotation marks ("), or a backslash (\).

Optional parameters

-m *QMgrName*
The name of the queue manager for which to record images. If you omit this parameter, the command operates on the default queue manager.

-z Suppresses error messages.

-l Writes messages containing the names of the oldest log files required to restart the queue manager and to perform media recovery. The messages are written to the error log and the standard error destination. (If you specify both the -z and -l parameters, the messages are sent to the error log, but not to the standard error destination.)

When issuing a sequence of **rcdmqimg** commands, include the -l parameter only on the last command in the sequence, so that the log file information is gathered only once.

Return codes

Return code	Description
0	Successful operation
26	Queue manager running as a standby instance.
28 V 9.0.2	Object not media recoverable.
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
58	Inconsistent use of installations detected
68	Media recovery not supported
69	Storage not available
71	Unexpected error
72	Queue manager name error
119	User not authorized
128	No objects processed
131	Resource problem
132	Object damaged
135	Temporary object cannot be recorded

Examples

The following command records an image of the queue manager object `saturn.queue.manager` in the log.

```
rcdmqimg -t qmgr -m saturn.queue.manager
```

Related commands

Command	Description
rcrmqobj	Re-create a queue manager object

rdqmadm (administer replicated data queue manager cluster)

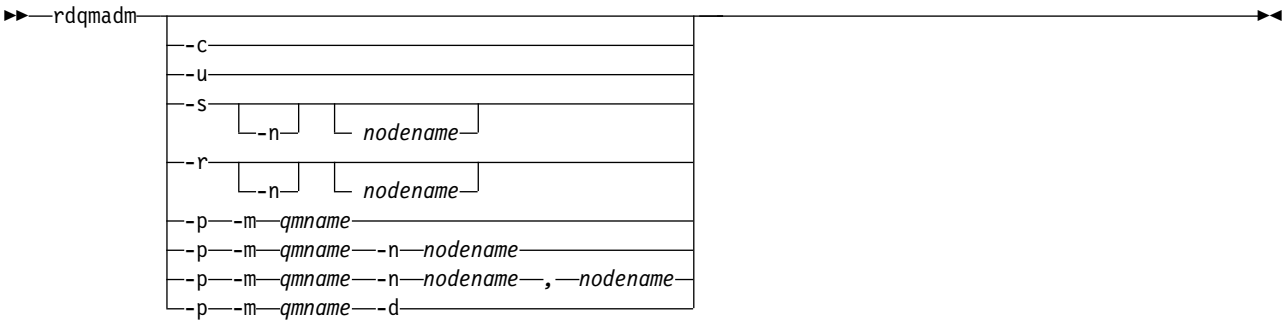
Linux V 9.0.4

Administer the cluster in a high availability RDQM configuration.

Purpose

Use the **rdqmadm** command to administer the Pacemaker cluster used in RDQM high availability configurations. (This command is not require for disaster recovery RDQM configurations.)

Syntax



Optional parameters

-c Initialize the Pacemaker cluster, using the settings specified in the `/var/mqm/rdqm.ini` file. The same command must be run on each of the three nodes by the `root` user. (You can also run this command as a user in the `mqm` group if you have configured sudo, see Requirements for RDQM HA solution.) The command fails if the node is already part of a Pacemaker cluster. A node cannot be a member of two Pacemaker clusters.

-u Delete the Pacemaker cluster configuration. The same command must be run on each of the three nodes by the `root` user. (You can also run this command as a user in the `mqm` group if you have configured sudo, see Requirements for RDQM HA solution.) The Pacemaker cluster configuration cannot be deleted if any replicated data queue managers (RDQMs) exist.

-s [-n *nodename*]
Suspend the local node (or the specified node if the **-n** *nodename* argument is supplied). The command can be run on any of the three nodes by a user in the haclient group, or by `root`. The node is taken offline. Any replicated data queue managers (RDQMs) running on that node are stopped and restarted on an active node. Queue manager data does not replicate to the offline node. The command fails if the specified node is the last active node.

-r [-n *nodename*]
Resume the local or specified node. The command can be run on any of the three nodes by a user in the haclient group, or by `root`. The node is brought online. If the node is the preferred location for any replicated data queue managers (RDQMs), the queue managers are stopped and restarted on this node.

-p -m *qmname* [-n *nodename*[,*nodename*]
Assign the local or specified node as the Preferred Location for the named queue manager. If the Pacemaker cluster is in a normal state and the Preferred Location is not the current primary node, the queue manager is stopped and restarted on the new Preferred Location. You can specify a comma-separated list of two node names to assign a second preference of Preferred Location.

-p -m *qmname* -d
Clear the Preferred Location so that the queue manager does not automatically return to a node when it is restored.

rdqmdr (manage DR RDQM instances)

Linux V 9.0.5

Change a primary disaster recovery replicated data queue manager (DR RDQM) to a secondary instance, or change a secondary instance to a primary.

Purpose

Use the **rdqmdr** command to control whether an instance of a DR RDQM has the primary or secondary role.

You can also use **rdqmdr** on the node where you created a primary DR RDQM to retrieve the command that you need to create the secondary instance on the recovery node.

Syntax

Parameters

-m *qmname*
Specify the name of the DR RDQM that you are issuing the command for.

-s Specify **-s** to make a DR RDQM that is currently in the primary role into the secondary.

-p
Specify **-p** to make a DR RDQM that is currently in the secondary role into the primary. This command fails if the primary instance of the queue manager is still running and the DR replication link is still functioning.

-d Specify **-d** to return the **crtmqm** command required to create a secondary instance of the specified DR RDQM.

rdqmint (add or delete floating IP address for RDQM)

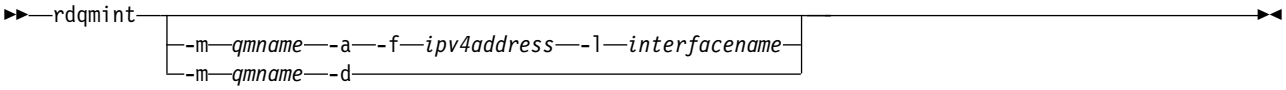
Linux V 9.0.4

Add or delete the floating IP address used to connect to a high availability replicated data queue manager (HA RDQM).

Purpose

Use the **rdqmint** command to add or delete the floating IP address that is used to connect to an HA RDQM regardless of which node in the high availability (HA) group is actually running the RDQM. (This command is not applicable to disaster recovery RDQM configurations.)

Syntax



Optional parameters

-m *qmname*
Specify the name of the RDQM for which you are adding or deleting a floating IP address.

-a Specify this option to add a floating IP address.

-d Specify this option to delete a floating IP address.

-f *ipv4address*
The IP address in dot decimal format.

-l *interfacename*
The name of the physical interface that the floating IP address is bound to.

To specify a floating IP address for the queue manager RDQM1, enter the following command:

```
rdqmint -m RDQM1 -a 192.168.7.5 -l MQCLI
```

To delete the floating IP address for the queue manager RDQM1, enter the following command:

```
rdqmint -m qmname -d
```

rdqmstatus (display RDQM status)

Linux V 9.0.4

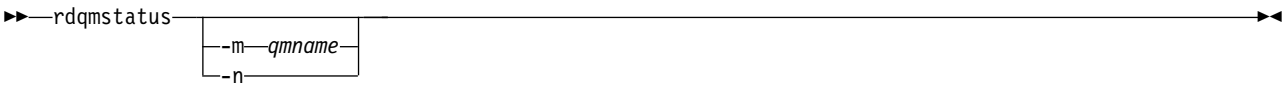
Display the status of all the replicated data queue managers (RDQMs) on a node or detailed status of specified individual RDQMs. You can also display the online/offline status of the nodes in an HA group.

Purpose

Use the **rdqmstatus** command on its own to view RDQM status on a node. You can specify a queue manager name to view detailed status for that RDQM. You can also view the availability status of all nodes in an HA group.

You can enter the command on any node in the Pacemaker cluster or DR pair.

Syntax



Optional parameters

-m *qmname*
Specify the name of the RDQM for which you are requesting status.

-n Specify `-n` to list the three nodes in the HA group, and their current online or offline status.

Related information:

Linux V 9.0.4 Viewing RDQM and HA group status

Linux V 9.0.4 Viewing DR RDQM status

rcrmqobj (re-create object)

Re-create an object, or group of objects, from their images contained in the log.

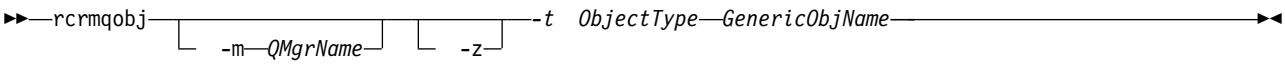
Purpose

Use the **rcrmqobj** command to re-create an object, or group of objects, from their images.

- With *ObjectType* argument of `clchltab` or `syncfile`, this command re-creates the object files from the internal queue manager state.
- For other *ObjectType* arguments, the command can only be used when the queue manager is configured to use linear logging. Use the associated command, `rcdmqimg`, to record the object images to the log. The object is re-created from images in the log.

  Use this command on a running queue manager. All activity on the queue manager after the image was recorded is logged. To re-create an object, replay the log to re-create events that occurred after the object image was captured.

Syntax



Required parameters

GenericObjName

The name of the object to re-create. This parameter can have a trailing asterisk to re-create any objects with names matching the portion of the name before the asterisk.

This parameter is required, unless the object type is the channel synchronization file; any object name supplied for this object type is ignored.

-t *ObjectType*

The types of object to re-create. Valid object types are:

Table 71. Valid object types.

Object Type	Description
`*` or `all`	All object types
`authinfo`	Authentication information object, for use with TLS channel security
`channel` or `chl`	Channels
`clntconn` or `clcn`	Client connection channels
`clchltab`	Client channel table
`comminfo`	Communication information object
`listener` or `lstr`	Listener
`namelist` or `nl`	Namelists
`process` or `prcs`	Processes
`queue` or `q`	All types of queue
`qalias` or `qa`	Alias queues

Table 71. Valid object types. (continued)

Object Type	Description
`qlocal` or `ql`	Local queues
`qmodel` or `qm`	Model queues
`qremote` or `qr`	Remote queues
`service` or `srvc`	Service
`syncfile`	Channel synchronization file. You can use this option when circular logs are configured but the `syncfile` fails if the channel scratchpad files, which are used to rebuild `syncfile`, are damaged or missing. You might want to do this if your system has reported the error message AMQ7353 (krcE_SYNCFILE_UPDATE_FAILED).
`topic` or `top`	Topics

Note: UNIX When using IBM MQ for UNIX systems, you must prevent the shell from interpreting the meaning of special characters, for example, an asterisk (*). How you do this depends on the shell you are using, but might involve the use of single quotation marks ('), double quotation marks ("), or a backslash (\).

Optional parameters

-m ***QMgrName***
The name of the queue manager for which to re-create objects. If omitted, the command operates on the default queue manager.

-z Suppresses error messages.

Return codes

Return code	Description
0	Successful operation
26	Queue manager running as a standby instance.
28 V 9.0.2	Object not media recoverable.
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
58	Inconsistent use of installations detected
66	Media image not available
68	Media recovery not supported
69	Storage not available
71	Unexpected error
72	Queue manager name error
119	User not authorized
128	No objects processed
135	Temporary object cannot be recovered
136	Object in use

Examples

1. The following command re-creates all local queues for the default queue manager:

```
rcrmqobj -t ql *
```

2. The following command re-creates all remote queues associated with queue manager `store`:

```
rcrmqobj -m store -t qr *
```

Related commands

Command	Description
rcdmqimg	Record an object in the log

rmvmqinf (remove configuration information)

UNIX Windows

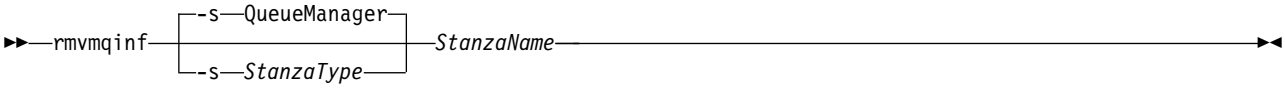
Remove IBM MQ configuration information (UNIX and Windows only).

Purpose

Use the **rmvmqinf** command to remove IBM MQ configuration information.

You must use the **rmvmqinf** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

Syntax



Required parameters

StanzaName
The name of the stanza. That is, the value of the key attribute that distinguishes between multiple stanzas of the same type.

Optional parameters

-s *StanzaType*
The type of stanza to remove. If omitted, a QueueManager stanza is removed.

The only supported value of *StanzaType* is `QueueManager`.

Return codes

Return code	Description
0	Successful operation
5	Queue manager is running
26	Queue manager is running as a standby instance
39	Bad command line parameters
44	Stanza does not exist
49	Queue manager is stopping
58	Inconsistent use of installations detected
69	Storage is not available
71	Unexpected error
72	Queue manager name error

Example

```
rmvmqinf QM.NAME
```

Usage notes

Use `rmvmqinf` to remove an instance of a multi-instance queue manager.

To use this command you must be an IBM MQ administrator and a member of the `mqm` group.

Related commands

Command	Description
"addmqinf" on page 196	Add queue manager configuration information
"dspmqinf (display configuration information)" on page 245	Display queue manager configuration information

rsvmqtrn (resolve transactions)

Resolve in-doubt and heuristically completed transactions

Purpose

The **rsvmqtrn** command is used to resolve two different transaction states.

in-doubt transactions

Use the **rsvmqtrn** command to commit or back out internally or externally coordinated in-doubt transactions.

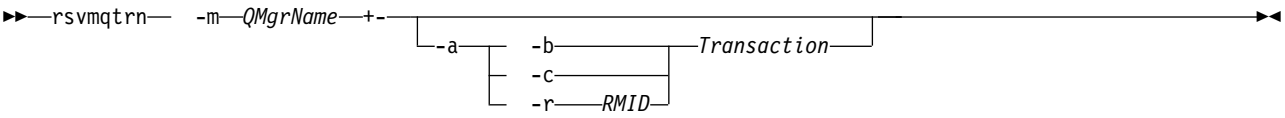
Note: Use this command only when you are certain that transactions cannot be resolved by the normal protocols. Issuing this command might result in the loss of transactional integrity between resource managers for a distributed transaction.

heuristically completed transactions

Use the **rsvmqtrn** command with the **-f** parameter for IBM MQ to remove all information about externally coordinated transactions that were previously resolved manually using the **rsvmqtrn** command, but the resolution has not been acknowledged by the transaction coordinator using the **xa-forget** command. Transactions that are manually resolved by a resource manager and unacknowledged by the transaction manager, are known as *heuristically completed* transactions by X/Open.

Note: Only use the **-f** option if the external transaction coordinator is permanently unavailable. The queue manager, as a resource manager, remembers the transactions that are committed or backed out manually by the `rsvmqtrn` command.

Syntax



Required parameters

-m *QMgrName*

The name of the queue manager.

Optional parameters

-a The queue manager resolves all internally coordinated, in-doubt transactions (that is, all global units of work).

-b Backs out the named transaction. This flag is valid for externally coordinated transactions (that is, for external units of work) only.

-c Commits the named transaction. This flag is valid for externally coordinated transactions (that is, external units of work) only.

-f Forgets the named heuristically completed transaction. This flag is valid only for externally coordinated transactions (that is, external units of work) that are resolved, but unacknowledged by the transaction coordinator.

Note: Use only if the external transaction coordinator is never going to be able to acknowledge the heuristically completed transaction. For example, if the transaction coordinator has been deleted.

-r *RMID*
The participation of the resource manager in the in-doubt transaction can be ignored. This flag is valid for internally coordinated transactions only, and for resource managers that have had their resource manager configuration entries removed from the queue manager configuration information.

Note: The queue manager does not call the resource manager. Instead, it marks the participation of the resource manager in the transaction as being complete.

Transaction
The transaction number of the transaction being committed or backed out. Use the **dspmqtrn** command to find the relevant transaction number. This parameter is required with the **-b**, **-c**, and **-r** *RMID* parameters and if used it must be the last parameter.

Return codes

Return code	Description
0	Successful operation
26	Queue manager running as a standby instance.
32	Transactions could not be resolved
34	Resource manager not recognized
35	Resource manager not permanently unavailable
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
58	Inconsistent use of installations detected
69	Storage not available
71	Unexpected error
72	Queue manager name error
85	Transactions not known

Related commands

Command	Description
dspmqtrn	Display list of prepared transactions

runmqbcb (run IBM MQ Bridge to blockchain)

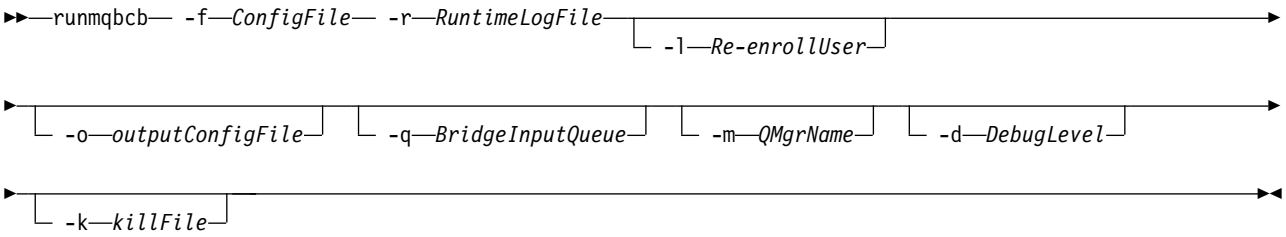
Linux z/OS V 9.0.3 MQ Adv. VUE

Configure and run the IBM MQ Bridge to blockchain.

- Syntax
- Usage notes
- Command line parameters
- Configuration parameters

Syntax

The diagram shows the syntax for the **runmqbcb** command usage as described in note 1.



Usage notes

1. You can run the **runmqbcb** command to start the IBM MQ Bridge to blockchain and connect to IBM Blockchain and IBM MQ. When the connections are made, the bridge is ready to receive and process query messages that are put on the queue manager input queue, send the correctly formatted queries and updates to the blockchain network, receive, process and put replies from the blockchain to the reply queue.

   ```
   runmqbcb -f ConfigFile -r RuntimeLogFile -m QMgrName -d DebugLevel -k killFile -r RuntimeLogFile -l Re-enrollUser
   ```

   When you use the command for runtime processing, the required parameters are **-f**, with the name of the previously created configuration file, and **-r** with the name of the log file. When the other command parameters are also given on the command line, they override the values in the configuration file. The same configuration file can be used by multiple bridges.
2. You can also use the **runmqbcb** command to generate a configuration file that is used to define the parameters that are needed for the bridge to connect to IBM Blockchain and IBM MQ.

   When you are creating the configuration file, the **-f** parameter is optional, the input configuration file `bcbConfig.json` is included in the IBM MQ Bridge to blockchain `samp` directory.

   ```
   runmqbcb -f inputConfigFile -o outputConfigFile
   ```

   When you run the command in this way, you are prompted to enter values for each of the configuration parameters. To keep an existing value press `Enter`. To remove an existing value press `Space`, then `Enter`. For more information, see “Configuration parameters” on page 295.

Command line parameters

-f ConfigFile

Configuration file. The **-f** parameter is required when you are running the **runmqbcb** command to start the IBM MQ Bridge to blockchain, as described in usage note 1. You can optionally use the **-f**

parameter to reuse some of the values from an existing *inputConfigFile*, as described in usage note 2 on page 294, and also enter some of the new values. If you do not specify the **-f** parameter when you are creating the configuration file, all the values for the parameters you are prompted for are empty.

-r ***RuntimeLogFile***
Required. Location and name of the log file for trace information. You can specify the log file path and name in the configuration file or on the command line.

-l ***ReenrollUser***
The **ReenrollUser** flag is used to force re-enrolment or password checking and credential download for the user. This is useful if you want to move to a different blockchain network but still use the same user and organization names, new credentials are then required and the process forces a discard of the old stored values.

-o ***outputConfigFile***
New configuration file. When you run the command with the **-o** parameter, **runmmbcb** command loads existing configuration values from the **-f** file and prompts for new values for each configuration parameter.

-q ***BridgeInputQueue***
Name of the queue that the bridge waits for messages on.

-m ***QMgrName***
Queue manager name.

-d ***debugLevel***
Debug level, `1`, or `2`.

1 Terse debug information is displayed.

2 Verbose debug information is displayed.

-k ***killFile***
A file to cause the bridge to exit. When you run the command with the **-k** parameter and specify a file, if the file exists, it causes the bridge program to exit. Using this file is an alternative way to stop the program when you don't want to use `Ctrl+C` or **kill** command. The file is deleted by the bridge on startup in case it exists. If the deletion fails, the bridge abends but monitors for the recreation of the file.

Configuration parameters

When you run the **runmqbcb** command to create the configuration file, the parameters are stepped through in six groups. Passwords are obfuscated and are not displayed as you type. The generated configuration file is in JSON format. You must use the **runmqbcb** command to create the configuration file. You cannot edit the passwords and security certificate information directly in the JSON file.

Connection to queue manager
Parameters relating to the IBM MQ queue manager.

IBM MQ Queue manager
Required. The z/OS queue manager that you are using with the IBM MQ Bridge to blockchain.

Bridge input queue

`SYSTEM.BLOCKCHAIN.INPUT.QUEUE` is the default queue where applications put request messages, this can be overridden in the configuration file or on the **runmqbcb** command line. User applications must have appropriate authorisation to put messages to this queue.

Bridge user identity queue
`SYSTEM.BLOCKCHAIN.IDENTITY.QUEUE` is used only by the bridge program to store the security credentials for the configured userid.

IBM MQ Channel
The bridge requires a svrcon channel to connect to the z/os queue manager remotely.

IBM MQ Conname
Uses standard connection name format of "host(port), host(port)" to enable multiple destinations such as for multi-instance queue managers.

IBM MQ CCDT URL
If a TLS connection is required to the queue manager, you must use a JNDI or CCDT definition.

JNDI implementation class name
The class name of your JNDI provider. The "queue manager name" parameter refers to the connection factory name when you are using JNDI.

JNDI provider URL
The endpoint of your JNDI service.

IBM MQ UserId
The **UserId** that is running the bridge must have permission to set identity context on the messages it sends as replies, these have the requester **UserId** set in the message. The bridge user must therefore have appropriate access to put to the reply queue.

IBM MQ Password
Password for the IBM MQ **UserId** that the bridge is using.

Blockchain - User identification
Parameters relating to blockchain user credentials that the bridge uses to connect to the IBM Blockchain network.

IBM Blockchain Userid
enrollID value from the credentials file from your IBM Blockchain network.

IBM Blockchain Enrollment Secret
enrollSecret value from the credentials file from your IBM Blockchain network.

Blockchain - Organisation identification
Parameters relating to the membership service provider (**MSPid**) that governs membership and identity rules for your blockchain network.

Organisation Name
MSPid name value from the credentials file from your blockchain network.

Organisation MSPId
MSPid value from the credentials file from your blockchain network.

Blockchain server locations
Parameters relating to the blockchain network certificate authority, peer, orderer, and peer event server addresses from your credential file and the location for the `.pem` certificate file.

Certificate Authority servers
From your blockchain network credentials file, provide the name, server (IP address) and port details for the certificate authority. For example:

```
ca.example.com Docker_container_host:7054 (for example ca.example.com localhost:7054)
```

or

```
CA1 your_blockchain_network_public_ip_address:30000 (for example CA1 123.456.789.10:30000)
```

Peer servers
From your blockchain network credentials file, provide the name, server (IP address) and port details for the peer servers. For example:

```
peer0 localhost:7051
```

or

```
blockchain-org1peer1 your_blockchain_network_public_ip_address:30110
```

Orderer servers
From your blockchain network credentials file, provide the name, server (IP address) and port details for the orderer servers. For example:

```
orderer0 localhost:7050
```

or

```
blockchain-orderer your_blockchain_network_public_ip_address:31010
```

Note: Include all the peer and orderer name-server:port values that appear in your credentials file.

Peer event servers
From your blockchain network credentials file, provide the name, server (IP address) and port details for the peer event servers. For example:

```
peer0 localhost:7053
```

or

```
blockchain-org1peer1 your_blockchain_network_public_ip_address:30111
```

Location of PEM file for IBM Blockchain certificate
When using a TLS connection to the Hyperledger Fabric instance, a single `PEM` file is used to hold the Hyperledger certificates to authenticate the bridge with the Hyperledger Fabric instance. This `PEM` file must be copied to the system where the IBM MQ Bridge to blockchain is running, and specified in the configuration file.

Certificate stores for TLS connections
Parameters relating to certificate stores for TLS connections.

Personal keystore for TLS certificates
Keystore for security certificates that are used for IBM MQ.

Keystore password
Password for the keystore.

Trusted store for signer certificates
If you do not add the trusted store, the personal keystore for TLS certificates is used.

Trusted store password
If the personal keystore for TLS certificates is used, this is the password for the keystore for TLS certificates.

Use TLS for MQ connection
The bridge can use TLS when it connects to the queue manager.

Timeout for Blockchain operations

If you don't provide a truststore parameter, the keystore is used for both roles. The stores can be the same as the one configured for the IBM MQ connection in the CCDT or JNDI.

Behavior of bridge program
Parameters relating to the behavior of the IBM MQ Bridge to blockchain.

Required. Runtime logfile for copy of stdout/stderr
Path to and name of the log file for the tracing information.

The configuration is only read on startup of the bridge process. Changes to the configuration require a restart, such as through the IBM MQ Service definitions.

runmqccred (obfuscate passwords for mqccred exit)

Obfuscate passwords in the `.ini` file used by the **mqccred** security exit.

Purpose

Use the **runmqccred** command to process the **mqccred** exit `.ini` file to change all plain text passwords into an obfuscated form. This command should be run before using the `.ini` with the exit to ensure the exit runs successfully.

Syntax

Optional Parameters

-f Specify a specific file to edit, other than the default file.

By default, the program locates the `.ini` file in the same way as the channel exit.

-p By default the program fails with an error, if the filemode enables others to access the file you edited.

Use the **-p** flag to continue processing even when the error appears.
This might be necessary in situations where you might, for example, have mounted a UNIX filesystem onto your Windows machine using NFS, or some other protocol, and are trying to use the `.ini` file from there (perhaps to share the same `.ini` file across multiple accounts).
Since NFS does not support the Windows NT FS Access Control Lists, the exit would fail unless you bypass the permissions check.

Usage notes

The **runmqccred** program locates the ini file in the same way as the channel exit. The program also writes console messages saying which file is being modified, and any success or failure status.

Note that the channel exit can work with either **Password** or **OPW** attributes, but the expectation is that you will protect passwords.

Important: The **runmqccred** program works only from IBM MQ Version 8.0 or later. You must run the program on a Version 8.0 or later system and then transfer the output `.ini` file manually to a system running a previous version if you want to use clients there.

By default the exit only works when there are no plain text passwords in the file. You can override this by using the **NOCHECKS SCYDATA** option.

The **runmqccred** program also checks that the `.ini` file does not have excessive permissions set that allow other users to access it. By default the program fails with an error if the filemode enables others to access it. Use the **-p** flag to continue processing even when the error appears.

The **runmqccred** program is installed in the following folder:

Windows **Windows platforms**

The *MQ_INSTALLATION_PATH*\Tools\c\Samples\mqccred\

UNIX **UNIX**

The *MQ_INSTALLATION_PATH*/usr/mqm/samp/mqccred/

If the file permissions are not secure enough **runmqccred** produces this message:

```
Configuration file 'C:\Users\User1\.mqs\mqccred.ini' is not secure.
Other users may be able to read it. No changes have been made to the file.
Use the -p option for runmqccred to bypass this error.
```

You can bypass this issue with the **-p** flag, but the exit will fail to run when put into production if you have not resolved this issue. When **runmqccred** runs successfully it informs you how many passwords have been obfuscated.

```
File 'C:\Users\User1\.mqs\mqccred.in' processed successfully.
Plaintext passwords found: 3
```

runmqchi (run channel initiator)

Run a channel initiator process to automate starting channels.

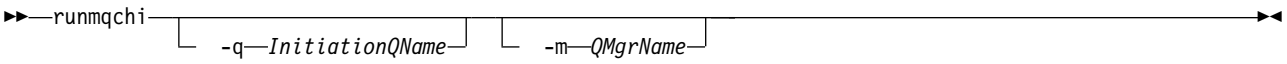
Purpose

Use the **runmqchi** command to run a channel initiator process.

You must use the **runmqchi** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the **dspmq -o installation** command.

The channel initiator is started by default as part of the queue manager.

Syntax



Optional parameters

-q *InitiationQName*
The name of the initiation queue to be processed by this channel initiator. If you omit it, SYSTEM.CHANNEL.INITQ is used.

-m *QMgrName*
The name of the queue manager on which the initiation queue exists. If you omit the name, the default queue manager is used.

Return codes

Return code	Description
0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

If errors occur that result in return codes of either 10 or 20, review the queue manager error log that the channel is associated with for the error messages, and the system error log for records of problems that occur before the channel is associated with the queue manager. For more information about error logs, see Error log directories.

runmqchl (run channel)

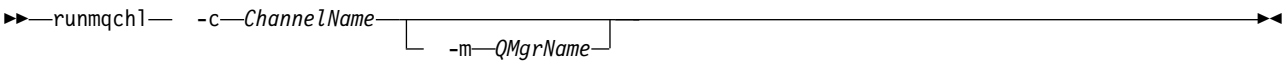
Start a sender or requester channel

Purpose

Use the **runmqchl** command to run either a sender (SDR) or a requester (RQSTR) channel.

The channel runs synchronously. To stop the channel, issue the MQSC command **STOP CHANNEL**.

Syntax



Required parameters

-c *ChannelName*
The name of the channel to run.

Optional parameters

-m *QMgrName*
The name of the queue manager with which this channel is associated. If you omit the name, the default queue manager is used.

Return codes

Return code	Description
0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

If return codes 10 or 20 are generated, review the error log of the associated queue manager for the error messages, and the system error log for records of problems that occur before the channel is associated with the queue manager.

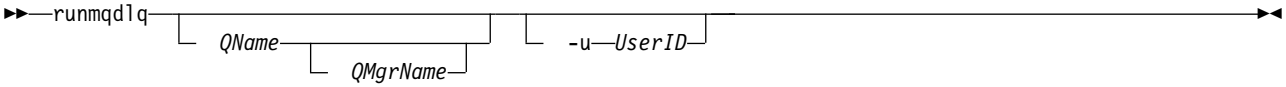
runmqdlq (run dead-letter queue handler)

Start the dead-letter queue handler to monitor and process messages on the dead-letter queue.

Purpose

Use the **runmqdlq** command to start the dead-letter queue (DLQ) handler, which monitors and handles messages on a dead-letter queue.

Syntax



Description

Use the dead-letter queue handler to perform various actions on selected messages by specifying a set of rules that can both select a message and define the action to be performed on that message.

The **runmqdlq** command takes its input from `stdin`. When the command is processed, the results and a summary are put into a report that is sent to `stdout`.

By taking `stdin` from the keyboard, you can enter **runmqdlq** rules interactively.

By redirecting the input from a file, you can apply a rules table to the specified queue. The rules table must contain at least one rule.

If you use the DLQ handler without redirecting stdin from a file (the rules table), the DLQ handler reads its input from the keyboard:

- UNIX Linux On UNIX and Linux, the DLQ handler does not start to process the named queue until it receives an end_of_file (Ctrl+D) character.
- Windows On Windows, the DLQ handler does not start to process the named queue until you press the following sequence of keys: Ctrl+Z, Enter, Ctrl+Z, Enter.

For more information about rules tables and how to construct them, see The DLQ handler rules table.

Optional parameters

The MQSC command rules for comment lines and for joining lines also apply to the DLQ handler input parameters.

QName
The name of the queue to be processed.

If you omit the name, the dead-letter queue defined for the local queue manager is used. If you enter one or more blanks (' '), the dead-letter queue of the local queue manager is explicitly assigned.

QMgrName
The name of the queue manager that owns the queue to be processed.

If you omit the name, the default queue manager for the installation is used. If you enter one or more blanks (' '), the default queue manager for this installation is explicitly assigned.

-u *UserID*
If you use the **-u** parameter to supply a user ID, you are prompted for a matching password.

If you have configured the `CONNAUTH AUTHINFO` record with `CHCKLOCL(REQUIRED)` or `CHCKLOCL(REQDADM)`, you must use the **-u** parameter otherwise you will not be able to start a dead-letter queue handler for your queue manager with **runmqdlq**.

If you specify this parameter and redirect `stdin`, a prompt will not be displayed and the first line of redirected input should contain the password.

runmqdnm (run .NET monitor)

Windows

Start processing messages on a queue using the .NET monitor (Windows only).

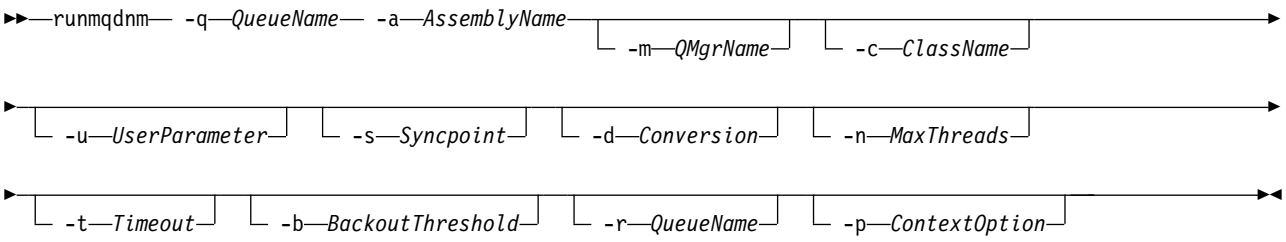
Purpose

Note: The **runmqdnm** command applies to IBM MQ for Windows only.

runmqdnm can be run from the command line, or as a triggered application.

Use the **runmqdnm** control command to start processing messages on an application queue with a .NET monitor.

Syntax



Required parameters

-q *QueueName*
The name of the application queue to monitor.

-a *AssemblyName*
The name of the .NET assembly.

Optional parameters

-m *QMgrName*
The name of the queue manager that hosts the application queue.

If omitted, the default queue manager is used.

-c *ClassName*
The name of the .NET class that implements the IMQObjectTrigger interface. This class must reside in the specified assembly.

If omitted, the specified assembly is searched to identify classes that implement the IMQObjectTrigger interface:

- If one class is found, then *ClassName* takes the name of this class.
- If no classes or multiple classes are found, then the .NET monitor is not started and a message is written to the console.

-u *UserData*
User-defined data. This data is passed to the Execute method when the .NET monitor calls it. User data must contain ASCII characters only, with no double quotation marks, NULLs, or carriage returns.

If omitted, null is passed to the Execute method.

-s *Syncpoint*
Specifies whether sync point control is required when messages are retrieved from the application queue. Possible values are:

Table 72. Syncpoint parameter values.

Value	Description
YES	Messages are retrieved under sync point control (MQGMO_SYNCPOINT).
NO	Messages are not retrieved under sync point control (MQGMO_NO_SYNCPOINT).
PERSISTENT	Persistent messages are retrieved under sync point control (MQGMO_SYNCPOINT_IF_PERSISTENT).

If omitted, the value of *Syncpoint* is dependent on your transactional model:

- If distributed transaction coordination (DTC) is being used, then *Syncpoint* is specified as YES.
- If distributed transaction coordination (DTC) is not being used, then *Syncpoint* is specified as PERSISTENT.

-d *Conversion*
Specifies whether data conversion is required when messages are retrieved from the application queue. Possible values are:

Table 73. Conversion parameter values.

Value	Description
YES	Data conversion is required (MQGMO_CONVERT).
NO	Data conversion is not required (no get message option specified).

If omitted, *Conversion* is specified as NO.

-n *MaxThreads*
The maximum number of active worker threads.

If omitted, *MaxThreads* is specified as 20.

-t *Timeout*
The time, in seconds, that the .NET monitor waits for further messages to arrive on the application queue. If you specify -1, the .NET monitor waits indefinitely.

If omitted when run from the command line, the .NET monitor waits indefinitely.

If omitted when run as a triggered application, the .NET monitor waits for 10 seconds.

-b *BackoutThreshold*
Specifies the backout threshold for messages retrieved from the application queue. Possible values are:

Table 74. BackoutThreshold parameter values.

Value	Description
-1	The backout threshold is taken from the application queue attribute, BOTHRESH.
0	The backout threshold is not set.
1 or more	Explicitly sets the backout threshold.

If omitted, *BackoutThreshold* is specified as -1.

-r *QueueName*
The queue to which messages, with a backout count exceeding the backout threshold, are put.

If omitted, the value of *QueueName* is dependent on the value of the BOQNAME attribute from the application queue:

- If BOQNAME is non-blank, then *QueueName* takes the value of BOQNAME.
- If BOQNAME is blank, then *QueueName* is specified as the queue manager dead letter queue. If a dead letter queue has not been assigned to the queue manager, then backout processing is not available.

-p *ContextOption*
Specifies whether context information from a message that is being backed out is passed to the backed out message. Possible values are:

Table 75. ContextOption parameter values.

Value	Description
NONE	No context information is passed.
IDENTITY	Identity context information is passed only.
ALL	All context information is passed.

If omitted, *ContextOption* is specified as ALL.

Return codes

Return code	Description
0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
58	Inconsistent use of installations detected
71	Unexpected error
72	Queue manager name error
133	Unknown object name error

Related information:

Using the .NET monitor

runmqlsr (run listener)

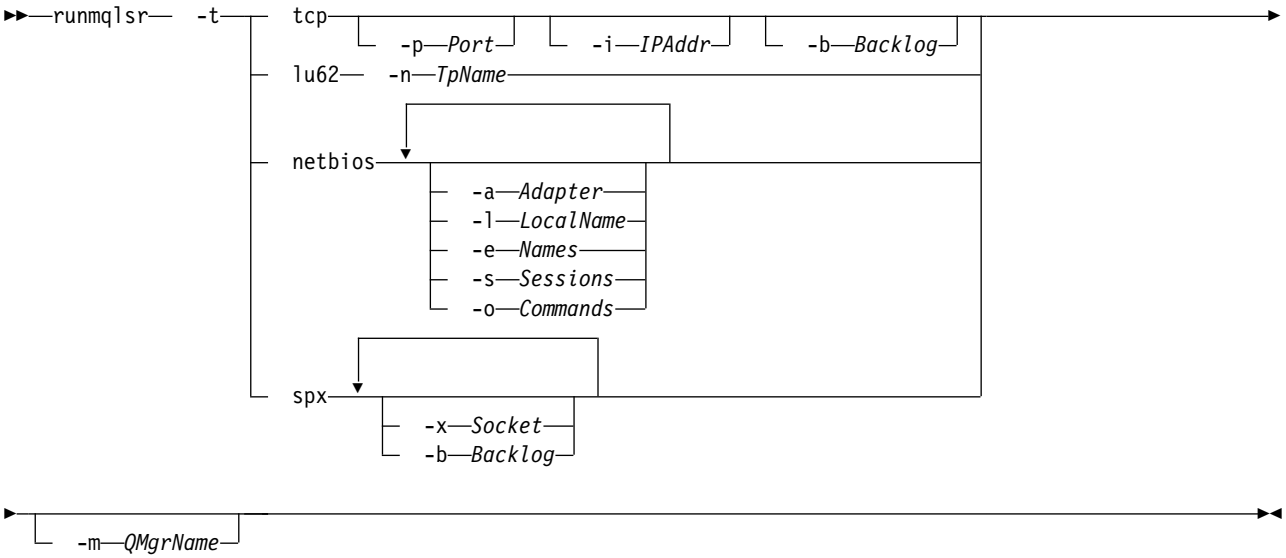
Run a listener process to listen for remote requests on various communication protocols.

Purpose

Use the **runmqlsr** command to start a listener process.

This command is run synchronously and waits until the listener process has finished before returning to the caller.

Syntax



Required parameters

-t The transmission protocol to be used:

Table 76. Transmission protocol values.

Header	Header
`tcp`	Transmission Control Protocol / Internet Protocol (TCP/IP)
`lu62`	Windows SNA LU 6.2 (Windows only)
`netbios`	Windows NetBIOS (Windows only)
`spx`	Windows SPX (Windows only)

Optional parameters

-p *Port*
The port number for TCP/IP. This flag is valid for TCP only. If you omit the port number, it is taken from the queue manager configuration information, or from defaults in the program. The default value is 1414. It must not exceed 65535.

-i *IPAddr*
The IP address for the listener, specified in one of the following formats:

- IPv4 dotted decimal
- IPv6 hexadecimal notation
- Alphanumeric format

This flag is valid for TCP/IP only.

On systems that are both IPv4 and IPv6 capable you can split the traffic by running two separate listeners. One listening on all IPv4 addresses and one listening on all IPv6 addresses. If you omit this parameter, the listener listens on all configured IPv4 and IPv6 addresses.

-n *TpName*
The LU 6.2 transaction program name. This flag is valid only for the LU 6.2 transmission protocol. If you omit the name, it is taken from the queue manager configuration information.

-a *Adapter*
The adapter number on which NetBIOS listens. By default the listener uses adapter 0.

-l *LocalName*
The NetBIOS local name that the listener uses. The default is specified in the queue manager configuration information.

-e *Names*
The number of names that the listener can use. The default value is specified in the queue manager configuration information.

-s *Sessions*
The number of sessions that the listener can use. The default value is specified in the queue manager configuration information.

-o *Commands*
The number of commands that the listener can use. The default value is specified in the queue manager configuration information.

-x *Socket*
The SPX socket on which SPX listens. The default value is hexadecimal 5E86.

-m *QMgrName*
The name of the queue manager. By default the command operates on the default queue manager.

-b *Backlog*
The number of concurrent connection requests that the listener supports. See TCP, LU62, NETBIOS, and SPX for a list of default values and further information.

Return codes

Return code	Description
0	Command completed normally
4	Command completed after being ended by the **endmqlsr** command
10	Command completed with unexpected results
20	An error occurred during processing: the AMQMSRVN process did not start.

Examples

The following command runs a listener on the default queue manager using the NetBIOS protocol. The listener can use a maximum of five names, five commands, and five sessions. These resources must be within the limits set in the queue manager configuration information.

```
runmqlsr -t netbios -e 5 -s 5 -o 5
```

Related reference:

"Listener commands" on page 191
A table of listener commands, showing the PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

runmqras (collect IBM MQ diagnostic information)

Use the **runmqras** command to gather IBM MQ diagnostic information together into a single archive, for example to submit to IBM Support.

Purpose

The **runmqras** command is used to gather diagnostic information from a machine, into a single archive. You can use this command to gather information about an application or IBM MQ failure, possibly for submitting to IBM when you report a problem.

V 9.0.2 The **runmqras** command requires a Java 7, or later, Java runtime environment (JRE) in order to run. If the IBM MQ JRE component (on Linux) or feature (on Windows) is not installed, then **runmqras** searches the system path for an alternative JRE and attempts to use that.

V 9.0.2 If no alternative could be found, error message AMQ8599 is output. In this case:

1. Install the IBM MQ JRE component, or install an alternative Java 7 JRE
2. Add the JRE to the system path
3. Rerun the command

.

By default, **runmqras** gathers information such as:

- IBM MQ FDC files
- Error logs (from all queue managers as well as the machine-wide IBM MQ error logs)
- Product versioning, status information, and output from various other operating system commands.

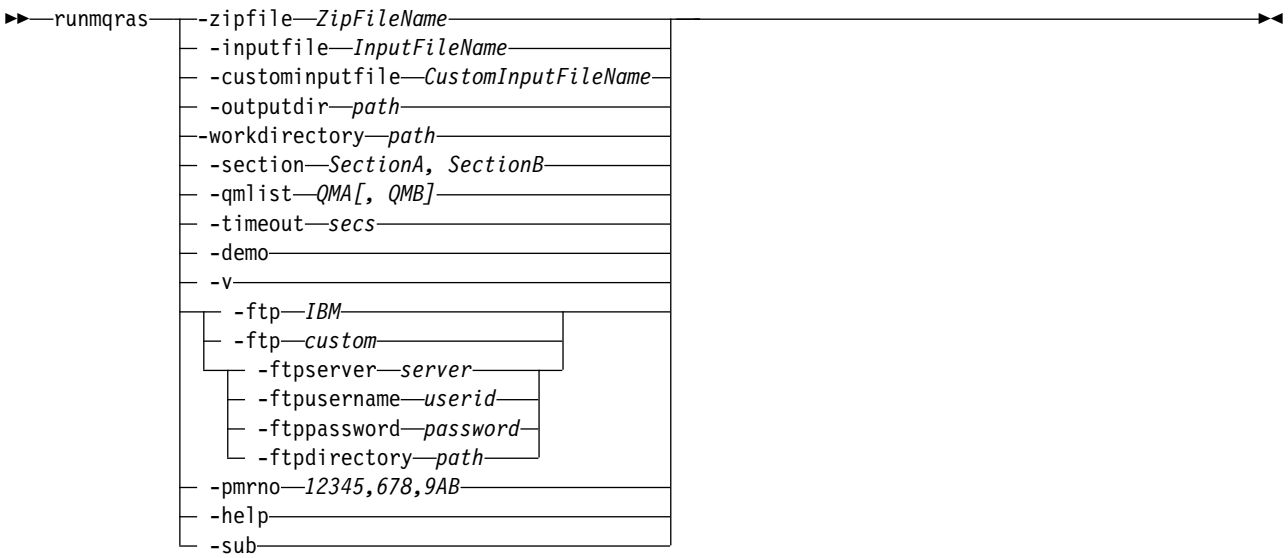
Note, for example, the **runmqras** command does not gather user information that is contained in messages on queues.

Running without requesting more sections is intended as a starting point for general problem diagnosis, however, you can request more *sections* through the command line.

These additional *sections* gather more detailed information, depending on the type of problem being diagnosed. If non-default sections are needed by IBM support personnel, they will tell you.

The **runmqras** command can be run under any user ID, but the command gathers only information that the user ID can gather manually. In general, when debugging IBM MQ problems, run the command under the `mqm` user ID to allow the command to gather queue manager files and command outputs.

Syntax



Keywords and parameters

All parameters are required unless the description states they are optional.

In every case, *QMgrName* is the name of the queue manager to which the command applies.

-inputfile *InputFileName*
Fully qualified name of the XML input file

-custominputfile *CustomInputFileName*
Fully qualified name of an additional XML input file

-zipfile *ZipFileName*
Supply the file name of the resulting archive.

By default, the name of the output archive is `runmqras.zip`.

-outputdir *path*
The directory in which the resulting output file is placed.

By default, the output directory is the same as the work directory.

-workdirectory *path*

The directory that is used for storing the output from commands that are run during the processing of the tool. If supplied, this directory must either not exist, in which case it is created, or must be empty.

If you do not supply the path, a directory whose name starts with **runmqras** and is suffixed by the date and time is used:

- UNIX On UNIX, the directory is under `/tmp`.
- Windows On Windows, the directory is under `%temp%`.

-section *SectionA, SectionB*

The optional sections about which to gather more specific information.

By default, a generic section of documentation is collected, whereas more specific information can be gathered for a specified problem type; for example, a section name of *trace* gathers all of the contents of the trace directory.

The default collections can be avoided by supplying a section name of *nodefault*.

IBM support generally supplies you with the sections to use. Example available sections are:

all Gathers all possible information, including all trace files, and diagnostics for many different types of problems. You must use this option only in certain circumstances and this option is not intended for general use.

default
IBM MQ logs, FDC files, basic configuration, and status.

Note: Always gathered unless you use the section name **nodefault**.

nodefault
Prevents the default collections from occurring, but other explicitly requested sections are still collected.

trace Gathers all the trace file information plus the default information.

Note: Does not enable tracing.

defs Gathers the queue manager definitions and status information.

cluster
Gathers cluster configuration and queue information.

dap Gathers transaction and persistence information.

kernel Gathers queue manager kernel information.

logger Gathers recovery logging information.

topic Gathers topic tree information.

QMGR
Gathers all queue manager files: queues, logs, and configuration files.

V 9.0.4 **mqweb**
Gathers trace and configuration data for the mqweb server.

For more information, see Section names and descriptions, in the IBM technote on using the IBM MQ **runmqras** command to collect data.

-qmlist *QMA[,QMB]*

A list of queue manager names on which the **runmqras** command is to be run.

NSS Client This parameter does not apply to a client product because there are no queue managers from which to request direct output. For example, this parameter does not apply to HP Integrity NonStop Server.

By supplying a comma-separated list, you can restrict the iteration across queue managers to a specific list of queue managers. By default, iteration of commands is across all queue managers.

-timeout *secs*
The default timeout to give an individual command before the command stops waiting for completion.

By default, a timeout of 10 seconds is used. A value of zero means wait indefinitely.

-demo
Run in demonstration mode where no commands are processed, and no files gathered.

By running in demonstration mode, you can see exactly which commands would have been processed, and what files would have been gathered. The output zip file contains a `console.log` file that documents exactly what would have been processed and gathered, should the command be run normally.

-v Extends the amount of information that is logged in the `console.log` file, contained in the output zip file.

-ftp *ibm/custom*
Allows the collected archive to be sent through basic FTP to a remote destination.

At the end of processing, the resultant archive can be sent through basic FTP, either directly into IBM, or to a site of your choosing. If you select the *ibm* option, anonymous FTP is used to deliver the archive into the IBM ECuRep server. This process is identical to submitting the file manually using FTP.

Note if you select the *ibm* option, you must also provide the *pmrno* option, and all other FTP* options are ignored.

-ftpserver ***server***
An FTP server name to connect to, when an FTP custom option is used.

-ftpusername ***userid***
The user ID to log in to the FTP server with, when an FTP custom option is used.

-ftppassword ***password***
The password to log in to the FTP server with, when an FTP custom option is used.

-ftpdirectory ***path***
The directory on the FTP server to place the resulting zip file into, used when an FTP custom option is used.

-pmrno ***12345,678,9AB***
A valid IBM PMR number (problem record number) against which to associate the documentation.

Use this option to ensure that the output is prefixed with your PMR Number, so that when the information is sent to IBM, the information is automatically associated with that problem record.

-help
Provide simple help.

-sub
Shows the keywords that will be substituted in the xml.

Examples

This command gathers the default documentation from the IBM MQ installation, and all queue managers on a machine:

```
runmqras
```

This command gathers the default documentation from the IBM MQ installation on a machine, and sends it directly into IBM to be associated with PMR number 11111,222,333 using the basic FTP capability:

```
runmqras -ftp ibm -pmrno 11111,222,333
```

This command gathers the default documentation from a machine, plus all trace files, the queue manager definitions, and status for all queue managers on the machine:

```
runmqras -section trace,defs
```

Return codes

A non zero return code indicates failure.

runmqsc (run MQSC commands)

Run IBM MQ commands on a queue manager.

Purpose

Use the **runmqsc** command to issue MQSC commands to a queue manager. MQSC commands enable you to perform administration tasks. For example, you can define, alter, or delete a local queue object. MQSC commands and their syntax are described in MQSC commands.

You must use the **runmqsc** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with by using the `dspmq -o installation` command.

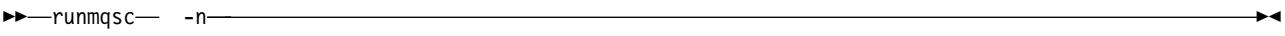
To stop the **runmqsc** command, use the **end** command. You can also use the **exit** or the **quit** command.

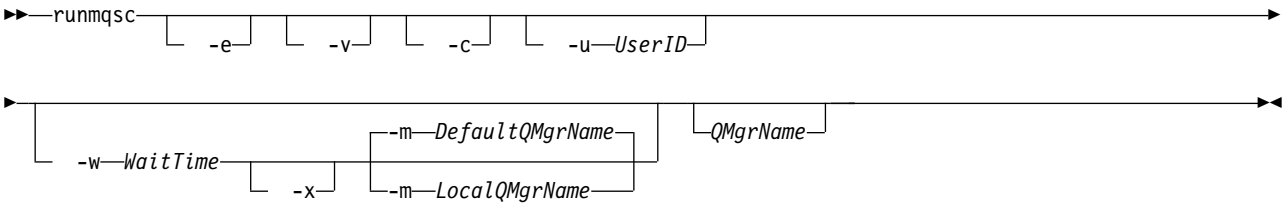
V 9.0.1 For Continuous Delivery, from IBM MQ Version 9.0.1, you can make it easier to see that you are in an MQSC environment and see some details of the current environment by setting a prompt of your choice by using the MQPROMPT environment variable. For more information, see Administration using MQSC commands.

V 9.0.0.1 From IBM MQ Version 9.0.0, Fix Pack 1, the MQPROMPT environment variable is also available in the Long Term Support release.

Syntax

You can use the **-n** parameter on its own, or you can use a number of other parameters in combination:





Description

You can start the **runmqsc** command in three ways:

Verify command
Verify MQSC commands but do not run them. An output report is generated indicating the success or failure of each command. This mode is available on a local queue manager only.

Run command directly
Send MQSC commands directly to a local queue manager.

Run command indirectly
Run MQSC commands on a remote queue manager. These commands are put on the command queue of a remote queue manager and run in the order in which they were queued. Reports from the commands are returned to the local queue manager.

The **runmqsc** command takes its input from `stdin`. When the commands are processed, the results and a summary are put into a report that is sent to `stdout`.

By taking `stdin` from the keyboard, you can enter MQSC commands interactively.

Alternatively, you can redirect `stdin` from a text file. By redirecting the input from a file, you can run a sequence of frequently used commands contained in the file. You can also redirect the output report to a file.

Note: If you run **runmqsc** in client mode by redirecting `stdin` from a text file, IBM MQ expects the first line of the input file to be a password.

Optional parameters

-c Modifies the **runmqsc** command to connect to a queue manager by using a client connection. The client channel definitions used to connect to the queue manager are located using the following environment variables in this order of precedence: **MQSERVER** , **MQCHLLIB** , and **MQCHLTAB** .

This option requires the client to be installed. If it is not installed an error message reporting the missing client libraries is issued.

-e Prevents source text for the MQSC commands from being copied into a report. This parameter is useful when you enter commands interactively.

-m *LocalQMgrName*
The local queue manager that you want to use to submit commands to the remote queue manager. If you omit this parameter the local default queue manager is used to submit commands to the remote queue manager. The **-w** parameter must also be specified.

-n Modifies the **runmqsc** command to not connect to a queue manager. If this parameter is specified, all other command parameters must be omitted, otherwise an error message is issued.

This option requires the client libraries to be installed. If they are not installed an error message is issued.

MQSC commands entered in this mode are limited to managing the local channel definition file, which is located through the **MQCHLLIB** and **MQCHLTAB** environment variables, or the default values if not defined.

Note: If you add new entries into the local channel definition file, or alter existing entries, these changes are not reflected inside the queue manager. The queue manager does not read the contents of the local channel definition file. The CCDT file is a write-only file from the perspective of the queue manager. The queue manager does not read the contents of the CCDT file.

Only the following MQSC commands are recognized:

ALTER, **DEFINE**, **DELETE**, **DISPLAY AUTHINFO** (Only of type CRLLDAP or OCSP)
ALTER, **DEFINE**, **DELETE**, **DISPLAY CHANNEL** (Only of type CLNTCONN)

For the AUTHINFO management commands, the names of existing AUTHINFO definitions are mapped and addressed using the names `CRLLDAP`*n* or `OCSP` *n* (according to type), where *n* is the numeric order in which they appear in the channel definition file. New AUTHINFO definitions are appended to the client channel table in order. For example, the the following commands are issued:

```
DEFINE AUTHINFO(XYZ) AUTHTYPE(CRLLDAP) CONNAME('xyz')

DEFINE AUTHINFO(ABC) AUTHTYPE(CRLLDAP) CONNAME('abc')
```

This results in the `'xyz'` LDAP server being checked for a CRL first, if that CRL server is unavailable then the `'abc'` server is checked.

Using the **DISPLAY AUTHINFO(*) CONNAME** command shows this:

```
AMQ8566: Display authentication information details.
AUTHINFO(CRLLDAP1)
AUTHTYPE(CRLLDAP)            CONNAME(xyz)
AMQ8566: Display authentication information details.
AUTHINFO(CRLLDAP2)
AUTHTYPE(CRLLDAP)            CONNAME(abc)
```

Note: The client mode only supports inserting new entries at the end of the client channel table. If you want to change the order of precedence of the CRL LDAP servers, you must remove the existing objects from the list and reinsert them in the correct order at the end.

-u ***UserID***
If you use the **-u** parameter to supply a user ID, you are prompted for a matching password.

If you have configured the `CONNAUTH AUTHINFO` record with `CHCKLOCL(REQUIRED)` or `CHCKLOCL(REQDADM)`, you must use the **-u** parameter otherwise you will not be able to administer your queue manager with **runmqsc**.

If you specify this parameter and redirect `stdin`, a prompt will not be displayed and the first line of redirected input should contain the password.

-v Verifies the specified commands without performing the actions. This mode is only available locally. The **-w** and **-x** parameters are ignored if they are specified at the same time as **-v**.

Important: The **-v** flag checks the syntax of the command only. Setting the flag does not check if any objects mentioned in the command actually exist.

For example, if the queue `Q1` does not exist in the queue manager, the following command is syntactically correct and does not generate any syntax errors: `runmqsc -v Qmgr display ql(Q1)`.

However, if you omit the **-v** flag, you receive error message AMQ8147.

-w *WaitTime*
Run the MQSC commands on another queue manager. You must have the required channel and transmission queues set up for this. See Preparing channels and transmission queues for remote administration for more information.

This parameter is ignored if the **-v** parameter is specified.

WaitTime
The time, in seconds, that **runmqsc** waits for replies. Any replies received after this are discarded, but the MQSC commands still run. Specify a time in the range 1 through 999999.

Each command is sent as an Escape PCF to the command queue (SYSTEM.ADMIN.COMMAND.QUEUE) of the target queue manager.

The replies are received on queue SYSTEM.MQSC.REPLY.QUEUE and the outcome is added to the report. This can be defined as either a local queue or a model queue.

-x The target queue manager is running under z/OS. This parameter applies only in indirect mode. The **-w** parameter must also be specified. In indirect mode, the MQSC commands are written in a form suitable for the IBM MQ for z/OS command queue.

QMgrName
The name of the target queue manager on which to run the MQSC commands. If not specified, the default queue manager is used.

Return codes

Return code	Description
00	MQSC command file processed successfully
10	MQSC command file processed with errors; report contains reasons for failing commands
20	Error; MQSC command file not run

Examples

1. Enter this command at a command prompt:

```
runmqsc
```

   Now you can enter MQSC commands directly at the command prompt. No queue manager name is specified, so the MQSC commands are processed on the default queue manager.

2. Use one of these commands, as appropriate in your environment, to specify that MQSC commands are to be verified only:

```
runmqsc -v BANK < "/u/users/commfile.in"

runmqsc -v BANK < "c:\users\commfile.in"
```

   The queue manager name is `BANK`. The command verifies the MQSC commands in file `commfile.in` and displays the output in the current window.

3. These commands run the MQSC command file `mqscfile.in` against the default queue manager.

```
runmqsc < "/var/mqm/mqsc/mqscfile.in" > "/var/mqm/mqsc/mqscfile.out"

runmqsc < "C:\Program Files\IBM\MQ\mqsc\mqscfile.in" >
        "C:\Program Files\IBM\MQ\mqsc\mqscfile.out"
```

   In this example, the output is directed to file `mqscfile.out`.

4. This command submits commands to the QMREMOTE queue manager, using QMLOCAL to submit the commands.

```
runmqsc -w 30 -m QMLOCAL QMREMOTE
```

Related information:
Using MQSC commands interactively
Running MQSC commands from text files
Administration using MQSC commands

runmqsfb (run IBM MQ Bridge to Salesforce)

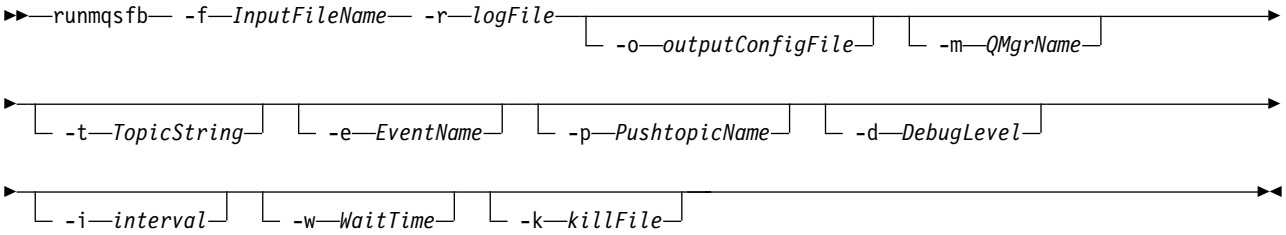
Linux V 9.0.2

Configure and run the IBM MQ Bridge to Salesforce.

- "Syntax"
- "Usage notes"
- "Command line parameters" on page 315
- Configuration parameters
- Examples

Syntax

The diagram shows the syntax for the **runmqsfb** command usage as described in note 1.



Usage notes

1. You can run the **runmqsfb** command to start the IBM MQ Bridge to Salesforce and connect to Salesforce and IBM MQ. When the connections are made, the bridge receives Salesforce generated events and publishes them to an IBM MQ network V 9.0.4 , or create event messages for Salesforce platform events.

   ```
   runmqsfb -f configFile -r logFile -m QMgrName -t TopicString -e EventName -p PushtopicName -d debugLevel -i interval -w
   ```

   When you use the command for runtime processing, the required parameters are **-f**, with the name of the previously created configuration file, and **-r** with the name of the log file. When the other command parameters are also given on the command line, they override the values in the configuration file. The option allows a core default configuration to be created and provides a simple way to handle minor variations such as the queue manager name.
2. You can also use the **runmqsfb** command to generate a configuration file that is used to define the parameters that are needed to connect to Salesforce and IBM MQ.

   When you are creating the configuration file, the **-f** parameter is optional, the input configuration file is included in the samples directory for the IBM MQ Bridge to Salesforce, `/opt/mqm/mqsf/samp`.

   ```
   runmqsfb -[f inputConfigFile] -o outputConfigFile
   ```

   When you run the command in this way, you are prompted to enter values for each of the configuration parameters. To keep an existing value press `Enter`. To remove an existing value press `Space`, then `Enter`. For more information, see "Configuration parameters" on page 315.

Command line parameters

-m ***QMgrName or ConnFactoryName***
Queue manager or Connection Factory name.

-r ***logFile***
Required. Location and name of the log file for trace information. You can specify the log file path and name in the configuration file or on the command line.

-t ***TopicString***
IBM MQ topic root.

-e ***EventName***
Salesforce platform event name (can repeat). You can specify multiple **-e** entries on the command line, one for each event type that the bridge listens for. You must provide the base part of the event name. The bridge automatically adds "`/event`" or "`/topic`" prefixes when it connects to Salesforce. Multiple **-e** parameters can be comma-separated.

-p ***PushtopicName***
Salesforce push topic name (can repeat). You can specify multiple **-p** entries on the command line, one for each topic type that the bridge listens for. You must provide the base part of the topic name. The bridge automatically adds "`/event`" or "`/topic`" prefixes when it connects to Salesforce. Multiple **-p** parameters can be comma-separated.

-i ***interval***
Monitor interval. Enter `0` to disable monitoring.

-f ***inputConfigFile***
Configuration file. The **-f** parameter is required when you are running the **runmqsfb** command to start the IBM MQ Bridge to Salesforce, as described in usage note 1 on page 314. You can optionally use the **-f** parameter to reuse some of the values from an existing *inputConfigFile*, as described in usage note 2 on page 314, and also enter some of the new values. If you do not specify the **-f** parameter when you are creating the configuration file, all the values for the parameters you are prompted for are empty.

-o ***outputConfigFile***
New configuration file. When you run the command with the **-o** parameter, **runmqsfb** command loads existing configuration values from the **-f** file and prompts for new values for each configuration parameter.

-k ***killFile***
A file to cause the bridge to exit. When you run the command with the **-k** parameter and specify a file, if the file exists, it causes the bridge program to exit. Using this file is an alternative way to stop the program when you don't want to use `Ctrl+C` or **kill** command. The file is deleted by the bridge on startup in case it exists. If the deletion fails, the bridge abends but monitors for the recreation of the file.

-d ***debugLevel***
Debug level, `1`, or `2`.

1 Terse debug information is displayed.

2 Verbose debug information is displayed.

-w ***WaitTime***
Wait before fully starting.

Configuration parameters

When you run the **runmqsfb** command to create the configuration file, the parameters are stepped through in four groups. Passwords are obfuscated and are not displayed as you type. The generated

configuration file is in JSON format. You must use the **runmqsfb** command to create the configuration file. You cannot edit the passwords and security certificate information directly in the JSON file.

Connection to queue manager

Parameters relating to the IBM MQ queue manager.

IBM MQ Queue manager or JNDI CF
Required.

IBM MQ Base Topic
Required. All events are published by using the topic root as the prefix to the Salesforce event name.

IBM MQ Channel
Blank **channel** implies local bindings.

IBM MQ Conname
Uses standard connection name format of "host(port), host(port)" to enable multiple destinations such as for multi-instance queue managers. Blank **conname** implies local bindings.

V 9.0.4 **IBM MQ Publication Error Queue**
Required for creating platform event messages. IBM MQ error queue for processing bad input messages. The default queue *SYSTEM.SALESFORCE.ERRORQ* is created when you run the **mqsfbSyncQ.mqsc** script command that also creates the required synchronization queue on the queue manager.

IBM MQ CCDT URL
If a TLS connection is required to the queue manager, you must use a JNDI or CCDT definition.

JNDI implementation class name
The class name of your JNDI provider. The "queue manager name" parameter refers to the connection factory name when you are using JNDI.

JNDI provider URL
The endpoint of your JNDI service.

IBM MQ UserId

IBM MQ Password

Connection to Salesforce

Parameters relating to Salesforce.

Salesforce Userid (required)
Required. Log in email for your Salesforce account.

Salesforce password (required)
Required. Password for your Salesforce account.

Salesforce security token (required)
Required. Security token that you can generate from the **Security controls** section of the **Administer** menu of your Salesforce **Force.com Home** page.

Login Endpoint
Salesforce login endpoint URL, https://login.salesforce.com.

Consumer key
The consumer key that you generate when you add the IBM MQ Bridge to Salesforce as a connected app in your Salesforce account. For more information, see step 5 in Configuring the IBM MQ Bridge to Salesforce

Consumer secret
The consumer secret that is generated along with the consumer key.

OAuth consumer key and secret values are optional but must be considered for production systems.

Certificate stores for TLS connections
Parameters relating to certificate stores for TLS connections.

Personal keystore for TLS certificates
Required. The keystore that you create in your Salesforce account. For more information, see Step 3 in Configuring the IBM MQ Bridge to Salesforce.

Keystore password
Required. The password that you create when you are exporting the keystore from your Salesforce account.

Trusted store for signer certificates
Required. If you do not add the trusted store, the personal keystore for TLS certificates is used.

Trusted store password
Required. If the personal keystore for TLS certificates is used, this is the password for the keystore for TLS certificates.

Use TLS for MQ connection
If you are using TLS for your IBM MQ connectivity, you can use the same keystore that you used to connect to Salesforce.

For the Salesforce connection, a truststore must be available and contain at least the signer certificates to validate the Salesforce system. Only TLS1.1 and TLS1.2 protocols are supported for connection to Salesforce. A user certificate is not required. If you don't provide a truststore parameter, the keystore is used for both roles. The stores can be the same as the one configured for the IBM MQ connection in the CCDT or JNDI.

Behavior of bridge program
Parameters relating to the behavior of the IBM MQ Bridge to Salesforce.

Push topic names
You can provide one push topic name at a time and then move on to the next parameter by pressing `enter`.

Platform event names
You can provide one platform event name at a time and then move on to the next parameter by pressing `enter`.

Monitoring frequency
IBM MQ Monitoring frequency.

At least once delivery
Quality of service. At-least-once or at-most-once delivery.

V 9.0.4 **Subscribe to IBM MQ publications for platform events**
Required. The default option is *N*. You must enter *Y* to enable the bridge feature for creating event messages for Salesforce platform events.

Publish control data with the payload
On republish, send full message not only subject.

Delay before starting to process events
Delay before the bridge starts to process events.

Runtime logfile for copy of stdout/stderr
Path to and name of the log file for the tracing information.

Push topic names and **Platform event names** can be entered individually or as a comma-separated list, in the same way as the command line **-p** and **-e** parameters are entered. The **Startup wait interval** provides an option to delay the initial processing of events. For example, if the bridge and the IBM MQ applications that use it are all run as services, the order in which they start cannot be sequenced. Therefore, events might be republished before applications are prepared to receive them. When you delay the bridge startup, you give the applications the time to start and subscribe to events and push topics.

The configuration is only read on startup of the bridge process. Changes to the configuration require a restart, such as through the IBM MQ Service definitions.

Examples

The **-f** parameter is optional when you use the **runmqsfb** to create the configuration file as described in the usage note 2 on page 314.

```
runmqsfb -f inputConfigFile -o outputConfigFile
```

In this example, the *outputConfigFile* is created:

```
runmqsfb -o outputConfigFile
```

The **-f** parameter is required when you use the **runmqsfb** command to run the IBM MQ Bridge to Salesforce, as described in the usage note 1 on page 314.

```
runmqsfb -f inputConfigFile -r logFile
```

Related information:

Configuring IBM MQ for use with Salesforce push topics and platform events

Tracing the IBM MQ Bridge to Salesforce

Monitoring the IBM MQ Bridge to Salesforce

runmqtmc (start client trigger monitor)

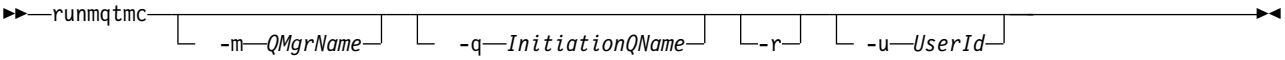
Start the trigger monitor on a client.

Purpose

Use the **runmqtmc** command to start a trigger monitor for a client. For further information about using trigger monitors, see Trigger monitors.

When a trigger monitor starts, it continuously monitors the specified initiation queue. The trigger monitor does not stop until the queue manager ends, see "endmqm (end queue manager)" on page 269. While the client trigger monitor is running it keeps the dead letter queue open.

Syntax



Optional parameters

-m ***QMgrName***
The name of the queue manager on which the client trigger monitor operates, by default the default queue manager.

-q ***InitiationQName***
The name of the initiation queue to be processed, by default SYSTEM.DEFAULT.INITIATION.QUEUE.

-r Specifies that the client trigger monitor automatically reconnects.

-u *UserId*
The ID of the user authorized to get the triggered message.

Note that using this option does not affect the authority of the triggered program, that might have its own authentication options.

Return codes

Return code	Description
0	Not used. The client trigger monitor is designed to run continuously and therefore not to end. The value is reserved.
10	Client trigger monitor interrupted by an error.
20	Error; client trigger monitor not run.

Examples

For examples of using this command, see The Triggering sample programs.

runmqtrm (start trigger monitor)

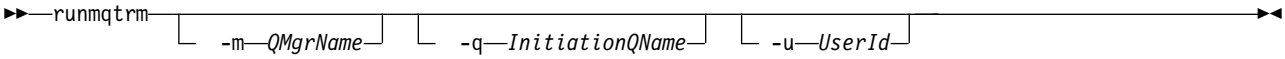
Start the trigger monitor on a server.

Purpose

Use the **runmqtrm** command to start a trigger monitor. For further information about using trigger monitors, see Trigger monitors.

When a trigger monitor starts, it continuously monitors the specified initiation queue. The trigger monitor does not stop until the queue manager ends, see "endmqm (end queue manager)" on page 269. While the trigger monitor is running it keeps the dead letter queue open.

Syntax



Optional parameters

-m *QMgrName*
The name of the queue manager on which the trigger monitor operates, by default the default queue manager.

-q *InitiationQName*
Specifies the name of the initiation queue to be processed, by default SYSTEM.DEFAULT.INITIATION.QUEUE.

-u *UserId*
The ID of the user authorized to read the initiation queue, and get the triggered message.

Note that using this option does not affect the authority of the triggered program, that might have its own authentication options.

Return codes

Return code	Description
0	Not used. The trigger monitor is designed to run continuously and therefore not to end. Hence a value of 0 would not be seen. The value is reserved.
10	Trigger monitor interrupted by an error.
20	Error; trigger monitor not run.

runswchl (switch cluster channel)

ULW

runswchl (switch cluster channel) on UNIX, Linux, and Windows.

Purpose

The command switches or queries the cluster transmission queues associated with cluster-sender channels.

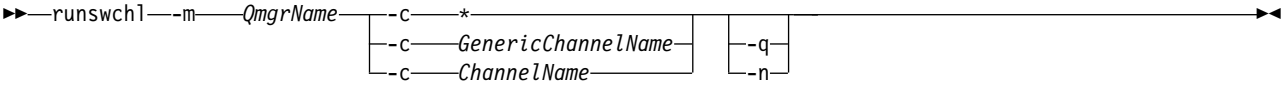
Usage notes

You must log on as an Administrator to run this command.

The command switches all the stopped or inactive cluster-sender channels that match the `-c` parameter, require switching, and can be switched. The command reports back on the channels that are switched, the channels that do not require switching, and the channels it cannot switch because they are not stopped or inactive.

If you set the `-q` parameter, the command does not perform the switch, but it provides the list of channels that would be switched.

Syntax



Required parameters

-m ***QmgrName***
The queue manager to run the command against. The queue manager must be started.

-c *
All the cluster-sender channels

-c ***GenericChannelName***
All matching cluster-sender channels

-c ***ChannelName***
Single cluster-sender channel.

Optional parameters

-q Display the state of one or more channels. If you omit this parameter, the commands switches any stopped or inactive channels that require switching.

-n When switching transmission queues, do not transfer messages from the old queue to the new transmission queue.

Note: Take care with the `-n` option: messages on the old transmission queue are not transferred unless you associate the transmission queue with another cluster-sender channel.

Return codes

0 The command completed successfully

10 The command completed with warnings.

20 The command completed with errors.

Examples

To display the configuration state of cluster-sender channel `TO.QM2`:

```
RUNSWCHL -m QM1 -c TO.QM2 -q
```

To switch the transmission queue for cluster-sender channel `TO.QM3` without moving the messages on it:

```
RUNSWCHL -m QM1 -c TO.QM3 -n
```

To switch the transmission queue for cluster-sender channel `TO.QM3` and move the messages on it:

```
RUNSWCHL -m QM1 -c TO.QM3
```

To display the configuration state of all cluster-sender channels on `QM1`:

```
RUNSWCHL -m QM1 -c * -q
```

To display the configuration state of all cluster-sender channels with a generic name of `TO.*`:

```
RUNSWCHL -m QM1 -c TO.* -q
```

Related information:

Clustering: Switching cluster transmission queues

setmqaut (grant or revoke authority)

Change the authorizations to a profile, object, or class of objects. Authorizations can be granted to, or revoked from, any number of principals or groups.

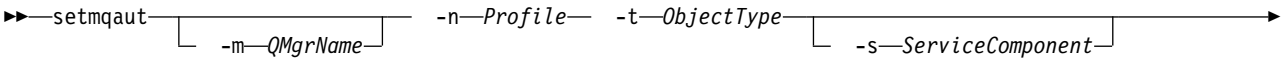
From IBM MQ Version 8.0, on UNIX and Linux systems, the object authority manager (OAM) can use user-based authorization as well as group-based authorization.

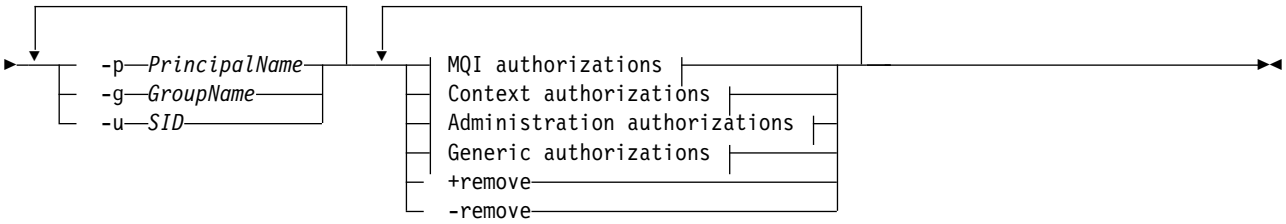
For more information about authorization service components, see Installable services, Service components, and Authorization service interface.

For more information about how authorizations work, see How authorizations work.

For more information about how user-based authorizations, see User-based permissions on Unix and Linux systems.

Syntax





MQI authorizations:

Context authorizations:

Administration authorizations:

Generic authorizations:

Description

Use **setmqaut** both to grant an authorization, that is, give a principal or user group permission to perform an operation, and to revoke an authorization, that is, remove the permission to perform an operation. You can specify a number of parameters:

- Queue manager name
- Principals and user groups
- Object type
- Profile name
- Service component

The authorizations that can be given are categorized as follows:

- Authorizations for issuing MQI calls
- Authorizations for MQI context
- Authorizations for issuing commands for administration tasks
- Generic authorizations

Each authorization to be changed is specified in an authorization list as part of the command. Each item in the list is a string prefixed by a plus sign (+) or a minus sign (-). For example, if you include +put in the authorization list, you grant authority to issue MQPUT calls against a queue. Alternatively, if you include -put in the authorization list, you revoke the authority to issue MQPUT calls.

You can specify any number of principals, user groups, and authorizations in a single command, but you must specify at least one principal or user group.

On UNIX, Linux, and Windows, you can use the installable services to control the queue manager authorization. For more information about the **SecurityPolicy** attribute that must be set to control the queue manager authorization, see Installable services. If the **SecurityPolicy** attribute is set to `group` or `default`, or if the **SecurityPolicy** attribute is not set, the following information is true:

- Windows If a principal is a member of more than one user group, the principal effectively has the combined authorities of all those user groups. On Windows systems, the principal also has all the authorities that are granted to it explicitly using the **setmqaut** command.
- UNIX On UNIX, all authorities are held by user groups internally, not by principals. Granting authorities to groups has the following implications:
  - If you use the **setmqaut** command to grant an authority to a principal, the authority is granted to the primary user group of the principal. This means that the authority is effectively granted to all members of that user group.
  - If you use the **setmqaut** command to revoke an authority from a principal, the authority is revoked from the primary user group of the principal. This means that the authority is effectively revoked from all members of that user group.

To alter authorizations for a cluster sender channel that has been automatically generated by a repository, see Channel definition commands.

Required parameters

-t *ObjectType*
The type of object for which to change authorizations.

Possible values are as follows:

Table 77. ObjectType values.

Value	Description
authinfo	An authentication information object
channel or **chl**	A channel
clntconn or **clcn**	A client connection channel
comminfo	A communication information object
listener or **lstr**	A listener
namelist or **nl**	A namelist
process or **prcs**	A process
queue or **q**	A queue
qmgr	A queue manager
rqmname or **rqmn**	A remote queue manager name
service or **srvc**	A service
topic or **top**	A topic

-n *Profile*
The name of the profile for which to change authorizations. The authorizations apply to all IBM MQ objects with names that match the profile name specified. The profile name can be generic, using wildcard characters to specify a range of names as explained in Using OAM generic profiles on UNIX, Linux, and Windows systems.

This parameter is required, unless you are changing the authorizations of a queue manager, in which case you must not include it. To change the authorizations of a queue manager use the queue manager name, for example

```
setmqaut -m QMGR -t qmgr -p user1 +connect
```

where *QMGR* is the name of the queue manager and *user1* is the user requesting the change.

Each class of object has authority records for each group or principal. These records have the profile name `@CLASS` and track the `crt` (create) authority common to all objects of that class. If the `crt` authority for any object of that class is changed then this record is updated. For example:

```
profile:   @class
object type: queue
entity:   test
entity type: principal
authority:  crt
```

This shows that members of the group `test` have `crt` authority to the class `queue`.

Optional parameters

-m *QMgrName*
The name of the queue manager of the object for which to change authorizations. The name can contain up to 48 characters.

This parameter is optional if you are changing the authorizations of your default queue manager.

-p *PrincipalName*
The name of the principal for which to change authorizations.

Windows For IBM MQ for Windows only, the name of the principal can optionally include a domain name, specified in the following format:

```
userid@domain
```

For more information about including domain names on the name of a principal, see Principals and groups.

You must have at least one principal or group.

-g *GroupName*
The name of the user group for which to change authorizations. You can specify more than one group name, but each name must be prefixed by the -g flag.

Windows For IBM MQ for Windows only, the group name can optionally include a domain name, specified in the following formats:

```
GroupName@domain
domain\GroupName
```

The IBM MQ Object Authority Manager validates the users and groups at the domain level, only if you set the **GroupModel** attribute to *GlobalGroups* in the Securing stanza of the queue manager.

-u *SID*
The SID for which authorities are to be removed. You can specify more than one SID, but each name must be prefixed by the -u flag.

This option must be used with either `+remove` or `-remove`.

This parameter is only valid on IBM MQ for Windows.

-s *ServiceComponent*
The name of the authorization service to which the authorizations apply (if your system supports installable authorization services). This parameter is optional; if you omit it, the authorization update is made to the first installable component for the service.

+remove or -remove

Remove all the authorities from IBM MQ objects that match the specified profile.

Authorizations

The authorizations to be granted or revoked. Each item in the list is prefixed by a plus sign (+) or a minus sign (-). The plus sign indicates that authority is to be granted. The minus sign indicates that authority is to be revoked.

For example, to grant authority to issue MQPUT calls, specify +put in the list. To revoke the authority to issue MQPUT calls, specify -put.

Table 78 shows the authorities that can be given to the different object types.

Table 78. Specifying authorities for different object types

Authority	Queue	Process	Queue manager	Remote queue manager name	Namelist	Topic	Auth info	Clntconn	Channel	Listener	Service	Comminfo
all [1]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
alladm [2]	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
allmqi [3]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No
none	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
altusr	No	No	Yes	No	No	No	No	No	No	No	No	No
browse	Yes	No	No	No	No	No	No	No	No	No	No	No
chg	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
clr	Yes	No	No	No	No	Yes	No	No	No	No	No	No
connect	No	No	Yes	No	No	No	No	No	No	No	No	No
crt	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ctrl	No	No	No	No	No	Yes	No	No	Yes	Yes	Yes	No
ctrlx	No	No	No	No	No	No	No	No	Yes	No	No	No
dlt	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
dsp	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
get	Yes	No	No	No	No	No	No	No	No	No	No	No
pub	No	No	No	No	No	Yes	No	No	No	No	No	No
put	Yes	No	No	Yes	No	No	No	No	No	No	No	No
inq	Yes	Yes	Yes	No	Yes	No	Yes	No	No	No	No	No
passall	Yes	No	No	No	No	No	No	No	No	No	No	No
passid	Yes	No	No	No	No	No	No	No	No	No	No	No
resume	No	No	No	No	No	Yes	No	No	No	No	No	No
set	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No
setall	Yes	No	Yes	No	No	No	No	No	No	No	No	No
setid	Yes	No	Yes	No	No	Yes	No	No	No	No	No	No
sub	No	No	No	No	No	Yes	No	No	No	No	No	No
system	No	No	Yes	No	No	No	No	No	No	No	No	No

Table 78. Specifying authorities for different object types (continued)

Authority	Queue	Process	Queue manager	Remote queue manager name	Namelist	Topic	Auth info	Clntconn	Channel	Listener	Service	Comminfo
Notes: 1. `all` authority is equivalent to the union of the authorities `alladm`, `allmqi`, and `system` appropriate to the object type. 2. `alladm` authority is equivalent to the union of the individual authorities `chg`, `clr`, `dlt`, `dsp`, `ctrl`, and `ctrlx` appropriate to the object type. `crt` authority is not included in the subset `alladm`. 3. `allmqi` authority is equivalent to the union of the individual authorities `altusr`, `browse`, `connect`, `get`, `inq`, `pub`, `put`, `resume`, `set`, and `sub` appropriate to the object type. 4. To use `setid` or `setall` authority, authorizations must be granted on both the appropriate queue object and also on the queue manager object.												

Description of specific authorities

You should not grant a user an authority (for example, `set` authority on a queue manager, or `system` authority) that allows the user to access IBM MQ privileged options, unless the required authority is specifically documented, and required to run any IBM MQ command, or IBM MQ API call.

For example, a user requires system authority to run the **setmqaut** command.

chg

A user needs `chg` authority to make any authorization changes on the queue manager. The authorization changes include:

- Changing the authorizations to a profile, object, or class of objects
- Creating and modifying channel authentication records, and so on

A user also needs `chg` authority to change or set the attributes of an IBM MQ object, using PCF or MQSC commands.

crt

If you grant an entity `+crt` authority to the queue manager, then that entity also gains `+crt` authority for each object class.

However, when you remove `+crt` authority against the queue manager object that only removes the authority on the queue manager object class; `crt` authority for other objects classes are not removed.

Note that `crt` authority on the queue manager object has no functional use, and is available for backwards-compatibility purposes only.

dlt

Note that the `dlt` authority against the queue manager object has no functional use, and is available for backwards-compatibility purposes only.

set

A user needs `set` authority against the queue to change or set the attributes of a queue using the MQSET API call.

`set` authority on the queue manager is not required for any administrative purpose, or for any application connecting to the queue manager.

However, a user needs `set` authority against the queue manager to set privileged connection options.

Note that `set` authority on the process object has no functional use, and is available for backwards-compatibility purposes only.

Important: Privileged connection options are internal to the queue manager and are not available in IBM MQ API calls used by IBM MQ applications.

system

The **setmqaut** command makes a privileged IBM MQ connection to the queue manager.

Any user who runs IBM MQ commands that makes a privileged IBM MQ connection needs `system` authority on the queue manager.

Return codes

Return code	Explanation
0	Successful operation
26	Queue manager running as a standby instance.
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
58	Inconsistent use of installations detected
69	Storage not available
71	Unexpected error
72	Queue manager name error
133	Unknown object name
145	Unexpected object name
146	Object name missing
147	Object type missing
148	Invalid object type
149	Entity name missing
150	Authorization specification missing
151	Invalid authorization specification

Examples

1. This example shows a command that specifies that the object on which authorizations are being given is the queue orange.queue on queue manager saturn.queue.manager.

   ```
   setmqaut -m saturn.queue.manager -n orange.queue -t queue
            -g tango +inq +alladm
   ```

   The authorizations are given to a user group called tango, and the associated authorization list specifies that the user group can:
   - Issue MQINQ calls
   - Perform all administration operations on that object
2. In this example, the authorization list specifies that a user group called foxy:
   - Cannot issue any MQI calls to the specified queue
   - Can perform all administration operations on the specified queue

   ```
   setmqaut -m saturn.queue.manager -n orange.queue -t queue
            -g foxy -allmqi +alladm
   ```
3. This example gives user1 full access to all queues with names beginning a.b. on queue manager qmgr1. The profile applies to any object with a name that matches the profile.

   ```
   setmqaut -m qmgr1 -n a.b.* -t q -p user1 +all
   ```
4. This example deletes the specified profile.

   ```
   setmqaut -m qmgr1 -n a.b.* -t q -p user1 -remove
   ```

5. This example creates a profile with no authority.

```
setmqaut -m qmgr1 -n a.b.* -t q -p user1 +none
```

Related reference:

"SET AUTHREC on Multiplatforms" on page 1007
Use the MQSC command SET AUTHREC to set authority records associated with a profile name.

Authorizations for MQI calls:

Table 79. Authorizations for MQI calls.

Value	Description
altusr	Use another user's authority for the queue manager. Also required for channel operations where the asserting userid is different from the one associated with the connection handle. (For example, an assigned dedicated profile on the receiver MCA end, or when processing a RESET CHL SEQNUM() request from remote systems.) If you are using IBM WebSphere MQ prior to IBM WebSphere MQ Version 7.0.1, Fix Pack 4.4, you must set +altusr for the group containing the user ID specified in MCAUSER on a receiver channel. This action prevents error message AMQ2035 appearing if you reset the sequence number of the corresponding sender channel.
browse	Retrieve a message from a queue using an MQGET call with the BROWSE option.
connect	Connect the application to the specified queue manager using an MQCONN call.
get	Retrieve a message from a queue using an MQGET call.
inq	Make an inquiry on a specific queue using an MQINQ call.
pub	Publish a message on a topic using the MQPUT call.
put	Put a message on a specific queue using an MQPUT call.
resume	Resume a subscription using the MQSUB call.
set	Set attributes on a queue from the MQI using an MQSET call.
sub	Create, alter or resume a subscription to a topic using the MQSUB call.

Note: If you open a queue for multiple options, you must be authorized for each option.

Authorizations for context:

Table 80. Authorizations for context.

Value	Description
passall	Pass all context on the specified queue. All the context fields are copied from the original request.
passid	Pass identity context on the specified queue. The identity context is the same as that of the request.
setall	Set all context on the specified queue. This is used by special system utilities.
setid	Set identity context on the specified queue. This is used by special system utilities. In order to modify any of the message context options, you must have the appropriate authorizations to issue the call. For example, in order to use MQOO_SET_IDENTITY_CONTEXT or MQPMO_SET_IDENTITY_CONTEXT, you must have `+setid` permission.

Note: To use `setid` or `setall` authority, authorizations must be granted on both the appropriate queue object and also on the queue manager object.

Authorizations for commands:

Table 81. Authorizations for commands.

Value	Description
chg	Change the attributes of the specified object.
clr	Clear the specified queue or a topic.
crt	Create objects of the specified type.
dlt	Delete the specified object. Note, that the dlt authority has no effect on a queue manager object.
dsp	Display the attributes of the specified object.
ctrl	For listeners and services, start and stop the specified channel, listener, or service. For channels, start, stop, and ping the specified channel. For topics, define, alter, or delete subscriptions.
ctrlx	Reset or resolve the specified channel.

Authorizations for generic operations:

Table 82. Authorizations for generic operations.

Value	Description
all	Use all operations applicable to the object. `all` authority is equivalent to the union of the authorities `alladm`, `allmqi`, and `system` appropriate to the object type.
alladm	Use all administration operations applicable to the object.
allmqi	Use all MQI calls applicable to the object.
none	No authority. Use this authorization to create profiles without authority. When an authority is given to an object or group that was previously showing "none", then the authorization changes to the authority just applied. However, when the "none" authorization is added to an object or group with an existing alternative authority, the authority does not change.
system	Use queue manager for internal system operations.

setmqcrl (set CRL LDAP server definitions)

Windows

Administer certificate revocation list (CRL) LDAP definitions in an Active Directory (Windows only).

Purpose

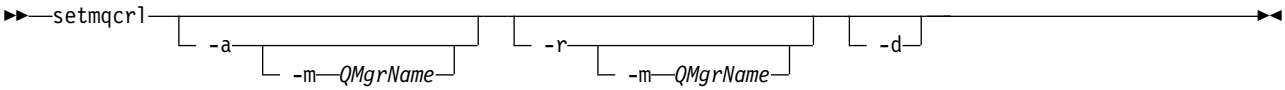
Note: The **setmqcrl** command applies to IBM MQ for Windows only.

Use the **setmqcrl** command to configure and administer support for publishing CRL (certificate revocation list) LDAP definitions in an Active Directory.

A domain administrator must use this command, or setmqscp**setmqcrl**, initially to prepare the Active Directory for IBM MQ usage and to grant IBM MQ users and administrators the relevant authorities to access and update the IBM MQ Active Directory objects. You can also use the `setmqcrl` command to display all the currently configured CRL server definitions available on the Active Directory, that is, those definitions referred to by the queue manager's CRL namelist.

The only types of CRL servers supported are LDAP servers.

Syntax



Optional parameters

You must specify one of `-a` (add), `-r` (remove) or `-d` (display).

-a Adds the IBM MQ MQI client connections Active Directory container, if it does not already exist. You must be a user with the appropriate privileges to create subcontainers in the *System* container of your domain. The IBM MQ folder is called `CN=IBM-MQClientConnections`. Do not delete this folder in any other way than by using the **setmqscp** command.

-d Displays the IBM MQ CRL server definitions.

-r Removes the IBM MQ CRL server definitions.

-m [* | qmgr]
Modifies the specified parameter (**-a** or **-r**) so that only the specified queue manager is affected. You must include this option with the **-a** parameter.

*** | qmgr**
* specifies that all queue managers are affected. This enables you to migrate a specific IBM MQ CRL server definitions file from one queue manager alone.

Examples

The following command creates the IBM-MQClientConnections folder and allocates the required permissions to IBM MQ administrators for the folder, and to child objects created subsequently. (In this, it is functionally equivalent to `setmqscp -a`.)

```
setmqcrl -a
```

The following command migrates existing CRL server definitions from a local queue manager, Paint.queue.manager, to the Active Directory.

Note: The command first deletes any other CRL definitions from the Active Directory.

```
setmqcrl -a -m Paint.queue.manager
```

setmqenv (set IBM MQ environment)

ULW

Use the **setmqenv** command to set up the IBM MQ environment on UNIX, Linux, and Windows.

Purpose

You can use the **setmqenv** command to automatically set up the environment for use with an installation of IBM MQ. Alternatively, you can use the **crtmqenv** command to create a list of environment variables and values to manually set each environment variable for your system; see "crtmqenv (create IBM MQ environment)" on page 208 for more information.

Note: Any changes you make to the environment are not persistent. If you log out, and log in again, your changes are lost.

You can specify which installation the environment is set up for by specifying a queue manager name, an installation name, or an installation path. You can also set up the environment for the installation that issues the **setmqenv** command by issuing the command with the **-s** parameter.

The **setmqenv** command sets the following environment variables, appropriate to your system:

- CLASSPATH
- INCLUDE
- LIB
- MANPATH
- MQ_DATA_PATH
- MQ_ENV_MODE
- MQ_FILE_PATH
- MQ_JAVA_INSTALL_PATH
- MQ_JAVA_DATA_PATH
- MQ_JAVA_LIB_PATH
- MQ_JAVA_JVM_FLAG
- MQ_JRE_PATH

- PATH

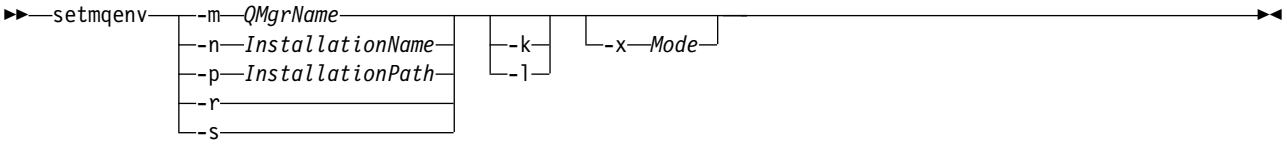
UNIX Linux On UNIX and Linux systems, if the **-l** or **-k** flag is specified:

- AIX The *LIBPATH* environment variable is set on AIX.
- The *LD_LIBRARY_PATH* environment variable is set on the following platforms:
  - HP-UX HP-UX
  - Linux Linux
  - Solaris Solaris

Usage notes

- The **setmqenv** command removes all directories for all IBM MQ installations from the environment variables before adding new references to the installation for which you are setting up the environment for. Therefore, if you want to set any additional environment variables that reference IBM MQ, set the variables after issuing the **setmqenv** command. For example, if you want to add *MQ_INSTALLATION_PATH*`/java/lib` to *LD_LIBRARY_PATH*, you must do so after running the **setmqenv** command.
- In some shells, command-line parameters cannot be used with **setmqenv** and any **setmqenv** command issued is assumed to be a `setmqenv -s` command. The command produces an informational message that the command has been run as if a `setmqenv -s` command had been issued. Therefore, in these shells you must ensure that you issue the command from the installation for which you want to set the environment for. In these shells, you must set the *LD_LIBRARY_PATH* variable manually. Use the **crtmqenv** command with the **-l** or **-k** parameter to list the *LD_LIBRARY_PATH* variable and value. Then use this value to set the *LD_LIBRARY_PATH*.

Syntax



Optional Parameters

-m *QMgrName*
Set the environment for the installation associated with the queue manager *QMgrName*.

-n *InstallationName*
Set the environment for the installation named *InstallationName*.

-p *InstallationPath*
Set the environment for the installation in the path *InstallationPath*.

-r Remove all installations from the environment.

-s Set the environment for the installation that issued the **setmqenv** command.

UNIX Linux **-k**
UNIX and Linux only.

Include the *LD_LIBRARY_PATH* or *LIBPATH* environment variable in the environment, adding the path to the IBM MQ libraries at the start of the current *LD_LIBRARY_PATH* or *LIBPATH* variable.

UNIX Linux **-l**
UNIX and Linux only.

Include the *LD_LIBRARY_PATH* or *LIBPATH* environment variable in the environment, adding the path to the IBM MQ libraries at the end of the current *LD_LIBRARY_PATH* or *LIBPATH* variable.

-x *Mode*

Mode can take the value 32 or 64.

Create a 32-bit or 64-bit environment. If this parameter is not specified, the environment matches that of the queue manager or installation specified in the command.

Any attempt to display a 64-bit environment with a 32-bit installation fails.

Return codes

Return code	Description
0	Command completed normally.
10	Command completed with unexpected results.
20	An error occurred during processing.

Examples

UNIX Linux The following examples assume that a copy of IBM MQ is installed in the `/opt/mqm` directory on a UNIX or Linux system.

Note: The period character (`.`) character used at the beginning of each command makes the **setmqenv** script run in the current shell. Therefore, the environment changes made by the **setmqenv** script are applied to the current shell. Without the period character (`.`), the environment variables are changed in another shell, and the changes are not applied to the shell from which the command is issued.

- The following command sets up the environment for an installation installed in the `/opt/mqm` directory:
  ```
  . /opt/mqm/bin/setmqenv -s
  ```
- The following command sets up the environment for an installation installed in the `/opt/mqm2` directory, and includes the path to the installation at the end of the current value of the *LD_LIBRARY_PATH* variable:
  ```
  . /opt/mqm/bin/setmqenv -p /opt/mqm2 -l
  ```
- The following command sets up the environment for queue manager `QM1` in a 32-bit environment:
  ```
  . /opt/mqm/bin/setmqenv -m QM1 -x 32
  ```

Windows The following example assumes that a copy of IBM MQ is installed in `C:\Program Files\IBM\MQ` on a Windows system. This command sets up the environment for an installation called `Installation1`:

```
"C:\Program Files\IBM\MQ\bin\setmqenv.cmd" -n Installation1
```

Related reference:

"crtmqenv (create IBM MQ environment)" on page 208
Create a list of environment variables for an installation of IBM MQ, on UNIX, Linux, and Windows.

Related information:

Choosing a primary installation

Multiple installations

setmqinst (set IBM MQ installation)

ULW

Set IBM MQ installations, on UNIX, Linux, and Windows.

Purpose

You can use the **setmqinst** command to change the installation description of an installation, or to set or unset an installation as the primary installation. To change the primary installation, you must unset the current primary installation before you can set a new primary installation. This command updates information contained in the `mqinst.ini` file.

You can use the **dspmqinst** command to display the installations.

After unsetting the primary installation, the **setmqinst** command will not be available unless you specify the full path or have an appropriate installation directory on your `PATH` (or equivalent). The default path in a system standard location will have been deleted.

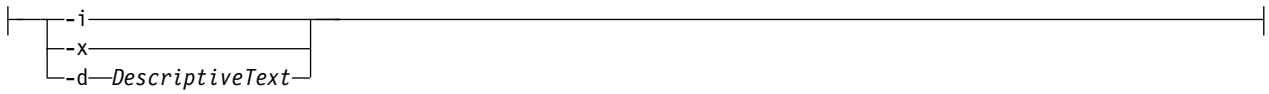
On UNIX platforms you should not assume that the current directory is in the path. If you are in `/opt/mqm/bin` and want to run, for example, `/opt/mqm/bin/dspmqver` you need to enter "**/opt/mqm/bin/dspmqver**" or "**./dspmqver**".

File `mqinst.ini` contains information about all IBM MQ installations on a system. For more information about `mqinst.ini`, see Installation configuration file, mqinst.ini.

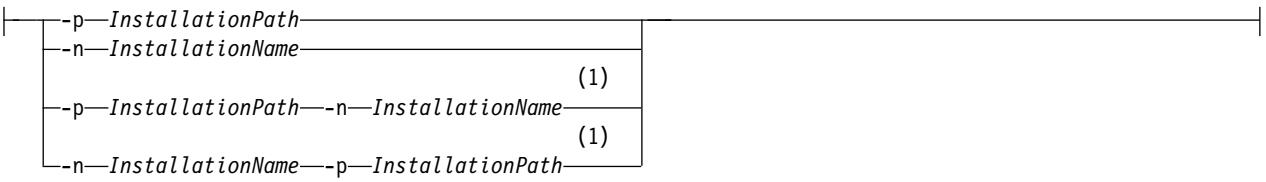
On UNIX or Linux systems, you must run this command as root. On Windows systems, you must run this command as a member of the Administrators group. The command does not have to be run from the installation you are modifying.

Syntax

Action:



Installation:



Notes:

1 When specified together, the installation name and installation path must refer to the same installation.

Parameters

-d *DescriptiveText*
Text that describes the installation.

The text can be up to 64 single-byte characters, or 32 double-byte characters. The default value is all blanks. You must use double quotation marks around the text if it contains spaces.

-i Set this installation as the primary installation.

-x Unset this installation as the primary installation.

-n *InstallationName*
The name of the installation to modify.

-p *InstallationPath*
The path of the installation to modify, for example, `opt/mqm`. You must use double quotation marks around the path if it contains spaces.

Return codes

Return code	Description
0	Entry set without error
36	Invalid arguments supplied
37	Descriptive text was in error
44	Entry does not exist
59	Invalid installation specified
71	Unexpected error
89	ini file error
96	Could not lock ini file
98	Insufficient authority to access ini file
131	Resource problem

Examples

1. This command sets the installation with the name of `myInstallation` as the primary installation:
   ```
   setmqinst -i -n myInstallation
   ```
2. This command sets the installation with an installation path of `/opt/myInstallation` as the primary installation:
   ```
   setmqinst -i -p /opt/myInstallation
   ```
3. This command unsets the installation named myInstallation as the primary installation:
   ```
   setmqinst -x -n myInstallation
   ```
4. This command unsets the installation with an installation path of `/opt/myInstallation` as the primary installation:

```
setmqinst -x -p /opt/myInstallation
```

5. This command sets the descriptive text for the installation named `myInstallation`:

```
setmqinst -d "My installation" -n myInstallation
```

The descriptive text is enclosed in quotation marks as it contains spaces.

Related information:

Choosing a primary installation

Changing the primary installation

setmqm (set queue manager)

ULW

Set the associated installation of a queue manager.

Purpose

Use the **setmqm** command to set the associated IBM MQ installation of a queue manager. The queue manager can then be administered using only the commands of the associated installation. For example, when a queue manager is started with **strmqm**, it must be the **strmqm** command of the installation that was specified by the **setmqm** command.

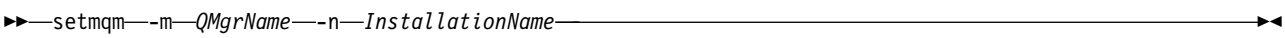
For more information about using this command, including information about when to use it, see Associating a queue manager with an installation.

This command is only applicable to UNIX, Linux and Windows.

Usage notes

- You must use the **setmqm** command from the installation you want to associate the queue manager with.
- The installation name specified by the **setmqm** command must match the installation from which the **setmqm** command is issued.
- You must stop the queue manager before executing the **setmqm** command. The command fails if the queue manager is still running.
- Once you have set the associated installation of a queue manager using the **setmqm** command, migration of the queue manager's data occurs when you start the queue manager using the **strmqm** command.
- Once you have started the queue manager on an installation, you cannot then use **setmqm** to set the associated installation to an earlier version of IBM MQ, as it is not possible to migrate back to earlier versions of IBM MQ.
- You can find out which installation is associated with a queue manager by using the **dspmq** command. See “dspmq (display queue managers)” on page 236 for more information.

Syntax



Required Parameters

-m ***QMgrName***

The name of the queue manager to set the associated installation for.

-n *InstallationName*
The name of the installation that the queue manager is to be associated with. The installation name is not case-sensitive.

Return codes

Return code	Description
0	Queue manager set to an installation without error
5	Queue manager running
36	Invalid arguments supplied
59	Invalid installation specified
60	Command not executed from the installation named by the -n parameter
61	Invalid installation name for this queue manager
69	Resource problem
71	Unexpected error
72	Queue manager name error
119	User not authorized

Examples

1. This command associates a queue manager QMGR1, with an installation with the installation name of myInstallation.

```
MQ_INSTALLATION_PATH/bin/setmqm -m QMGR1 -n myInstallation
```

setmqprd (enroll production license)

Enroll an IBM MQ production license.

A license is normally enrolled as part of the installation process.

Note: You must have the appropriate privileges to run this command on your system. UNIX requires root access, and Windows with UAC (User Account Control) requires Administrator access to run this command.

Syntax

Required parameters

LicenseFile
Specifies the fully-qualified name of the production license certificate file.

The full license file is `amqpcert.lic`:

- UNIX Linux On UNIX and Linux, it is in the */MediaRoot*/licenses directory on the installation media.
- Windows On Windows it is in the *MediaRoot*\licenses directory on the installation media. It is installed into the bin directory on the IBM MQ installation path.
- IBM i On IBM i, issue the command

```
CALL PGM(QMQM/SETMQPRD) PARM('/QOPT/OPT01/amqpcert.lic')
```

Trial license conversion

A trial license installation is identical to a production license installation, except for the "count-down" message that is displayed when you start a queue manager on an installation with a trial license. Parts of

IBM MQ that are not installed on the server, such as the IBM MQ MQI client, continue to work after the expiry of the trial license. You do not need to run **setmqprd** to enroll them with a production license.

When a trial license expires, you can still uninstall IBM MQ. You can also reinstall IBM MQ with a full production license.

Run **setmqprd** to enroll a production license after installing and using a installation with a trial license.

Related information:

Converting a trial license on AIX

Converting a trial license on HP-UX

Converting a trial license on Linux

Converting a trial license on Solaris

Converting a trial license on Windows

setmqscp (set service connection points)

Windows

Publish client connection channel definitions in an Active Directory (Windows only).

Purpose

Note: The **setmqscp** command applies to IBM MQ for Windows only.

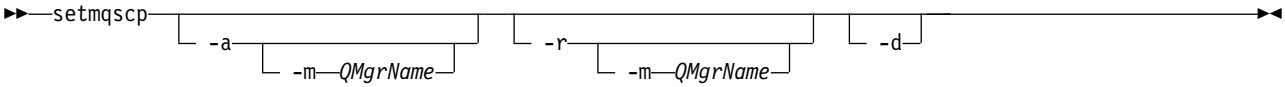
Use the **setmqscp** command to configure and administer support for publishing client connection channel definitions in an Active Directory.

Initially, this command is used by a domain administrator to:

- Prepare the Active Directory for IBM MQ use
- Grant IBM MQ users and administrators the relevant authorities to access and update the IBM MQ Active Directory objects

You can also use the **setmqscp** command to display all the currently configured client connection channel definitions available on the Active Directory.

Syntax



Optional parameters

You must specify one of -a (add), -r (remove) or -d (display).

-a Adds the IBM MQ MQI client connections Active Directory container, if it does not already exist. You must be a user with the appropriate privileges to create subcontainers in the *System* container of your domain. The IBM MQ folder is called `CN=IBM-MQClientConnections`. Do not delete this folder in any other way than by using the `setmqscp -r` command.

-d Displays the service connection points.

-r Removes the service connection points. If you omit **-m**, and no client connection definitions exist in the IBM-MQClientConnections folder, the folder itself is removed from the Active Directory.

-m [* | qmgr]
Modifies the specified parameter (-a or -r) so that only the specified queue manager is affected.

*** | qmgr**
* specifies that all queue managers are affected. This enables you to migrate a specific client connection table file from one queue manager alone, if required.

Examples

The following command creates the IBM-MQClientConnections folder and allocates the required permissions to IBM MQ administrators for the folder, and to child objects created subsequently:

```
setmqscp -a
```

The following command migrates existing client connection definitions from a local queue manager, Paint.queue.manager, to the Active Directory:

```
setmqscp -a -m Paint.queue.manager
```

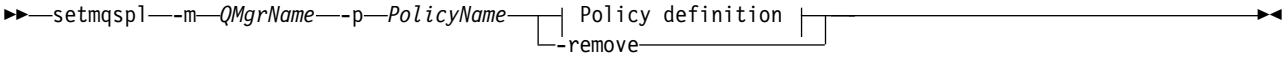
The following command migrates all client connection definitions on the local server to the Active Directory:

```
setmqscp -a -m *
```

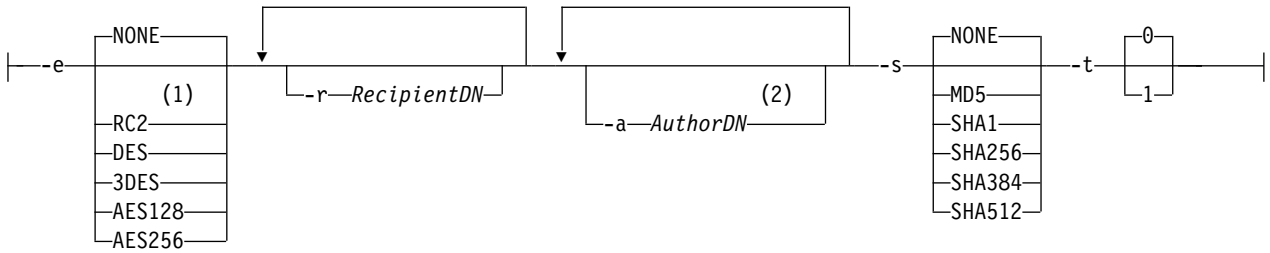
setmqspl (set security policy)

Use the **setmqspl** command to define a new security policy, alter an already existing one, or remove an existing policy.

Syntax



Policy definition:



Notes:

1 If an encryption algorithm is selected, a recipient DN must also be provided.

2 If an author DN is provided, a signing algorithm must also be selected.

Table 83. `setmqspl` command flags

Command flag	Explanation
-m	Queue manager name. This flag is mandatory for all actions on security policies.
-p	Policy name. Set the policy name to the name of the queue you want the policy to apply to.
-s	Digital signature algorithm. Advanced Message Security supports the following values: `MD5`, `SHA1`, `SHA256`, `SHA384`, and `SHA512`. All must be in uppercase. The default value is `NONE`. **Important:** • For the SHA384 and SHA512 cryptographic hash functions, keys used for signing must be longer than 768 bits. • The name of the signature algorithm must be provided in uppercase. • V 9.0.0 From Version 9.0, with the `Confidentiality` policy, the signature algorithm must be `NONE`. For more information about the `Confidentiality` policy, see Qualities of protection available with AMS.
-e	Digital encryption algorithm. Advanced Message Security supports the following encryption algorithms: `RC2`, `DES`, `3DES`, `AES128`, `AES256`. The default value is `NONE`. **Important:** The name of the encryption algorithm must be provided in uppercase
-r	The distinguished name (DN) of the message recipient (the certificate of a provided DN is used to encrypt a given message). Recipients can be specified only if the encryption algorithm is different from `NONE`. Multiple recipients can be included for a message. Each DN must be provided with a separate **-r** flag. **Important:** • DN attribute names must be in uppercase. • Commas must be used as a name separators. • To avoid command interpreter errors, place quotation marks around the DNs. For example: `-r "CN=alice, O=ibm, C=US"`

Table 83. `setmqspl` command flags (continued)

Command flag	Explanation
-a	Signature DN that is validated during message retrieval. Only messages signed by a user with a provided DN are accepted during the retrieval. Signature DNs can be specified only if the signature algorithm is different from `NONE`. Multiple authorized signers can be specified, each authorized signer needs to have a separate **-a** flag. **Important:** The attribute in the DN name must be in upper case. Specify `CN=` rather than `cn=`. The attribute values in the DN are case sensitive so, for example, `CN=USERID1` is different from `CN=userid1`.
-t	The toleration flag indicates whether messages that do not meet the requirements of the policy can still be successfully browsed or retrieved by an application. Toleration may be useful for example when introducing a policy to a queue which already contains unprotected messages. Valid values include: • **0 (default)** Toleration flag off. • **1** Toleration flag on. Toleration is optional and facilitates staged implementation, where policies were applied to queues but those queues may already contain messages that have no policy, or still receive messages from remote systems that do not have the security policy set.
V 9.0.0 **-c**	The key reuse count can be provided as an integer from 1 through 9,999,999. Special values are: • **0** Keys are not reused. • ***** Allows applications to reuse an encryption key an unlimited number of times. If you omit the **-c** parameter when defining a policy, a key reuse count of 0 is assumed for backwards compatibility with previous versions of Advanced Message Security and IBM WebSphere MQ Extended Security Edition. Note that a non-zero key reuse count is only valid for a confidentiality policy. If you attempt to create or modify an integrity or privacy policy, with a non-zero key reuse count, you receive error message `AMQ9091: Key reuse is not valid for policy` and the policy operation fails.
-remove	Delete policy. Only the policy name flag, **-p** is valid for use in combination with this flag.

V 9.0.0

Examples

The following list shows examples of some valid **setmqspl** commands on Multiplatforms:

```
setmqspl -m QMGR -p PROT -s SHA256
setmqspl -m QMGR -p PROT -s SHA256 -a "CN=Alice, O=IBM, C=US"
setmqspl -m QMGR -p PROT -s SHA256 -e AES128 -a "CN=Alice, O=IBM, C=US" -r "CN=Bob, O=IBM, C=GB"
setmqspl -m QMGR -p PROT -e AES128 -r "CN=Bob, O=IBM, C=GB" -c 50
```

The following list shows examples of **setmqspl** commands that are not valid:

- No recipients specified:

```
setmqspl -m QMGR -p PROT -e AES128
```

- Key reuse not valid for an `Integrity` policy:

```
setmqspl -m QMGR -p PROT -s SHA256 -c 1
```

- Key reuse is not valid for a `Privacy` policy:

```
setmqspl -m QMGR -p PROT -s SHA256 -e AES128 -r "CN=Bob, O=IBM, C=GB" -c 1
```

z/OS On z/OS, you can use the **setmqspl** command with the CSQ0UTIL utility. For more information, see The message security policy utility (CSQ0UTIL).

setmqweb (set mqweb server configuration)

V 9.0.4

Add or remove a known configuration property from the `mqwebuser.xml` file.

Purpose

You can use the **setmqweb properties** command to configure the mqweb server. The mqweb server is used to support the IBM MQ Console and REST API.

z/OS

Using the command on z/OS

Before issuing either the **setmqweb** or **dspmqweb** commands on z/OS, you must set the WLP_USER_DIR environment variable, so that the variable points to your mqweb server configuration.

To do this, issue the following command:

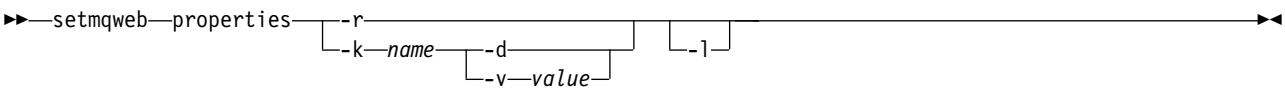
```
export WLP_USER_DIR=WLP_user_directory
```

where *WLP_user_directory* is the name of the directory passed to `crtmqweb.sh`. For example:

```
export WLP_USER_DIR=/var/mqm/web/installation1
```

See Create the Liberty server definition for more information.

Syntax



Parameters

-r Reset to default values. This parameter removes all user-modified configuration properties from the `mqwebuser.xml` file.

-k *name*

The name of the configuration property to add, update, or remove to or from the `mqwebuser.xml` file. The following values are the valid values for *name* on all platforms, including the IBM MQ Appliance:

ltpaExpiration
This configuration property is used to specify the time, in seconds, before the LTPA token expires.

The value for this property is an integer value.

maxTraceFiles
This configuration property is used to specify the maximum number of trace files that are generated by the mqweb server.

The value for this property is an integer value.

maxTraceFileSize
This configuration property is used to specify the maximum size, in MB, that each log file can reach.

The value for this property is an integer value.

mqRestCorsAllowedOrigins
This configuration property is used to specify the origins that are allowed to access the REST API. For more information about CORS, see Configuring CORS for the REST API.

The value for this property is a string value.

mqRestCorsMaxAgeInSeconds
This configuration property is used to specify the time, in seconds, that a web browser can cache the results of any CORS pre-flight checks.

The value for this property is an integer value.

V 9.0.5 **mqRestCsrfExpirationInMinutes**
This configuration property no longer exists at Version 9.0.5.

Applicable to Version 9.0.4 only, and used to specify the time, in minutes, before the CSRF token expires.

The value for this property is an integer value.

mqRestCsrfValidation
This configuration property is used to specify whether CSRF validation checks are performed. A value of `false` removes the CSRF token validation checks.

The value for this property is a boolean value.

mqRestGatewayEnabled
This configuration property is used to specify whether the administrative REST API gateway is enabled.

The value for this property is a boolean value.

mqRestGatewayQmgr
This configuration property is used to specify the name of the queue manager to use as the gateway queue manager. This queue manager must be in the same installation as the mqweb server. A blank value indicates that no queue manager is configured as the gateway queue manager.

The value for this property is a string value.

mqRestMessagingEnabled
This configuration property is used to specify whether the messaging REST API is enabled.

The value for this property is a boolean value.

mqRestRequestTimeout
This configuration property is used to specify the time, in seconds, before a REST request times out.

The value for this property is an integer value.

traceSpec
This configuration property is used to specify the level of trace that is generated by the mqweb server. For a list of possible values, see Configuring logging for the IBM MQ Console and REST API.

The value for this property is a string value.

ULW z/OS

The following values are the additional valid values for *name* on z/OS, UNIX, Linux, and Windows:

httpHost
This configuration property is used to specify the HTTP host name as an IP address, domain name server (DNS) host name with domain name suffix, or the DNS host name of the server where IBM MQ is installed.

You can use an asterisk in double quotation marks to specify all available network interfaces.

You can use the value `localhost` to allow only local connections.

The value for this property is a string value.

httpPort
This configuration property is used to specify the HTTP port number that is used for HTTP connections.

You can use a value of `-1` to disable the port.

The value for this property is an integer value.

httpsPort
This configuration property is used to specify the HTTPS port number that is used for HTTPS connections.

You can use a value of `-1` to disable the port.

The value for this property is an integer value.

mqConsoleAutostart
This configuration property is used to specify whether the IBM MQ Console automatically starts when the mqweb server starts.

The value for this property is a boolean value.

mqRestAutostart
This configuration property is used to specify whether the REST API automatically starts when the mqweb server starts.

The value for this property is a boolean value.

-d Deletes the specified configuration property from the `mqwebuser.xml` file.

-v *value*
The value of the configuration property to add to, or update in, the `mqwebuser.xml` file. Any existing configuration properties of the same *name* are overwritten. Duplicate configuration properties are removed.

The value is case-sensitive. To specify an asterisk, multiple tokens, or an empty value, enclose the value in double quotation marks.

The *value* that is specified is not validated. If incorrect values are specified a subsequent attempt to start the mqweb server might fail.

-l Enable verbose logging. Diagnostic information is written to an mqweb server log file.

Return codes

Return code	Description
0	Command successful
>0	Command not successful.

For a full list of server command exit codes, see Liberty:server command options in the WebSphere Application Server documentation.

Related commands

Command	Description
strmqweb	Start the mqweb server.
endmqweb	Stop the mqweb server.
dspmqweb	Display the status or configuration of the mqweb server.

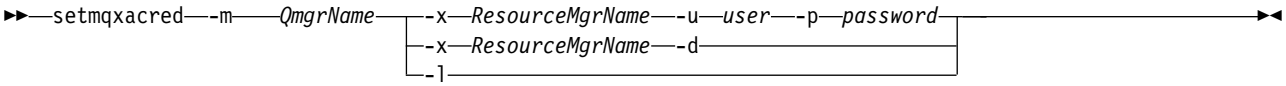
`setmqxacred` (add XA credentials)

Use the **setmqxacred** command to add or modify credentials in the IBM MQ XA credentials store.

Purpose

The **setmqxacred** command adds new credentials to the IBM MQ XA credentials store, or modifies or deletes existing credentials.

Syntax



Required parameters

-m ***QmgrName***

The queue manager for which authentication details are stored.

Optional parameters

-x ***ResourceMgrName***

Specifies the resource manager name as defined in the `qm.ini` file.

-u ***user***

Specifies the user name to use to connect to the database.

-p ***password***

Specifies the password for the user.

-d

Deletes the credentials for the named resource manager.

-l

Lists the credentials in the queue manager store.

Examples

To add credentials for the queue manager QM1 for the resource mqdb2:

```
# setmqxacred -m QM1 -x mydb2 -u user1 -p Password1
Successfully added credentials for XA Resource Manager mydb2
```

To delete the credentials for the queue manager QM1 for the resource mqdb2:

```
# setmqxacred -m QM1 -x mydb2 -d
Successfully removed credentials for XA Resource Manager mydb2
```

To list details about the credentials stored in the credentials store.

```
# setmqxacred -m QM1 -l
ResourceName(mydb2) UserName(user1)
ResourceName(myora) UserName(user2)
```

strmqcfg (start IBM MQ Explorer)

Linux Windows

Start IBM MQ Explorer (Windows and Linux x86-64 platforms only).

Purpose

Windows For IBM MQ for Windows only, note that if you use runas to execute this command, you must define the Environment Variable *APPDATA* to set a path to a directory that the user you are running as has access. For example:

```
set APPDATA=C:\Users\user_name\AppData\Roaming
```

You can use the following command to identify the path that *APPDATA* is set to:

```
set APPDATA
```

Linux On Linux, to start IBM MQ Explorer successfully, you must be able to write a file to your home directory, and the home directory must exist.

Note: The preferred way to start IBM MQ Explorer on Windows and Linux systems is by using the system menu, or the MQExplorer executable file.

Syntax

The syntax of this command follows:

Optional parameters

-c **-clean** is passed to Eclipse. This parameter causes Eclipse to delete any cached data used by the Eclipse runtime.

-i **-clean -initialize** is passed to Eclipse. This parameter causes Eclipse to delete any cached data as well as discard configuration information used by the Eclipse runtime.

IBM MQ Explorer starts briefly and then ends without displaying the user interface.

-x Output debug messages to the console.

strmqbrk (migrate IBM WebSphere MQ Version 6.0 publish/subscribe broker to later version)

Multi

Migrate the persistent state of an IBM MQ publish/subscribe broker to a later version queue manager.

Purpose

Use the **strmqbrk** command to migrate the state of a IBM WebSphere MQ Version 6.0 publish/subscribe broker to a later version queue manager. If the queue manager has already been migrated, no action is taken.

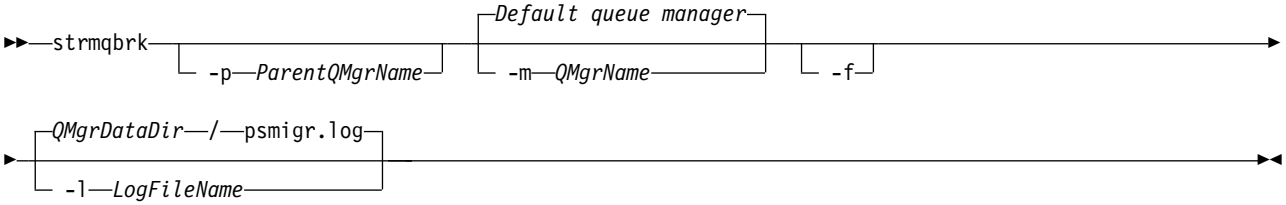
In IBM WebSphere MQ Version 6.0, **strmqbrk** started a broker. IBM MQ Version 8.0 publish/subscribe cannot be started in this manner. To enable publish/subscribe for a queue manager, use the **ALTER QMGR** command.

You can also use the **runmqbrk** command. This has the same parameters as **strmqbrk** and exactly the same effect.

ULW

Syntax

This syntax diagram applies to UNIX, Linux, and Windows



ULW

Optional parameters for UNIX, Linux, and Windows

-p *ParentQMgrName*

Note: This option is deprecated. **strmqbrk** migrates the parent connection automatically. If you specify the current parent queue manager, a warning message is issued and migration continues. If you specify a different queue manager, a error is issued and migration is not performed.

-m *QMgrName*
The name of the queue manager to be migrated. If you do not specify this parameter, the command is routed to the default queue manager.

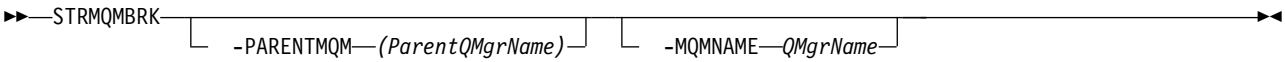
-f Force migration. This option specifies that objects created during the migration replace existing objects with the same name. If this option is not specified, if migration would create a duplicate object, a warning is issued, the object is not created, and migration continues.

-l *LogFileName*
Log migration activity to the file specified in *LogFileName*.

IBM i

Syntax

This syntax diagram applies to IBM i



IBM i

Optional parameters for IBM i

-PARENTMQM *ParentQMgrName)*

Note: This option is deprecated.
If you specify the current parent queue manager, a warning message is issued and migration continues. If you specify a different queue manager, a warning is issued and migration is not performed.

-MQMNAME *QMgrName*
The name of the queue manager to be migrated. If you do not specify this parameter, the command is routed to the default queue manager.

Related information:

ALTER QMGR
Use the MQSC command **ALTER QMGR** to alter the queue manager parameters for the local queue manager.

strmqcsv (start command server)

Start the command server for a queue manager.

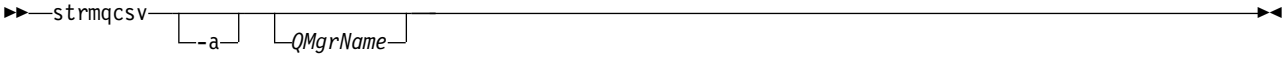
Purpose

Use the **strmqcsv** command to start the command server for the specified queue manager. This enables IBM MQ to process commands sent to the command queue.

You must use the **strmqcsv** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

If the queue manager attribute, SCMDSERV, is specified as QMGR then changing the state of the command server using **strmqcsv** does not effect how the queue manager acts upon the SCMDSERV attribute at the next restart.

Syntax



Required parameters

None

Optional parameters

-a Blocks the following PCF commands from modifying or displaying authority information:

- Inquire authority records (`MQCMD_INQUIRE_AUTH_RECS`)
- Inquire entity authority (`MQCMD_INQUIRE_ENTITY_AUTH`)
- Set authority record (`MQCMD_SET_AUTH_REC`).
- Delete authority record (`MQCMD_DELETE_AUTH_REC`).

QMgrName
The name of the queue manager on which to start the command server. If omitted, the default queue manager is used.

Return codes

Return code	Description
0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

The following command starts a command server for queue manager `earth`:

```
strmqcsv earth
```

Related commands

Command	Description
endmqcsv	End a command server
dspmqcsv	Display the status of a command server

Related reference:

"Command server commands" on page 188
A table of command server commands, showing the PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

strmqsvc (start IBM MQ service)

Windows

Start the IBM MQ service on Windows.

Purpose

The command starts the IBM MQ service on Windows. Run the command on Windows only.

If you are running IBM MQ on Windows systems with User Account Control (UAC) enabled, you must invoke **strmqsvc** with elevated privileges.

Run the command to start the service, if it has not been started automatically, or if the service has ended.

Restart the service for IBM MQ processes to pick up a new environment, including new security definitions.

Syntax

strmqsvc

Parameters

The **strmqsvc** command has no parameters.

You must set the path to the installation that contains the service. Either make the installation primary, run the **setmqenv** command, or run the command from the directory containing the **strmqsvc** binary file.

Related reference:
"endmqsvc" on page 273
End the IBM MQ service on Windows.

strmqm (start queue manager)

Start a queue manager or ready it for standby operation.

Purpose

Use the **strmqm** command to start a queue manager.

You must use the **strmqm** command from the installation that is associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

If a queue manager has no associated installation and there is no installation of IBM WebSphere MQ Version 7.0.1 on the system, the **strmqm** command associates the queue manager with the installation that issued the **strmqm** command.

If the queue manager startup takes more than a few seconds IBM MQ shows intermittent messages detailing the startup progress.

Usage notes

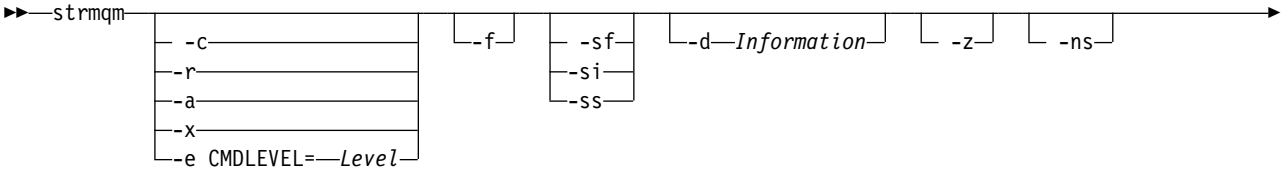
V 9.0.2 From IBM MQ Version 9.0.2, IBM MQ supports the use of back-up queue managers. That is, a queue manager where log extents are asynchronously copied to a backup machine, and where replay of the log records is periodically driven by use of the command **strmqm -r**. When the backup queue manager needs to be activated, use the command **strmqm -a** and then start the queue manager normally.

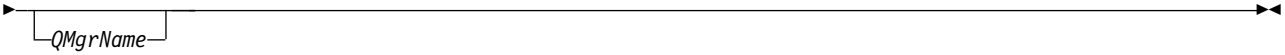
Attention: You cannot use **LogManagement**=*Automatic*, along with a backup queue manager, as extents might be reused before they are backed up. Furthermore, if you run the command **strmqm -r** together with **LogManagement**=*Automatic*, the command fails.

V 9.0.3 From IBM MQ Version 9.0.3, the security of *data path*`/log/`*qm*, on UNIX, has been changed from 2775 to 2770.

V 9.0.4 V 9.0.0.2 From IBM MQ Version 9.0.4 and IBM MQ Version 9.0.0, Fix Pack 2, the **strmqm** command checks the syntax of the CHANNELS and SSL stanzas in the `qm.ini` file early on, before starting the queue manager fully. If the `qm.ini` file contains any errors, this check makes it much easier to see what is wrong, and correct it quickly. If an error is found, **strmqm** outputs an AMQ9224 error message, describing the full details of the position of the error in the `qm.ini` file. It also ends immediately without starting the queue manager.

Syntax





Optional parameters

-a Activate the specified backup queue manager. The backup queue manager is not started.

When activated, a backup queue manager can be started by using the control command `strmqm` *QMgrName*. The requirement to activate a backup queue manager prevents accidental startup.

When activated, a backup queue manager can no longer be updated.

For more information about using backup queue managers, see Backing up and restoring IBM MQ queue manager data.

-c Starts the queue manager, redefines the default and system objects, then stops the queue manager. Any existing system and default objects that belong to the queue manager are replaced if you specify this flag, and any non-default system object values are reset (for example, the value of MCAUSER is set to blank).

Use the **crtmqm** command to create the default and system objects for a queue manager.

Note: If you run `strmqm -c` on a queue manager that is being used as a Managed File Transfer coordination queue manager you must re-run the MQSC script that defines the coordination queue manager objects. This script is in a file called *queue_manager_name*`.mqsc`, which is in the Managed File Transfer configuration directory.

-d *Information*
Specifies whether information messages are displayed. Possible values for *Information* are as follows:

Value	Description
all	All information messages are displayed. This value is the default value.
minimal	The minimal number of information messages are displayed.
none	No information messages are displayed. This parameter is equivalent to **-z**.

The *-z* parameter takes precedence over this parameter.

-e CMDLEVEL = *Level*
Enables a command level for this queue manager, and then stops the queue manager.

The queue manager is now able to use all functions that are provided by the specified command level. You can start the queue manager only with an installation that supports the new command level.

This option is only valid if the current command level that is used by the queue manager is lower than the maximum command level supported by the installation. Specify a command level that is greater than the current command level of the queue manager and less than or equal to the maximum command level supported by the installation.

Use exactly the command level as a value for *Level* that is associated with the function you want to enable.

This flag cannot be specified with -a, -c, -r or -x.

-f Use this option if you know that a queue manager is not starting because its data directories are missing or corrupted.

The **strmqm** **-f** *qmname* command attempts to re-create the queue manager data directory and reset file permissions. If it is successful, the queue manager starts, unless the queue manager configuration information is missing. If the queue manager fails to start because the configuration information is missing, re-create the configuration information, and restart the queue manager.

In releases of the product before IBM WebSphere MQ Version 7.0.1, **strmqm**, with no **-f** option, automatically repaired missing data directories and then tried to start. This behavior is changed.

From IBM WebSphere MQ Version 7.0.1 onwards, the default behavior of **strmqm**, with no **-f** option, is not to recover missing or corrupted data directories automatically, but to report an error, such as AMQ6235 or AMQ7001, and not start the queue manager.

You can think of the **-f** option as performing the recover actions that used to be performed automatically by **strmqm**.

The reason for the change to the behavior of **strmqm** is that with the support for networked file storage in IBM WebSphere MQ Version 7.0.1, the most likely cause of missing or corrupted queue manager data directories is a configuration error that can be rectified, rather than the data directories are corrupted or irretrievably unavailable.

You must not use **strmqm** **-f** to re-create the queue manager data directories if you can restore the directories by correcting the configuration.

Possible solutions to problems with **strmqm** are to make the networked file storage location accessible to the queue manager, or to ensure the gid and uid of the mqm group and user ID on the server hosting the queue manager match the gid and uid of the mqm group and user ID on the server that is hosting the queue manager data directory.

From IBM WebSphere MQ Version 7.0.1, if you are performing media recovery for a queue manager, then you must use the **-f** option to re-create the queue manager data directory.

-ns

Prevents any of the following processes from starting automatically when the queue manager starts:

- The channel initiator
- The command server
- Listeners
- Services

This parameter also runs the queue manager as if the CONNAUTH attribute is blank, regardless of its current value. This allows unauthenticated access to the queue manager for locally bound applications; client applications cannot connect because there are no listeners. Administrative changes must be made by using **runmqsc** because the command server is not running.

-r Updates the backup queue manager. The backup queue manager is not started.

IBM MQ updates the objects of the backup queue manager by reading the queue manager log and replaying updates to the object files.

For more information about using backup queue managers, see Backing up and restoring IBM MQ queue manager data.

Windows **-si**

Interactive (manual) queue manager startup type. This option is available on IBM MQ for Windows only.

The queue manager runs under the logged on (interactive) user. Queue managers that are configured with interactive startup end when the user who started them logs off.

If you set this parameter, it overrides any startup type set previously by the **crtmqm** command, the **amqmdain** command, or the IBM MQ Explorer Explorer.

If you do not specify a startup type of either **-si** or **-ss**, the queue manager startup type that is specified on the **crtmqm** command is used.

Windows **-ss**

Service (manual) queue manager startup type. This option is available on IBM MQ for Windows only.

The queue manager runs as a service. Queue managers that are configured with service startup continue to run even after the interactive user is logged off.

If you set this parameter, it overrides any startup type set previously by the **crtmqm** command, the **amqmdain** command, or the IBM MQ Explorer.

V 9.0.5 **-sf**

Foreground queue manager startup.

In a docker environment, it is better to run a queue manager in the foreground, as it blocks until the queue manager ends. In addition, this option also allows `stderr` to be more easily intercepted.

Windows On Windows platforms this option implies interactive startup.

IBM i On IBM i, starting in the foreground is not supported using the STRMQM CL command.

You can still use the **endmqm** command. However, on:

- UNIX UNIX, when the queue manager receives a SIGTERM (used by docker stop) or SIGINT signal while running in the foreground, the queue manager enters immediate shutdown mode.
- Windows Windows, if a CTRL_C_EVENT, CTRL_BREAK_EVENT, or CTRL_SHUTDOWN_EVENT event is received, the queue manager enters immediate shutdown mode.

-x

Start an instance of a multi-instance queue manager on the local server, permitting it to be highly available. If an instance of the queue manager is not already running elsewhere, the queue manager starts and the instance becomes active. The active instance is ready to accept local and remote connections to the queue manager on the local server.

If a multi-instance queue manager instance is already active on a different server the new instance becomes a standby, permitting it to takeover from the active queue manager instance. While it is in standby, it cannot accept local or remote connections.

You must not start a second instance of a queue manager on the same server.

The default behavior, omitting the `-x` optional parameter, is to start the instance as a single instance queue manager, forbidding standby instances from being started.

-z Suppresses error messages.

This flag is used within IBM MQ to suppress unwanted information messages. Because using this flag can result in loss of information, do not use it when you are entering commands on a command line.

This parameter takes precedence over the `-d` parameter.

QMgrName

The name of a local queue manager. If omitted, the default queue manager is used.

Return codes

Return code	Description
0	Queue manager started.
3	Queue manager being created.
5	Queue manager running.
16	Queue manager does not exist.
23	Log not available.
24	A process that was using the previous instance of the queue manager has not yet disconnected.
30	A standby instance of the queue manager started. The active instance is running elsewhere.
31	The queue manager already has an active instance. The queue manager permits standby instances.
39	Invalid parameter specified.
43	The queue manager already has an active instance. The queue manager does not permit standby instances.
47	The queue manager already has the maximum number of standby instances.
49	Queue manager stopping.
58	Inconsistent use of installations detected.
62	The queue manager is associated with a different installation.
69	Storage not available.
71	Unexpected error.
72	Queue manager name error.
74	The IBM MQ service is not started.
91	The command level is outside the range of acceptable values.
92	The queue manager's command level is greater or equal to the specified value.
100	Log location invalid.
114 V 9.0.2	Invalid QM.INI file stanza.
119	User not authorized to start the queue manager.

Examples

The following command starts the queue manager `account`:

```
strmqm account
```

Related reference:

crtmqm (create queue manager)
Create a queue manager.

dltmqm (delete queue manager)
Delete a queue manager.

dspmqver (display IBM MQ version information)
Display IBM MQ version and build information.

endmqm (end queue manager)
Stop a queue manager or switch to a standby queue manager.

Related information:

Applying maintenance level updates to multi-instance queue managers on Windows

Applying maintenance level updates to multi-instance queue managers on UNIX and Linux

strmqtrc (Start trace)

Enable trace at a specified level of detail, or report the level of tracing in effect.

Purpose

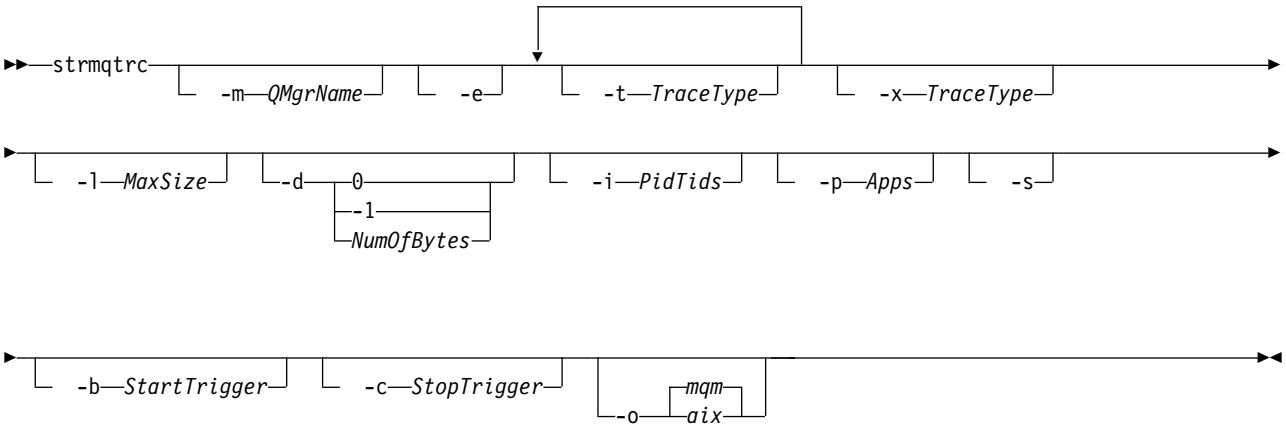
Use the **strmqtrc** command to enable tracing.

You must use the **strmqtrc** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

NSS Client This does not apply to a client product (for example, HP Integrity NonStop Server) because there are no queue managers from which to request direct output.

Syntax

The syntax of this command is as follows:



Description

The strmqtrc command enables tracing. The command has optional parameters that specify the level of tracing you want:

- One or more queue managers
- Levels of trace detail
- One or more IBM MQ processes. The processes can be either part of the IBM MQ product or customer applications that use the IBM MQ API
- Specific threads within customer applications, either by IBM MQ thread number or by operating system thread number
- Events. These can be either the entry or exit from internal IBM MQ functions or the occurrence of a first failure data capture (FDC).

Each combination of parameters on an individual invocation of the command are interpreted by IBM MQ as having a logical AND between them. You can start the strmqtrc command multiple times, regardless of whether tracing is already enabled. If tracing is already enabled, the trace options that are in effect are modified to those specified on the most recent invocation of the command. Multiple invocations of the command, without an intervening enqmqtrc command, are interpreted by IBM MQ as having a logical OR between them. The maximum number of concurrent strmqtrc commands that can be in effect at one time is 16.

NSS Client For the IBM MQ client on HP Integrity NonStop Server, you must direct your trace commands to specific processors. For example, if your client is running on processor 2 and your shell is on processor 1, initiating trace with **strmqtrc** *options* does not trace the client. In this case, `run -cpu=2 strmqtrc` is required.

Optional parameters

-m *QMgrName*

The name of the queue manager to trace. NSS Client This parameter applies to server products only.

The following wildcards are allowed: asterisk (*), replacing zero or more characters, and question mark (?), replacing any single character. In command environments such as the UNIX shell, where the asterisk (*) and question mark (?) characters have special meaning, you must either escape the wildcard character or enclose it in quotation marks to prevent the command environment from operating on the wildcard character.

-e Requests early tracing of all processes, making it possible to trace the creation or startup of a queue manager. If you include this flag, any process belonging to any component of any queue manager traces its early processing. The default is not to perform early tracing.

Use the following command to trace a client:

```
strmqtrc -e
```

You cannot use the -e flag with the -m flag, -i flag, the -p flag, the -c flag, or the -b flag. If you try to use the -e flag with the -m flag, the -i flag, the -p flag, the -c flag, or the -b flag, then an error message is issued.

-t *TraceType*

The points to trace and the amount of trace detail to record. By default **all** trace points are enabled and a default-detail trace is generated.

Alternatively, you can supply one or more of the options in the following list. For each *Tracetype* value you specify, including `-t all`, specify either `-t parms` or `-t detail` to obtain the appropriate level of trace detail. If you do not specify either `-t parms` or `-t detail` for any particular trace type, only a default-detail trace is generated for that trace type.

If you supply multiple trace types, each must have its own -t flag. You can include any number of -t flags, if each has a valid trace type associated with it.

It is not an error to specify the same trace type on multiple -t flags.

Table 84. TraceType parameter values.

Value	Description
`all`	Output data for every trace point in the system (the default). The all parameter activates tracing at default detail level.
`api`	Output data for trace points associated with the MQI and major queue manager components.
`commentary`	Output data for trace points associated with comments in the IBM MQ components.
`comms`	Output data for trace points associated with data flowing over communications networks.
`csdata`	Output data for trace points associated with internal data buffers in common services.
`csflows`	Output data for trace points associated with processing flow in common services.
`detail`	Activate tracing at high-detail level for flow processing trace points.
`Explorer`	Output data for trace points associated with the IBM MQ Explorer.

Table 84. TraceType parameter values. (continued)

Value	Description
`java`	Output data for trace points associated with applications using the IBM MQ classes for Java API.
`lqmdata`	Output data for trace points associated with internal data buffers in the local queue manager.
`lqmflows`	Output data for trace points associated with processing flow in the local queue manager.
`otherdata`	Output data for trace points associated with internal data buffers in other components.
`otherflows`	Output data for trace points associated with processing flow in other components.
`parms`	Activate tracing at default-detail level for flow processing trace points.
remotedata	Output data for trace points associated with internal data buffers in the communications component.
`remoteflows`	Output data for trace points associated with processing flow in the communications component.
`servicedata`	Output data for trace points associated with internal data buffers in the service component.
`serviceflows`	Output data for trace points associated with processing flow in the service component.
`spldata`	Output data for trace points associated with buffers and control blocks that use a security policy (AMS) operation.
`splflows`	Output data for trace points associated with entry and exit data for functions that use a security policy (AMS) operation.
`soap`	Output data for trace points associated with IBM MQ Transport for SOAP.
`TLS`	Output data associated with using GSKit to enable TLS channel security.
`versiondata`	Output data for trace points associated with the version of IBM MQ running.

-x ***TraceType***

The points **not** to trace. By default **all** trace points are enabled and a default-detail trace is generated. The trace points you can specify are those listed for the **-t** flag.

You can use the **-x** flag with *Tracetype* values to exclude those entry points you do not want to record. This is useful in reducing the amount of trace produced.

If you supply multiple trace types, each must have its own **-x** flag. You can include any number of **-x** flags, if each has a valid *Tracetype* associated with it.

-l ***MaxSize***

The maximum size of a trace file (`AMQ` *`ppppp`*`.` *`qq`*`.TRC`) in megabytes (MB). For example, if you specify a *MaxSize* of 1, the size of the trace is limited to 1 MB.

When a trace file reaches the specified maximum, it is renamed to `AMQ` *`ppppp`*`.` *`qq`*`.TRS` and a new `AMQ` *`ppppp`*`.` *`qq`*`.TRC` file is started. If a previous copy of an `AMQ` *`ppppp`*`.` *`qq`*`.TRS` file exists, it is deleted.

The highest value that *MaxSize* can be set to is 2048 MB.

-d

Trace options. The value can be:

0 Trace no user data.

-1 or all

Trace all user data.

NumOfBytes

- For a communication trace; trace the specified number of bytes of data including the transmission segment header (TSH).
- For an MQPUT or MQGET call; trace the specified number of bytes of message data held in the message buffer.
- Values in the range 1 through 15 are not allowed.

-i *PidTids*

Process identifier (PID) and thread identifier (TID) to which the trace generation is restricted. You cannot use the **-i** flag with the **-e** flag. If you try to use the **-i** flag with the **-e** flag, then an error message is issued. This parameter must only be used under the guidance of IBM Service personnel.

This parameter is not supported for .NET clients if NMQ_MQ_LIB is set to `managed`, so that the client uses managed IBM MQ problem diagnostics.

-p *Apps*

The named processes to which the trace generation is restricted. *Apps* is a comma-separated list. You must specify each name in the list exactly as the program name would be displayed in the "Program Name" FDC header. Asterisk (*) or question mark (?) wildcards are allowed. You cannot use the **-p** flag with the **-e** flag. If you try to use the **-p** flag with the **-e** flag, then an error message is issued.

This parameter is not supported for .NET clients if NMQ_MQ_LIB is set to `managed`, so that the client uses managed IBM MQ problem diagnostics.

-s Reports the tracing options that are currently in effect. You must use this parameter on its own with no other parameters.

A limited number of slots are available for storing trace commands. When all slots are in use, then no more trace commands can be accepted unless they replace an existing slot. Slot numbers are not fixed, so if the command in slot number 0 is removed, for example by an endmqtrc command, then all the other slots move up, with slot 1 becoming slot 0, for example. An asterisk (*) in a field means that no value is defined, and is equivalent to the asterisk wildcard.

An example of the output from this command is as follows:

```
Listing Trace Control Array

Used slots = 2 of 15

EarlyTrace    [OFF]
TimedTrace    [OFF]
TraceUserData [0]
MaxSize       [0]
Trace Type    [1]

Slot position 1

Untriggered
Queue Manager [avocet]
Application   [*]
PID.TID       [*]
TraceOptions  [1f4ffff]
TraceInterval    [0]
Trace Start Time [0]
Trace Stop  Time [0]
Start Trigger [KN346050K]
Start Trigger [KN346080]
```

```
Slot position 2

Untriggered
Queue Manager [*]
Application   [*]
PID.TID       [*]
TraceOptions  [1fcffff]
TraceInterval    [0]
Trace Start Time [0]
Trace Stop  Time [0]
Start Trigger [KN346050K]
Start Trigger [KN346080]
```

This parameter is not supported for .NET clients if NMQ_MQ_LIB is set to `managed`, so that the client uses managed IBM MQ problem diagnostics.

-b *Start_Trigger*

FDC probe IDs for which tracing must be turned on. *Start_Trigger* is a comma-separated list of FDC probe IDs. You can use asterisk (*) and question mark (?) wildcards in the specification of probe IDs. You cannot use the -b flag with the -e flag. If you try to use the -b flag with the -e flag, then an error message is issued. This parameter must only be used under the guidance of IBM Service personnel.

Start_Trigger	Effect
FDC=comma-separated list of FDC probe IDs.	Turns on tracing when any FDCs with the specified FDC probe IDs are generated.

This parameter is not supported for .NET clients if NMQ_MQ_LIB is set to `managed`, so that the client uses managed IBM MQ problem diagnostics.

-c *Stop_Trigger*

FDC probe IDs for which tracing must be turned off, or interval in seconds after which tracing must be turned off. *Stop_Trigger* is a comma-separated list of FDC probe IDs. You can use asterisk (*) and question mark (?) wildcards in the specification of probe IDs. This parameter should be used only under the guidance of IBM Service personnel.

Stop_Trigger	Effect
FDC=comma-separated list of FDC probe IDs.	Turns tracing off when any FDCs with the specified FDC probe IDs are generated.
interval=n where n is an unsigned integer between 1 and 32,000,000.	Turns tracing off n seconds after it starts or, if it tracing is already enabled, turns tracing off n seconds after this instance of the command is issued.

This parameter is not supported for .NET clients if NMQ_MQ_LIB is set to `managed`, so that the client uses managed IBM MQ problem diagnostics.

-o

mqm Enables IBM MQ trace as in previous releases.

This is the default value if no -o option is supplied.

AIX **aix**

Enables IBM MQ to write AIX system trace, provided AIX system trace is enabled.

As previously, you must use the AIX operating system trace command for any output to actually be produced.

This is a legacy option, and you should use this option only when directed to do so by IBM service personnel.

Return codes

Return code	Description
AMQ7024	Non-valid arguments supplied to the command.
AMQ7077	You are not authorized to perform the requested operation.
AMQ8304	Nine concurrent traces (the maximum) already running.
58	Inconsistent use of installations detected

Examples

UNIX This command enables tracing of processing flow from common services and the local queue manager for a queue manager called QM1 in IBM MQ for UNIX systems. Trace data is generated at the default level of detail.

```
strmqtrc -m QM1 -t csflows -t lqmflows -t parms
```

This command disables tracing of TLS activity on a queue manager called QM1. Other trace data is generated at the parms level of detail.

```
strmqtrc -m QM1 -x ssl -t parms
```

This command enables high-detail tracing of the processing flow for all components:

```
strmqtrc -t all -t detail
```

This command enables tracing when FDC KN346050 or FDC KN346080 occur on any process that is using queue manager QM1:

```
strmqtrc -m QM1 -b FDC=KN346050,KN346080
```

This command enables tracing when FDC KN34650 occurs, and stops tracing when FDC KN346080 occurs. In both cases the FDC must occur on a process that is using queue manager QM1:

```
strmqtrc -m QM1 -b FDC=KN346050 -c FDC=KN346080
```

The next examples use the -p and -m flags to show the following:

- How a combination of parameters on an individual invocation of the command are interpreted by IBM MQ as having a logical AND between them.
- How multiple invocations of the command, without an intervening enqmqtrc command, are interpreted by IBM MQ as having a logical OR between them:

1. This command enables tracing for all threads that result from any executing process called amqxxx.exe:

   ```
   strmqtrc -p amqxxx.exe
   ```

2.
   - If you start the following command after the command in step 1, without an intervening endmqtrc command, then tracing is limited to all threads that result from any executing process called amqxxx.exe *and* that are using queue manager QM2:

     ```
     strmqtrc -p amqxxx.exe -m QM2
     ```

   - If you start the following command after the command in step 1, without an intervening endmqtrc command, then tracing is limited to all processes and threads that result from executing amqxxx.exe *or* that are using queue manager QM2:

     ```
     strmqtrc -m QM2
     ```

Related commands

Command	Description
dspmqtrc	Display formatted trace output
endmqtrc	End trace

Related reference:

Command sets comparison: Other commands
A table of other commands, showing the command description, and its PCF command, MQSC command, and control command equivalents. The REST API resource and HTTP method equivalents, and IBM MQ Explorer equivalents, are included if available.

strmqweb (start mqweb server)

V 9.0.1

Start the mqweb server that is used to support the IBM MQ Console and REST API.

Purpose

Use the **strmqweb** command to start the mqweb server. You must start the mqweb server as a privileged user to use the IBM MQ Console or the REST API.

Syntax

Optional parameters

None.

Return codes

Return code	Description
0	Command successful
>0	Command not successful.

For a full list of server command exit codes, see Liberty:server command options in the WebSphere Application Server documentation.

Related commands

Command	Description
dspmqweb	Display the status of the mqweb server.
endmqweb	Stop the mqweb server.

MQSC reference

Use MQSC commands to manage queue manager objects, including the queue manager itself, queues, process definitions, channels, client connection channels, listeners, services, namelists, clusters, and authentication information objects.

For an overview of using MQSC commands for administering IBM MQ, see Administration using MQSC commands.

MQSC commands use certain special characters to have certain meanings. For more information about these special characters and how to use them, see "Generic values and characters with special meanings."

To find out how you can build scripts using MQSC commands, see "Building command scripts" on page 364.

z/OS For more information on building commands on z/OS, see "Using commands on z/OS" on page 365.

For the full list of MQSC commands, see "MQSC commands" on page 366.

Related concepts:

"IBM MQ Control commands reference" on page 195
Reference information about the IBM MQ control commands.

"Programmable command formats reference" on page 1068
Programmable Command Formats (PCFs) define command and reply messages that can be exchanged between a program and any queue manager (that supports PCFs) in a network. PCFs simplify queue manager administration and other network administration.

Related reference:

IBM i "CL commands reference for IBM i" on page 1065
A list of CL commands for IBM i, grouped according to command type.

Related information:

Administration using MQSC commands

Generic values and characters with special meanings

The following information describes generic values, and characters that have special meaning when you build MQSC commands.

Wherever a parameter can have a generic value, it is entered ending with an asterisk (*), for example ABC*. A generic value means 'all values beginning with'; so ABC* means 'all values beginning with ABC'.

If characters that require quotation marks are used in the value, the asterisk must be placed inside the quotation marks, thus 'abc*'. The asterisk must be the last or only character in the value.

The question mark (?) and colon (:) are not allowed in generic values.

Character	Description
	Blanks are used as separators. Multiple blanks are equivalent to a single blank, except in strings that have apostrophes (') round them. Any trailing blanks in those string attributes which are based on MQCHARV types are treated as significant.
,	Commas are used as separators. Multiple commas are equivalent to a single comma, except in strings that have apostrophes (') round them.

Character	Description
'	An apostrophe indicates the beginning or end of a string. IBM MQ leaves all characters that have quotation marks round them exactly as they are entered. The containing apostrophes are not included when calculating the length of the string.
''	Single quotation marks inside a string are treated by IBM MQ as one character when calculating the length of the string and the string is not terminated.
=	On z/OS, an equals sign indicates the start of a parameter value which is ended by a comma or blank.
(	An open parenthesis indicates the beginning of a parameter value or list of values.
)	A close parenthesis indicates the end of a parameter value or list of values.
:	A colon indicates an inclusive range. For example (1:5) means (1,2,3,4,5). This notation can be used only in TRACE commands.
*	An asterisk means "all". For example, DISPLAY TRACE (*) means display all traces, and DISPLAY QUEUE (PAY*) means display all queues with names that begin with PAY.

When you need to use any of these special characters in a field (for example as part of a description), you must enclose the whole string in single quotation marks.

Building command scripts

Use this information to learn how to build command scripts.

You might want to build the MQSC commands into a script when you use:

- z/OS The CSQINP1, CSQINP2, and CSQINPX initialization data sets or the CSQUTIL batch utility on z/OS.
- IBM i The **STRMQM** command on IBM i.
- ULW The **runmqsc** command on UNIX, Linux, and Windows.

When you do this, follow these rules:

- Each command must start on a new line.
- On each platform, there might be platform-specific rules about the line length and record format. If scripts are to be readily portable to different platforms, the significant length of each line should be restricted to 72 characters.
  - z/OS On z/OS, scripts are held in a fixed-format data set, with a record length of 80. Only columns 1 through 72 can contain meaningful information; columns 73 through 80 are ignored.
  - Multi On the Multiplatforms, each line can be of any length up to a maximum of 2048 characters.
- A line must not end in a keyboard control character (for example, a tab).
- If the last nonblank character on a line is:
  - A minus sign (-), this indicates that the command is to be continued from the start of the next line.
  - A plus sign (+), this indicates that the command is to be continued from the first nonblank character in the next line. If you use + to continue a command remember to leave at least one blank before the next parameter z/OS (except on z/OS where this is not necessary).

  Either of these can occur within a parameter, data value, or a string enclosed in quotation marks. For example,

```
'Fr+
 ed'
```

  and

```
   'Fr-
ed'
```

(where the 'e' of the second line of the second example is in the first position of the line) are both equivalent to

```
   'Fred'
```

MQSC commands that are contained within an Escape PCF (Programmable Command Format) command cannot be continued in this way. The entire command must be contained within a single Escape command. (For information about the PCF commands, see Introduction to Programmable Command Formats).

- + and - values used at the ends of lines are discarded when the command is reassembled into a single string.
- Multi On Multiplatforms you can use a semicolon character (;) to terminate a command, even if you have entered a plus sign (+) at the end of the previous line.
- z/OS You can also use the semicolon in the same way on z/OS for commands issued from the CSQUTIL batch utility program.
- A line starting with an asterisk (*) in the first position is ignored. This can be used to insert comments into the file.

  A blank line is also ignored.

  If a line ends with a continuation character (- or +), the command continues with the next line that is not a comment line or a blank line.
- When running MQSC commands interactively, you end the interactive session by typing the END command. This applies to:
  - ULW UNIX, Linux, and Windows systems, where you start the interactive session by entering `runmqsc`
  - IBM i IBM i systems, where you start the interactive session from the WRKMQM panel
- Windows On Windows, if certain special characters such as the pound sign ('£') and the logical NOT ('¬') are used in a command script (for example, as part of an object description), they are displayed differently in the output from a command such as **DISPLAY QLOCAL**.

Using commands on z/OS

z/OS

MQSC commands can be issued from various sources, depending on the command.

Commands can be issued from the following sources:

- The z/OS console or equivalent
- The initialization input data sets CSQINP1, CSQINP2, CSQINPT and CSQINPX
- The CSQUTIL batch utility
- Suitably authorized applications, sending commands as messages to the SYSTEM.COMMAND.INPUT queue

z/OS For further details, see Issuing commands,

However, not all commands can be issued from all these sources. Commands can be classified according to whether they can be issued from:

1 CSQINP1
2 CSQINP2
C The z/OS console

R The command server and command queue, by means of CSQUTIL, CSQINPT, CSQINPX, or applications

Within the command descriptions that follow, these sources are identified by the use of the characters 1, 2, C, and R in each command description.

MQSC commands

Use this topic as a reference to the MQSC commands.

This section describes, in alphabetical order, all the MQSC commands that can be issued by operators and administrators.

"STOP CHANNEL" on page 1046
"STOP CHANNEL (MQTT)" on page 1050
"STOP CHINIT on z/OS" on page 1051
"STOP CMDSERV on z/OS" on page 1052
"STOP CONN on Multiplatforms" on page 1053
"STOP LISTENER" on page 1054
"STOP QMGR on z/OS" on page 1056
"STOP SERVICE on Multiplatforms" on page 1057
"STOP SMDSCONN on z/OS" on page 1058
"STOP TRACE on z/OS" on page 1059
"SUSPEND QMGR" on page 1062

Related information:
Clustering: Using REFRESH CLUSTER best practices

ALTER AUTHINFO:

Use the MQSC command **ALTER AUTHINFO** to alter an authentication information object. These objects contain the definitions required to perform certificate revocation checking using OCSP or Certificate Revocation Lists (CRLs) on LDAP servers.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

Parameters not specified in the **ALTER AUTHINFO** command result in the existing values for those parameters being left unchanged.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

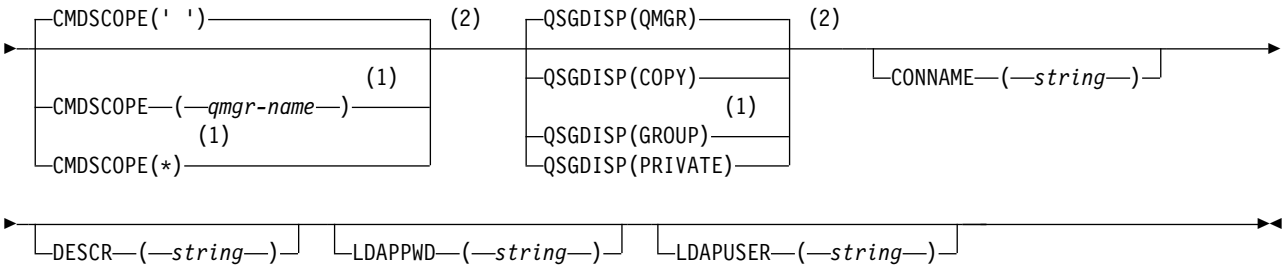
There are separate syntax diagrams for each **AUTHTYPE** parameter option:

- Syntax diagram for TYPE(CRLLDAP)
- Syntax diagram for TYPE(OCSP)
- Syntax diagram for TYPE(IDPWOS)
- Syntax diagram for TYPE(IDPWLDAP)
- "Parameter descriptions for ALTER AUTHINFO" on page 372

Synonym: **ALT AUTHINFO**

Syntax diagram for AUTHTYPE(CRLLDAP)

ALTER AUTHINFO



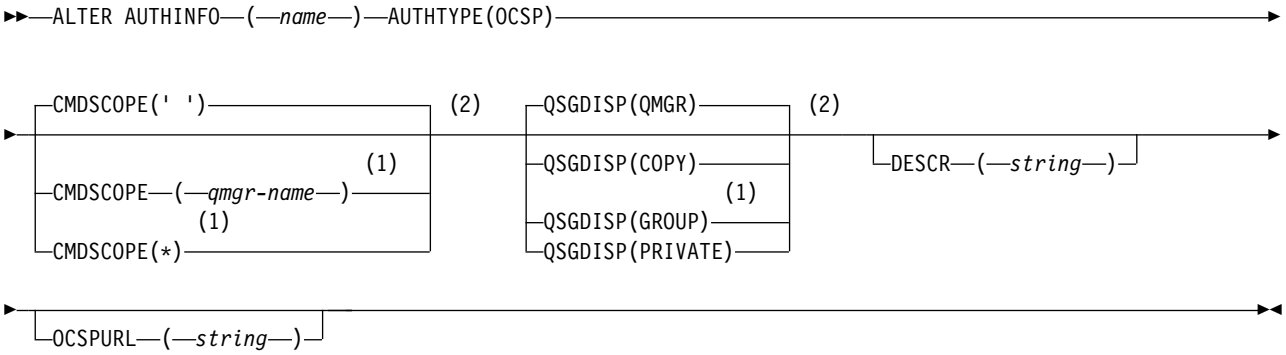
Notes:

1 Valid only when the queue manager is a member of a queue-sharing group. You can use queue-sharing groups only on IBM MQ for z/OS.

2 Valid only on z/OS.

Syntax diagram for AUTHTYPE(OCSP)

ALTER AUTHINFO



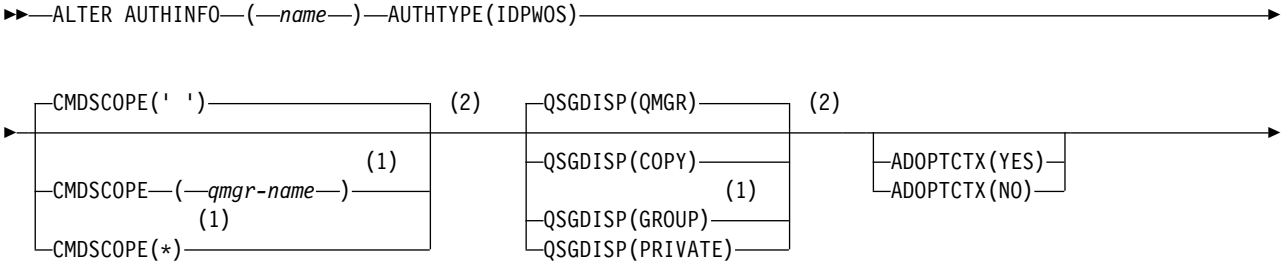
Notes:

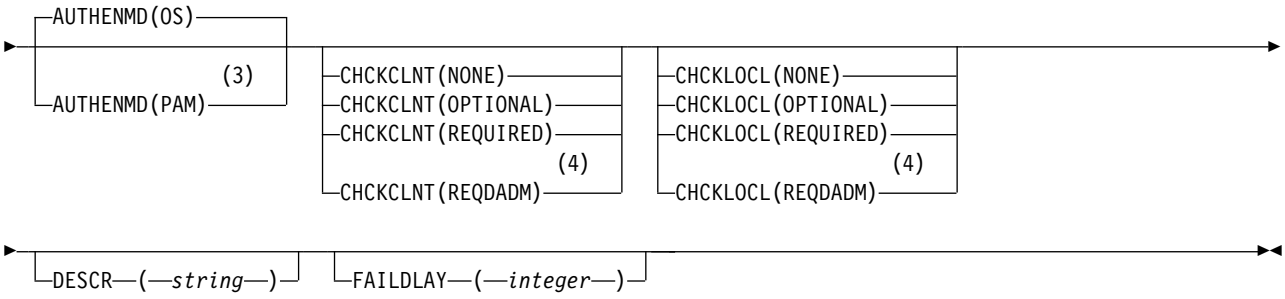
1 Valid only when the queue manager is a member of a queue-sharing group. You can use queue-sharing groups only on IBM MQ for z/OS.

2 Valid only on z/OS.

Syntax diagram for AUTHTYPE(IDPWOS)

ALTER AUTHINFO





Notes:

1 Valid only when the queue manager is a member of a queue-sharing group. You can use queue-sharing groups only on IBM MQ for z/OS.

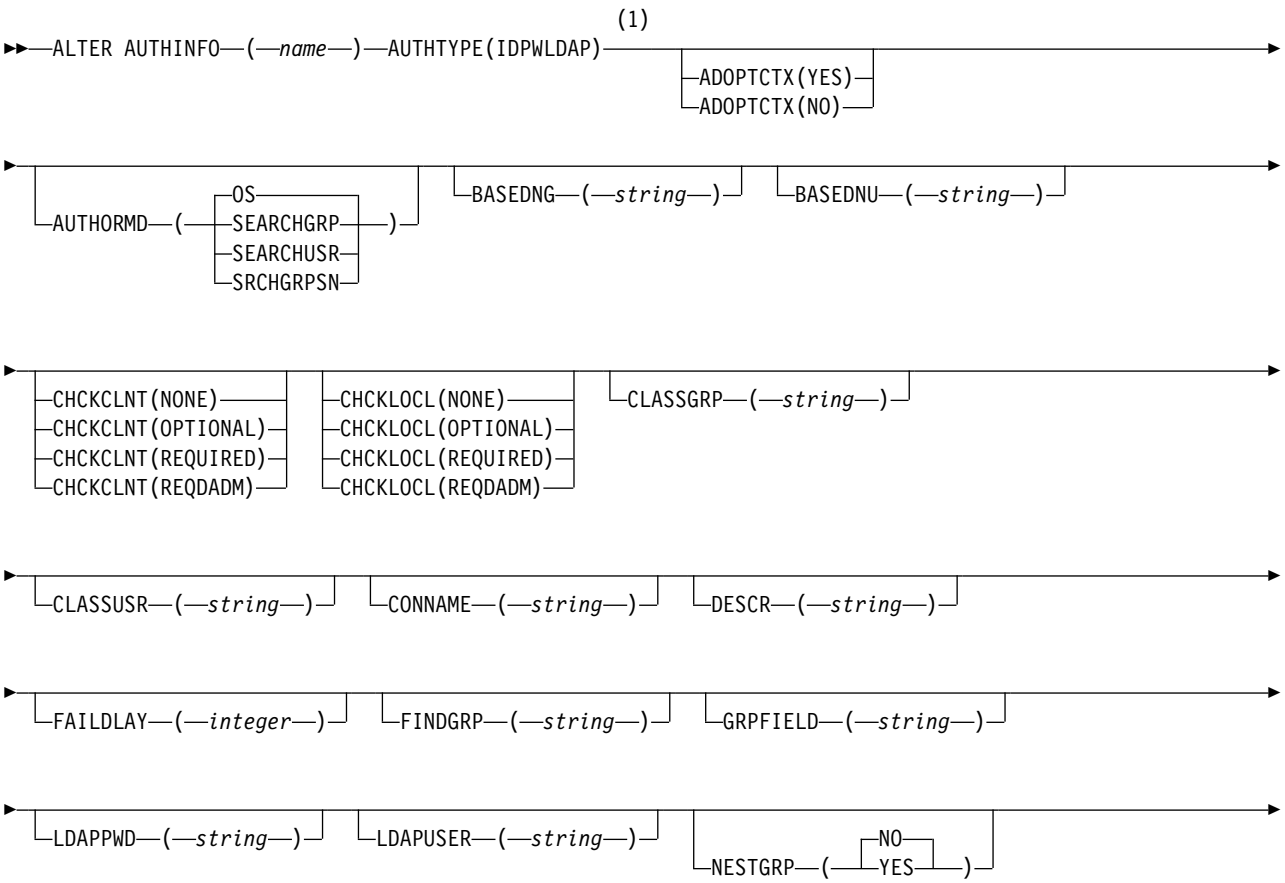
2 Valid only on z/OS.

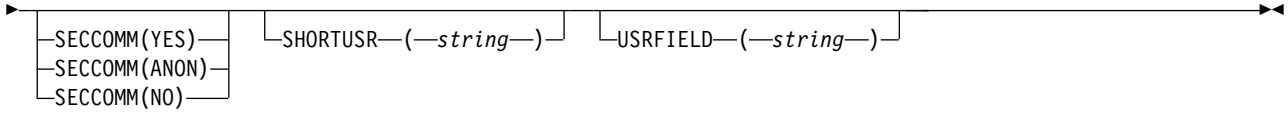
3 Not valid on z/OS and PAM value can be set only on UNIX.

4 Not valid on z/OS.

Syntax diagram for AUTHTYPE(IDPWLDAP)

ALTER AUTHINFO





Notes:

1 Not valid on z/OS.

Parameter descriptions for ALTER AUTHINFO

name Name of the authentication information object. This parameter is required.

The name must not be the same as any other authentication information object name currently defined on this queue manager (unless **REPLACE** or **ALTER** is specified). See Rules for naming IBM MQ objects.

ADOPTCTX

Whether to use the presented credentials as the context for this application. This means that they are used for authorization checks, shown on administrative displays, and appear in messages.

YES The user ID presented in the MQCSP structure, which has been successfully validated by password, is adopted as the context to use for this application. Therefore, this user ID will be the credentials checked for authorization to use IBM MQ resources.

If the user ID presented is an LDAP user ID, and authorization checks are done using operating system user IDs, the SHORTUSR associated with the user entry in LDAP will be adopted as the credentials for authorization checks to be done against.

NO Authentication is performed on the user ID and password presented in the MQCSP structure, but then the credentials are not adopted for further use. Authorization is performed using the user ID that the application is running under.

The **ADOPTCTX** attribute is only valid for an **AUTHTYPE** of IDPWOS and IDPWLDAP.

AUTHENMD

Authentication method. Whether to use the operating system or Pluggable Authentication Method (PAM) to authenticate user passwords.

OS UNIX Use the traditional UNIX password verification method.

PAM

Use the PAM to authenticate the user password.

UNIX Linux You can set the PAM value only on UNIX and Linux.

Changes to this attribute are effective only after you run the REFRESH SECURITY TYPE(CONNAUTH) command.

The **AUTHENMD** attribute is valid only for an **AUTHTYPE** of IDPWOS.

AUTHORMD

Authorization method.

OS Use operating system groups to determine permissions associated with a user.

This is how IBM MQ has previously worked, and is the default value.

SEARCHGRP

A group entry in the LDAP repository contains an attribute listing the Distinguished Name of all the users belonging to that group. Membership is indicated by the attribute defined in FINDGRP. This value is typically *member* or *uniqueMember*.

SEARCHUSR
A user entry in the LDAP repository contains an attribute listing the Distinguished Name of all the groups to which the specified user belongs. The attribute to query is defined by the FINDGRP value, typically *memberOf*.

V 9.0.5 **SRCHGRPSN**
A group entry in the LDAP repository contains an attribute listing the short user name of all the users belonging to that group. The attribute in the user record that contains the short user name is specified by SHORTUSR.

Membership is indicated by the attribute defined in FINDGRP. This value is typically *memberUid*.

Note: This authorization method should only be used if all user short names are distinct.

Many LDAP servers use an attribute of the group object to determine group membership and you should, therefore, set this value to SEARCHGRP.

Microsoft Active Directory typically stores group memberships as a user attribute. The IBM Tivoli Directory Server supports both methods.

In general, retrieving memberships through a user attribute will be faster than searching for groups that list the user as a member.

AUTHTYPE
The type of authentication information.

CRLLDAP
Certificate Revocation List checking is done using LDAP servers.

IDPWLDAP
Connection authentication user ID and password checking is done using an LDAP server.

IDPWOS
Connection authentication user ID and password checking is done using the operating system.

OCSP
Certificate revocation checking is done using OCSP.

IBM i z/OS An authentication information object with **AUTHTYPE(OCSP)** does not apply for use on IBM i or z/OS queue managers. However, it can be specified on those platforms to be copied to the client channel definition table (CCDT) for client use.

The **AUTHTYPE** parameter is required.

You cannot define an authentication information object as LIKE another authentication object with a different **AUTHTYPE**. You cannot alter the **AUTHTYPE** of an authentication information object after you have created it.

BASEDNG
Base DN for groups

In order to be able to find group names, this parameter must be set with the base DN to search for groups in the LDAP server.

BASEDNU(*base DN*)
In order to be able to find the short user name attribute, SHORTUSR, this parameter must be set with the base DN to search for users within the LDAP server.

The **BASEDNU** attribute is valid only for an **AUTHTYPE** of IDPWLDAP.

CHCKCLNT

This attribute determines the authentication requirements for client applications, and is valid only for an **AUTHTYPE** of IDPWOS or IDPWLDAP. The possible values are:

NONE No user ID and password checks are made. If any user ID or password is supplied by a client application, the credentials are ignored.

OPTIONAL

Client applications are not required to provide a user ID and password.

Any applications that do provide a user ID and password in the MQCSP structure have them authenticated by the queue manager against the password store indicated by the **AUTHTYPE**.

The connection is only allowed to continue if the user ID and password are valid.

This option might be useful during migration, for example.

REQUIRED

All client applications must provide a user ID and password in the MQCSP structure. This user ID and password is authenticated by the queue manager against the password store indicated by the **AUTHTYPE**.

The connection will only be allowed to continue if the user ID and password are valid.

REQDADM

All client applications using a privileged user ID must provide a user ID and password in the MQCSP structure. Any locally bound applications using a non-privileged user ID are not required to provide a user ID and password and are treated as with the OPTIONAL setting.

Any provided user ID and password are authenticated by the queue manager against the password store indicated by the **AUTHTYPE**. The connection is only allowed to continue if the user ID and password are valid.

Note: The REQDADM value for the **CHCKCLNT** attribute is irrelevant if the authentication type is LDAP. This is because there is no concept of privileged user ID when using LDAP user accounts. LDAP user accounts and groups must be assigned permission explicitly.

z/OS (This setting is not allowed on z/OS systems.)

Important:

1. This attribute can be overridden by the **CHCKCLNT** attribute of the CHLAUTH rule that matches the client connection. The CONNAUTH *AUTHINFO CHCKCLNT* attribute on the queue manager therefore determines the default client checking behavior for client connections that do not match a CHLAUTH rule, or where the CHLAUTH rule matched has **CHCKCLNT** ASQMGR.
2. If you select NONE and the client connection matches a CHLAUTH record with **CHCKCLNT** REQUIRED (or REQDADM on platforms other than z/OS), the connection fails. You receive the following message:
   - Multi AMQ9793 on Multiplatforms.
   - z/OS CSQX793E on z/OS.
3. This parameter is valid only with **TYPE(USERMAP)**, **TYPE(ADDRESSMAP)** and TYPE (SSLPEERMAP), and only when **USERSRC** is not set to NOACCESS.
4. This parameter applies only to inbound connections that are server-connection channels.

CHCKLOCL

This attribute determines the authentication requirements for locally bound applications, and is valid only for an **AUTHTYPE** of IDPWOS or IDPWLDAP.

IBM MQ Appliance For information about use of this attribute on IBM MQ Appliance, see Control commands on the IBM MQ Appliance in the IBM MQ Appliance documentation. The possible values are:

NONE No user ID and password checks are made. If any user ID or password is supplied by a locally bound application, the credentials are ignored.

OPTIONAL
Locally bound applications are not required to provide a user ID and password.

Any applications that do provide a user ID and password in the MQCSP structure have them authenticated by the queue manager against the password store indicated by the **AUTHTYPE**.

The connection is only allowed to continue if the user ID and password are valid.

This option might be useful during migration, for example.

REQUIRED
All locally bound applications must provide a user ID and password in the MQCSP structure. This user ID and password will be authenticated by the queue manager against the password store indicated by the **AUTHTYPE**. The connection will only be allowed to continue if the user ID and password are valid.

z/OS If your user ID has UPDATE access to the BATCH profile in the MQCONN class, you can treat **CHCKLOCL(REQUIRED)** as if it is **CHCKLOCL(OPTIONAL)**. That is, you do not have to supply a password, but if you do, the password must be the correct one.
See Using **CHCKLOCL** on locally bound applications.

REQDADM
All locally bound applications using a privileged user ID must provide a user ID and password in the MQCSP structure. Any locally bound applications using a non-privileged user ID are not required to provide a user ID and password and are treated as with the OPTIONAL setting.

Any provided user ID and password will be authenticated by the queue manager against the password store indicated by the **AUTHTYPE**. The connection will only be allowed to continue if the user ID and password are valid.

z/OS (This setting is not allowed on z/OS systems.)

CLASSGRP
The LDAP object class used for group records in the LDAP repository.

If the value is blank, groupOfNames is used.

Other commonly used values include groupOfUniqueNames or group.

CLASSUSR(*LDAP class user*)
The LDAP object class used for user records in the LDAP repository.

If blank, the value defaults to inetOrgPerson, which is generally the value needed.

For Microsoft Active Directory, the value you require is often *user*.

This attribute is valid only for an **AUTHTYPE** of IDPWLDAP.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if **QSGDISP** is set to GROUP.

'' The command runs on the queue manager on which it was entered.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

*
The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

CONNAME(*connection name*)
The host name, IPv4 dotted decimal address, or IPv6 hexadecimal notation of the host on which the LDAP server is running, with an optional port number.

If you specify the connection name as an IPv6 address, only systems with an IPv6 stack are able to resolve this address. If the **AUTHINFO** object is part of the CRL namelist of the queue manager, ensure that any clients using the client channel table generated by the queue manager can resolve the connection name.

z/OS On z/OS, if a **CONNAME** is to resolve to an IPv6 network address, a level of z/OS that supports IPv6 for connection to an LDAP server is required.

The syntax for **CONNAME** is the same as for channels. For example,

```
conname('hostname (nnn)')
```

where *nnn* is the port number.

The maximum length for the field is:

- Multi 264 characters on Multiplatforms.
- z/OS 48 characters on z/OS.

This attribute is valid only for an **AUTHTYPE** of CRLLDAP and IDPWLDAP, when the attribute is mandatory.

When used with an **AUTHTYPE** of IDPWLDAP, this can be a comma separated list of connection names.

DESCR(*string*)
Plain-text comment. It provides descriptive information about the authentication information object when an operator issues the **DISPLAY AUTHINFO** command (see "DISPLAY AUTHINFO" on page 738).

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

FAILDLAY(*delay time*)
When a user ID and password are provided for connection authentication, and the authentication fails due to the user ID or password being incorrect, this is the delay, in seconds, before the failure is returned to the application.

This can aid in avoiding busy loops from an application that simply retries, continuously, after receiving a failure.

The value must be in the range 0 - 60 seconds. The default value is 1.

The **FAILDLAY** attribute is valid only for an **AUTHTYPE** of IDPWOS and IDPWLDAP.

FINDGRP

Name of the attribute used within an LDAP entry to determine group membership.

When AUTHORMD = *SEARCHGRP*, the **FINDGRP** attribute is typically set to *member* or *uniqueMember*.

When AUTHORMD = *SEARCHUSR*, the **FINDGRP** attribute is typically set to *memberOf*.

V 9.0.5 When AUTHORMD = *SRCHGRPSN*, the **FINDGRP** attribute is typically set to *memberUid*.

When left blank, if:

- AUTHORMD = *SEARCHGRP*, the **FINDGRP** attribute defaults to *memberOf*
- AUTHORMD = *SEARCHUSR*, the **FINDGRP** attribute defaults to *member*
- V 9.0.5 AUTHORMD = *SRCHGRPSN*, the **FINDGRP** attribute defaults to *memberUid*

GRPFIELD

LDAP attribute that represents a simple name for the group.

If the value is blank, commands like **setmqaut** must use a qualified name for the group. The value can either be a full DN, or a single attribute.

LDAPPWD(*LDAP password*)

The password associated with the Distinguished Name of the user who is accessing the LDAP server. Its maximum size is 32 characters.

z/OS On z/OS, the **LDAPPWD** used for accessing the LDAP server might not be the one defined in the AUTHINFO object. If more than one AUTHINFO object is placed in the namelist referred to by the QMGR parameter **SSLCRLNL**, the **LDAPPWD** in the first AUTHINFO object is used for accessing all LDAP Servers.

The **GRPFIELD** attribute is valid only for an **AUTHTYPE** of CRLLDAP and IDPWLDAP.

LDAPUSER(*LDAP user*)

The Distinguished Name of the user who is accessing the LDAP server. (See the SSLPEER parameter for more information about distinguished names.)

The maximum size for the user name is:

- Multi 1024 characters on Multiplatforms.
- z/OS 256 characters on z/OS.

z/OS On z/OS, the **LDAPUSER** used for accessing the LDAP server might not be the one defined in the AUTHINFO object. If more than one AUTHINFO object is placed in the namelist referred to by the QMGR parameter **SSLCRLNL**, the **LDAPUSER** in the first AUTHINFO object is used for accessing all LDAP Servers.

Multi On Multiplatforms, the maximum accepted line length is defined to be BUFSIZ, which can be found in stdio.h.

The **LDAPUSER** attribute is valid only for an **AUTHTYPE** of CRLLDAP and IDPWLDAP.

NESTGRP

Group nesting.

NO Only the initially discovered groups are considered for authorization.

YES
The group list is searched recursively to enumerate all the groups to which a user belongs.

The group's Distinguished Name is used when searching the group list recursively, regardless of the authorization method selected in AUTHORMD.

OCSPURL(*Responder URL*)

The URL of the OCSP responder used to check for certificate revocation. This value must be an HTTP URL containing the host name and port number of the OCSP responder. If the OCSP responder is using port 80, which is the default for HTTP, then the port number can be omitted. HTTP URLs are defined in RFC 1738.

This field is case sensitive. It must start with the string `http://` in lowercase. The rest of the URL might be case sensitive, depending on the OCSP server implementation. To preserve case, use single quotation marks to specify the OCSPURL parameter value, for example:

```
OCSPURL ('http://ocsp.example.ibm.com')
```

This parameter is applicable only for **AUTHTYPE(OCSP)**, when it is mandatory.

z/OS **QSGDISP**

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

QSGDISP	ALTER
COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters **QSGDISP(COPY)**. Any object residing in the shared repository, or any object defined using a command that had the parameters **QSGDISP(QMGR)**, is not affected by this command.
GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameters **QSGDISP(GROUP)**. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command. If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero: `DEFINE AUTHINFO(name)` `REPLACE QSGDISP(COPY)` The ALTER for the group object takes effect regardless of whether the generated command with **QSGDISP(COPY)** fails.
PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with **QSGDISP(QMGR)** or **QSGDISP(COPY)**. Any object residing in the shared repository is unaffected.
QMGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters **QSGDISP(QMGR)**. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

SECCOMM

Whether connectivity to the LDAP server should be done securely using TLS

YES Connectivity to the LDAP server is made securely using TLS.

The certificate used is the default certificate for the queue manager, named in CERTLABL on the queue manager object, or if that is blank, the one described in Digital certificate labels, understanding the requirements.

The certificate is located in the key repository specified in SSLKEYR on the queue manager object. A cipherspec will be negotiated that is supported by both IBM MQ and the LDAP server.

If the queue manager is configured to use **SSLFIPS(YES)** or SUITEB cipher specs, then this is taken account of in the connection to the LDAP server as well.

ANON
Connectivity to the LDAP server is made securely using TLS just as for **SECCOMM(YES)** with one difference.

No certificate is sent to the LDAP server; the connection will be made anonymously. To use this setting, ensure that the key repository specified in SSLKEYR, on the queue manager object, does not contain a certificate marked as the default.

NO Connectivity to the LDAP server does not use TLS.

The **SECCOMM** attribute is valid only for an **AUTHTYPE** of IDPWLDAP.

SHORTUSR(*user name*)
A field in the user record to be used as a short user name in IBM MQ.

This field must contain values of 12 characters or less. This short user name is used for the following purposes:

- If LDAP authentication is enabled, but LDAP authorization is not enabled, this is used as an operating system user ID for authorization checks. In this case, the attribute must represent an operating system user ID.
- If LDAP authentication and authorization are both enabled, this is used as the user ID carried with the message in order for the LDAP user name to be rediscovered when the user ID inside the message needs to be used.

  For example, on another queue manager, or when writing report messages. In this case, the attribute does not need to represent an operating system user ID, but must be a unique string. An employee serial number is an example of a good attribute for this purpose.

The **SHORTUSR** attribute is valid only for an **AUTHTYPE** of IDPWLDAP and is mandatory.

USRFIELD(*user field*)
If the user ID provided by an application for authentication does not contain a qualifier for the field in the LDAP user record, that is, it does not contain an ' = ' sign, this attribute identifies the field in the LDAP user record that is used to interpret the provided user ID.

This field can be blank. If this is the case, any unqualified user IDs use the SHORTUSR parameter to interpret the provided user ID.

The contents of this field are concatenated with an ' = ' sign, together with the value provided by the application, to form the full user ID to be located in an LDAP user record. For example, the application provides a user of fred and this field has the value cn, then the LDAP repository will be searched for cn=fred.

The **USRFIELD** attribute is valid only for an **AUTHTYPE** of IDPWLDAP.

ALTER BUFFPOOL on z/OS: z/OS

Use the MQSC command **ALTER BUFFPOOL** to dynamically change the settings of a predefined buffer pool on z/OS.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

Parameters not specified in the **ALTER BUFFPOOL** command result in the existing values for those parameters being left unchanged.

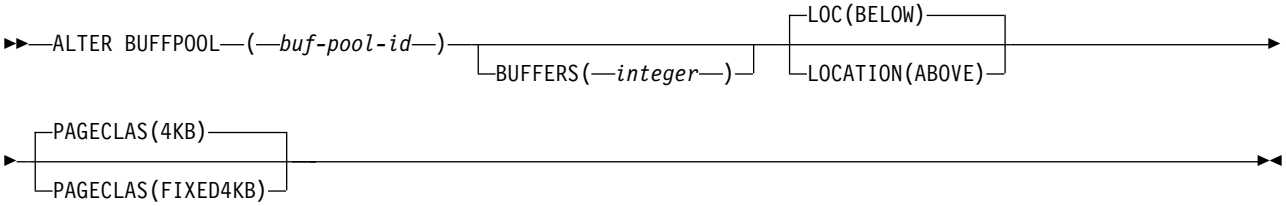
You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for ALTER BUFFPOOL"
- "Parameter descriptions for ALTER BUFFPOOL"

Syntax diagram

Synonym: **ALT BP**

ALTER BUFFPOOL



Usage notes for ALTER BUFFPOOL

1. Buffers are added or removed according to whether the value is more than or less than the current allocation (which can be shown by the DISPLAY USAGE command).
2. If there is insufficient storage, of the type specified by the PAGECLAS attribute to add the requested number, as many as possible are added.
3. The command runs asynchronously. Message CSQP023I is sent to the console when the command is complete.
4. **ALTER BUFFPOOL** cannot be issued from CSQINPT.

Parameter descriptions for ALTER BUFFPOOL

(buf-pool-id)
Buffer pool identifier.

CD If IBM MQ Version 8.0 new functions are enabled with OPMODE, this parameter is an integer in the range zero through 99. Otherwise, this parameter is an integer in the range zero through 15.

BUFFERS(*integer*)
This parameter is optional and is the number of 4096 byte buffers to be used in this buffer pool.

If the value of the **LOCATION** parameter is BELOW, the minimum value of buffers is 100 and the maximum value is 500,000. If the value of the **LOCATION** parameter is ABOVE, then valid values are in the range of 100 to 999999999 (nine nines). The storage used for buffers in a buffer pool with **LOCATION ABOVE** is obtained in multiples of 4MB. Therefore specifying a **BUFFERS** value which is a multiple of 1024 will make the most efficient use of storage.
See Buffers and buffer pools for guidance on the number of buffers you can define in each buffer pool.
When defining a buffer pool care should be taken to ensure that there is sufficient storage available for it either above or below the bar. For more information, see Address space storage.

Note: Creating a large buffer pool can take several minutes depending on size of the buffer pool and machine configuration. In some cases message CSQP061I might be output.

LOCATION(LOC)(*BELOW* or *ABOVE*)

LOCATION and **LOC** are synonyms and either, but not both, can be used.

The **LOCATION** or **LOC** parameter specifies where the memory used by the specified buffer pool is located.

This memory location can be either ABOVE (64 bit) or BELOW (31 bit) the bar. Valid values for this parameter are BELOW or ABOVE, with BELOW being the default.

ABOVE can only be specified if IBM MQ Version 8.0 new functions are enabled with **OPMODE**. BELOW can be specified regardless of the value of **OPMODE** and has the same effect as not specifying the **LOCATION** parameter.

When altering a buffer pool, you should take care to make sure that there is sufficient storage available if increasing the number of buffers, or changing the **LOCATION** value. Switching the location of the buffer pool can be a CPU and I/O intensive task. You should perform this task when the queue manager is not being heavily used.

For more information, see Address space storage.

PAGECLAS(*4KB* or *FIXED4KB*)

Optional parameter that describes the type of virtual storage pages used for backing the buffers in the buffer pool.

This attribute applies to all buffers in the buffer pool, including any that are added later as a result of using the **ALTER BUFFPOOL** command. The default value is 4KB, which means that pageable 4KB pages are used to back the buffers in the pool.

4KB is the only valid value if the buffer pool has its location attribute set to BELOW. If the buffer pool has its LOCATION attribute set to ABOVE, it is also possible to specify FIXED4KB. This means that fixed 4KB pages, which are permanently in real storage and will never be paged out to auxiliary storage, are used to back the buffers in the buffer pool.

FIXED4KB can only be specified if IBM MQ Version 8.0 new functions are enabled with **OPMODE**, whereas 4KB can be specified regardless of the value of **OPMODE**.

The **PAGECLAS** attribute of a buffer pool can be altered at any time. However, the alteration only takes place when the buffer pool switches location from above the bar, to below the bar, or the other way round. Otherwise, the value is stored in the log of the queue manager and is applied when the queue manager next restarts.

The current value of **PAGECLAS** can be checked by issuing the **DISPLAY USAGE PSID(*)** command. Doing this also results in a CSQP062I message being output, if the current value of **PAGECLAS** is different from the value in the log of the queue manager.

For example:

- Buffer pool 7 currently has **LOCATION(ABOVE)** and **PAGECLAS(4KB)** specified. If **ALTER BUFFPOOL(7) PAGECLAS(FIXED4KB)** is specified, the buffer pool continues to be backed by pageable 4KB pages as the **LOCATION** has not been changed.
- Buffer pool 8 currently has **LOCATION(BELOW)** and **PAGECLAS(4KB)** specified. If **ALTER BUFFPOOL(8) LOCATION(ABOVE) PAGECLAS(FIXED4KB)** is specified, the buffer pool is moved above the bar and has its buffers backed by fixed 4KB pages, if any are available.

When you specify **PAGECLAS(FIXED4KB)** the whole buffer pool is backed by page-fixed 4KB pages, so ensure that there is sufficient real storage available on the LPAR. Otherwise, the queue manager might not start, or other address spaces might be impacted; for more information, see Address space storage.

See IBM MQ Support Pac MP16: IBM MQ for z/OS - Capacity planning & tuning for advice on when to use the FIXED4KB value of the **PAGECLAS** attribute.

Attention: The queue manager records the current buffer pool settings in checkpoint log records. These buffer pool settings are automatically restored when a queue manager is later restarted. This restoration occurs after processing of the CSQINP1 data set. Therefore, if you have used **ALTER BUFFPOOL** since the buffer pool was last defined, any **DEFINE BUFFPOOL** command in CSQINP1 has been ignored at restart, unless the **REPLACE** attribute has been specified.

ALTER CFSTRUCT on z/OS: z/OS

On z/OS, use the MQSC command **ALTER CFSTRUCT** to alter the CF application structure backup and recovery parameters, and offload environment parameters for any specified application structure.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

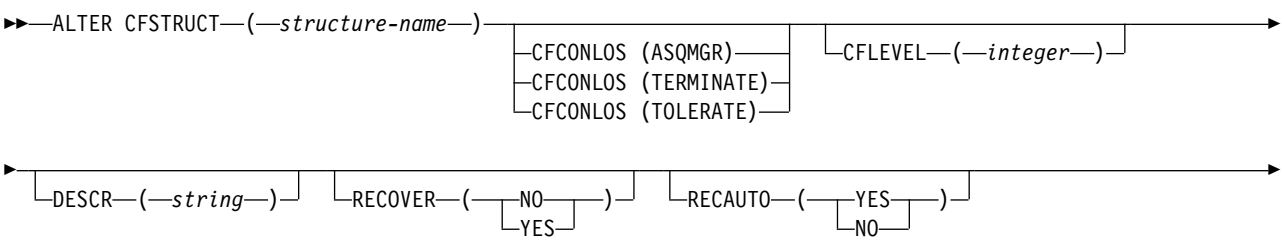
Parameters not specified in the **ALTER CFSTRUCT** command result in the existing values for those parameters being left unchanged.

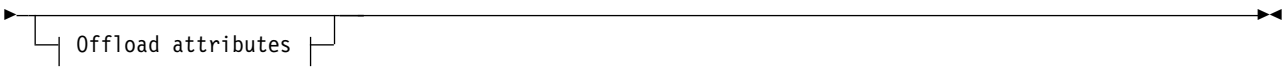
You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

Syntax diagram

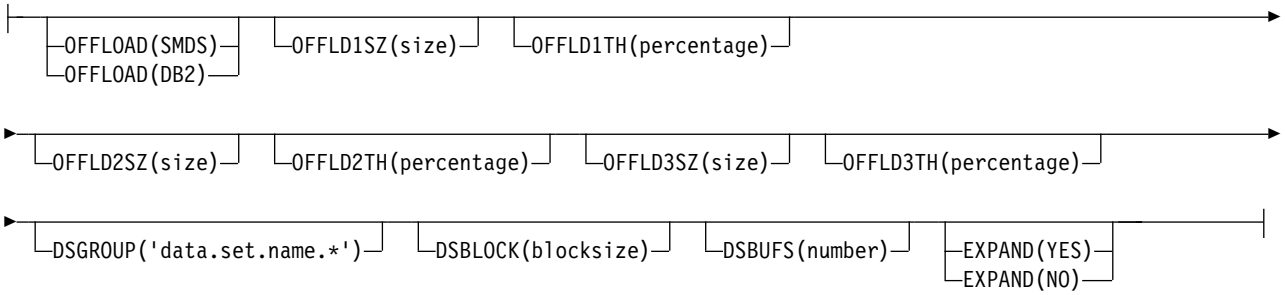
Synonym: **ALT CFSTRUCT**

ALTER CFSTRUCT





Offload attributes:



Usage notes

- This command cannot specify the CF administration structure (CSQ_ADMIN).
- This command is valid only when the queue manager is a member of a queue-sharing group.

Parameter descriptions for ALTER CFSTRUCT

(structure-name)

Name of the coupling facility application structure with queue manager CF level capability and backup and recovery parameters you want to define. This parameter is required.

The name:

- Cannot have more than 12 characters.
- Must start with an uppercase letter (A through Z).
- Can include only the characters A through Z and 0 through 9.

The name of the queue-sharing group to which the queue manager is connected is prefixed to the name you supply. The name of the queue-sharing group is always four characters, padded with @ symbols if necessary. For example, if you use a queue-sharing group named NY03 and you supply the name PRODUCT7, the resultant coupling facility structure name is NY03PRODUCT7. The administrative structure for the queue-sharing group (in this case NY03CSQ_ADMIN) cannot be used for storing messages.

CFCONLOS

This parameter specifies the action to be taken when a queue manager loses connectivity to the CF structure. The value can be:

ASQMGR

The action taken is based on the setting of the **CFCONLOS** queue manager attribute.

TERMINATE

The queue manager terminates when connectivity to the structure is lost. This is the default value when **CFLEVEL** is increased to 5.

TOLERATE

The queue manager tolerates loss of connectivity to the structure without terminating.

The **CFCONLOS** parameter is only valid from **CFLEVEL(5)**.

CFLEVEL(*integer*)

Specifies the functional capability level for this CF application structure. Value can be one of the following:

1 A CF structure that can be "auto-created" by a queue manager at command level 520.

2 A CF structure at command level 520 that can only be created or deleted by a queue manager at command level 530 or greater.

3

A CF structure at command level 530. This **CFLEVEL** is required if you want to use persistent messages for either one or both of the following reasons:

- On shared queues, if **RECOVER(YES)** is set.
- For message grouping when a local queue is defined with **INDXTYPE(GROUPID)**.

You can only increase the value of **CFLEVEL** to 3 if all the queue managers in the queue-sharing group are at command level 530 or greater - this is to ensure that there are no latent command level 520 connections to queues referencing the structure.

You can only decrease the value of **CFLEVEL** from 3 if all the queues that reference the CF structure are both empty (have no messages or uncommitted activity) and closed.

4

This **CFLEVEL** supports all the **CFLEVEL(3)** functions. **CFLEVEL(4)** allows queues defined with CF structures at this level to have messages with a length greater than 63 KB.

Only a queue manager with a command level of 600 or greater can connect to a CF structure at **CFLEVEL(4)**.

You can only increase the value of **CFLEVEL** to 4 if all the queue managers in the queue-sharing group are at command level 600 or greater.

You can only decrease the value of **CFLEVEL** from 4 if all the queues that reference the CF structure are both empty (have no messages or uncommitted activity) and closed.

5

This **CFLEVEL** supports all functions for **CFLEVEL(4)**. In addition, **CFLEVEL(5)** enables the following new functions. If altering an existing **CFSTRUCT** to **CFLEVEL(5)**, you must review other attributes as indicated:

- Queues defined with CF structures at this level can have message data offloaded to either shared message data sets (SMDS), or Db2, under control of the **OFFLOAD** attribute. The offload threshold and size parameters (such as **OFFLD1TH**, and **OFFLD1SZ**) determine whether any particular messages are offloaded given its size and current CF structure utilization. If using SMDS offload, the **DSGROUP**, **DSBUFS**, **DSEXPAND** and **DSBLOCK** attributes are respected.
- Structures at **CFLEVEL(5)** allow the queue manager to tolerate a loss of connectivity to the CF structure. The **CFCONLOS** attribute determines queue manager behavior when a loss of connectivity is detected, and the **RECAUTO** attribute controls subsequent automatic structure recovery behavior.
- Messages containing IBM MQ message properties are stored in a different format on shared queues in a **CFLEVEL(5)** structure. This format leads to internal processing optimizations. Additional application migration capabilities are also available and these are enabled via the queue **PROPCTL** attribute.

Only a queue manager with a command level of 710 or above can connect to a CF structure at **CFLEVEL(5)**.

Note:

You can only increase the value of **CFLEVEL** to 5 if all the queue managers in the queue-sharing group are at command level 710 or greater and have IBM WebSphere MQ Version 7.1.0 new functions enabled with OPMODE.

You can decrease the value of **CFLEVEL** from 5 if all the queues that reference the CF structure are both empty, that is the queues, and CF structure have no messages or uncommitted activity, and are closed.

DESCR(*string*)

Plain-text comment that provides descriptive information about the object when an operator issues the **DISPLAY CFSTRUCT** command.

The string should contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

OFFLOAD

Specify whether offloaded message data is to be stored in a group of shared message data sets or in Db2.

SMDS
Offload messages from coupling facility to shared message data set (SMDS).

Db2 Offload messages from coupling facility to Db2. This value is the default assumption when **CFLEVEL** is increased to 5.

Offloading messages using Db2 has significant performance impact. If you want to use offload rules as a means of increasing capacity, the SMDS option should be specified.

This parameter is only valid from **CFLEVEL(5)**. **At CFLEVEL(4)** any message offloading is always to Db2, and only applies to messages greater than the maximum coupling facility entry size.

Note:

If you change the offload technique (from Db2 to SMDS or the other way) then all new messages will be written using the new method but any existing large messages stored using the previous technique can still be retrieved. The relevant Db2 message table or shared message data sets will continue to be used until the queue managers have detected that there are no further messages stored in the old format.

If SMDS is specified, then the **DSGROUP** parameter is also required. It can be specified either on the same command or on a previous **DEFINE** or **ALTER** command for the same structure.

OFFLD1TH(percentage) OFFLD1SZ(size)
OFFLD2TH(percentage) OFFLD2SZ(size)
OFFLD3TH(percentage) OFFLD3SZ(size)

Specify rules for when messages smaller than the maximum coupling facility entry size are to be offloaded to external storage (shared message data sets or Db2 tables) instead of being stored in the application structure. These rules can be used to increase the effective capacity of the structure. The offloaded message still requires an entry in the coupling facility containing message control information, and a descriptor referring to the offloaded message data, but the amount of structure space required is less than the amount that would be needed to store the whole message.

If the message data is very small (less than approximately 140 bytes) it may fit into the same coupling facility entry as the message control information, without needing additional data elements. In this case, no space can be saved, so any offload rules are ignored and the message data is not offloaded.

Messages exceeding the maximum coupling facility entry size (63.75 KB including control information) are always offloaded as they cannot be stored in a coupling facility entry. Messages where the message body exceeds 63 KB are also offloaded to ensure that enough space is available for the control information. Additional rules to request offloading of smaller messages can be specified using these pairs of keywords. Each rule indicates that when the usage of the structure (in either elements or entries) exceeds the specified threshold percentage value, the message data will be offloaded if the total size of the coupling facility entry required to store the whole message (including message data, headers and descriptors) exceeds the specified size value. Headers and descriptors typically require approximately 400 bytes.

percentage
The usage threshold percentage value is an integer in the range 0 (meaning this rule always applies) to 100 (meaning this rule only applies when the structure is full).

size The message size value should be specified as an integer followed by K, giving the number of kilobytes in the range 0K to 64K. As messages exceeding 63.75 KB are always offloaded, the value 64K is allowed as a simple way to indicate that the rule is not being used.

In general, the smaller the numbers, the more messages are offloaded.

A message is offloaded if any offload rule matches. The normal convention is that a later rule would be for a higher usage level and a smaller message size than an earlier one, but no check is made for consistency or redundancy between the rules.

When structure **ALTER** processing is active, the number of used elements or entries can temporarily exceed the reported total number, giving a percentage exceeding 100, because the new elements or entries are made available during **ALTER** processing but the total is only updated when the **ALTER** completes. At such times, a rule specifying 100 for the threshold may temporarily take effect. If a rule is not intended to be used at all, it should specify 64K for the size.

The default values assumed for the offload rules when defining a new structure at **CFLEVEL(5)** or upgrading an existing structure to **CFLEVEL(5)** depend on the **OFFLOAD** method option. For **OFFLOAD(SMDS)**, the default rules specify increasing amounts of offloading as the structure becomes full. This increases the effective structure capacity with minimal performance impact. For **OFFLOAD(Db2)**, the default rules have the same threshold values as for SMDS but the size values are set to 64K so that the rules never apply and messages are offloaded only if they are too large to be stored in the structure, as for **CFLEVEL(4)**.

For **OFFLOAD(SMDS)** the defaults are:

- **OFFLD1TH(70) OFFLD1SZ(32K)**
- **OFFLD2TH(80) OFFLD2SZ(4K)**
- **OFFLD3TH(90) OFFLD3SZ(0K)**

For **OFFLOAD(Db2)** the defaults are:

- **OFFLD1TH(70) OFFLD1SZ(64K)**
- **OFFLD2TH(80) OFFLD2SZ(64K)**
- **OFFLD3TH(90) OFFLD3SZ(64K)**

If the **OFFLOAD** method option is changed from Db2 to SMDS or back when the current offload rules all match the default values for the old method, the offload rules are switched to the default values for the new method. However, if any of the rules have been changed, the current values are kept when switching method.

These parameters are only valid from **CFLEVEL(5)**. At **CFLEVEL(4)**, any message offloading is always to Db2, and only applies to messages greater than the maximum coupling facility entry size.

DSGROUP

For **OFFLOAD(SMDS)**, specify the generic data set name to be used for the group of shared message data sets associated with this structure (one for each queue manager), with exactly one asterisk indicating where the queue manager name should be inserted to form the specific data set name.

'data.set.name.*'
The value must be a valid data set name when the asterisk is replaced by a queue manager name of up to four characters. The queue manager name can form all or part of any qualifier in the data set name.

The entire parameter value must be enclosed in quotation marks.

This parameter cannot be changed after any data sets have been activated for the structure.

If SMDS is specified, then the **DSGROUP** parameter must also be specified.

The **DSGROUP** parameter is only valid from **CFLEVEL(5)**.

DSBLOCK

For **OFFLOAD(SMDS)**, specify the logical block size, which is the unit in which shared message data set space is allocated to individual queues.

8K
16K
32K
64K
128K
256K
512K
1M Each message is written starting at the next page within the current block and is allocated further blocks as needed. A larger size decreases space management requirements and reduces I/O for large messages, but increases buffer space requirements and disk space requirements for small queues.

This parameter cannot be changed after any data sets have been activated for the structure.

The **DSBLOCK** parameter is only valid from **CFLEVEL(5)**.

DSBUFS

For **OFFLOAD(SMDS)**, specify the number of buffers to be allocated in each queue manager for accessing shared message data sets, as a number in the range 1 - 9999. The size of each buffer is equal to the logical block size. SMDS buffers are allocated in memory objects residing in z/OS 64-bit storage (above the bar).

number
This parameter can be overridden for individual queue managers using the **DSBUFS** parameter on **ALTER SMDS**.

When this parameter is altered, any queue managers which are already connected to the structure (and which do not have an individual DSBUFS override value) dynamically increase or decrease the number of data set buffers being used for this structure to match the new value. If the specified target value cannot be reached, the affected queue manager adjusts the DSBUFS parameter associated with its own individual SMDS definition (as for the **ALTER SMDS** command) to match the actual new number of buffers.

These buffers use virtual storage. You should work with the z/OS systems programmer to ensure there is sufficient auxiliary storage available before increasing the number of buffers.

The **DSBUFS** parameter is only valid from **CFLEVEL(5)**.

DSEXPAND

For **OFFLOAD(SMDS)**, this parameter controls whether the queue manager should expand a shared message data set when it becomes nearly full, and further blocks are required in the data set.

YES Expansion is supported.

Each time expansion is required, the data set is expanded by the secondary allocation specified when the data set was defined. If no secondary allocation was specified, or it was specified as zero, then a secondary allocation amount of approximately 10% of the existing size is used

NO No automatic data set expansion is to take place.

This parameter can be overridden for individual queue managers using the **DSEXPAND** parameter on **ALTER SMDS**.

If an expansion attempt fails, the **DSEXPAND** override for the affected queue manager is automatically changed to NO to prevent further expansion attempts, but it can be changed back to YES using the **ALTER SMDS** command to enable further expansion attempts.

When this parameter is altered, any queue managers which are already connected to the structure (and which do not have an individual **DSEXPAND** override value) immediately start using the new parameter value.

The **DSEXPAND** parameter is only valid from **CFLEVEL(5)**.

RECOVER

Specifies whether CF recovery is supported for the application structure. Values are:

NO CF application structure recovery is not supported. (The synonym is N.)

YES CF application structure recovery is supported. (The synonym is Y.)

You can only set **RECOVER(YES)** if the structure has a **CFLEVEL** of 3 or higher. Set **RECOVER(YES)** if you intend to use persistent messages.

You can only change **RECOVER(NO)** to **RECOVER(YES)** if all the queue managers in the queue-sharing group are at command level 530 or greater ; this is to ensure that there are no latent command level 520 connections to queues referencing the **CFSTRUCT**.

You can only change **RECOVER(YES)** to **RECOVER(NO)** if all the queues that reference the CF structure are both empty (have no messages or uncommitted activity) and closed.

RECAUTO

Specifies the automatic recovery action to be taken when a queue manager detects that the structure is failed or when a queue manager loses connectivity to the structure and no systems in the SysPlex have connectivity to the Coupling Facility that the structure is allocated in. Values can be:

YES The structure and associated shared message data sets which also need recovery are automatically recovered. (The synonym is Y).

NO The structure is not automatically recovered. (The synonym is N). This is the default value when **CFLEVEL** is increased to 5.

This parameter has no effect for structures defined with **RECOVER(NO)**.

The **RECAUTO** parameter is only valid from **CFLEVEL(5)**.

ALTER CHANNEL:

Use the MQSC command **ALTER CHANNEL** to alter the parameters of a channel.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

Parameters not specified in the **ALTER CHANNEL** command result in the existing values for those parameters being left unchanged.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

Synonym: **ALT CHL**

- "Syntax diagrams"
- "Usage notes"
- "Parameter descriptions for ALTER CHANNEL"

Syntax diagrams

The syntax diagrams for **ALTER CHANNEL** are in the subtopics. There is a separate syntax diagram for each channel type.

Usage notes

- Changes take effect after the channel is next started.
- For cluster channels (the CLUSSDR and CLUSRCVR columns in the table), if an attribute can be set on both channels, set it on both and ensure that the settings are identical. If there is any discrepancy between the settings, those that you specify on the CLUSRCVR channel are likely to be used. This is explained in Cluster channels.
- If you change the **XMITQ** name or the **CONNAME**, you must reset the sequence number at both ends of the channel. (See "RESET CHANNEL" on page 982 for information about the **SEQNUM** parameter.)
- Successful completion of the command does not mean that the action completed. To check for true completion, see the ALTER CHANNEL step in Checking that async commands for distributed networks have finished.

Parameter descriptions for ALTER CHANNEL

The following table shows the parameters that are relevant for each type of channel. There is a description of each parameter after the table. Parameters are optional unless the description states that they are required.

Table 85. DEFINE and ALTER CHANNEL parameters

Parameter	SDR	SVR	RCVR	RQSTR	CLNTCONN	SVRCONN	CLUSSDR	CLUSRCVR	AMQP V9.0.0
AFFINITY					✓				
AMQPKA									✓ 9.0.0
BATCHHB	✓	✓					✓	✓	

Table 85. DEFINE and ALTER CHANNEL parameters (continued)

Parameter	SDR	SVR	RCVR	RQSTR	CLNTCONN	SVRCONN	CLUSSDR	CLUSRCVR	AMQP 9.0.0
BATCHINT	✓	✓					✓	✓	
BATCHLIM	✓	✓					✓	✓	
BATCHSZ	✓	✓	✓	✓			✓	✓	
CERTLABL	✓	✓	✓	✓	✓	✓		✓	✓ 9.0.0
channel-name	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
CHLTYPE	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
CLNTWGHT					✓				
CLUSNL							✓	✓	
CLUSTER							✓	✓	
CLWLPRTY							✓	✓	
CLWLRANK							✓	✓	
CLWLWGHT							✓	✓	
z/OS CMDSCOPE	✓	✓	✓	✓	✓	✓	✓	✓	
COMPHDR	✓	✓	✓	✓	✓	✓	✓	✓	
COMPMSG	✓	✓	✓	✓	✓	✓	✓	✓	
CONNAME	✓	✓		✓	✓		✓	✓	
CONVERT	✓	✓					✓	✓	
DEFCDISP	✓	✓	✓	✓		✓			
DEFRECON					✓				
DESCR	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
DISCINT	✓	✓				✓	✓	✓	
HBINT	✓	✓	✓	✓	✓	✓	✓	✓	
KAINT	✓	✓	✓	✓	✓	✓	✓	✓	
LIKE	✓	✓	✓	✓	✓	✓	✓	✓	

Table 85. DEFINE and ALTER CHANNEL parameters (continued)

Parameter	SDR	SVR	RCVR	RQSTR	CLNTCONN	SVRCONN	CLUSSDR	CLUSRCVR	AMQP V 9.0.0
LOCLADDR	✓	✓		✓	✓		✓	✓	✓ 9.0.0
LONGRTY	✓	✓					✓	✓	
LONGTMR	✓	✓					✓	✓	
MAXINST						✓			✓ 9.0.0
MAXINSTC						✓			
MAXMSGL	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
MCANAME	✓	✓		✓			✓	✓	
MCATYPE	✓	✓		✓			✓	✓	
MCAUSER			✓	✓		✓		✓	✓ 9.0.0
MODENAME	✓	✓		✓	✓		✓	✓	
MONCHL	✓	✓	✓	✓		✓	✓	✓	
MRDATA			✓	✓				✓	
MREXIT			✓	✓				✓	
MRRTY			✓	✓				✓	
MRTMR			✓	✓				✓	
MSGDATA	✓	✓	✓	✓			✓	✓	
MSGEXIT	✓	✓	✓	✓			✓	✓	
NETPRTY								✓	
NPMSPEED	✓	✓	✓	✓			✓	✓	
PASSWORD	✓	✓		✓	✓		✓	✓	
PORT									✓ 9.0.0
PROPCTL	✓	✓					✓	✓	
PUTAUT			✓	✓		✓		✓	
QMNAME					✓				

Table 85. DEFINE and ALTER CHANNEL parameters (continued)

Parameter	SDR	SVR	RCVR	RQSTR	CLNTCONN	SVRCONN	CLUSSDR	CLUSRCVR	AMQP V 9.0.0
z/OS QSGDISP	✓	✓	✓	✓	✓	✓	✓	✓	
RCVDATA	✓	✓	✓	✓	✓	✓	✓	✓	
RCVEXIT	✓	✓	✓	✓	✓	✓	✓	✓	
REPLACE	✓	✓	✓	✓	✓	✓	✓	✓	
SCYDATA	✓	✓	✓	✓	✓	✓	✓	✓	
SCYEXIT	✓	✓	✓	✓	✓	✓	✓	✓	
SENDDATA	✓	✓	✓	✓	✓	✓	✓	✓	
SENDEXIT	✓	✓	✓	✓	✓	✓	✓	✓	
SEQWRAP	✓	✓	✓	✓			✓	✓	
SHARECNV					✓	✓			
SHORTRTY	✓	✓					✓	✓	
SHORTTMR	✓	✓					✓	✓	
SSLCAUTH		✓	✓	✓		✓		✓	
SSLCIPH	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
SSLPEER	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
STATCHL	✓	✓	✓	✓			✓	✓	
TPNAME	✓	✓		✓	✓	✓	✓	✓	
TPROOT									✓ 9.0.0
TRPTYPE	✓	✓	✓	✓	✓	✓	✓	✓	
USECLTID									✓ 9.0.0
USEDLQ	✓	✓	✓	✓			✓	✓	
USERID	✓	✓		✓	✓		✓		
XMITQ	✓	✓							

AFFINITY

The channel affinity attribute is used so client applications that connect multiple times using the

same queue manager name can choose whether to use the same client channel definition for each connection. This attribute is intended to be used when multiple applicable channel definitions are available.

PREFERRED
The first connection in a process reading a client channel definition table (CCDT) creates a list of applicable definitions based on the weighting with any applicable **CLNTWGHT(0)** definitions first and in alphabetical order. Each connection in the process attempts to connect using the first definition in the list. If a connection is unsuccessful the next definition is used. Unsuccessful non-**CLNTWGHT(0)** definitions are moved to the end of the list. **CLNTWGHT(0)** definitions remain at the start of the list and are selected first for each connection. For C, C++ and .NET (including fully managed .NET) clients the list is updated if the CCDT has been modified since the list was created. Each client process with the same host name creates the same list.

NONE
The first connection in a process reading a CCDT creates a list of applicable definitions. All connections in a process select an applicable definition based on the weighting with any applicable **CLNTWGHT(0)** definitions selected first in alphabetical order. For C, C++ and .NET (including fully managed .NET) clients the list is updated if the CCDT has been modified since the list was created.

For example, suppose the CCDT includes the following definitions:

```
CHLNAME(A) QMNAME(QM1) CLNTWGHT(3)
CHLNAME(B) QMNAME(QM1) CLNTWGHT(4)
CHLNAME(C) QMNAME(QM1) CLNTWGHT(4)
```

The first connection in a process creates its own ordered list based on the weightings. So it might, for example, create the ordered list `CHLNAME(B), CHLNAME(A), CHLNAME(C)`.
For **AFFINITY(PREFERRED)**, each connection in the process attempts to connect using **CHLNAME(B)**. If a connection is unsuccessful the definition is moved to the end of the list which now becomes `CHLNAME(A), CHLNAME(C), CHLNAME(B)`. Each connection in the process then attempts to connect using **CHLNAME(A)**.
For **AFFINITY(NONE)**, each connection in the process attempts to connect using one of the three definitions selected at random based on the weightings.
When sharing conversations is enabled with a non-zero channel weighting and **AFFINITY(NONE)**, multiple connections in a process using the same queue manager name can connect using different applicable definitions rather than sharing an existing channel instance.

Multi V 9.0.0 **AMQPKA(*integer*)**
The keep alive time for an AMQP channel in seconds. If the AMQP client has not sent any frames within the keep alive interval, then the connection is closed with a `amqp:resource-limit-exceeded` AMQP error condition.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of `AMQP`

BATCHHB(*integer*)
Specifies whether batch heartbeats are to be used. The value is the length of the heartbeat in milliseconds.

Batch heartbeats allow a sending channel to verify that the receiving channel is still active just before committing a batch of messages, so that if the receiving channel is not active, the batch can be backed out rather than becoming in-doubt, as would otherwise be the case. By backing out the batch, the messages remain available for processing so they could, for example, be redirected to another channel.

If the sending channel has had a communication from the receiving channel within the batch heartbeat interval, the receiving channel is assumed to be still active. If not, a 'heartbeat' is sent to the receiving channel to check.

The value must be in the range zero through 999999. A value of zero indicates that batch heartbeating is not used.

The **BATCHHB** parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, CLUSSDR, and CLUSRCVR.

BATCHINT(*integer*)

The minimum amount of time, in milliseconds, that a channel keeps a batch open.

The batch is terminated when one of the following conditions is met:

- **BATCHSZ** messages have been sent.
- **BATCHLIM** bytes have been sent.
- The transmission queue is empty and **BATCHINT** is exceeded.

The value must be in the range 0 - 999999999. Zero means that the batch is terminated as soon as the transmission queue becomes empty, or the **BATCHSZ** or **BATCHLIM** limit is reached.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, CLUSSDR, or CLUSRCVR.

BATCHLIM(*integer*)

The limit, in kilobytes, of the amount of data that can be sent through a channel before taking a sync point. A sync point is taken after the message that caused the limit to be reached has flowed across the channel. A value of zero in this attribute means that no data limit is applied to batches over this channel.

The batch is terminated when one of the following conditions is met:

- **BATCHSZ** messages have been sent.
- **BATCHLIM** bytes have been sent.
- The transmission queue is empty and **BATCHINT** is exceeded.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, CLUSSDR, or CLUSRCVR.

The value must be in the range 0 - 999999. The default value is 5000.

The **BATCHLIM** parameter is supported on all platforms.

BATCHSZ(*integer*)

The maximum number of messages that can be sent through a channel before taking a sync point.

The maximum batch size used is the lowest of the following values:

- The **BATCHSZ** of the sending channel.
- The **BATCHSZ** of the receiving channel.
- z/OS On z/OS, three less than the maximum number of uncommitted messages allowed at the sending queue manager (or one if this value is zero or less).
- Multi On Multiplatforms, the maximum number of uncommitted messages allowed at the sending queue manager (or one if this value is zero or less).
- z/OS On z/OS, three less than the maximum number of uncommitted messages allowed at the receiving queue manager (or one if this value is zero or less).
- Multi On Multiplatforms, the maximum number of uncommitted messages allowed at the receiving queue manager (or one if this value is zero or less).

The maximum number of uncommitted messages is specified by the **MAXUMSGS** parameter of the **ALTER QMGR** command.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, RCVR, RQSTR, CLUSSDR, or CLUSRCVR.

The value must be in the range 1 through 9999.

CERTLABL

Certificate label for this channel to use.

The label identifies which personal certificate in the key repository is sent to the remote peer. If this attribute is blank, the certificate is determined by the queue manager **CERTLABL** parameter.

Note that inbound channels (including receiver, cluster-receiver, unqualified server, and server-connection channels) only send the configured certificate if the IBM MQ version of the remote peer fully supports certificate label configuration, and the channel is using a TLS CipherSpec. See Interoperability of Elliptic Curve and RSA CipherSpecs for further information.

In all other cases, the queue manager **CERTLABL** parameter determines the certificate sent. In particular, the following only ever receive the certificate configured by the **CERTLABL** parameter of the queue manager, regardless of the channel-specific label setting:

- All current Java and JMS clients.
- Versions of IBM MQ prior to Version 8.0.

You do not need to run the **REFRESH SECURITY TYPE(SSL)** command if you make any changes to **CERTLABL** on a channel. However, you must run a **REFRESH SECURITY TYPE(SSL)** command if you make any changes to **CERTLABL** on the queue manager.

Note: It is an error to inquire, or set, this attribute for cluster-sender channels. If you attempt to do so, you receive the error MQRCCF_WRONG_CHANNEL_TYPE. However, the attribute is present in cluster-sender channel objects (including MQCD structures) and a channel auto-definition (CHAD) exit might set it programmatically if required.

channel-name)

The name of the new channel definition.

This parameter is required on all types of channel.

Multi On CLUSSDR channels, it can take a different form from the other channel types. If your convention for naming cluster-sender channels includes the name of the queue manager, you can define a cluster-sender channel using the +QMNAME+ construction. After connection to the matching cluster-receiver channel, IBM MQ substitutes the correct repository queue manager name in place of +QMNAME+ in the cluster-sender channel definition. For more information see Components of a cluster.

The name must not be the same as any existing channel defined on this queue manager (unless **REPLACE** or **ALTER** is specified).

z/OS On z/OS, client-connection channel names can duplicate others.

The maximum length of the string is 20 characters, and the string must contain only valid characters; see Rules for naming IBM MQ objects.

CHLTYPE

Channel type. This parameter is required. It must follow immediately after the *channel-name)* parameter on all platforms except z/OS.

SDR Sender channel

SVR Server channel

RCVR Receiver channel

RQSTR Requester channel

CLNTCONN
Client-connection channel

SVRCONN
Server-connection channel

CLUSSDR
Cluster-sender channel

CLUSRCVR
Cluster-receiver channel

Note: If you are using the **REPLACE** option, you cannot change the channel type.

CLNTWGHT
The client channel weighting attribute is used so client channel definitions can be selected at random based on their weighting when more than one suitable definition is available. Specify a value in the range 0 - 99.

The special value 0 indicates that no random load balancing is performed and applicable definitions are selected in alphabetical order. To enable random load balancing the value can be in the range 1 through 99 where 1 is the lowest weighting and 99 is the highest.

When a client issues an MQCONN with queue manager name "**name*" and more than one suitable definition is available in the CCDT the choice of definition to use is randomly selected based on the weighting with any applicable **CLNTWGHT(0)** definitions selected first in alphabetical order. The distribution is not guaranteed.

For example, suppose the CCDT includes the following two definitions:

```
CHLNAME(TO.QM1) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address1) CLNTWGHT(2)
CHLNAME(TO.QM2) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address2) CLNTWGHT(4)
```

A client MQCONN with queue manager name "*GRP1" would choose one of the two definitions based on the weighting of the channel definition. (A random integer 1 - 6 would be generated. If the integer was in the range 1 through 2 address1 would be used otherwise address2 would be used). If this connection was unsuccessful the client would then use the other definition.

The CCDT might contain applicable definitions with both zero and non-zero weighting. In this situation, the definitions with zero weightings are chosen first and in alphabetical order. If these connections are unsuccessful the definitions with non-zero weighting are chosen based on their weighting.

For example, suppose the CCDT includes the following four definitions:

```
CHLNAME(TO.QM1) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address1) CLNTWGHT(1)
CHLNAME(TO.QM2) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address2) CLNTWGHT(2)
CHLNAME(TO.QM3) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address3) CLNTWGHT(0)
CHLNAME(TO.QM4) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address4) CLNTWGHT(0)
```

A client MQCONN with queue manager name "*GRP1" would first choose definition "TO.QM3". If this connection was unsuccessful the client would then choose definition "TO.QM4". If this connection was also unsuccessful the client would then randomly choose one of the remaining two definitions based on their weighting.

CLNTWGHT support is added for all supported transport protocols.

CLUSNL(*nlname*)
The name of the namelist that specifies a list of clusters to which the channel belongs.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of CLUSSDR and CLUSRCVR channels. Only one of the resultant values of CLUSTER or CLUSNL can be nonblank, the other must be blank.

CLUSTER(*clustername*)
The name of the cluster to which the channel belongs. The maximum length is 48 characters conforming to the rules for naming IBM MQ objects.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of CLUSSDR or CLUSRCVR. Only one of the resultant values of CLUSTER or CLUSNL can be nonblank, the other must be blank.

CLWLPRTY(*integer*)
Specifies the priority of the channel for the purposes of cluster workload distribution. The value must be in the range zero through 9 where zero is the lowest priority and 9 is the highest.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of CLUSSDR or CLUSRCVR.

For more information about this attribute, see CLWLPRTY queue attribute.

CLWLRANK(*integer*)
Specifies the rank of the channel for the purposes of cluster workload distribution. The value must be in the range zero through 9 where zero is the lowest rank and 9 is the highest.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of CLUSSDR or CLUSRCVR.

For more information about this attribute, see CLWLRANK channel attribute.

CLWLWGHT(*integer*)
Specifies the weighting to be applied to the channel for the purposes of cluster workload distribution so that the proportion of messages sent down the channel can be controlled. The value must be in the range 1 through 99 where 1 is the lowest rank and 99 is the highest.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of CLUSSDR or CLUSRCVR.

For more information about this attribute, see CLWLWGHT channel attribute.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if **QSGDISP** is set to GROUP.

' ' The command is executed on the queue manager on which it was entered.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group. You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

COMPHDR
The list of header data compression techniques supported by the channel. For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference with the first compression technique supported by the remote end of the channel being used.

The mutually supported compression techniques of the channel are passed to the message exit of the sending channel where the compression technique used can be altered on a per message basis. Compression alters the data passed to send and receive exits.

NONE No header data compression is performed.

SYSTEM Header data compression is performed.

COMPMSG
The list of message data compression techniques supported by the channel. For sender, server,

cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference with the first compression technique supported by the remote end of the channel being used.

The mutually supported compression techniques of the channel are passed to the message exit of the sending channel where the compression technique used can be altered on a per message basis. Compression alters the data passed to send and receive exits.

NONE No message data compression is performed.

RLE Message data compression is performed using run-length encoding.

ZLIBFAST
Message data compression is performed using ZLIB encoding with speed prioritized.

ZLIBHIGH
Message data compression is performed using ZLIB encoding with compression prioritized.

ANY Any compression technique supported by the queue manager can be used. This value is only valid for receiver, requester, and server-connection channels.

CONNAME(*string*)
Connection name.

For cluster-receiver channels (when specified) **CONNAME** relates to the local queue manager, and for other channels it relates to the target queue manager.

z/OS On z/OS, the maximum length of the string is 48 characters.

Multi On Multiplatforms, the maximum length of the string is 264 characters

A workaround to the 48 character limit might be one of the following suggestions:

- Set up your DNS servers so that you use, for example, host name of "myserver" instead of "myserver.location.company.com", ensuring you can use the short host name.
- Use IP addresses.

Specify **CONNAME** as a comma-separated list of names of machines for the stated **TRPTYPE**. Typically only one machine name is required. You can provide multiple machine names to configure multiple connections with the same properties. The connections are usually tried in the order they are specified in the connection list until a connection is successfully established. The order is modified for clients if the **CLNTWGHT** attribute is provided. If no connection is successful, the channel attempts the connection again, as determined by the attributes of the channel. With client channels, a connection-list provides an alternative to using queue manager groups to configure multiple connections. With message channels, a connection list is used to configure connections to the alternative addresses of a multi-instance queue manager.

This parameter is required for channels with a channel type (**CHLTYPE**) of SDR, RQSTR, CLNTCONN, and CLUSSDR. It is optional for SVR channels, and for CLUSRCVR channels of **TRPTYPE(TCP)**, and is not valid for RCVR or SVRCONN channels.

Providing multiple connection names in a list was first supported in IBM WebSphere MQ Version 7.0.1. It changes the syntax of the **CONNAME** parameter. Earlier clients and queue managers connect using the first connection name in the list, and do not read the rest of the connection names in the list. In order for the earlier clients and queue managers to parse the new syntax, you must specify a port number on the first connection name in the list. Specifying a port number avoids problems when connecting to the channel from a client or queue manager that is running at a level earlier than IBM WebSphere MQ Version 7.0.1.

Multi On Multiplatforms, the TCP/IP connection name parameter of a cluster-receiver channel is optional. If you leave the connection name blank, IBM MQ generates a connection name for you, assuming the default port and using the current IP address of the system. You can

override the default port number, but still use the current IP address of the system. For each connection name leave the IP name blank, and provide the port number in parentheses; for example:

```
(1415)
```

The generated **CONNAME** is always in the dotted decimal (IPv4) or hexadecimal (IPv6) form, rather than in the form of an alphanumeric DNS host name.

Note: If you are using any of the special characters in your connection name (for example, parentheses) you must enclose the string in single quotation marks.

The value you specify depends on the transport type (**TRPTYPE**) to be used:

LU 6.2

- Multi On Multiplatforms, **CONNAME** is the name of the CPI-C communications side object. Or, if the **TPNAME** is not blank, **CONNAME** is the fully qualified name of the partner logical unit.
- z/OS On z/OS, there are two forms in which to specify the value:

  Logical unit name
  The logical unit information for the queue manager, comprising the logical unit name, TP name, and optional mode name. Logical unit name can be specified in one of three forms:

Form	Example
luname	IGY12355
luname/TPname	IGY12345/APING
luname/TPname/modename	IGY12345/APINGD/#INTER

  For the first form, the TP name and mode name must be specified for the **TPNAME** and **MODENAME** parameters; otherwise these parameters must be blank.

  Note: For client-connection channels, only the first form is allowed.

  Symbolic name
  The symbolic destination name for the logical unit information for the queue manager, as defined in the side information data set. The **TPNAME** and **MODENAME** parameters must be blank.

  Note: For cluster-receiver channels, the side information is on the other queue managers in the cluster. Alternatively, in this case it can be a name that a channel auto-definition exit can resolve into the appropriate logical unit information for the local queue manager.

  The specified or implied LU name can be that of a VTAM generic resources group.

  For more information, see Configuration parameters for an LU 6.2 connection.

NetBIOS
A unique NetBIOS name (limited to 16 characters).

SPX The 4 byte network address, the 6 byte node address, and the 2 byte socket number. These values must be entered in hexadecimal, with a period separating the network and node addresses. The socket number must be enclosed in brackets, for example:

```
CONNAME('0a0b0c0d.804abcde23a1(5e86)')
```

TCP Either the host name, or the network address of the remote machine (or the local machine for cluster-receiver channels). This address can be followed by an optional port number, enclosed in parentheses.

If the **CONNAME** is a host name, the host name is resolved to an IP address.

The IP stack used for communication depends on both the value specified for **CONNAME** and the value specified for **LOCLADDR**. See LOCLADDR for information about how this value is resolved.

z/OS On z/OS, the connection name can include the IP_name of an z/OS dynamic DNS group or a Network Dispatcher input port.

Important: Do not include the IP_name or input port for channels with a channel type (**CHLTYPE**) of CLUSSDR.

On all platforms, when you define a channel with a channel type (**CHLTYPE**) of CLUSRCVR that is using TCP/IP, you do not need to specify the network address of your queue manager. IBM MQ generates a **CONNAME** for you, assuming the default port and using the current IPv4 address of the system. If the system does not have an IPv4 address, the current IPv6 address of the system is used.

Note: If you are using clustering between IPv6-only and IPv4-only queue managers, do not specify an IPv6 network address as the **CONNAME** for CLUSRCVR channels. A queue manager that is capable only of IPv4 communication is unable to start a cluster sender channel definition that specifies the CONNAME in IPv6 hexadecimal form. Consider, instead, using host names in a heterogeneous IP environment.

CONVERT

Specifies whether the sending message channel agent attempts conversion of the application message data, if the receiving message channel agent cannot perform this conversion.

NO No conversion by sender

YES Conversion by sender

z/OS On z/OS, N and Y are accepted as synonyms of NO and YES.

The **CONVERT** parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, CLUSSDR, or CLUSRCVR.

DEFCDISP

Specifies the default channel disposition of the channel.

PRIVATE
The intended disposition of the channel is as a PRIVATE channel.

FIXSHARED
The intended disposition of the channel is as a FIXSHARED channel.

SHARED The intended disposition of the channel is as a SHARED channel.

This parameter does not apply to channels with a channel type (**CHLTYPE**) of CLNTCONN, CLUSSDR, or CLUSRCVR.

DEFRECON

Specifies whether a client connection automatically reconnects a client application if its connection breaks.

NO Unless overridden by **MQCONNX**, the client is not reconnected automatically.

YES Unless overridden by **MQCONNX**, the client reconnects automatically.

QMGR Unless overridden by **MQCONNX**, the client reconnects automatically, but only to the same queue manager. The QMGR option has the same effect as MQCNO_RECONNECT_Q_MGR.

DISABLED
Reconnection is disabled, even if requested by the client program using the **MQCONNX** MQI call.

Table 86. Automatic reconnection depends on the values set in the application and in the channel definition

DEFRECON	**Reconnection options set in the application**			
	MQCNO_RECONNECT	MQCNO_RECONNECT_Q_MGR	MQCNO_RECONNECT_AS_DEF	MQCNO_RECONNECT_DISABLED
NO	YES	QMGR	NO	NO
YES	YES	QMGR	YES	NO
QMGR	YES	QMGR	QMGR	NO
DISABLED	NO	NO	NO	NO

DESCR(*string*)
Plain-text comment. It provides descriptive information about the channel when an operator issues the **DISPLAY CHANNEL** command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

DISCINT(*integer*)
The minimum time in seconds for which the channel waits for a message to arrive on the transmission queue, after a batch ends, before terminating the channel. A value of zero causes the message channel agent to wait indefinitely.

The value must be in the range zero through 999 999.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SVRCONN , SDR, SVR, CLUSSDR, CLUSRCVR.

For SVRCONN channels using the TCP protocol, this parameter is the minimum time in seconds for which the SVRCONN instance remains active without any communication from its partner client. A value of zero disables this disconnect processing. The SVRCONN inactivity interval only applies between IBM MQ API calls from a client, so no client is disconnected during an extended MQGET with wait call. This attribute is ignored for SVRCONN channels using protocols other than TCP.

HBINT(*integer*)

This attribute specifies the approximate time between heartbeat flows that are to be passed from a sending MCA when there are no messages on the transmission queue.

Heartbeat flows unblock the receiving MCA, which is waiting for messages to arrive or for the disconnect interval to expire. When the receiving MCA is unblocked it can disconnect the channel without waiting for the disconnect interval to expire. Heartbeat flows also free any storage buffers that have been allocated for large messages and close any queues that have been left open at the receiving end of the channel.

The value is in seconds and must be in the range 0 through 999999. A value of zero means that no heartbeat flows are to be sent. The default value is 300. To be most useful, the value needs to be less than the disconnect interval value.

For server-connection and client-connection channels, heartbeats can flow from both the server side as well as the client side independently. If no data has been transferred across the channel for the heartbeat interval, the client-connection MQI agent sends a heartbeat flow and the server-connection MQI agent responds to it with another heartbeat flow. This happens irrespective of the state of the channel, for example, irrespective of whether it is inactive while making an API call, or is inactive waiting for client user input. The server-connection MQI agent is also capable of initiating a heartbeat to the client, again irrespective of the state of the channel. To prevent both server-connection and client-connection MQI agents heart beating to each other at the same time, the server heartbeat is flowed after no data has been transferred across the channel for the heartbeat interval plus 5 seconds.

For server-connection and client-connection channels working in the channel mode before IBM WebSphere MQ Version 7.0, heartbeats flow only when a server MCA is waiting for an MQGET command with the `WAIT` option specified, which it has issued on behalf of a client application.

For more information, see Heartbeat interval (HBINT).

KAINT(*integer*)

The value passed to the communications stack for KeepAlive timing for this channel.

For this attribute to be effective, TCP/IP keepalive must be enabled both in the queue manager and in TCP/IP.

z/OS On z/OS, you enable TCP/IP keepalive in the queue manager by issuing the **ALTER QMGR TCPKEEP(YES)** command; if the**TCPKEEP** queue manager parameter is `NO`, the value is ignored, and the KeepAlive facility is not used.

Multi On Multiplatforms, TCP/IP keepalive is enabled when the **KEEPALIVE=YES** parameter is specified in the TCP stanza in the distributed queuing configuration file, `qm.ini`, or through the IBM MQ Explorer.

Keepalive must also be enabled within TCP/IP itself. Refer to your TCP/IP documentation for information about configuring keepalive:

- AIX On AIX, use the **no** command.
- Windows On Windows, edit the registry.
- z/OS On z/OS, update your TCP/IP PROFILE data set and add or change the **INTERVAL** parameter in the TCPCONFIG section.

z/OS Although this parameter is available on all platforms, its setting is implemented only on z/OS.

Multi On Multiplatforms, you can access and modify the parameter, but it is only stored and forwarded; there is no functional implementation of the parameter. This functionality is useful in a clustered environment where a value set in a cluster-receiver channel definition on AIX, for example, flows to (and is implemented by) z/OS queue managers that are in, or join, the cluster.

Multi On Multiplatforms, if you need the functionality provided by the **KAINT** parameter, use the Heartbeat Interval (**HBINT**) parameter, as described in HBINT.

(*integer*)

The KeepAlive interval to be used, in seconds, in the range 1 through 99 999.

0

The value used is that specified by the INTERVAL statement in the TCP profile configuration data set.

AUTO

The KeepAlive interval is calculated based upon the negotiated heartbeat value as follows:

- If the negotiated **HBINT** is greater than zero, KeepAlive interval is set to that value plus 60 seconds.
- If the negotiated **HBINT** is zero, the value used is that specified by the INTERVAL statement in the TCP profile configuration data set.

This parameter is valid for all channel types. It is ignored for channels with a **TRPTYPE** other than TCP or SPX.

LIKE(*channel-name*)
The name of a channel. The parameters of this channel are used to model this definition.

If this field is not completed, and you do not complete the parameter fields related to the command, the values are taken from one of the following default channels, depending upon the channel type:

SYSTEM.DEF.SENDER
Sender channel

SYSTEM.DEF.SERVER
Server channel

SYSTEM.DEF.RECEIVER
Receiver channel

SYSTEM.DEF.REQUESTER
Requester channel

SYSTEM.DEF.SVRCONN
Server-connection channel

SYSTEM.DEF.CLNTCONN
Client-connection channel

SYSTEM.DEF.CLUSSDR
Cluster-sender channel

SYSTEM.DEF.CLUSRCVR
Cluster-receiver channel

This parameter is equivalent to defining the following object for a sender channel, and similarly for other channel types:

```
LIKE(SYSTEM.DEF.SENDER)
```

These default channel definitions can be altered by the installation to the default values required.

z/OS On z/OS, the queue manager searches page set zero for an object with the name you specify and a disposition of QMGR or COPY. The disposition of the **LIKE** object is not copied to the object and channel type you are defining.

Note:

1. **QSGDISP(GROUP)** objects are not searched.
2. # **LIKE** is ignored if **QSGDISP(COPY)** is specified. However, the group object defined is used as a **LIKE** object.

LOCLADDR(*string*)
LOCLADDR is the local communications address for the channel. For channels other than AMQP channels, use this parameter if you want a channel to use a particular IP address, port, or port range for outbound communications. **LOCLADDR** might be useful in recovery scenarios where a channel is restarted on a different TCP/IP stack. **LOCLADDR** is also useful to force a channel to use an IPv4 or IPv6 stack on a dual-stack system. You can also use **LOCLADDR** to force a channel to use a dual-mode stack on a single-stack system.

Note: AMQP channels do not support the same format of **LOCLADDR** as other IBM MQ channels. For the format supported by AMQ, see the next parameter **AMQP: LOCLADDR**.

For channels other than AMQP channels, the **LOCLADDR** parameter is valid only for channels with a transport type (**TRPTYPE**) of TCP. If **TRPTYPE** is not TCP, the data is ignored and no error message is issued.

The value is the optional IP address, and optional port or port range used for outbound TCP/IP communications. The format for this information is as follows:

```
LOCLADDR([ip-addr][(low-port[,high-port])][,[ip-addr][(low-port[,high-port])]])
```

The maximum length of **LOCLADDR**, including multiple addresses, is MQ_LOCAL_ADDRESS_LENGTH.

If you omit **LOCLADDR**, a local address is automatically allocated.

Note, that you can set **LOCLADDR** for a C client using the Client Channel Definition Table (CCDT).

All the parameters are optional. Omitting the ip-addr part of the address is useful to enable the configuration of a fixed port number for an IP firewall. Omitting the port number is useful to select a particular network adapter without having the identify a unique local port number. The TCP/IP stack generates a unique port number.

Specify *[,[ip-addr][(low-port[,high-port])]]* multiple times for each additional local address. Use multiple local addresses if you want to specify a specific subset of local network adapters. You can also use *[,[ip-addr][(low-port[,high-port])]]* to represent a particular local network address on different servers that are part of a multi-instance queue manager configuration.

ip-addr
ip-addr is specified in one of three forms:

IPv4 dotted decimal
For example, 192.0.2.1

IPv6 hexadecimal notation
For example, 2001:DB8:0:0:0:0:0:0

Alphanumeric host name form
For example WWW.EXAMPLE.COM

low-port and high-port
low-port and high-port are port numbers enclosed in parentheses.

The following table shows how the **LOCLADDR** parameter can be used:

Table 87. Examples of how the **LOCLADDR** *parameter can be used*

LOCLADDR	**Meaning**
9.20.4.98	Channel binds to this address locally
9.20.4.98, 9.20.4.99	Channel binds to either IP address. The address might be two network adapters on one server, or a different network adapter on two different servers in a multi-instance configuration.
9.20.4.98(1000)	Channel binds to this address and port 1000 locally
9.20.4.98(1000,2000)	Channel binds to this address and uses a port in the range 1000 - 2000 locally
(1000)	Channel binds to port 1000 locally
(1000,2000)	Channel binds to port in range 1000 - 2000 locally

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, RQSTR, CLNTCONN, CLUSSDR, OR CLUSRCVR.

On CLUSSDR channels, the IP address and port to which the outbound channel binds, is a combination of fields. It is a concatenation of the IP address, as defined in the **LOCLADDR**

parameter, and the port range from the cluster cache. If there is no port range in the cache, the port range defined in the **LOCLADDR** parameter is used.

z/OS This port range does not apply to z/OS systems.

Even though this parameter is similar in form to **CONNAME**, it must not be confused with it. The **LOCLADDR** parameter specifies the characteristics of the local communications, whereas the **CONNAME** parameter specifies how to reach a remote queue manager.

When a channel is started, the values specified for **CONNAME** and **LOCLADDR** determine the IP stack to be used for communication; see Table 3 and Local Address (**LOCLADDR**).

If the TCP/IP stack for the local address is not installed or configured, the channel does not start and an exception message is generated.

z/OS For example, on z/OS systems, the message is "CSQO015E: Command issued but no reply received." The message indicates that the connect() request specifies an interface address that is not known on the default IP stack. To direct the connect() request to the alternative stack, specify the **LOCLADDR** parameter in the channel definition as either an interface on the alternative stack, or a DNS host name. The same specification also works for listeners that might not use the default stack. To find the value to code for **LOCLADDR**, run the **NETSTAT HOME** command on the IP stacks that you want to use as alternatives.

Table 88. How the IP stack to be used for communication is determined

Protocols supported	CONNAME	Action of channel	
IPv4 only	IPv4 address [1]		Channel binds to IPv4 stack
	IPv6 address [2]		Channel fails to resolve **CONNAME**
	IPv4 and 6 host name [3]		Channel binds to IPv4 stack
	IPv4 address	IPv4 address	Channel binds to IPv4 stack
	IPv6 address	IPv4 address	Channel fails to resolve **CONNAME**
	IPv4 and 6 host name	IPv4 address	Channel binds to IPv4 stack
	Any address [4]	IPv6 address	Channel fails to resolve **LOCLADDR**
	IPv4 address	IPv4 and 6 host name	Channel binds to IPv4 stack
	IPv6 address	IPv4 and 6 host name	Channel fails to resolve **CONNAME**
	IPv4 and 6 host name	IPv4 and 6 host name	Channel binds to IPv4 stack
IPv4 and IPv6	IPv4 address		Channel binds to IPv4 stack
	IPv6 address		Channel binds to IPv6 stack
	IPv4 and 6 host name		Channel binds to stack determined by **IPADDRV**
	IPv4 address	IPv4 address	Channel binds to IPv4 stack
	IPv6 address	IPv4 address	Channel fails to resolve **CONNAME**
	IPv4 and 6 host name	IPv4 address	Channel binds to IPv4 stack
	IPv4 address	IPv6 address	Channel maps **CONNAME** to IPv6 [5]
	IPv6 address	IPv6 address	Channel binds IPv6 stack
	IPv4 and 6 host name	IPv6 address	Channel binds IPv6 stack
	IPv4 address	IPv4 and 6 host name	Channel binds to IPv4 stack
	IPv6 address	IPv4 and 6 host name	Channel binds to IPv6 stack
	IPv4 and 6 host name	IPv4 and 6 host name	Channel binds to stack determined by **IPADDRV**

Table 88. How the IP stack to be used for communication is determined (continued)

Protocols supported	CONNAME	Action of channel	
IPv6 only	IPv4 address		Channel maps **CONNAME** to IPv6 [5]
	IPv6 address		Channel binds to IPv6 stack
	IPv4 and 6 host name		Channel binds to IPv6 stack
	Any address	IPv4 address	Channel fails to resolve **LOCLADDR**
	IPv4 address	IPv6 address	Channel maps **CONNAME** to IPv6 [5]
	IPv6 address	IPv6 address	Channel binds to IPv6 stack
	IPv4 and 6 host name	IPv6 address	Channel binds to IPv6 stack
	IPv4 address	IPv4 and 6 host name	Channel maps **CONNAME** to IPv6 [5]
	IPv6 address	IPv4 and 6 host name	Channel binds to IPv6 stack
	IPv4 and 6 host name	IPv4 and 6 host name	Channel binds to IPv6 stack

Notes:

1. IPv4 address. An IPv4 host name that resolves only to an IPv4 network address or a specific dotted notation IPv4 address, for example `1.2.3.4`. This note applies to all occurrences of ' IPv4 address' in this table.
2. IPv6 address. An IPv6 host name that resolves only to an IPv6 network address or a specific hexadecimal notation IPv6 address, for example `4321:54bc`. This note applies to all occurrences of ' IPv6 address' in this table.
3. IPv4 and 6 host name. A host name that resolves to both IPv4 and IPv6 network addresses. This note applies to all occurrences of ' IPv4 and 6 host name' in this table.
4. Any address. IPv4 address, IPv6 address, or IPv4 and 6 host name. This note applies to all occurrences of 'Any address' in this table.
5. Maps IPv4 **CONNAME** to IPv4 mapped IPv6 address. IPv6 stack implementations that do not support IPv4 mapped IPv6 addressing fail to resolve the **CONNAME**. Mapped addresses might require protocol translators in order to be used. The use of mapped addresses is not recommended.

AMQP: LOCLADDR(*ip-addr*)

Note: For the format of **LOCLADDR** that other IBM MQ channels use, see the previous parameter **LOCLADDR**.

For AMQP channels, **LOCLADDR** is the local communications address for the channel. Use this parameter if you want to force the client to use a particular IP address. **LOCLADDR** is also useful to force a channel to use an IPv4 or IPv6 address if a choice is available, or to use a particular network adapter on a system with multiple network adapters.

The maximum length of **LOCLADDR** is `MQ_LOCAL_ADDRESS_LENGTH`.

If you omit **LOCLADDR**, a local address is automatically allocated.

ip-addr
`ip-addr` is a single network address, specified in one of three forms:

IPv4 dotted decimal
For example, `192.0.2.1`

IPv6 hexadecimal notation
For example, `2001:DB8:0:0:0:0:0:0`

Alphanumeric host name form
For example, `WWW.EXAMPLE.COM`

If an IP address is entered, only the address format is validated. The IP address itself is not validated.

LONGRTY(*integer*)
When a sender, server, or cluster-sender channel is attempting to connect to the remote queue

manager, and the count specified by **SHORTRTY** has been exhausted, this parameter specifies the maximum number of further attempts that are made to connect to the remote queue manager, at intervals specified by **LONGTMR**.

If this count is also exhausted without success, an error is logged to the operator, and the channel is stopped. The channel must then be restarted with a command (it is not started automatically by the channel initiator).

The value must be in the range zero through 999999999.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, CLUSSDR, or CLUSRCVR.

LONGTMR(*integer*)
For long retry attempts, this parameter is the maximum number of seconds to wait before reattempting connection to the remote queue manager.

The time is approximate; zero means that another connection attempt is made as soon as possible.

The interval between retries might be extended if the channel has to wait to become active.

The value must be in the range zero through 999999999.

Note: For implementation reasons, the maximum retry interval that can be used is 999,999; values exceeding this maximum are treated as 999,999. Similarly, the minimum retry interval that can be used is 2; values less than this minimum are treated as 2.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, CLUSSDR, or CLUSRCVR.

MAXINST(*integer*)

The maximum number of simultaneous instances of an individual server-connection channel or AMQP channel that can be started.

The value must be in the range zero through 999999999.

A value of zero prevents all client access on this channel.

If the value of this parameter is reduced to a number that is less than the number of instances of the server-connection channel that are currently running, then those running instances are not affected. However, new instances cannot start until sufficient existing instances have ceased to run so that the number of currently running instances is less than the value of this parameter.

V 9.0.0 If an AMQP client attempts to connect to an AMQP channel, and the number of connected clients has reached **MAXINST**, the channel closes the connection with a close frame. The close frame contains the following message: amqp:resource-limit-exceeded. If a client connects with an ID that is already connected (that is, it performs a client-takeover), and the client is permitted to take over the connection, the takeover will succeed regardless of whether the number of connected clients has reached **MAXINST**.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SVRCONN or AMQP.

MAXINSTC(*integer*)
The maximum number of simultaneous individual server-connection channels that can be started from a single client. In this context, connections that originate from the same remote network address are regarded as coming from the same client.

The value must be in the range zero through 999999999.

A value of zero prevents all client access on this channel.

If the value of this parameter is reduced to a number that is less than the number of instances of the server-connection channel that is currently running from individual clients, then those

running instances are not affected. However, new instances from those clients cannot start until sufficient instances have ceased to run that the number of running instances is less than the value of this parameter.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SVRCONN.

MAXMSGL(*integer*)
Specifies the maximum message length that can be transmitted on the channel. This parameter is compared with the value for the partner and the actual maximum used is the lower of the two values. The value is ineffective if the MQCB function is being executed and the channel type (**CHLTYPE**) is SVRCONN.

The value zero means the maximum message length for the queue manager.

Multi On Multiplatforms, specify a value in the range zero through to the maximum message length for the queue manager.

z/OS On z/OS, specify a value in the range zero through 104857600 bytes (100 MB).

See the **MAXMSGL** parameter of the **ALTER QMGR** command for more information.

MCANAME(*string*)
Message channel agent name.

This parameter is reserved, and if specified must only be set to blanks (maximum length 20 characters).

MCATYPE
Specifies whether the message-channel-agent program on an outbound message channel runs as a thread or a process.

PROCESS
The message channel agent runs as a separate process.

THREAD The message channel agent runs as a separate thread

In situations where a threaded listener is required to service many incoming requests, resources can become strained. In this case, use multiple listener processes and target incoming requests at specific listeners through the port number specified on the listener.

Multi On Multiplatforms, this parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, RQSTR, CLUSSDR, or CLUSRCVR.

z/OS On z/OS, this parameter is supported only for channels with a channel type of CLUSRCVR. When specified in a CLUSRCVR definition, **MCATYPE** is used by a remote machine to determine the corresponding CLUSSDR definition.

MCAUSER(*string*)
Message channel agent user identifier.

Note: An alternative way of providing a user ID for a channel to run under is to use channel authentication records. With channel authentication records, different connections can use the same channel while using different credentials. If both **MCAUSER** on the channel is set and channel authentication records are used to apply to the same channel, the channel authentication records take precedence. The **MCAUSER** on the channel definition is only used if the channel authentication record uses **USERSRC(CHANNEL)**. For more details, see Channel authentication records.

This parameter interacts with **PUTAUT**, see the definition of that parameter for more information.

If it is nonblank, it is the user identifier that is to be used by the message channel agent for authorization to access IBM MQ resources, including (if **PUTAUT** is DEF) authorization to put the message to the destination queue for receiver or requester channels.

If it is blank, the message channel agent uses its default user identifier.

The default user identifier is derived from the user ID that started the receiving channel. The possible values are:

- z/OS On z/OS, the user ID assigned to the channel-initiator started task by the z/OS started-procedures table.
- Multi For TCP/IP, on Multiplatforms, the user ID from the `inetd.conf` entry, or the user that started the listener.
- Multi For SNA, on Multiplatforms, the user ID from the SNA server entry or, in the absence of this user ID the incoming attach request, or the user that started the listener.
- For NetBIOS or SPX, the user ID that started the listener.

The maximum length of the string is:

- Windows 64 characters on Windows. On Windows, you can optionally qualify a user identifier with the domain name in the format `user@domain`.
- 12 characters on other platforms.

This parameter is not valid for channels with a channel type (**CHLTYPE**) of `SDR`, `SVR`, `CLNTCONN`, `CLUSSDR`.

MODENAME(*string*)
LU 6.2 mode name (maximum length 8 characters).

This parameter is valid only for channels with a transport type (**TRPTYPE**) of LU 6.2. If **TRPTYPE** is not LU 6.2, the data is ignored and no error message is issued.

If specified, this parameter must be set to the SNA mode name unless the **CONNAME** contains a side-object name, in which case it must be set to blanks. The actual name is then taken from the CPI-C Communications Side Object, or APPC side information data set.

z/OS See Configuration parameters for an LU 6.2 connection for more information about configuration parameters for an LU 6.2 connection for your platform.

This parameter is not valid for channels with a channel type (**CHLTYPE**) of `RCVR` or `SVRCONN`.

MONCHL
Controls the collection of online monitoring data for channels:

QMGR Collect monitoring data according to the setting of the queue manager parameter MONCHL.

OFF Monitoring data collection is turned off for this channel.

LOW If the value of the queue manager **MONCHL** parameter is not `NONE`, online monitoring data collection is turned on, with a low rate of data collection, for this channel.

MEDIUM If the value of the queue manager **MONCHL** parameter is not `NONE`, online monitoring data collection is turned on, with a moderate rate of data collection, for this channel.

HIGH If the value of the queue manager **MONCHL** parameter is not `NONE`, online monitoring data collection is turned on, with a high rate of data collection, for this channel.

For cluster channels, the value of this parameter is not replicated in the repository and, therefore, not used in the auto-definition of cluster-sender channels.

For auto-defined cluster-sender channels, the value of this parameter is taken from the queue manager attribute **MONACLS**. If you want to modify the value, use the command `ALTER QMGR MONACLS(HIGH)`, then restart the auto-defined sender channel.

Changes to this parameter take effect only on channels started after the change occurs.

MRDATA(*string*)
Channel message-retry exit user data. The maximum length is 32 characters.

This parameter is passed to the channel message-retry exit when it is called.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of RCVR, RQSTR, or CLUSRCVR.

MREXIT(*string*)
Channel message-retry exit name.

The format and maximum length of the name is the same as for MSGEXIT, however you can only specify one message-retry exit.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of RCVR, RQSTR, or CLUSRCVR.

MRRTY(*integer*)
The number of times the channel tries again before it decides it cannot deliver the message.

This parameter controls the action of the MCA only if the message-retry exit name is blank. If the exit name is not blank, the value of **MRRTY** is passed to the exit to use, but the number of retries performed (if any) is controlled by the exit, and not by this parameter.

The value must be in the range zero through 999999999. A value of zero means that no retries are performed.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of RCVR, RQSTR, or CLUSRCVR.

MRTMR(*integer*)
The minimum interval of time that must pass before the channel can try the MQPUT operation again. This time interval is in milliseconds.

This parameter controls the action of the MCA only if the message-retry exit name is blank. If the exit name is not blank, the value of **MRTMR** is passed to the exit to use, but the retry interval is controlled by the exit, and not by this parameter.

The value must be in the range zero through 999 999 999. A value of zero means that the retry is performed as soon as possible (if the value of **MRRTY** is greater than zero).

This parameter is valid only for channels with a channel type (**CHLTYPE**) of RCVR, RQSTR, or CLUSRCVR.

MSGDATA(*string*)
User data for the channel message exit. The maximum length is 32 characters.

This data is passed to the channel message exit when it is called.

ULW On UNIX, Linux, and Windows, you can specify data for more than one exit program by specifying multiple strings separated by commas. The total length of the field must not exceed 999 characters.

IBM i On IBM i, you can specify up to 10 strings, each of length 32 characters. The first string of data is passed to the first message exit specified, the second string to the second exit, and so on.

z/OS On z/OS, you can specify up to eight strings, each of length 32 characters. The first string of data is passed to the first message exit specified, the second string to the second exit, and so on.

On other platforms, you can specify only one string of message exit data for each channel.

Note: This parameter is accepted but ignored for server-connection and client-connection channels.

MSGEXIT(*string*)

Channel message exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately after a message has been retrieved from the transmission queue (sender or server), or immediately before a message is put to a destination queue (receiver or requester).

  The exit is given the entire application message and transmission queue header for modification.
- At initialization and termination of the channel.

ULW On UNIX, Linux, and Windows, you can specify the name of more than one exit program by specifying multiple strings separated by commas. However, the total number of characters specified must not exceed 999.

IBM i On IBM i, you can specify the names of up to 10 exit programs by specifying multiple strings separated by commas.

z/OS On z/OS, you can specify the names of up to eight exit programs by specifying multiple strings separated by commas.

On other platforms, you can specify only one message exit name for each channel.

For channels with a channel type (**CHLTYPE**) of CLNTCONN or SVRCONN, this parameter is accepted but ignored, because message exits are not invoked for such channels.

The format and maximum length of the name depends on the environment:

- UNIX Linux On UNIX, and Linux, it is of the form:

  ```
  libraryname(functionname)
  ```

  The maximum length of the string is 128 characters.
- Windows On Windows, it is of the form:

  ```
  dllname(functionname)
  ```

  where *dllname* is specified without the suffix .DLL. The maximum length of the string is 128 characters.
- IBM i On IBM i, it is of the form:

  ```
  progname libname
  ```

  where *progname* occupies the first 10 characters and *libname* the second 10 characters (both padded to the right with blanks if necessary). The maximum length of the string is 20 characters.
- z/OS On z/OS, it is a load module name, maximum length 8 characters (128 characters are allowed for exit names for client-connection channels, subject to a maximum total length including commas of 999).

NETPRTY(*integer*)

The priority for the network connection. Distributed queuing chooses the path with the highest priority if there are multiple paths available. The value must be in the range zero through 9; zero is the lowest priority.

This parameter is valid only for CLUSRCVR channels.

NPMSPEED

The class of service for nonpersistent messages on this channel:

FAST Fast delivery for nonpersistent messages; messages might be lost if the channel is lost. Messages are retrieved using MQGMO_SYNCPOINT_IF_PERSISTENT and so are not included in the batch unit of work.

NORMAL Normal delivery for nonpersistent messages.

If the sending side and the receiving side do not agree about this parameter, or one does not support it, NORMAL is used.

Notes:

1. If the active recovery logs for IBM MQ for z/OS are switching and archiving more frequently than expected, given that the messages being sent across a channel are non-persistent, setting NPMSPEED(FAST) on both the sending and receiving ends of the channel can minimize the SYSTEM.CHANNEL.SYNCQ updates.
2. If you are seeing high CPU usage relating to updates to the SYSTEM.CHANNEL.SYNCQ, setting NPMSPEED(FAST) can significantly reduce the CPU usage.

This parameter is valid only for channels with a **CHLTYPE** of SDR, SVR, RCVR, RQSTR, CLUSSDR, or CLUSRCVR.

PASSWORD(*string*)

Password used by the message channel agent when attempting to initiate a secure LU 6.2 session with a remote message channel agent. The maximum length is 12 characters.

Multi On Multiplatforms, this parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR,RQSTR, CLNTCONN, or CLUSSDR.

z/OS On z/OS, it is supported only for channels with a channel type (**CHLTYPE**) of CLNTCONN.

Although the maximum length of the parameter is 12 characters, only the first 10 characters are used.

V 9.0.0 **PORT(*integer*)**

The port number used to connect an AMQP channel. The default port for AMQP 1.0 connections is 5672. If you are already using port 5672, you can specify a different port.

PROPCTL

Property control attribute.

Specifies what happens to properties of messages when the message is about to be sent to a V6 or prior queue manager (a queue manager that does not understand the concept of a property descriptor).

This parameter is applicable to Sender, Server, Cluster Sender, and Cluster Receiver channels.

This parameter is optional.

Permitted values are:

COMPAT COMPAT allows applications which expect JMS-related properties to be in an MQRFH2 header in the message data to continue to work unmodified.

Message properties	Result
The message contains a property with a prefix of `mcd.`, `jms.`, `usr.` or `mqext.`	All optional message properties (where the **Support** value is MQPD_SUPPORT_OPTIONAL), except properties in the message descriptor or extension, are placed in one or more MQRFH2 headers in the message data before the message it sent to the remote queue manager.
The message does not contain a property with a prefix of `mcd.`, `jms.`, `usr.` or `mqext.`	All message properties, except properties in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.
The message contains a property where the **Support** field of the property descriptor is not set to MQPD_SUPPORT_OPTIONAL	The message is rejected with reason MQRC_UNSUPPORTED_PROPERTY and treated in accordance with its report options.
The message contains one or more properties where the **Support** field of the property descriptor is set to MQPD_SUPPORT_OPTIONAL but other fields of the property descriptor are set to non-default values.	The properties with non-default values are removed from the message before the message is sent to the remote queue manager.
The MQRFH2 folder that would contain the message property needs to be assigned with the *content='properties'* attribute	The properties are removed to prevent MQRFH2 headers with unsupported syntax flowing to a V6 or prior queue manager.

NONE All properties of the message, except properties in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.

If the message contains a property where the **Support** field of the property descriptor is not set to MQPD_SUPPORT_OPTIONAL then the message is rejected with reason MQRC_UNSUPPORTED_PROPERTY and treated in accordance with its report options.

ALL All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except properties in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

PUTAUT

Specifies which user identifiers are used to establish authority to put messages to the destination queue (for messages channels) or to execute an MQI call (for MQI channels).

DEF The default user ID is used.

z/OS On z/OS, `DEF` might involve using both the user ID received from the network and that derived from **MCAUSER**.

CTX The user ID from the *UserIdentifier* field of the message descriptor is used.

z/OS On z/OS, `CTX` might involve also using the user ID received from the network or that derived from **MCAUSER**, or both.

z/OS **ONLYMCA**

The user ID derived from **MCAUSER** is used. Any user ID received from the network is not used. This value is supported only on z/OS.

z/OS **ALTMCA**

The user ID from the *UserIdentifier* field of the message descriptor is used. Any user ID received from the network is not used. This value is supported only on z/OS.

z/OS On z/OS, the user IDs that are checked, and how many user IDs are checked, depends on the setting of the MQADMIN RACF class hlq.RESLEVEL profile. Depending on the level of access the user ID of the channel initiator has to hlq.RESLEVEL, zero, one or two user IDs are

checked. To see how many user IDs are checked, see RESLEVEL and channel initiator connections. For more information about which user IDs are checked, see User IDs used by the channel initiator.

z/OS On z/OS, this parameter is valid only for channels with a channel type (**CHLTYPE**) of RCVR, RQSTR, CLUSRCVR, or SVRCONN. CTX and ALTMCA are not valid for SVRCONN channels.

Multi On Multiplatforms, this parameter is valid only for channels with a channel type (**CHLTYPE**) of RCVR, RQSTR, or CLUSRCVR.

QMNAME(*string*)
Queue manager name.

For channels with a channel type (**CHLTYPE**) of CLNTCONN, this parameter is the name of a queue manager to which an application that is running in a client environment and using the client channel definition table can request connection. This parameter need not be the name of the queue manager on which the channel is defined, to allow a client to connect to different queue managers.

For channels of other types, this parameter is not valid.

z/OS **QSGDISP**
This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

QSGDISP	ALTER
COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters **QSGDISP(COPY)**. Any object residing in the shared repository, or any object defined using a command that had the parameters **QSGDISP(QMGR)**, is not affected by this command.
GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameters **QSGDISP(GROUP)**. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command. If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero: `DEFINE CHANNEL(channel-name) CHLTYPE(type)` `REPLACE QSGDISP(COPY)` The **ALTER** for the group object takes effect regardless of whether the generated command with **QSGDISP(COPY)** fails.
PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with **QSGDISP(QMGR)** or **QSGDISP(COPY)**. Any object residing in the shared repository is unaffected.
QMGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters **QSGDISP(QMGR)**. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

RCVDATA(*string*)
Channel receive exit user data (maximum length 32 characters).

This parameter is passed to the channel receive exit when it is called.

ULW On UNIX, Linux, and Windows, you can specify data for more than one exit program by specifying multiple strings separated by commas. The total length of the field must not exceed 999 characters.

IBM i On IBM i, you can specify up to 10 strings, each of length 32 characters. The first string of data is passed to the first receive exit specified, the second string to the second exit, and so on.

z/OS On z/OS, you can specify up to eight strings, each of length 32 characters. The first string of data is passed to the first receive exit specified, the second string to the second exit, and so on.

On other platforms, you can specify only one string of receive exit data for each channel.

RCVEXIT(*string*)
Channel receive exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately before the received network data is processed.

  The exit is given the complete transmission buffer as received. The contents of the buffer can be modified as required.
- At initialization and termination of the channel.

ULW On UNIX, Linux, and Windows, you can specify the name of more than one exit program by specifying multiple strings separated by commas. However, the total number of characters specified must not exceed 999.

IBM i On IBM i, you can specify the names of up to 10 exit programs by specifying multiple strings separated by commas.

z/OS On z/OS, you can specify the names of up to eight exit programs by specifying multiple strings separated by commas.

On other platforms, you can specify only one receive exit name for each channel.

The format and maximum length of the name is the same as for **MSGEXIT**.

REPLACE and NOREPLACE

Whether the existing definition z/OS (and on z/OS, with the same disposition) is to be replaced with this one. This parameter is optional. Any object with a different disposition is not changed.

REPLACE
The definition replaces any existing definition of the same name. If a definition does not exist, one is created. REPLACE does not alter the channel status.

NOREPLACE
The definition does not replace any existing definition of the same name.

SCYDATA(*string*)
Channel security exit user data (maximum length 32 characters).

This parameter is passed to the channel security exit when it is called.

SCYEXIT(*string*)
Channel security exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately after establishing a channel.

  Before any messages are transferred, the exit is able to instigate security flows to validate connection authorization.

- Upon receipt of a response to a security message flow.

  Any security message flows received from the remote processor on the remote queue manager are given to the exit.
- At initialization and termination of the channel.

The format and maximum length of the name is the same as for **MSGEXIT** but only one name is allowed.

SENDDATA(*string*)

Channel send exit user data. The maximum length is 32 characters.

This parameter is passed to the channel send exit when it is called.

ULW On UNIX, Linux, and Windows, you can specify data for more than one exit program by specifying multiple strings separated by commas. The total length of the field must not exceed 999 characters.

IBM i On IBM i, you can specify up to 10 strings, each of length 32 characters. The first string of data is passed to the first send exit specified, the second string to the second exit, and so on.

z/OS On z/OS, you can specify up to eight strings, each of length 32 characters. The first string of data is passed to the first send exit specified, the second string to the second exit, and so on.

On other platforms, you can specify only one string of send exit data for each channel.

SENDEXIT(*string*)

Channel send exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately before data is sent out on the network.

  The exit is given the complete transmission buffer before it is transmitted. The contents of the buffer can be modified as required.
- At initialization and termination of the channel.

ULW On UNIX, Linux, and Windows, you can specify the name of more than one exit program by specifying multiple strings separated by commas. However, the total number of characters specified must not exceed 999.

IBM i On IBM i, you can specify the names of up to 10 exit programs by specifying multiple strings separated by commas.

z/OS On z/OS, you can specify the names of up to eight exit programs by specifying multiple strings separated by commas.

On other platforms, you can specify only one send exit name for each channel.

The format and maximum length of the name is the same as for **MSGEXIT**.

SEQWRAP(*integer*)

When this value is reached, sequence numbers wrap to start again at 1.

This value is nonnegotiable and must match in both the local and remote channel definitions.

The value must be in the range 100 through 999999999.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, RCVR, RQSTR, CLUSSDR, or CLUSRCVR.

SHARECNV(*integer*)
Specifies the maximum number of conversations that can be sharing each TCP/IP channel instance. A **SHARECNV** value of:

1 Specifies no sharing of conversations over a TCP/IP channel instance. Client heartbeating is available whether in an MQGET call or not. Read ahead and client asynchronous consumption are also available, and channel quiescing is more controllable.

0 Specifies no sharing of conversations over a TCP/IP channel instance.

The value must be in the range zero through 999999999.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of CLNTCONN or SVRCONN. If the client-connection **SHARECNV** value does not match the server-connection **SHARECNV** value, the lower of the two values is used. This parameter is ignored for channels with a transport type (**TRPTYPE**) other than TCP.

All the conversations on a socket are received by the same thread.

High **SHARECNV** limits have the advantage of reducing queue manager thread usage. However, if many conversations sharing a socket are all busy, there is a possibility of delays as the conversations contend with one another to use the receiving thread. In this situation, a lower **SHARECNV** value is better.

The number of shared conversations does not contribute to the **MAXINST** or **MAXINSTC** totals.

Note: You should restart the client for this change to take effect.

SHORTRTY(*integer*)
The maximum number of attempts that are made by a sender, server, or cluster-sender channel to connect to the remote queue manager, at intervals specified by **SHORTTMR**, before the (normally longer) **LONGRTY** and **LONGTMR** are used.

Retry attempts are made if the channel fails to connect initially (whether it is started automatically by the channel initiator or by an explicit command), and also if the connection fails after the channel has successfully connected. However, if the cause of the failure is such that more attempts are unlikely to be successful, they are not attempted.

The value must be in the range zero through 999999999.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, CLUSSDR, or CLUSRCVR.

SHORTTMR(*integer*)
For short retry attempts, this parameter is the maximum number of seconds to wait before reattempting connection to the remote queue manager.

The time is approximate; zero means that another connection attempt is made as soon as possible.

The interval between retries might be extended if the channel has to wait to become active.

The value must be in the range zero through 999999999.

Note: For implementation reasons, the maximum retry interval that can be used is 999999; values exceeding this maximum are treated as 999999. Similarly, the minimum retry interval that can be used is 2; values less than this minimum are treated as 2.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, CLUSSDR, or CLUSRCVR.

SSLCAUTH
Defines whether IBM MQ requires a certificate from the TLS client. The initiating end of the channel acts as the TLS client, so this parameter applies to the end of the channel that receives the initiation flow, which acts as the TLS server.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of RCVR, SVRCONN, CLUSRCVR, SVR, or RQSTR.

The parameter is used only for channels with **SSLCIPH** specified. If **SSLCIPH** is blank, the data is ignored and no error message is issued.

REQUIRED
IBM MQ requires and validates a certificate from the TLS client.

OPTIONAL
The peer TLS client system might still send a certificate. If it does, the contents of this certificate are validated as normal.

SSLCIPH*(string)*

SSLCIPH specifies the CipherSpec that is used on the channel. The maximum length is 32 characters. This parameter is valid on all channel types which use transport type **TRPTYPE(TCP)**. If the **SSLCIPH** parameter is blank, no attempt is made to use TLS on the channel.

Note: When **SSLCIPH** is used with a telemetry channel, it means TLS Cipher Suite. See the **SSLCIPH** description for **ALTER CHANNEL** (MQTT).

Specify the name of the CipherSpec you are using. The CipherSpecs that can be used with IBM MQ SSL support are shown in the following table. The **SSLCIPH** values must specify the same CipherSpec on both ends of the channel.

Note: IBM i z/OS On IBM MQ for z/OS, you can also specify the two digit hexadecimal code of a CipherSpec, whether or not it appears in the table. On IBM i, you can also specify the two digit hexadecimal code of a CipherSpec, whether or not it appears in the table. Also, on IBM i, installation of AC3 is a prerequisite for the use of SSL.

Platform support[1]	CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[2]	Suite B
ULW z/OS	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	SHA-1	AES	128	Yes	No
ULW z/OS	TLS_RSA_WITH_AES_256_CBC_SHA [3]	TLS 1.0	SHA-1	AES	256	Yes	No
All	ECDHE_ECDSA_AES_128_CBC_SHA256	TLS 1.2	SHA-256	AES	128	Yes	No
All	ECDHE_ECDSA_AES_256_CBC_SHA384 [3]	TLS 1.2	SHA-384	AES	256	Yes	No
Multi	ECDHE_ECDSA_AES_128_GCM_SHA256 [4]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	128 bit
Multi	ECDHE_ECDSA_AES_256_GCM_SHA384 [3] [4]	TLS 1.2	AEAD AES-128 GCM	AES	256	Yes	192 bit
All	ECDHE_RSA_AES_128_CBC_SHA256	TLS 1.2	SHA-256	AES	128	Yes	No
All	ECDHE_RSA_AES_256_CBC_SHA384 [3]	TLS 1.2	SHA-384	AES	256	Yes	No
Multi	ECDHE_RSA_AES_128_GCM_SHA256 [4]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	No
Multi	ECDHE_RSA_AES_256_GCM_SHA384 [3] [4]	TLS 1.2	AEAD AES-128 GCM	AES	SHA384	Yes	No

Platform support[1]	CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[2]	Suite B
IBM i	ECDHE_ECDSA_RC4_128_SHA256	TLS 1.2	AEAD AES-128 GCM	AES	SHA256	Yes	No
IBM i	ECDHE_ECDSA_3DES_EDE_CBC_SHA256	TLS 1.2	AEAD AES-128 GCM	3DES	SHA256	Yes	No
IBM i	ECDHE_ECDSA_NULL_SHA256	TLS 1.2	AEAD AES-128 GCM	ECDSA	SHA256	Yes	No
IBM i	ECDHE_ECDSA_AES_256_GCM_SHA384 [3] [4]	TLS 1.2	AEAD AES-128 GCM	AES	SHA384	Yes	No
ULW z/OS	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	SHA-256	AES	128	Yes	No
ULW z/OS	TLS_RSA_WITH_AES_256_CBC_SHA256 [3]	TLS 1.2	SHA-256	AES	256	Yes	No
Multi	TLS_RSA_WITH_AES_128_GCM_SHA256 [4]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	No
Multi	TLS_RSA_WITH_AES_256_GCM_SHA384 [3] [4]	TLS 1.2	AEAD AES-128 GCM	AES	256	Yes	No

Notes:

1. If no specific platform is noted, the CipherSpec is available on all platforms. For a list of platforms covered by each platform icon, see Release and platform icons in the product documentation.
2. Specifies whether the CipherSpec is FIPS-certified on a FIPS-certified platform. See Federal Information Processing Standards (FIPS) for an explanation of FIPS.
3. This CipherSpec cannot be used to secure a connection from the IBM MQ Explorer to a queue manager unless the appropriate unrestricted policy files are applied to the JRE used by the Explorer.
4. Following a recommendation by NIST, GCM CipherSpecs now have a restriction which means that after 2^22 TLS records are sent, using the same session key, the connection will be terminated with message AMQ9288.

   To prevent this error from happening: avoid using GCM Ciphers, enable secret key reset, or start your IBM MQ queue manager with the environment variable GSK_ENFORCE_GCM_RESTRICTION=GSK_FALSE set.
   Important: The GCM restriction is active, regardless of the FIPS mode being used.
5. IBM i The CipherSpecs listed as supported on IBM i, apply to Versions 7.2 and 7.3 of IBM i.

When you request a personal certificate, you specify a key size for the public and private key pair. The key size that is used during the SSL handshake can depend on the size stored in the certificate and on the CipherSpec:

- ULW z/OS On z/OS, UNIX, Linux, and Windows, when a CipherSpec name includes `_EXPORT`, the maximum handshake key size is 512 bits. If either of the certificates exchanged during the SSL handshake has a key size greater than 512 bits, a temporary 512-bit key is generated for use during the handshake.
- ULW On UNIX, Linux, and Windows, when a CipherSpec name includes `_EXPORT1024`, the handshake key size is 1024 bits.
- Otherwise the handshake key size is the size stored in the certificate.

SSLPEER(*string*)

Specifies the filter to use to compare with the Distinguished Name of the certificate from the peer queue manager or client at the other end of the channel. (A Distinguished Name is the identifier of the TLS certificate.) If the Distinguished Name in the certificate received from the peer does not match the **SSLPEER** filter, the channel does not start.

Note: An alternative way of restricting connections into channels by matching against the TLS Subject Distinguished Name, is to use channel authentication records. With channel authentication records, different TLS Subject Distinguished Name patterns can be applied to the same channel. If both **SSLPEER** on the channel and a channel authentication record are used to apply to the same channel, the inbound certificate must match both patterns in order to connect. For more information, see Channel authentication records.

This parameter is optional; if it is not specified, the Distinguished Name of the peer is not checked at channel startup. (The Distinguished Name from the certificate is still written into the **SSLPEER** definition held in memory, and passed to the security exit). If **SSLCIPH** is blank, the data is ignored and no error message is issued.

This parameter is valid for all channel types.

The **SSLPEER** value is specified in the standard form used to specify a Distinguished Name. For example:

```
SSLPEER('SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN="H1_C_FR1",O=IBM,C=GB')
```

You can use a semi-colon as a separator instead of a comma.

The possible attribute types supported are:

Table 89. Attribute types supported by SSLPEER

Summary attribute	Description
SERIALNUMBER	Certificate serial number
MAIL	Email address
E	Email address (Deprecated in preference to MAIL)
UID or USERID	User identifier
CN	Common Name
T	Title
OU	Organizational Unit name
DC	Domain component
O	Organization name
STREET	Street / First line of address
L	Locality name
ST (or SP or S)	State or Province name
PC	Postal code / zip code
C	Country
UNSTRUCTUREDNAME	Host name
UNSTRUCTUREDADDRESS	IP address
DNQ	Distinguished name qualifier

IBM MQ accepts only uppercase letters for the attribute types.

If any of the unsupported attribute types are specified in the **SSLPEER** string, an error is output either when the attribute is defined or at run time (depending on which platform you are running on), and the string is deemed not to have matched the Distinguished Name of the flowed certificate.

If the Distinguished Name of the flowed certificate contains multiple OU (organizational unit) attributes, and **SSLPEER** specifies these attributes to be compared, they must be defined in descending hierarchical order. For example, if the Distinguished Name of the flowed certificate contains the OUs `OU=Large Unit, OU=Medium Unit, OU=Small Unit`, specifying the following **SSLPEER** values works:

```
('OU=Large Unit,OU=Medium Unit')
('OU=*,OU=Medium Unit,OU=Small Unit')
('OU=*,OU=Medium Unit')
```

but specifying the following **SSLPEER** values fails:

```
('OU=Medium Unit,OU=Small Unit')
('OU=Large Unit,OU=Small Unit')
('OU=Medium Unit')
('OU=Small Unit, Medium Unit, Large Unit')
```

As indicated in these examples, attributes at the low end of the hierarchy can be omitted. For example, `('OU=Large Unit,OU=Medium Unit')` is equivalent to `('OU=Large Unit,OU=Medium Unit,OU=*')`

If two DNs are equal in all respects except for their DC values, the same matching rules apply as for OUs except that in DC values the left-most DC is the lowest-level (most specific) and the comparison ordering differs accordingly.

Any or all the attribute values can be generic, either an asterisk (*) on its own, or a stem with initiating or trailing asterisks. Asterisks allow the **SSLPEER** to match any Distinguished Name value, or any value starting with the stem for that attribute.

If an asterisk is specified at the beginning or end of any attribute value in the Distinguished Name on the certificate, you can specify '*' to check for an exact match in **SSLPEER**. For example, if you have an attribute of `CN='Test*'` in the Distinguished Name of the certificate, you can use the following command:

```
SSLPEER('CN=Test\*')
```

The maximum length of the parameter is 1024 bytes on UNIX, Linux, and Windows.

The maximum length of the parameter is 1024 bytes on IBM i.

The maximum length of the parameter is 256 bytes on z/OS.

Channel authentication records provide greater flexibility when using **SSLPEER** and support 1024 bytes on all platforms.

STATCHL

Controls the collection of statistics data for channels:

QMGR The value of the **STATCHL** parameter of the queue manager is inherited by the channel.

OFF Statistics data collection is turned off for this channel.

LOW If the value of the **STATCHL** parameter of the queue manager is not NONE, statistics data collection is turned on, with a low rate of data collection, for this channel.

MEDIUM If the value of the **STATCHL** parameter of the queue manager is not NONE, statistics data collection is turned on, with a moderate rate of data collection, for this channel.

HIGH If the value of the **STATCHL** parameter of the queue manager is not NONE, statistics data collection is turned on, with a high rate of data collection, for this channel.

Changes to this parameter take effect only on channels started after the change occurs.

z/OS On z/OS systems, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying LOW, MEDIUM, or HIGH makes no difference to your results.

For cluster channels, the value of this parameter is not replicated in the repository and used in the auto-definition of cluster-sender channels. For auto-defined cluster-sender channels, the value of this parameter is taken from the attribute **STATACLS** of the queue manager. This value might then be overridden in the channel auto-definition exit.

TPNAME(*string*)
LU 6.2 transaction program name (maximum length 64 characters).

This parameter is valid only for channels with a transport type (**TRPTYPE**) of LU 6.2.

Set this parameter to the SNA transaction program name, unless the **CONNAME** contains a side-object name in which case set it to blanks. The actual name is taken instead from the CPI-C Communications Side Object, or the APPC side information data set.

z/OS See Configuration parameters for an LU 6.2 connection for more information about configuration parameters for an LU 6.2 connection for your platform.

Windows z/OS On Windows SNA Server, and in the side object on z/OS, the **TPNAME** is wrapped to uppercase.

This parameter is not valid for channels with a channel type (**CHLTYPE**) of RCVR.

V 9.0.0 **TPROOT**
The topic root for an AMQP channel. The default value for **TPROOT** is SYSTEM.BASE.TOPIC. With this value, the topic string an AMQP client uses to publish or subscribe has no prefix, and the client can exchange messages with other IBM MQ publish/subscribe applications. To have AMQP clients publish and subscribe under a topic prefix, first create an IBM MQ topic object with a topic string set to the prefix you want, then set **TPROOT** to the name of the IBM MQ topic object you created.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of AMQP

TRPTYPE
Transport type to be used.

On UNIX, IBM i, Linux, Windows, and z/OS, this parameter is optional because, if you do not enter a value, the value specified in the SYSTEM.DEF.*channel-type* definition is used. However, no check is made that the correct transport type has been specified if the channel is initiated from the other end.

z/OS On z/OS, if the SYSTEM.DEF.*channel-type* definition does not exist, the default is LU62.

This parameter is required on all other platforms.

LU62 SNA LU 6.2

NETBIOS
Windows NetBIOS (supported only on Windows, and DOS).

z/OS This attribute also applies to z/OS for defining client-connection channels that connect to servers on the platforms supporting NetBIOS.

SPX Windows Sequenced packet exchange (supported only on Windows, and DOS).

z/OS This attribute also applies to z/OS for defining client-connection channels that connect to servers on the platforms supporting SPX.

TCP Transmission Control Protocol - part of the TCP/IP protocol suite

Multi V 9.0.0 **USECLTID**

Specifies that the client ID should be used for authorization checks for an AMQP channel, instead of the **MCAUSER** attribute value.

NO The MCA user ID should be used for authorization checks.

YES The client ID should be used for authorization checks.

USEDLQ

Determines whether the dead-letter queue is used when messages cannot be delivered by channels.

NO Messages that cannot be delivered by a channel are treated as a failure. The channel either discards the message, or the channel ends, in accordance with the **NPMSPEED** setting.

YES When the **DEADQ** queue manager attribute provides the name of a dead-letter queue, then it is used, else the behavior is as for NO. YES is the default value.

USERID(*string*)

Task user identifier. The maximum length is 12 characters.

This parameter is used by the message channel agent when attempting to initiate a secure LU 6.2 session with a remote message channel agent.

Multi On Multiplatforms, this parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, RQSTR, CLNTCONN, or CLUSSDR.

z/OS On z/OS, it is supported only for CLNTCONN channels.

Although the maximum length of the parameter is 12 characters, only the first 10 characters are used.

On the receiving end, if passwords are kept in encrypted format and the LU 6.2 software is using a different encryption method, an attempt to start the channel fails with invalid security details. You can avoid invalid security details by modifying the receiving SNA configuration to either:

- Turn off password substitution, or
- Define a security user ID and password.

XMITQ(*string*)

Transmission queue name.

The name of the queue from which messages are retrieved. See Rules for naming IBM MQ objects.

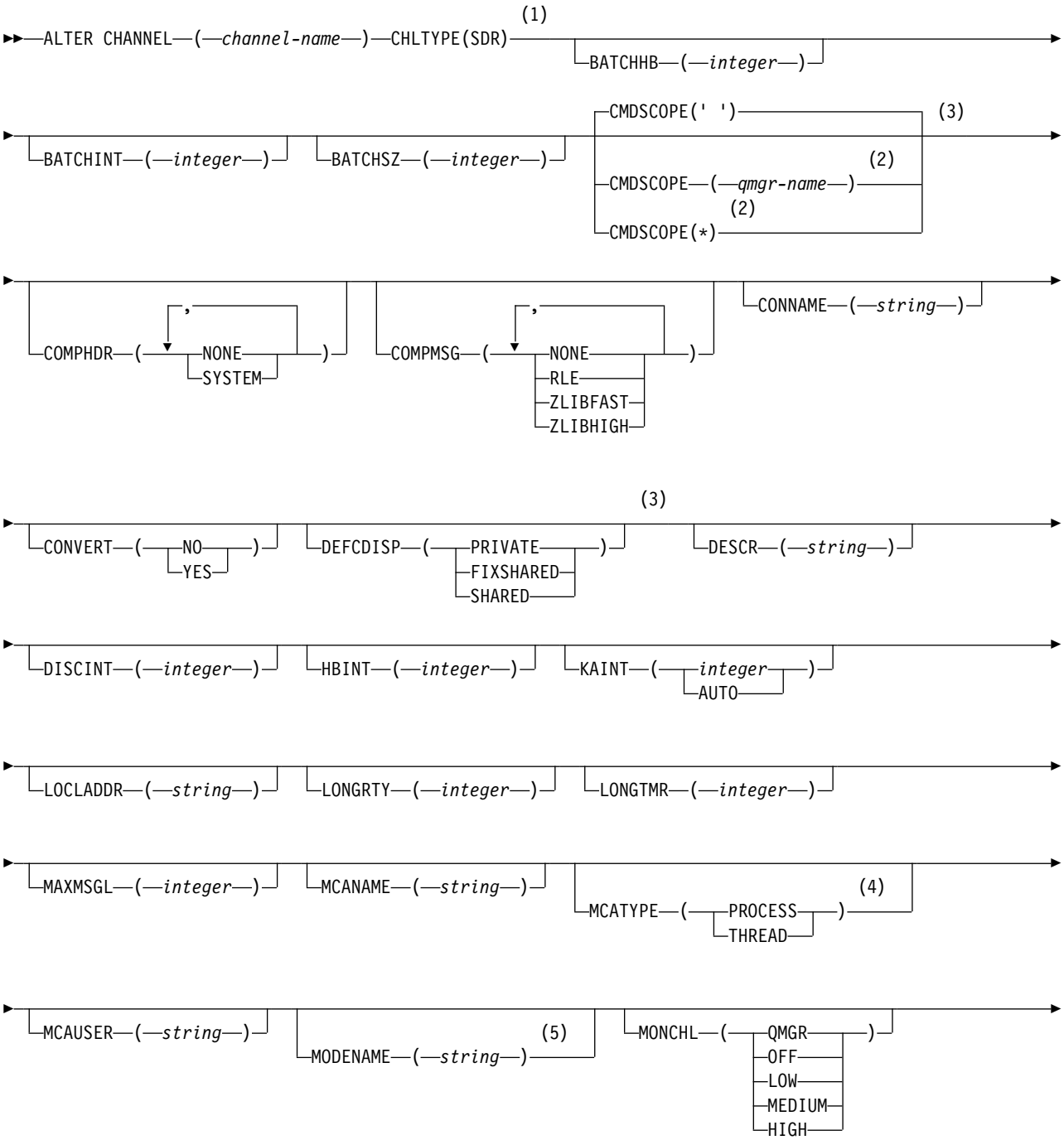
This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR or SVR. For these channel types, this parameter is required.

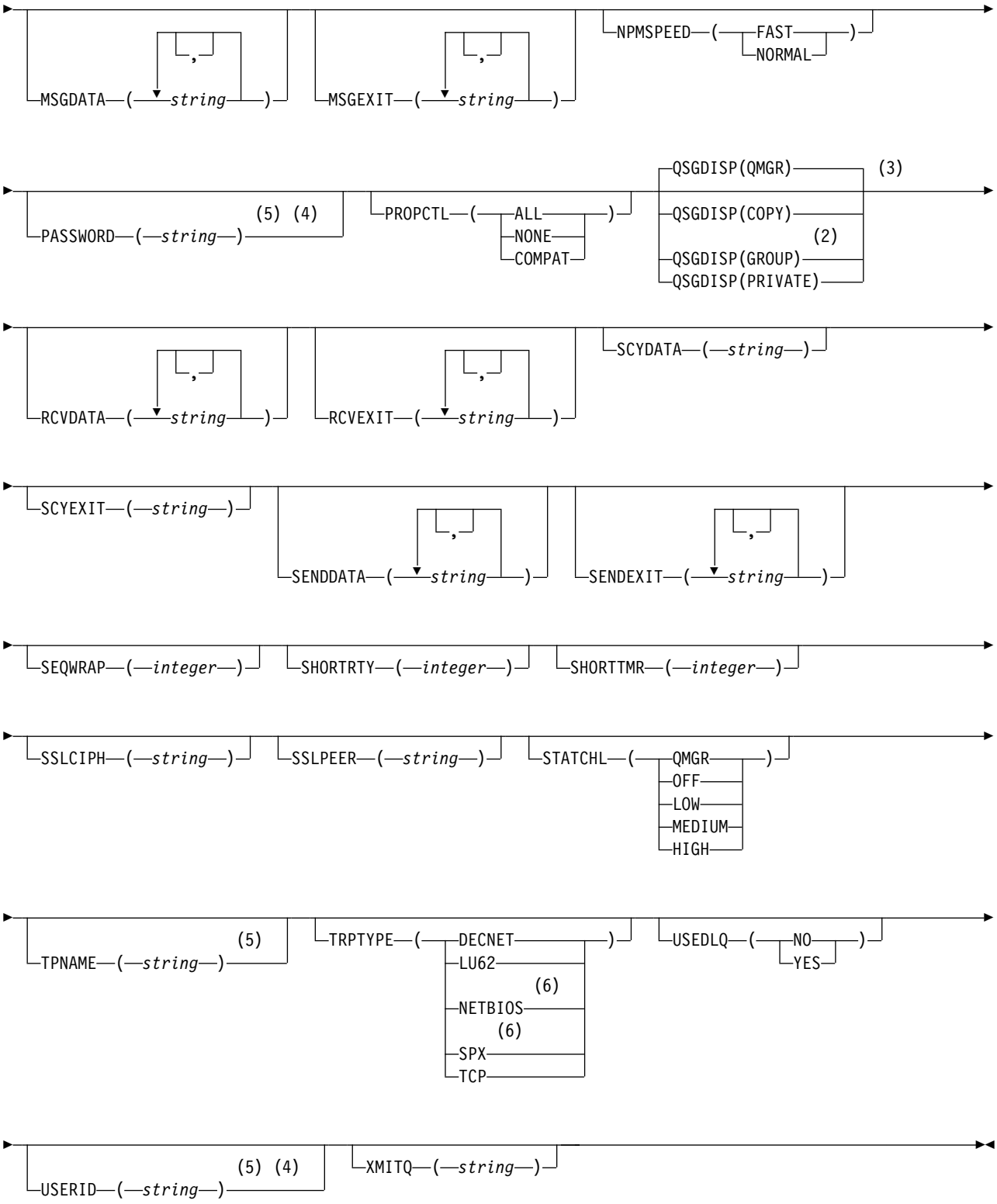
There is a separate syntax diagram for each type of channel:

Sender channel:

Syntax diagram for a sender channel when using the **ALTER CHANNEL** command.

ALTER CHANNEL





Notes:

1 This parameter must follow immediately after the channel name z/OS except on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Not valid on z/OS.

5 Valid only if TRPTYPE is LU62.

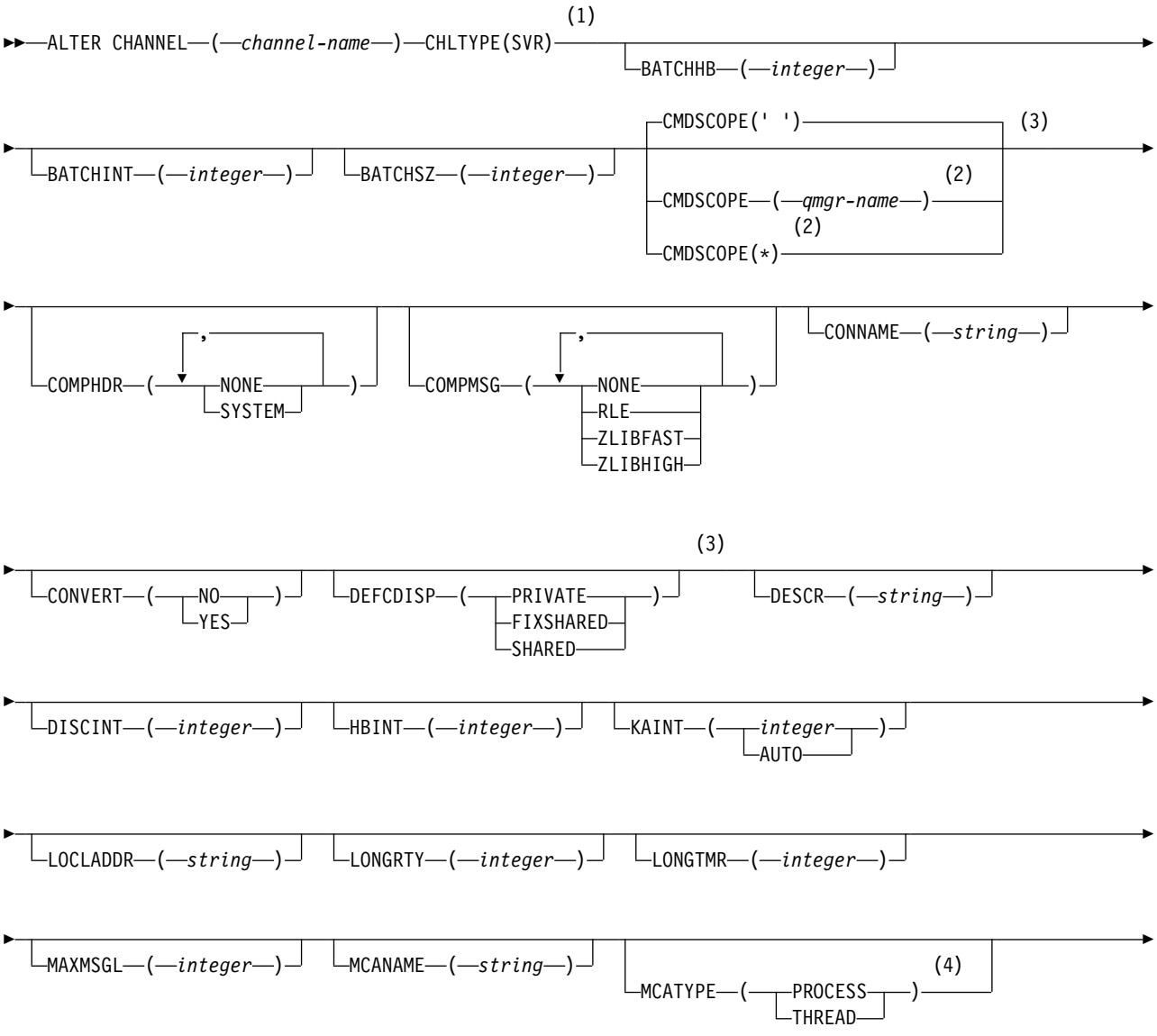
6 Valid only Windows.

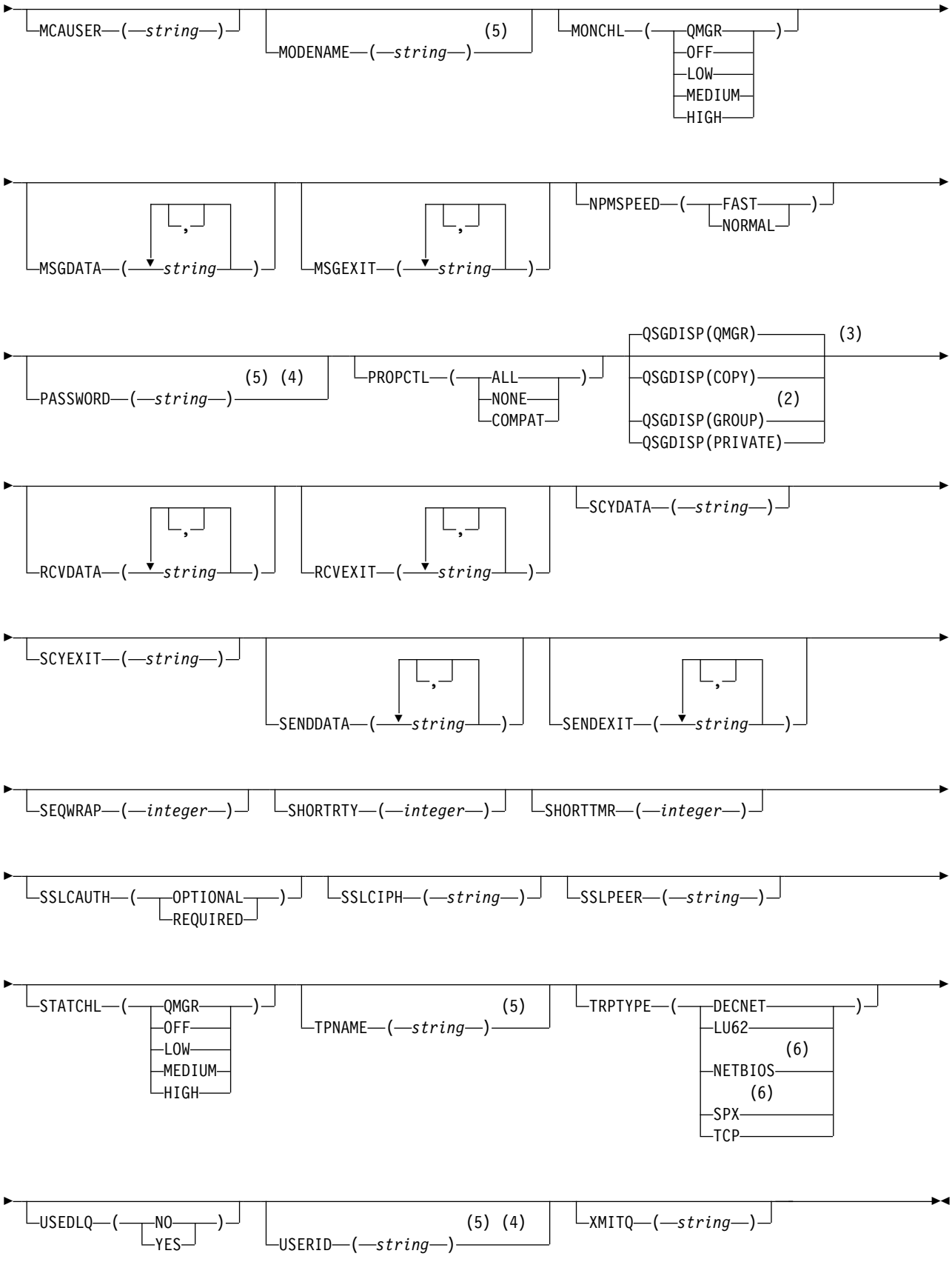
The parameters are described in "ALTER CHANNEL" on page 389.

Server channel:

Syntax diagram for a server channel when using the **ALTER CHANNEL** command.

ALTER CHANNEL




MCAUSER (string)
MODENAME (string) (5)
MONCHL (QMGR, OFF, LOW, MEDIUM, HIGH)
MSGDATA (string)
MSGEXIT (string)
NPMSPEED (FAST, NORMAL)
PASSWORD (string) (5) (4)
PROPCTL (ALL, NONE, COMPAT)
QSGDISP(QMGR) (3)
QSGDISP(COPY)
QSGDISP(GROUP) (2)
QSGDISP(PRIVATE)
RCVDATA (string)
RCVEXIT (string)
SCYDATA (string)
SCYEXIT (string)
SENDDATA (string)
SENDEXIT (string)
SEQWRAP (integer)
SHORTRTY (integer)
SHORTTMR (integer)
SSLCAUTH (OPTIONAL, REQUIRED)
SSLCIPH (string)
SSLPEER (string)
STATCHL (QMGR, OFF, LOW, MEDIUM, HIGH)
TPNAME (string) (5)
TRPTYPE (DECNET, LU62, NETBIOS (6), SPX (6), TCP)
USEDLQ (NO, YES)
USERID (string) (5) (4)
XMITQ (string)

Notes:

1 This parameter must follow immediately after the channel name z/OS except on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Not valid on z/OS.

5 Valid only if TRPTYPE is LU62.

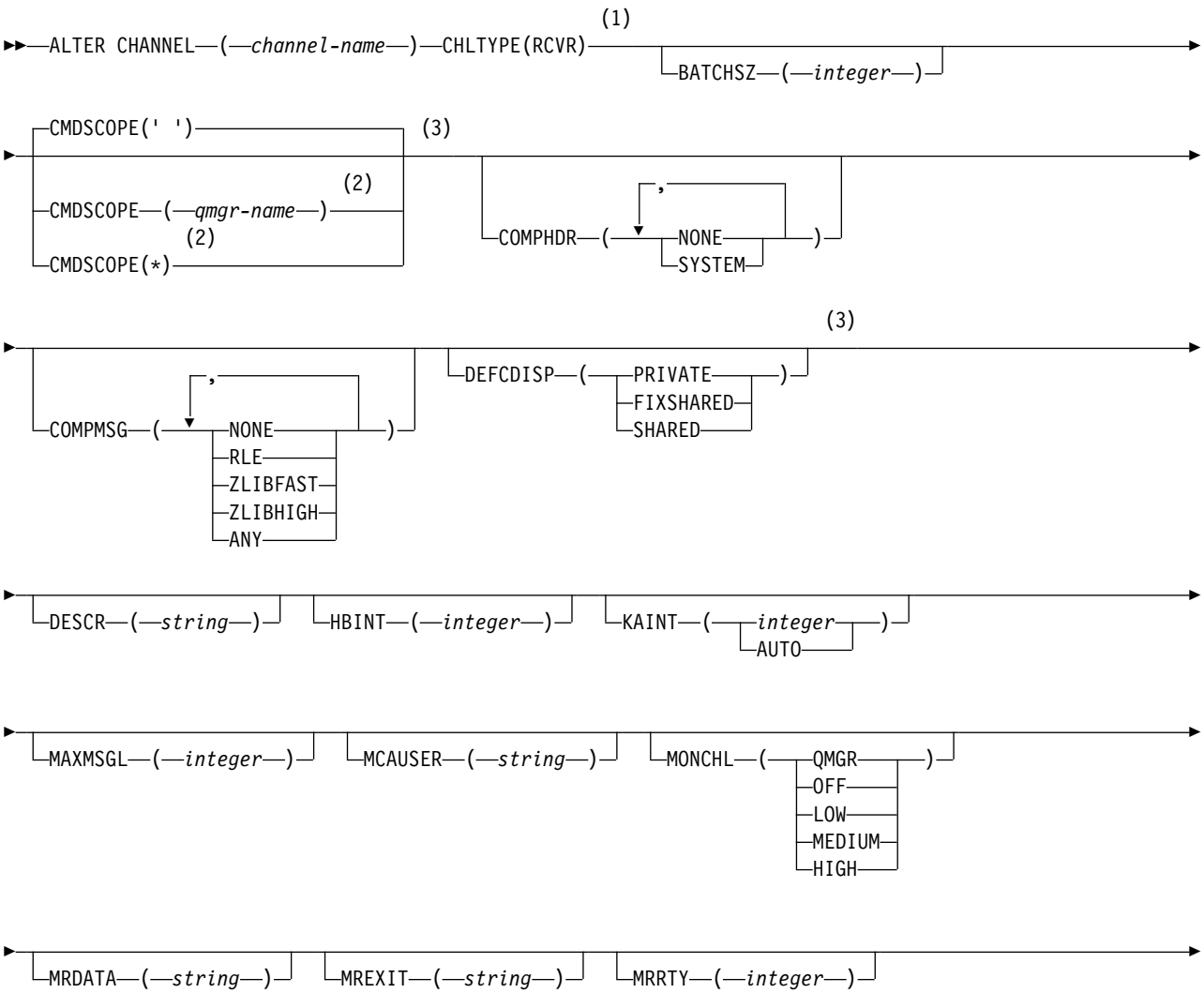
6 Valid only on Windows.

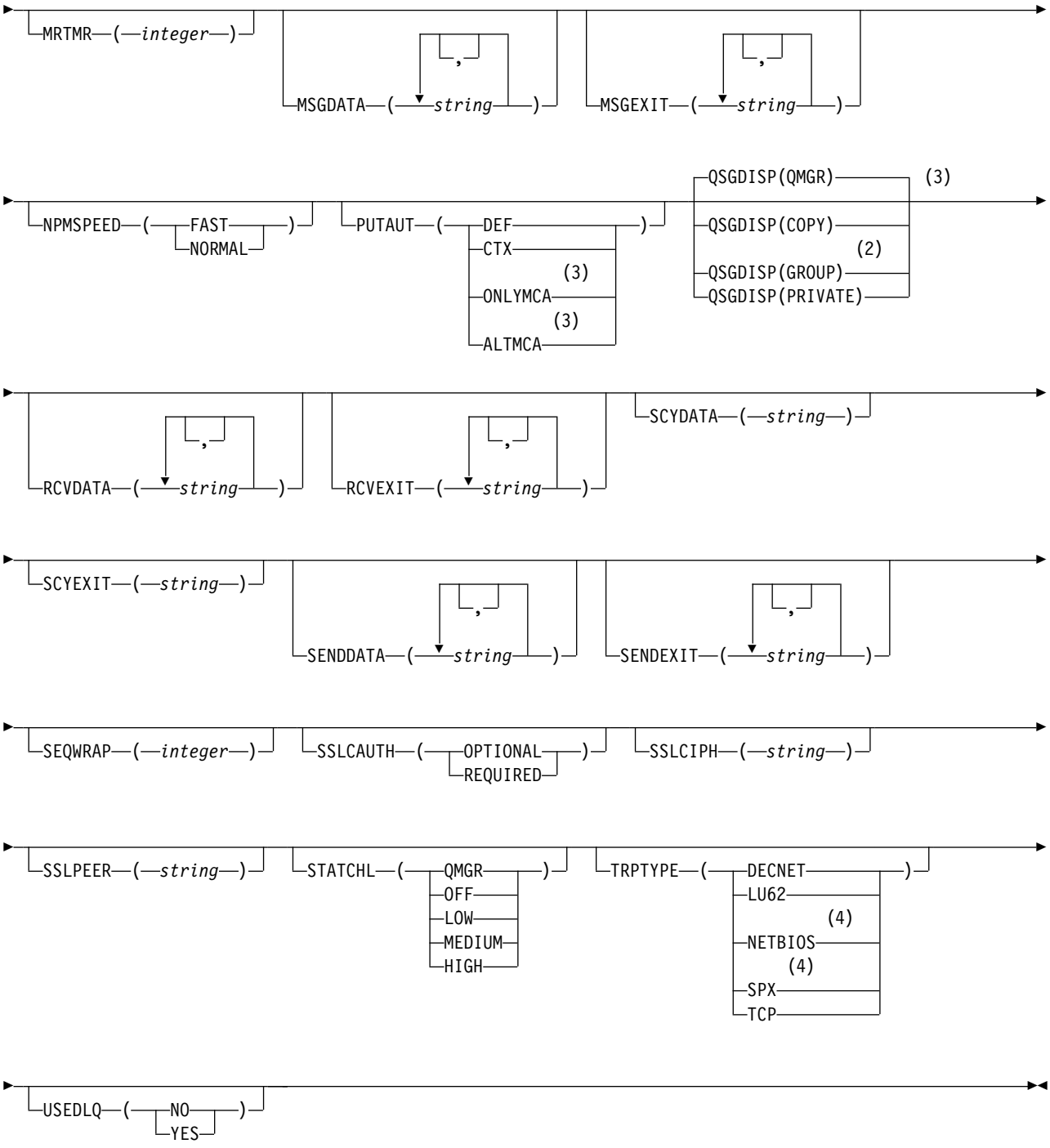
The parameters are described in "ALTER CHANNEL" on page 389.

Receiver channel:

Syntax diagram for a receiver channel when using the **ALTER CHANNEL** command.

ALTER CHANNEL





Notes:

1 This parameter must follow immediately after the channel name z/OS except on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

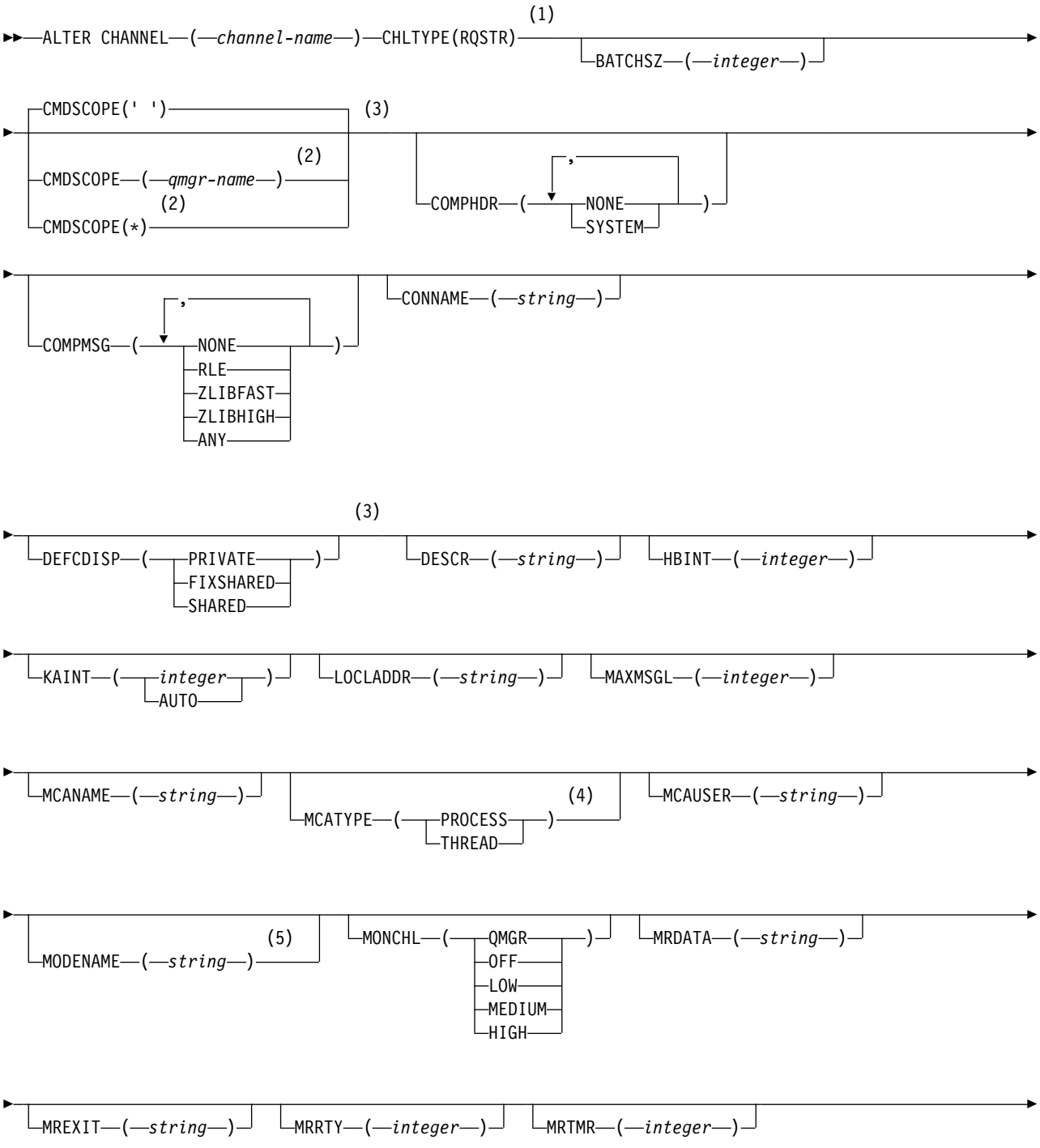
4 Valid only on Windows.

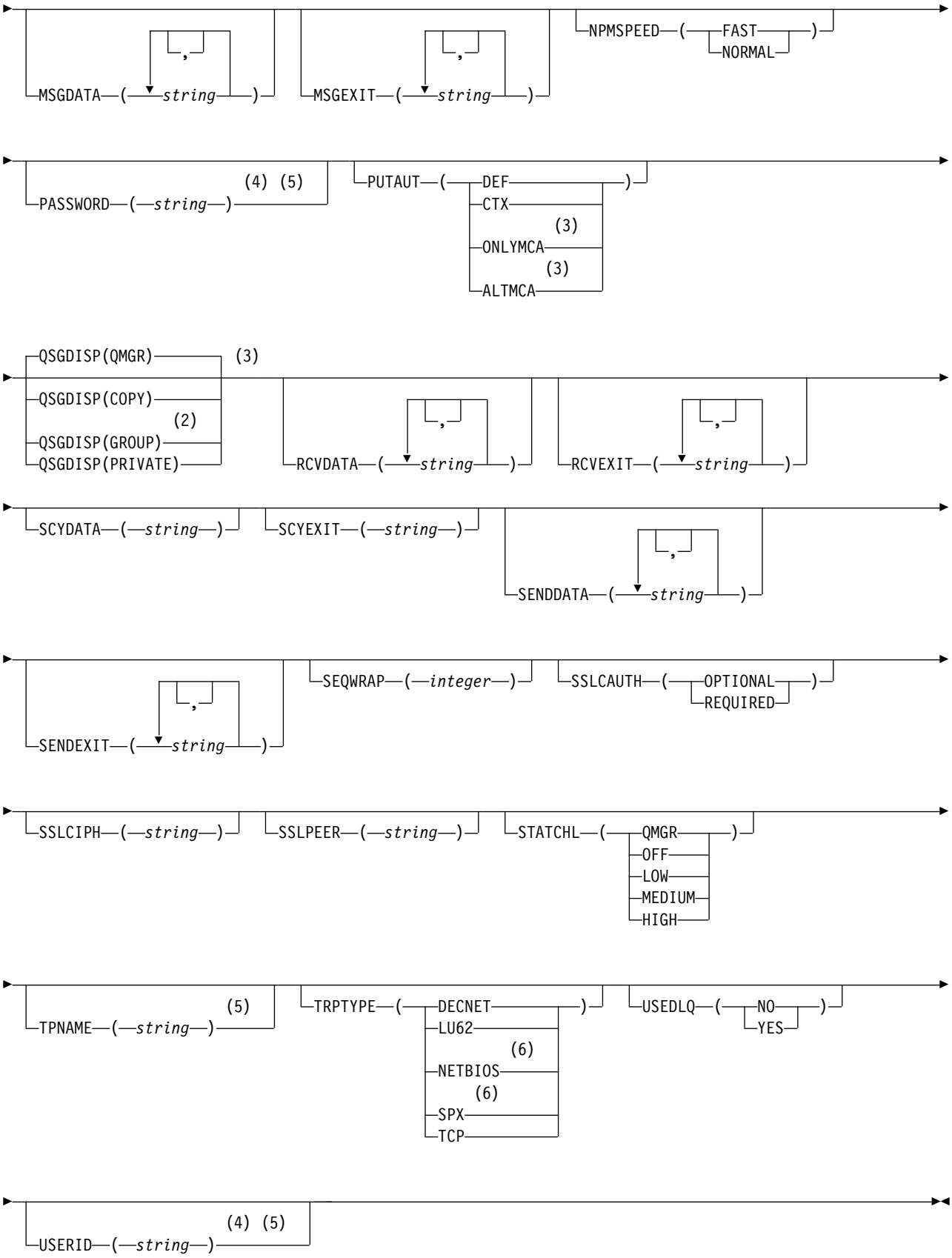
The parameters are described in "ALTER CHANNEL" on page 389.

Requester channel:

Syntax diagram for a requester channel when using the **ALTER CHANNEL** command.

ALTER CHANNEL




MSGDATA
string
MSGEXIT
string
NPMSPEED
FAST
NORMAL
PASSWORD
string
(4) (5)
PUTAUT
DEF
CTX
ONLYMCA
(3)
ALTMCA
(3)
QSGDISP(QMGR)
QSGDISP(COPY)
QSGDISP(GROUP)
(2)
QSGDISP(PRIVATE)
(3)
RCVDATA
string
RCVEXIT
string
SCYDATA
string
SCYEXIT
string
SENDDATA
string
SENDEXIT
string
SEQWRAP
integer
SSLCAUTH
OPTIONAL
REQUIRED
SSLCIPH
string
SSLPEER
string
STATCHL
QMGR
OFF
LOW
MEDIUM
HIGH
TPNAME
string
(5)
TRPTYPE
DECNET
LU62
NETBIOS
(6)
SPX
(6)
TCP
USEDLQ
NO
YES
USERID
string
(4) (5)

Notes:

1 This parameter must follow immediately after the channel name z/OS except on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Not valid on z/OS.

5 Valid only if TRPTYPE is LU62.

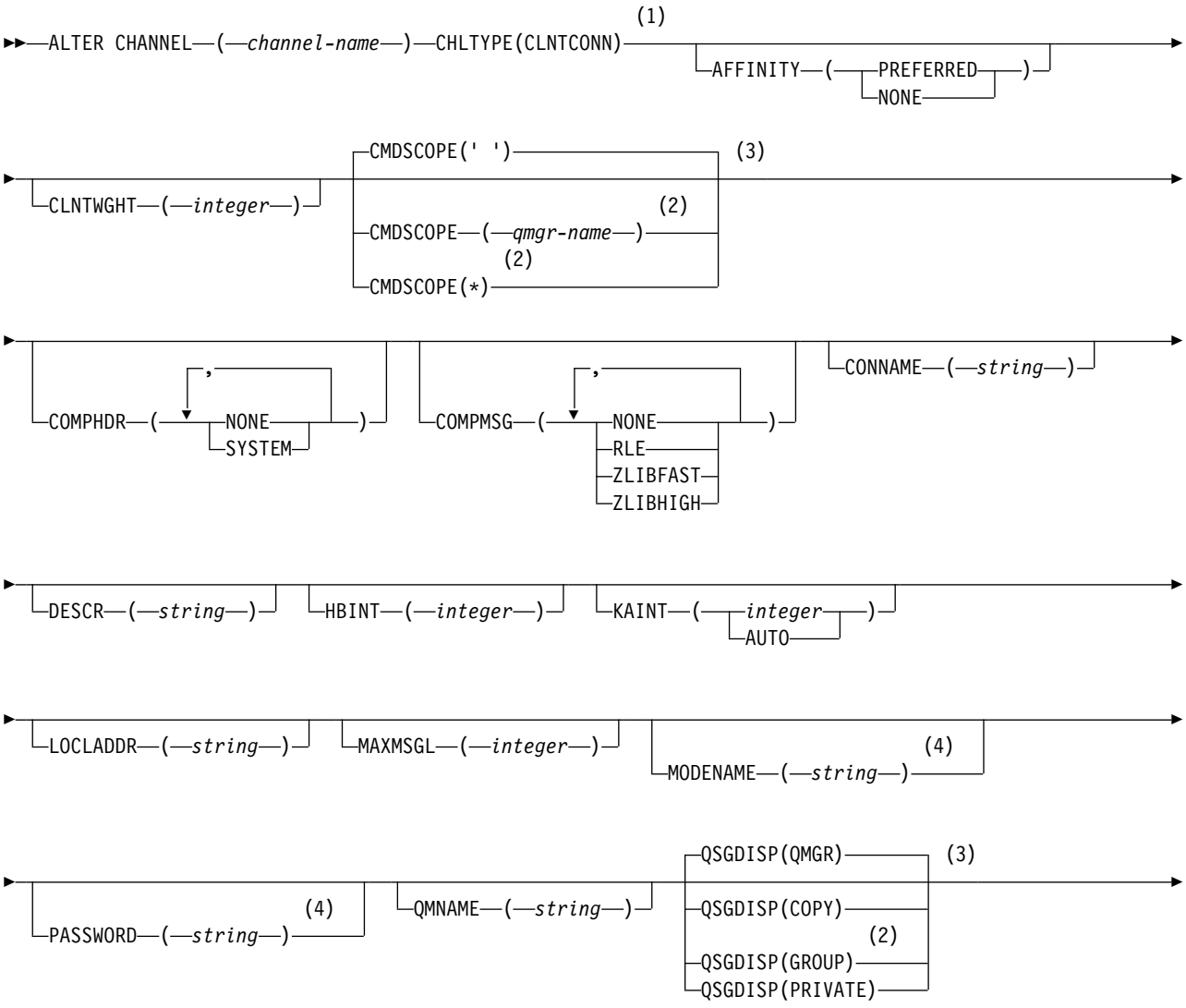
6 Valid only on Windows.

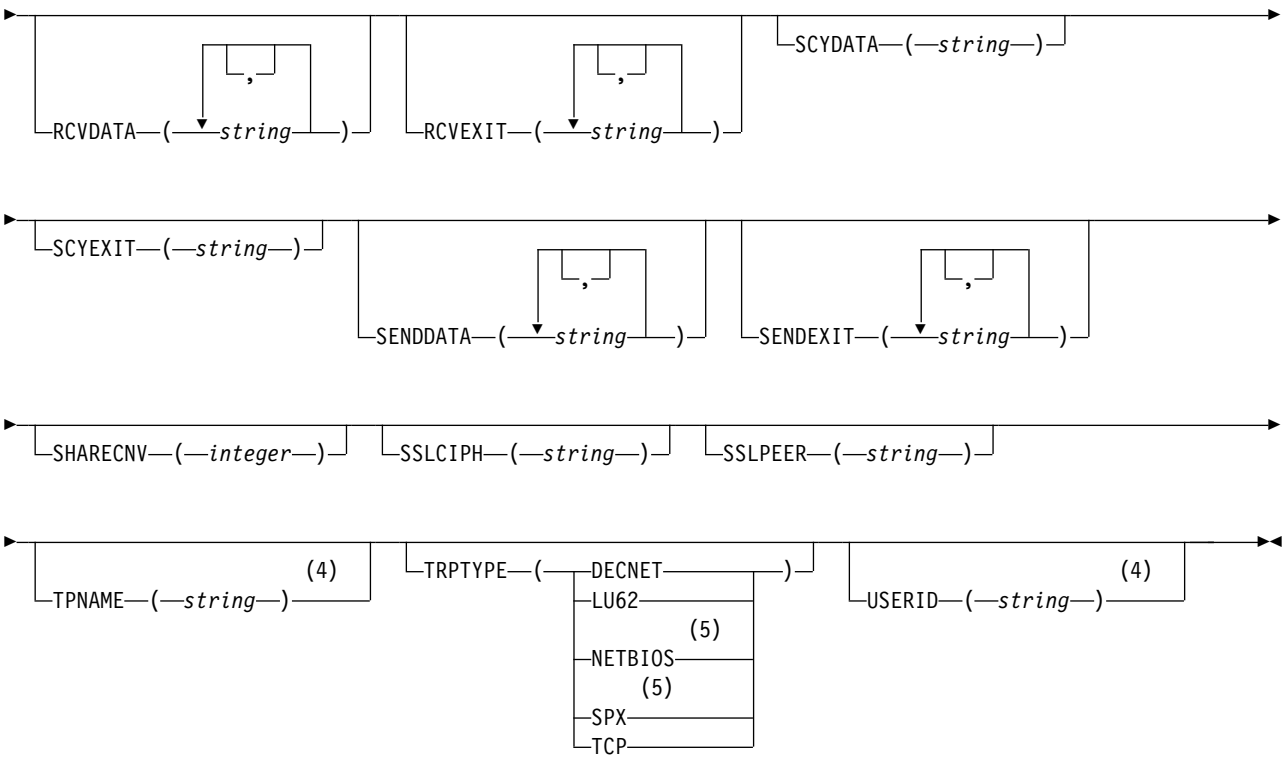
The parameters are described in "ALTER CHANNEL" on page 389.

Client-connection channel:

Syntax diagram for a client-connection channel when using the **ALTER CHANNEL** command.

ALTER CHANNEL





Notes:

1 This parameter must follow immediately after the channel name z/OS except on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Valid only if TRPTYPE is LU62.

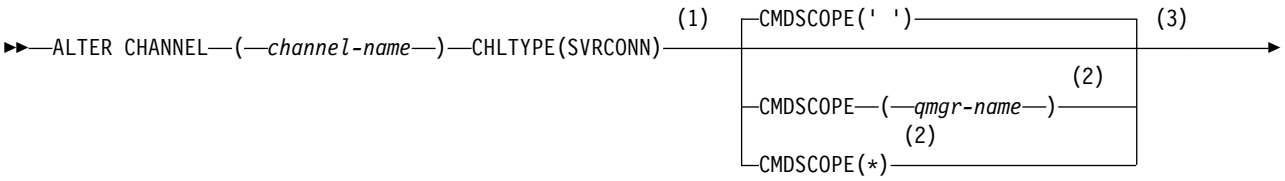
5 Valid only for clients to be run on DOS and Windows.

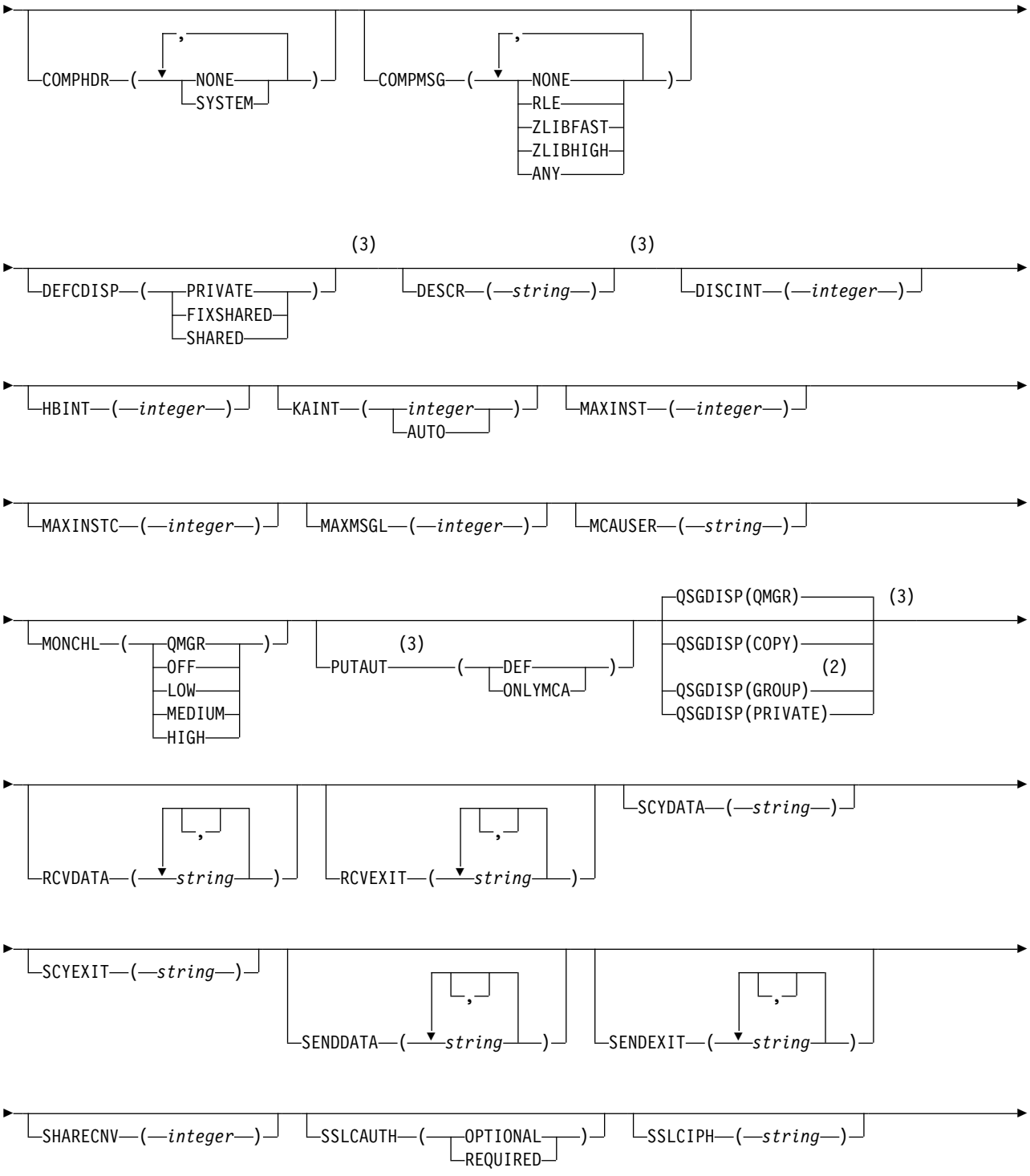
The parameters are described in "ALTER CHANNEL" on page 389.

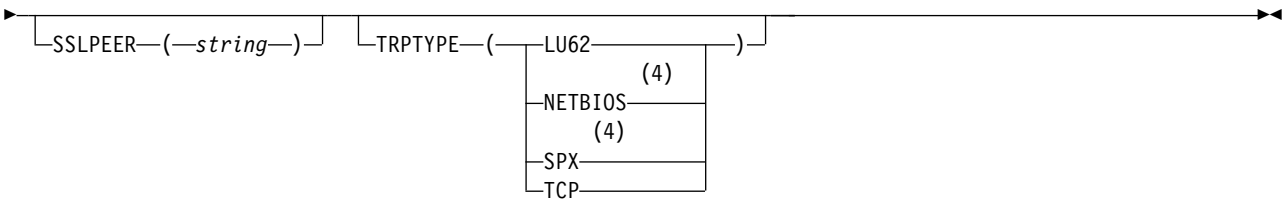
Server-connection channel:

Syntax diagram for a server-connection channel when using the **ALTER CHANNEL** command.

ALTER CHANNEL




COMPHDR
NONE
SYSTEM
COMPMSG
NONE
RLE
ZLIBFAST
ZLIBHIGH
ANY
DEFCDISP
PRIVATE
FIXSHARED
SHARED
(3)
DESCR
string
(3)
DISCINT
integer
HBINT
integer
KAINT
integer
AUTO
MAXINST
integer
MAXINSTC
integer
MAXMSGL
integer
MCAUSER
string
MONCHL
QMGR
OFF
LOW
MEDIUM
HIGH
PUTAUT
(3)
DEF
ONLYMCA
QSGDISP(QMGR)
(3)
QSGDISP(COPY)
QSGDISP(GROUP)
(2)
QSGDISP(PRIVATE)
RCVDATA
string
RCVEXIT
string
SCYDATA
string
SCYEXIT
string
SENDDATA
string
SENDEXIT
string
SHARECNV
integer
SSLCAUTH
OPTIONAL
REQUIRED
SSLCIPH
string



Notes:

1 This parameter must follow immediately after the channel name z/OS except on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

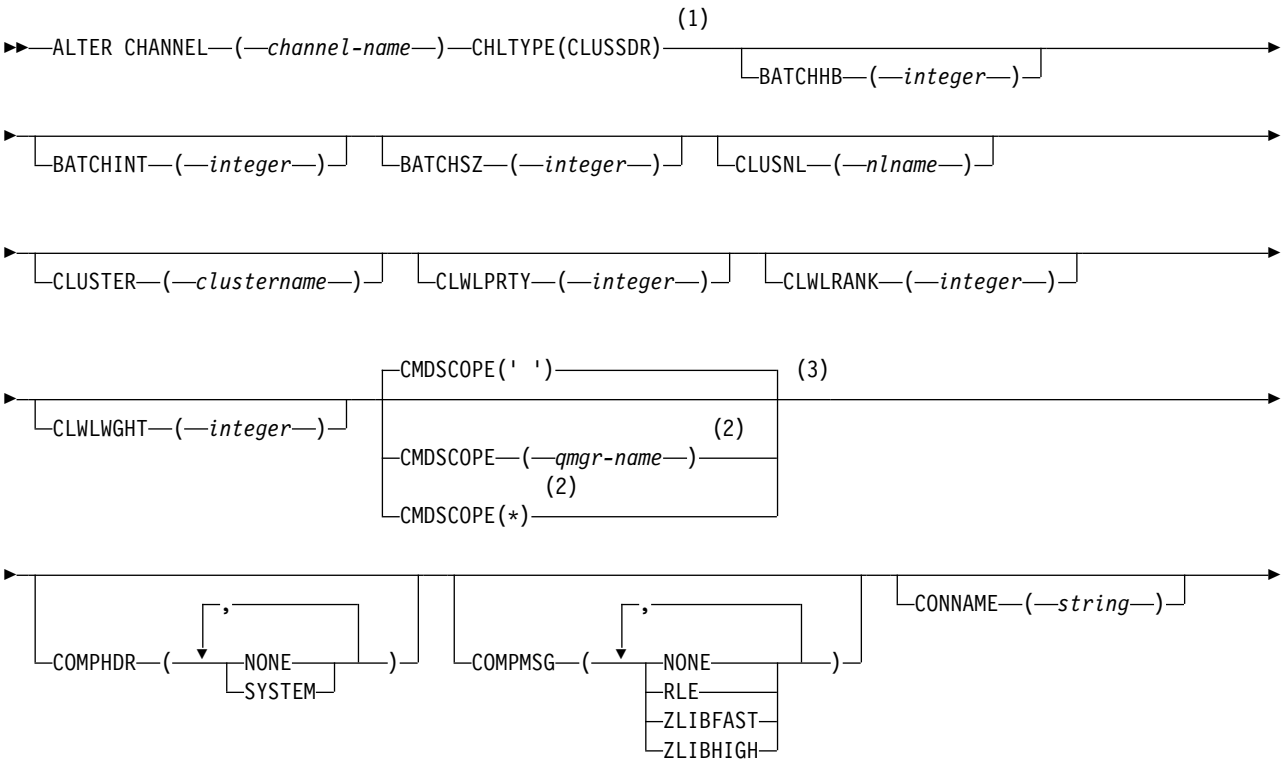
4 Valid only for clients to be run on Windows.

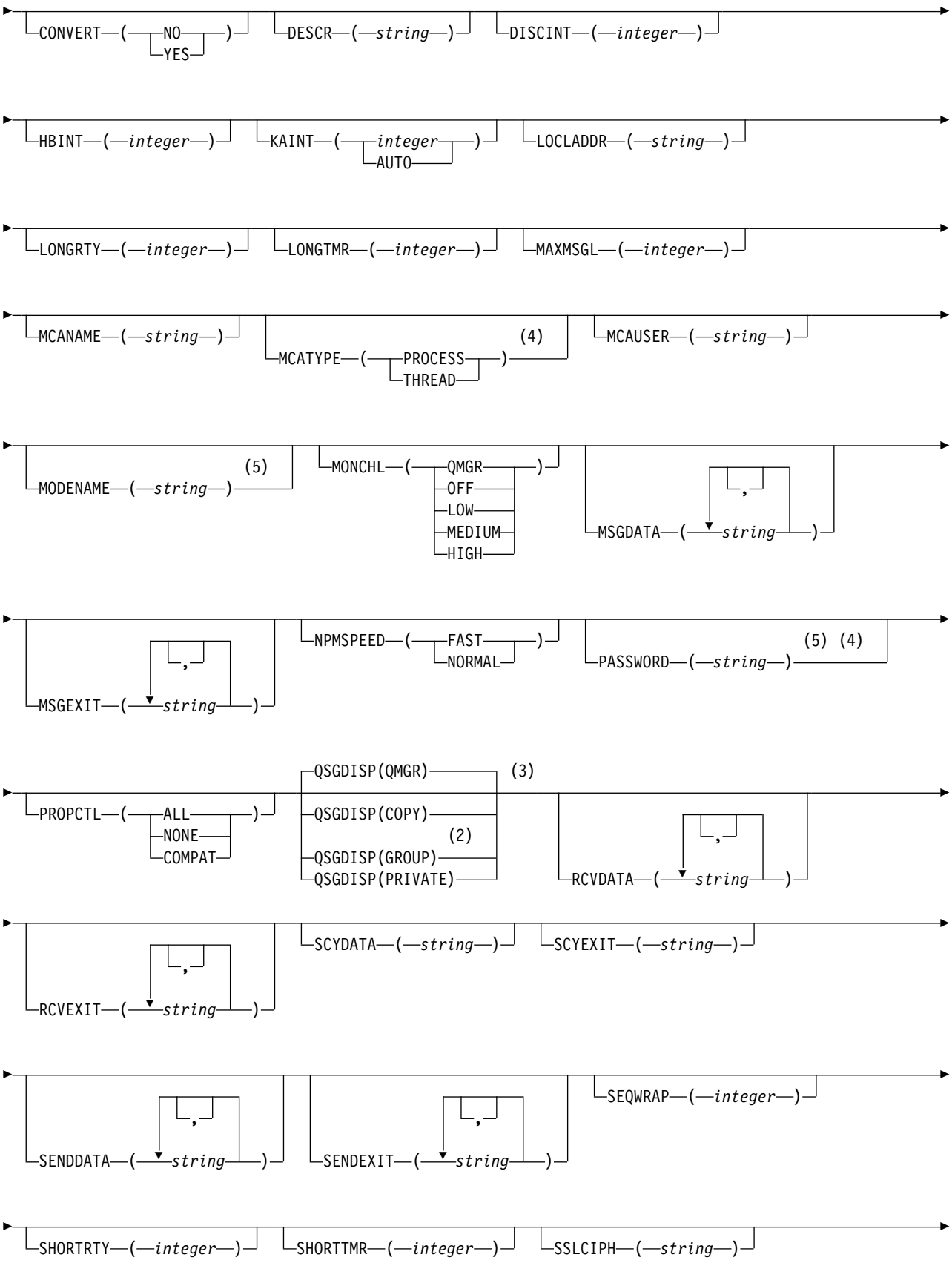
The parameters are described in "ALTER CHANNEL" on page 389.

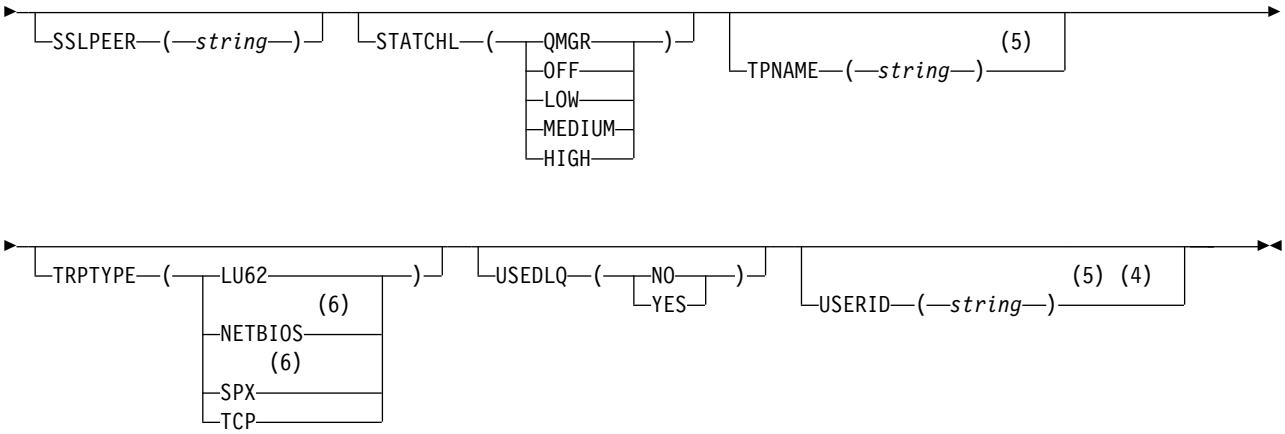
Cluster-sender channel:

Syntax diagram for a cluster-sender channel when using the **ALTER CHANNEL** command.

ALTER CHANNEL




CONVERT—(—NO—)
YES
DESCR—(—string—)
DISCINT—(—integer—)
HBINT—(—integer—)
KAINT—(—integer—)
AUTO
LOCLADDR—(—string—)
LONGRTY—(—integer—)
LONGTMR—(—integer—)
MAXMSGL—(—integer—)
MCANAME—(—string—)
MCATYPE—(—PROCESS—)
THREAD
(4)
MCAUSER—(—string—)
MODENAME—(—string—)
(5)
MONCHL—(—QMGR—)
OFF
LOW
MEDIUM
HIGH
MSGDATA—(—string—)
,
MSGEXIT—(—string—)
,
NPMSPEED—(—FAST—)
NORMAL
PASSWORD—(—string—)
(5) (4)
PROPCTL—(—ALL—)
NONE
COMPAT
QSGDISP(QMGR)
QSGDISP(COPY)
QSGDISP(GROUP)
QSGDISP(PRIVATE)
(2)
(3)
RCVDATA—(—string—)
,
RCVEXIT—(—string—)
,
SCYDATA—(—string—)
SCYEXIT—(—string—)
SENDDATA—(—string—)
,
SENDEXIT—(—string—)
,
SEQWRAP—(—integer—)
SHORTRTY—(—integer—)
SHORTTMR—(—integer—)
SSLCIPH—(—string—)



Notes:

1 This parameter must follow immediately after the channel name z/OS except on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Not valid on z/OS.

5 Valid only if TRPTYPE is LU62.

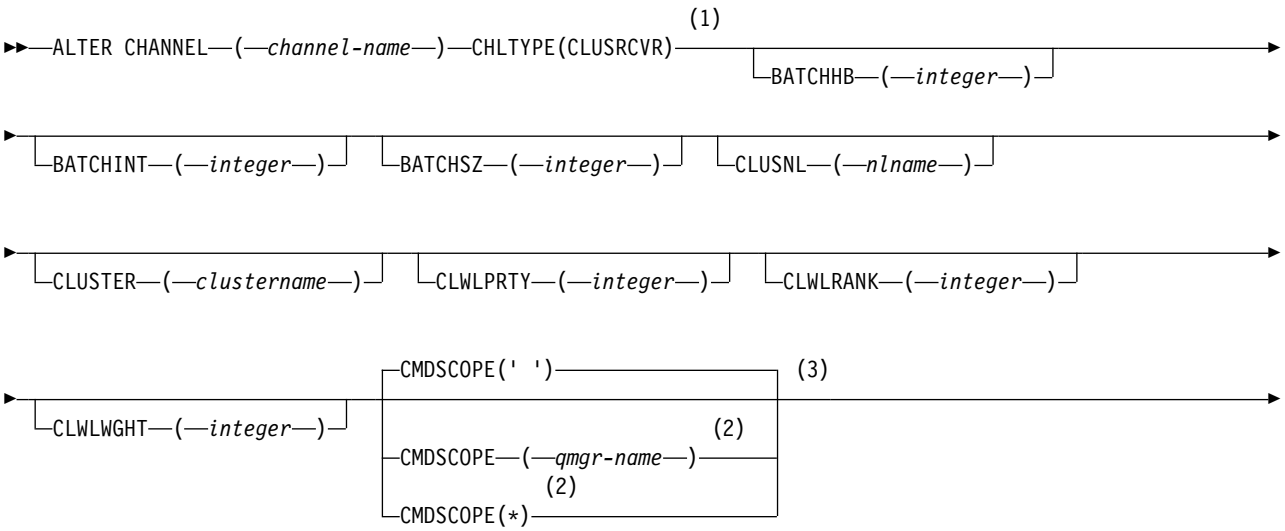
6 Valid only Windows.

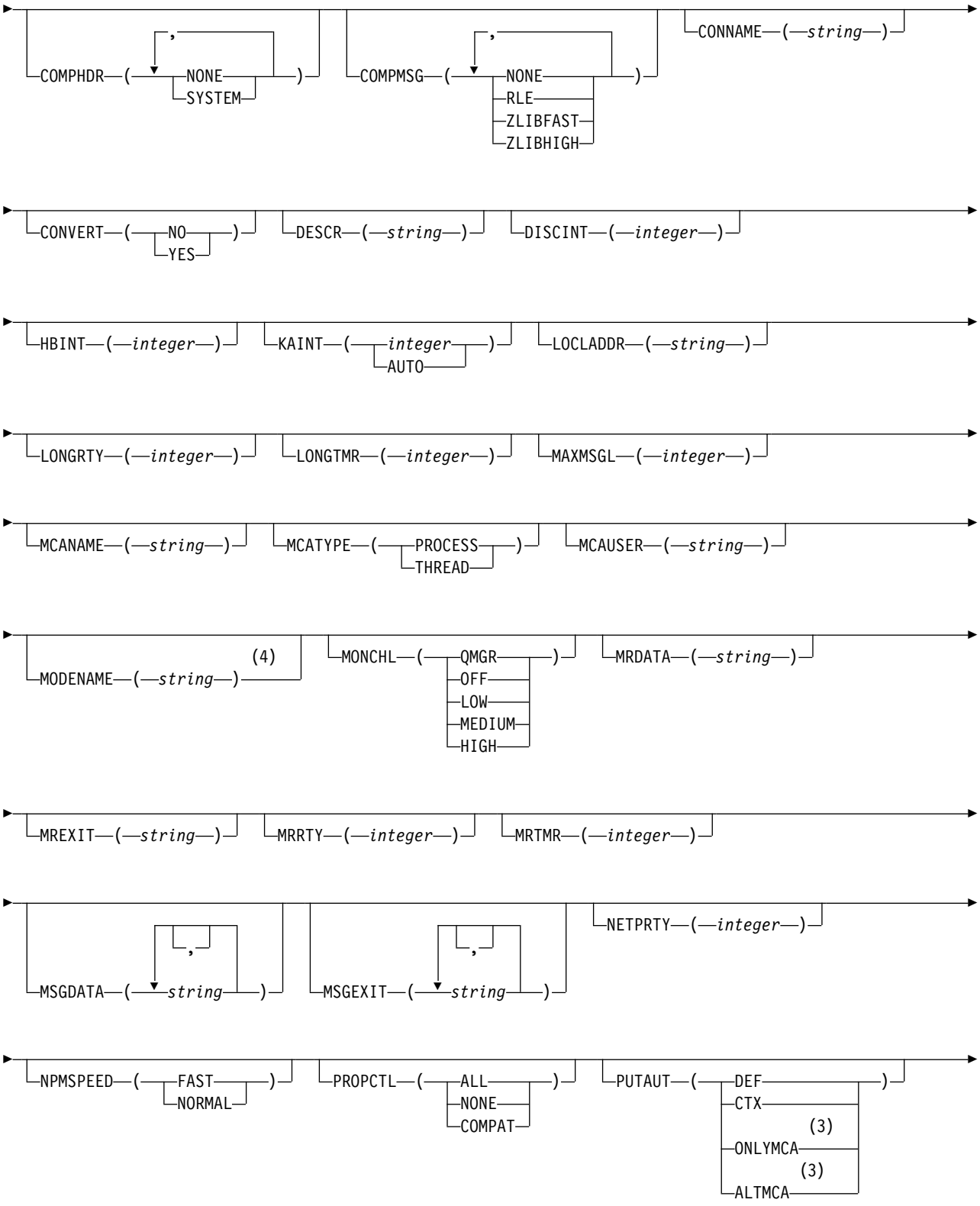
The parameters are described in "ALTER CHANNEL" on page 389.

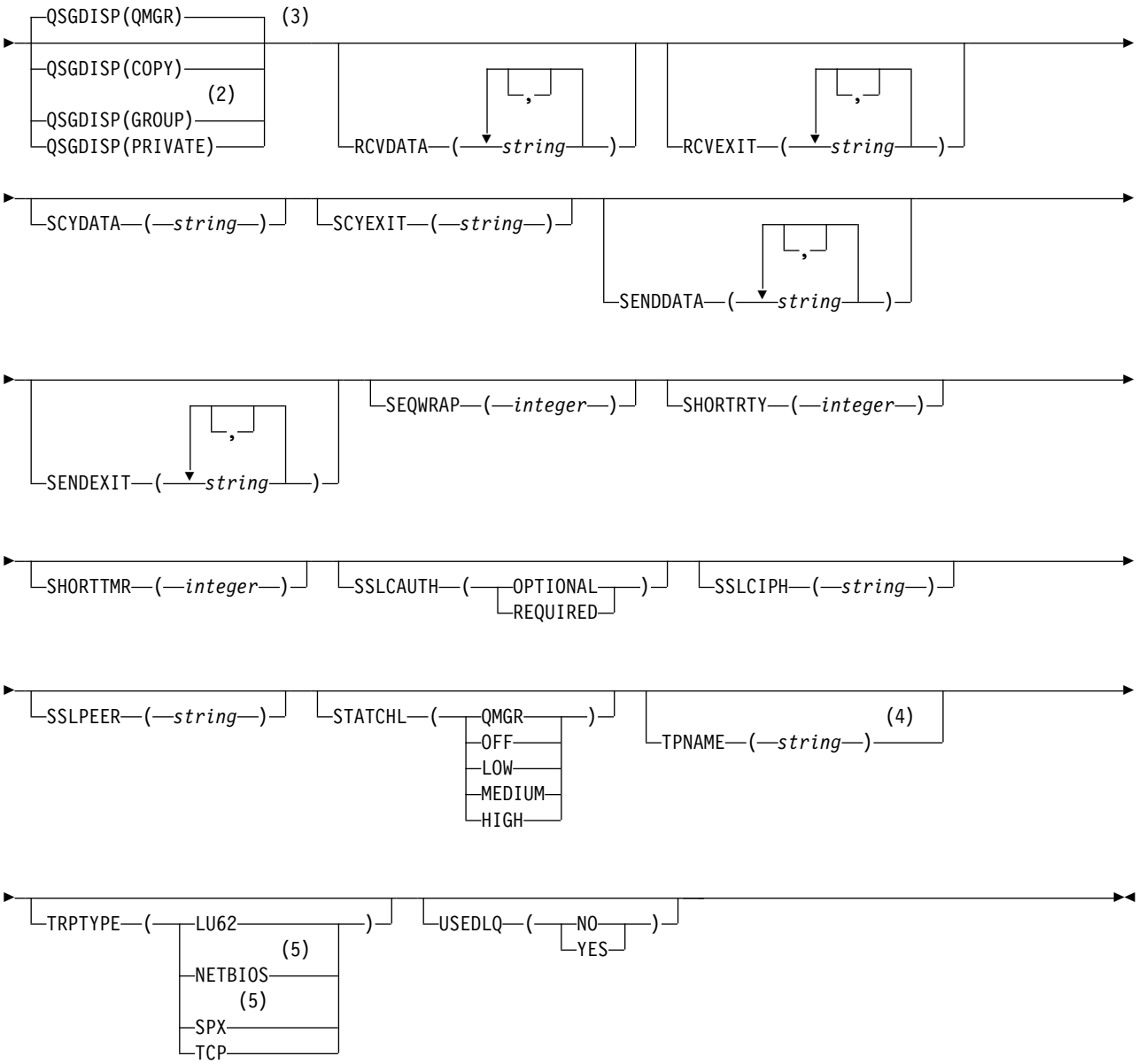
Cluster-receiver channel:

Syntax diagram for a cluster-receiver channel when using the **ALTER CHANNEL** command.

ALTER CHANNEL




COMPHDR
NONE
SYSTEM
COMPMSG
NONE
RLE
ZLIBFAST
ZLIBHIGH
CONNAME
string
CONVERT
NO
YES
DESCR
string
DISCINT
integer
HBINT
integer
KAINT
integer
AUTO
LOCLADDR
string
LONGRTY
integer
LONGTMR
integer
MAXMSGL
integer
MCANAME
string
MCATYPE
PROCESS
THREAD
MCAUSER
string
MODENAME
string
(4)
MONCHL
QMGR
OFF
LOW
MEDIUM
HIGH
MRDATA
string
MREXIT
string
MRRTY
integer
MRTMR
integer
MSGDATA
string
MSGEXIT
string
NETPRTY
integer
NPMSPEED
FAST
NORMAL
PROPCTL
ALL
NONE
COMPAT
PUTAUT
DEF
CTX
ONLYMCA
(3)
ALTMCA
(3)



Notes:

1 This parameter must follow immediately after the channel name z/OS except on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Valid only if TRPTYPE is LU62.

5 Valid only on Windows.

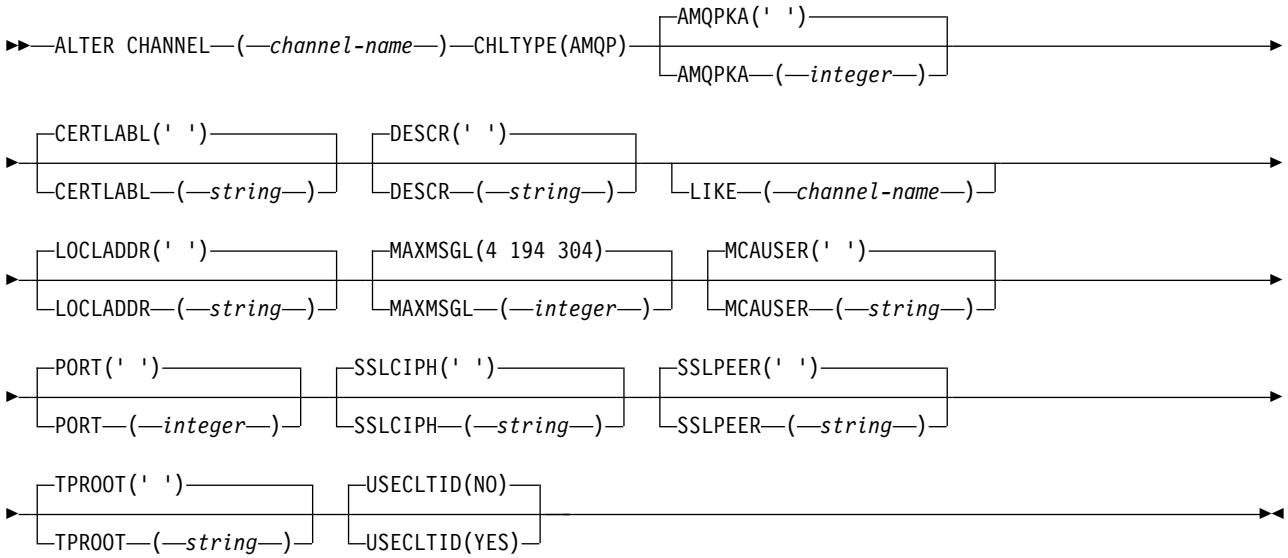
The parameters are described in "ALTER CHANNEL" on page 389.

AMQP channel: ULW V 9.0.0

Syntax diagram for an AMQP channel when using the **ALTER CHANNEL** command.

Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

DEFINE CHANNEL



The parameters are described in "ALTER CHANNEL" on page 389.

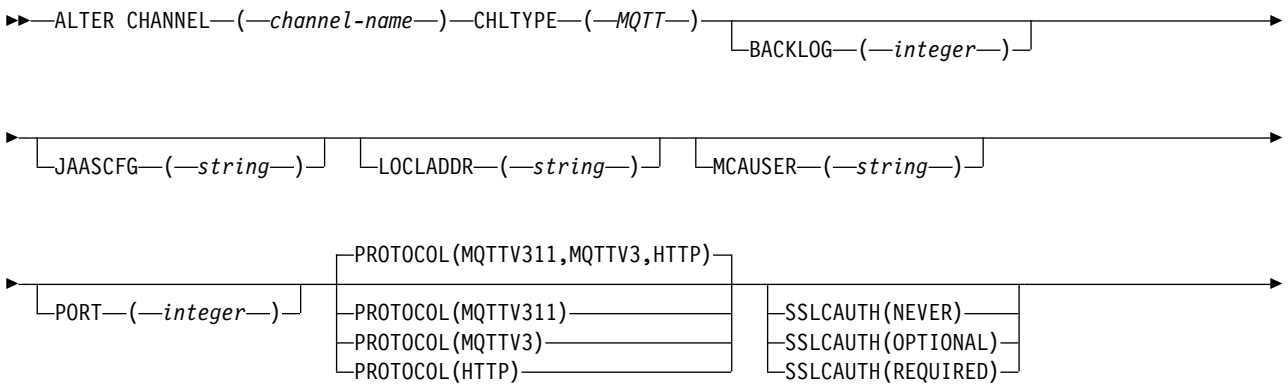
ALTER CHANNEL (MQTT): AIX Linux Windows

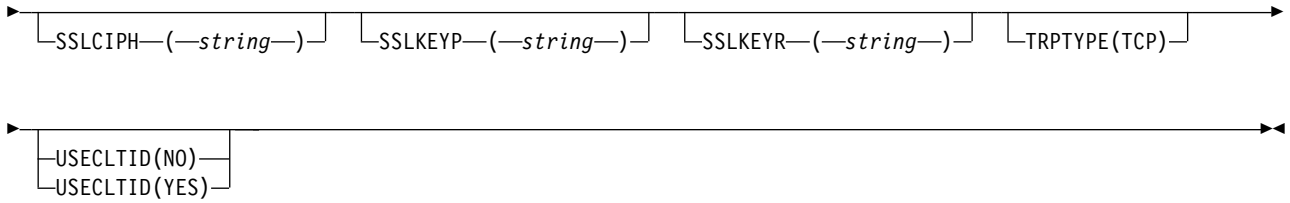
Syntax diagram for a telemetry channel when using the **ALTER CHANNEL** command.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

ALTER CHANNEL (MQTT)





Usage notes

The telemetry (MQXR) service must be running when you issue this command. For instructions on how to start the telemetry (MQXR) service, see Configuring a queue manager for telemetry on Linux or Configuring a queue manager for telemetry on Windows.

Parameter descriptions for `ALTER CHANNEL` (MQTT)

(channel-name)

The name of the channel definition.

BACKLOG(*integer*)

The number of outstanding connection requests that the telemetry channel can support at any one time. When the backlog limit is reached, any further clients trying to connect will be refused connection until the current backlog is processed.

The value is in the range 0 - 999999999.

The default value is 4096.

CHLTYPE

Channel type. MQTT (telemetry) channel.

JAASCFG (*string*)

The name of a stanza in the JAAS configuration file.

See Authenticating an MQTT client Java app with JAAS

LOCLADDR (*ip-addr*)

LOCLADDR is the local communications address for the channel. Use this parameter if you want to force the client to use a particular IP address. LOCLADDR is also useful to force a channel to use an IPv4 or IPv6 address if a choice is available, or to use a particular network adapter on a system with multiple network adapters.

The maximum length of **`LOCLADDR`** is `MQ_LOCAL_ADDRESS_LENGTH`.

If you omit **`LOCLADDR`**, a local address is automatically allocated.

`ip-addr`
ip-addr is a single network address, specified in one of three forms:

IPv4 dotted decimal
For example `192.0.2.1`

IPv6 hexadecimal notation
For example `2001:DB8:0:0:0:0:0:0`

Alphanumeric host name form
For example `WWW.EXAMPLE.COM`

If an IP address is entered, only the address format is validated. The IP address itself is not validated.

MCAUSER(*string*)

Message channel agent user identifier.

The maximum length of the string is 64 characters on Windows and 12 characters on other platforms. On Windows, you can optionally qualify a user identifier with the domain name in the format `user@domain`.

If this parameter is nonblank, and **USECLNTID** is set to `NO`, then this user identifier is used by the telemetry service for authorization to access IBM MQ resources.

If this parameter is blank, and **USECLNTID** is set to `NO`, then the user name flowed in the MQTT CONNECT Packet is used. See MQTT client identity and authorization.

PORT(*integer*)

The port number on which the telemetry (MQXR) service accepts client connections. The default port number for a telemetry channel is 1883; and the default port number for a telemetry channel secured using SSL is 8883. Specifying a port value of 0 causes MQTT to dynamically allocate an available port number.

PROTOCOL

The following communication protocols are supported by the channel:

MQTTV311
The channel accepts connections from clients using the protocol defined by the MQTT Version 3.1.1 Oasis standard. The functionality provided by this protocol is almost identical to that provided by the pre-existing MQTTV3 protocol.

MQTTV3 The channel accepts connections from clients using the MQTT V3.1 Protocol Specification from mqtt.org.

HTTP The channel accepts HTTP requests for pages, or WebSockets connections to MQ Telemetry.

To accept connections from clients using different protocols, specify the acceptable values as a comma-delimited list. For example if you specify `MQTTV3,HTTP` the channel accepts connections from clients using either MQTTV3 or HTTP. If you specify no client protocols, the channel accepts connections from clients using any of the supported protocols.

If you are using IBM MQ Version 8.0.0, Fix Pack 3 or later, and your configuration includes an MQTT channel that was last modified in an earlier version of the product, you must explicitly change the protocol setting to prompt the channel to use the `MQTTV311` option. This is so even if the channel does not specify any client protocols, because the specific protocols to use with the channel are stored at the time the channel is configured, and previous versions of the product have no awareness of the `MQTTV311` option. To prompt a channel in this state to use the `MQTTV311` option, explicitly add the option then save your changes. The channel definition is now aware of the option. If you subsequently change the settings again, and specify no client protocols, the `MQTTV311` option is still included in the stored list of supported protocols.

SSLCAUTH

Defines whether IBM MQ requires a certificate from the TLS client. The initiating end of the channel acts as the TLS client, so this parameter applies to the end of the channel that receives the initiation flow, which acts as the TLS server.

NEVER IBM MQ never requests a certificate from the TLS client.

REQUIRED
IBM MQ requires and validates a certificate from the TLS client.

OPTIONAL
IBM MQ lets the TLS client decide whether to provide a certificate. If the client sends a certificate, the contents of this certificate are validated as normal.

SSLCIPH(*string*)

When **SSLCIPH** is used with a telemetry channel, it means TLS Cipher Suite. The TLS cipher suite is the one supported by the JVM that is running the telemetry (MQXR) service. If the **SSLCIPH** parameter is blank, no attempt is made to use TLS on the channel.

If you plan to use SHA-2 cipher suites, see System requirements for using SHA-2 cipher suites with MQTT channels.

SSLKEYP(*string*)

The passphrase for the TLS key repository.

SSLKEYR(*string*)

The full path name of the TLS key repository file, the store for digital certificates and their associated private keys. If you do not specify a key file, TLS is not used.

The maximum length of the string is 256 characters;

- AIX Linux On AIX and Linux, the name is of the form *pathname/keyfile*.
- Windows On Windows, the name is of the form *pathname\keyfile*.

where *keyfile* is specified without the suffix `.kdb`, and identifies a Java keystore file.

TRPTYPE (*string*)

The transmission protocol to be used:

TCP
TCP/IP.

USECLTID

Decide whether you want to use the MQTT client ID for the new connection as the IBM MQ user ID for that connection. If this property is specified, the user name supplied by the client is ignored.

If you set this parameter to `YES`, then **MCAUSER** must be blank.

If **USECLNTID** is set to `NO`, and **MCAUSER** is blank, then the user name flowed in the MQTT CONNECT Packet is used. See MQTT client identity and authorization.

Related reference:

"DEFINE CHANNEL (MQTT)" on page 634
Syntax diagram for a telemetry channel when using the **DEFINE CHANNEL** command.

Related information:

Telemetry channel configuration for MQTT client authentication using TLS

Telemetry channel configuration for channel authentication using TLS

CipherSpecs and CipherSuites

System requirements for using SHA-2 cipher suites with MQTT channels

ALTER COMMINFO:

Use the MQSC command ALTER COMMINFO to alter the parameters of a communication information object.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

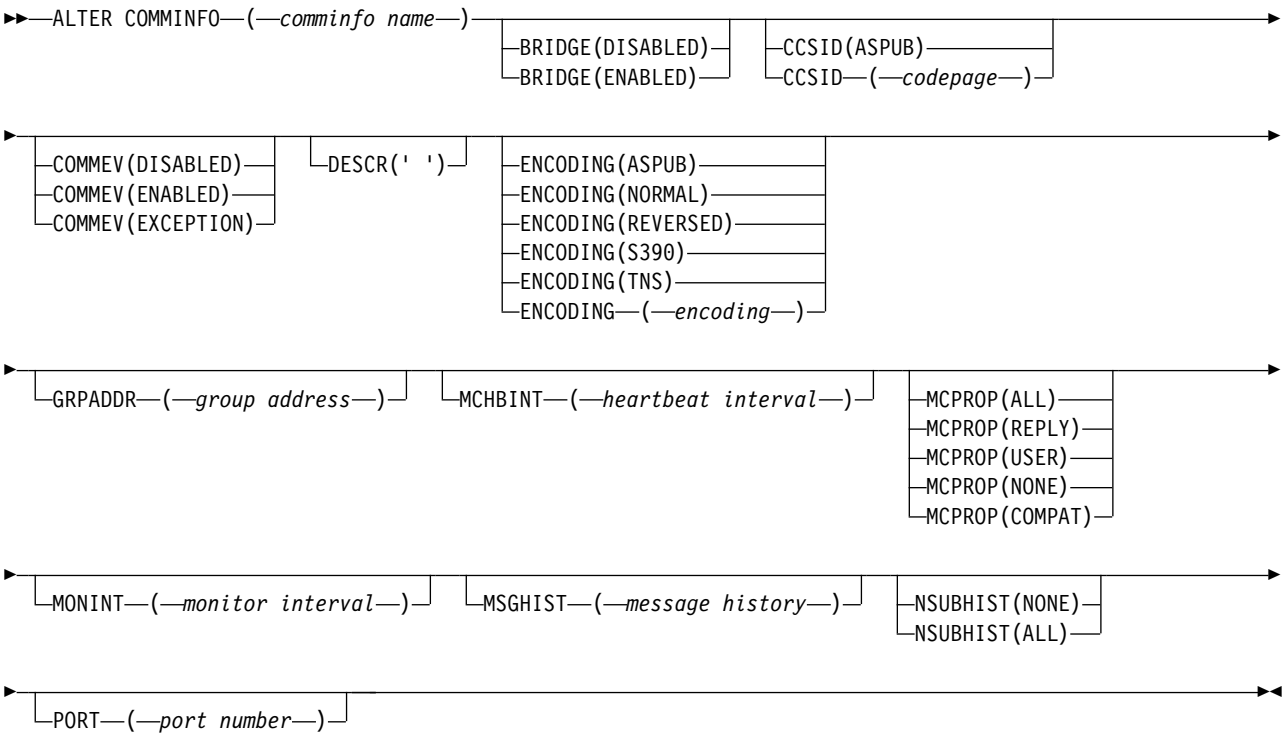
Parameters not specified in the **ALTER COMMINFO** command result in the existing values for those parameters being left unchanged.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for ALTER COMMINFO"

Synonym: ALT COMMINFO

ALTER COMMINFO



Parameter descriptions for ALTER COMMINFO

(comminfo name)
Name of the communications information object. This parameter is required.

The name must not be the same as any other communications information object name currently defined on this queue manager. See Rules for naming IBM MQ objects.

BRIDGE
Controls whether publications from applications not using Multicast are bridged to applications using Multicast. Bridging does not apply to topics that are marked as **MCAST(ONLY)**. As these topics can only be Multicast traffic, it is not applicable to bridge to the queue's publish/subscribe domain.

DISABLED
Publications from applications not using Multicast are not bridged to applications that do use Multicast.

ENABLED
Publications from applications not using Multicast are bridged to applications that do use Multicast.

CCSID(*integer*)
The coded character set identifier that messages are transmitted on. Specify a value in the range 1 through 65535.

The CCSID must specify a value that is defined for use on your platform, and use a character set that is appropriate to the queue manager's platform. If you use this parameter to change the CCSID, applications that are running when the change is applied continue to use the original CCSID therefore you must stop and restart all running applications before you continue. Running applications include the command server and channel programs. Stop and restart all running applications, stop and restart the queue manager after changing this parameter.

The CCSID can also be set to `ASPUB`, which means that the coded character set is taken from that supplied in the published message.

COMMEV
Controls whether event messages are generated for Multicast handles that are created using this COMMINFO object. Events are only generated if they are enabled using the **MONINT** parameter.

DISABLED
Publications from applications not using Multicast are not bridged to applications that do use Multicast.

ENABLED
Publications from applications not using Multicast are bridged to applications that do use Multicast.

EXCEPTION
Event messages are written if the message reliability is below the reliability threshold. The reliability threshold is set to 90 by default.

DESCR(*string*)
Plain-text comment. It provides descriptive information about the communication information object when an operator issues the DISPLAY COMMINFO command (see "DISPLAY COMMINFO on Multiplatforms" on page 820).

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

ENCODING
The encoding that the messages are transmitted in.

ASPUB
The encoding of the message is taken from that supplied in the published message.

NORMAL

REVERSED

S390

TNS

encoding

GRPADDR
The group IP address or DNS name.

It is the responsibility of the administrator to manage the group addresses. It is possible for all multicast clients to use the same group address for every topic; only the messages that match outstanding subscriptions on the client are delivered. Using the same group address can be

inefficient because every client has to examine and process every multicast packet in the network. It is more efficient to allocate different IP group addresses to different topics or sets of topics, but this allocation requires careful management, especially if other non-MQ multicast applications are in use on the network.

MCHBINT
The heartbeat interval is measured in milliseconds, and specifies the frequency at which the transmitter notifies any receivers that there is no further data available.

MCPROP
The multicast properties control how many of the MQMD properties and user properties flow with the message.

All
All user properties and all the fields of the MQMD are transported.

Reply
Only user properties, and MQMD fields that deal with replying to the messages, are transmitted. These properties are:

- MsgType
- MessageId
- CorrelId
- ReplyToQ
- ReplyToQmgr

User
Only the user properties are transmitted.

NONE
No user properties or MQMD fields are transmitted.

COMPAT
This value causes the transmission of the message to be done in a compatible mode to RMM allowing some inter-operation with the current XMS applications and Broker RMM applications.

MONINT(*integer* **)**
How frequently, in seconds, that monitoring information is updated. If events messages are enabled, this parameter also controls how frequently event messages are generated about the status of the Multicast handles created using this COMMINFO object.

A value of 0 means that there is no monitoring.

MSGHIST
The maximum message history is the amount of message history that is kept by the system to handle retransmissions in the case of NACKs (negative acknowledgments).

A value of `0` gives the least level of reliability.

NSUBHIST
The new subscriber history controls whether a subscriber joining a publication stream receives as much data as is currently available, or receives only publications made from the time of the subscription.

NONE
A value of `NONE` causes the transmitter to transmit only publication made from the time of the subscription.

ALL
A value of `ALL` causes the transmitter to retransmit as much history of the topic as is known. In some circumstances, this retransmission can give a similar behavior to retained publications.

Note: Using the value of ALL might have a detrimental effect on performance if there is a large topic history because all the topic history is retransmitted.

PORT(*integer*)
The port number to transmit on.

ALTER LISTENER on Multiplatforms: Multi

Use MQSC command **ALTER LISTENER** to alter the parameters of an existing IBM MQ listener definition. If the listener is already running, any changes you make to its definition are effective only after the next time that the listener is started.

Using MQSC commands

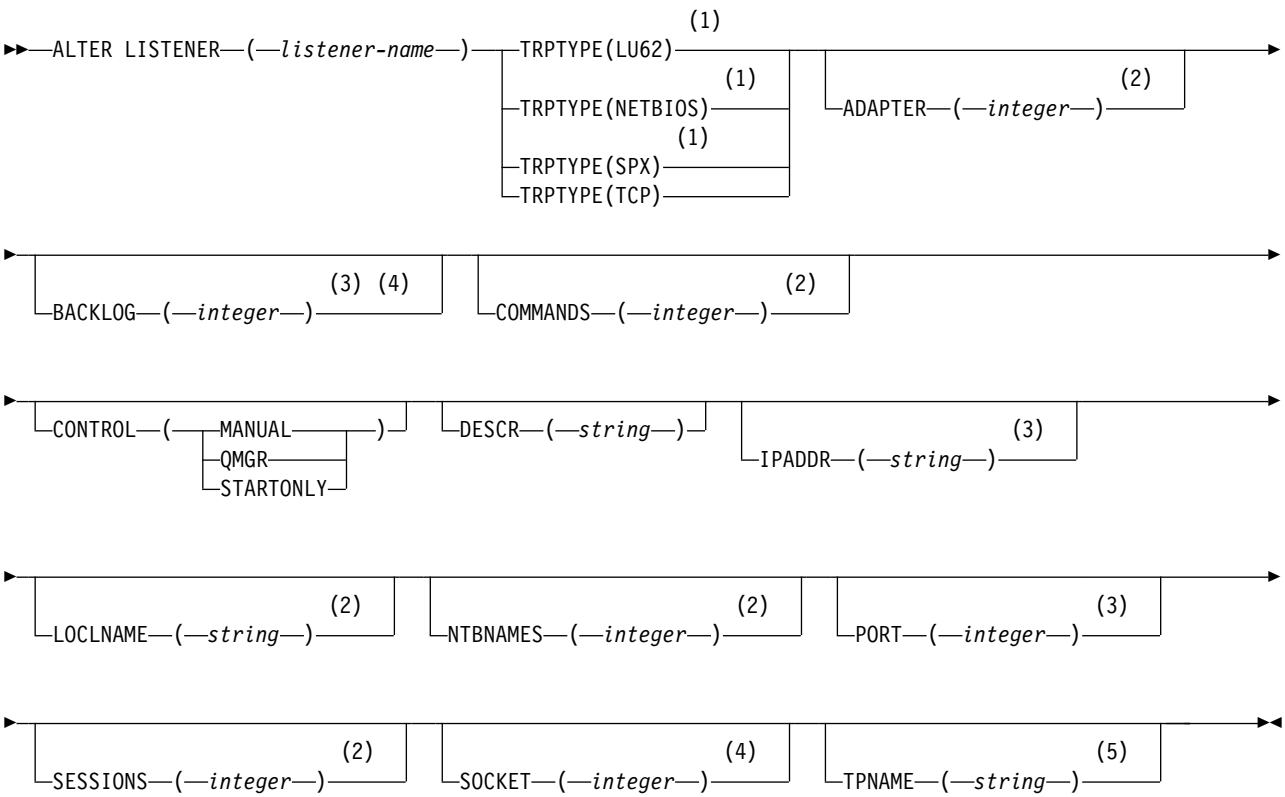
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

Parameters not specified in the **ALTER LISTENER** command result in the existing values for those parameters being left unchanged.

- Syntax diagram
- "Parameter descriptions for ALTER LISTENER" on page 448

Synonym: ALT LSTR

ALTER LISTENER



Notes:

1 Valid only on Windows.

2 Valid only on Windows when TRPTYPE is NETBIOS.

3 Valid when TRPTYPE is TCP.

4 Valid on Windows when TRPTYPE is SPX.

5 Valid only on Windows when TRPTYPE is LU62.

Parameter descriptions for ALTER LISTENER

listener-name)
Name of the IBM MQ listener definition (see Rules for naming IBM MQ objects). This is required.

The name must not be the same as any other listener definition currently defined on this queue manager (unless REPLACE is specified).

Windows **ADAPTER(*integer*)**
The adapter number on which NetBIOS listens. This parameter is valid only on Windows when TRPTYPE is NETBIOS.

BACKLOG(*integer*)
The number of concurrent connection requests that the listener supports.

Windows **COMMANDS(*integer*)**
The number of commands that the listener can use. This parameter is valid only on Windows when **TRPTYPE** is NETBIOS.

CONTROL(*string*)
Specifies how the listener is to be started and stopped.:

MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by use of the **START LISTENER** and **STOP LISTENER** commands.

QMGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY
The listener is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

DESCR(*string*)
Plain-text comment. It provides descriptive information about the listener when an operator issues the **DISPLAY LISTENER** command (see “DISPLAY LISTENER on Multiplatforms” on page 839).

It should contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

IPADDR(*string*)
IP address for the listener specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric host name form. If you do not specify a value for this parameter, the listener listens on all configured IPv4 and IPv6 stacks.

LIKE(*listener-name*)
The name of a listener, with parameters that are used to model this definition.

This parameter applies only to the **DEFINE LISTENER** command.

If this field is not filled in, and you do not complete the parameter fields related to the command, the values are taken from the default definition for listeners on this queue manager. This is equivalent to specifying:

```
LIKE(SYSTEM.DEFAULT.LISTENER)
```

A default listener is provided but it can be altered by the installation of the default values required. See Rules for naming IBM MQ objects.

Windows **LOCLNAME(*string*)**
The NetBIOS local name that the listener uses. This parameter is valid only on Windows when **TRPTYPE** is NETBIOS.

Windows **NTBNAMES(*integer*)**
The number of names that the listener can use. This parameter is valid only on Windows when **TRPTYPE** is NETBIOS.

PORT(*integer*)
The port number for TCP/IP. This is valid only when TRPTYPE is TCP. It must not exceed 65535.

Windows **SESSIONS(*integer*)**
The number of sessions that the listener can use. This parameter is valid only on Windows when **TRPTYPE** is NETBIOS.

SOCKET(*integer*)
The SPX socket on which to listen. This is valid only if **TRPTYPE** is SPX.

Windows **TPNAME(*string*)**
The LU 6.2 transaction program name (maximum length 64 characters). This parameter is valid only on Windows when **TRPTYPE** is LU62.

TRPTYPE(*string*)
The transmission protocol to be used:

Windows **LU62**
SNA LU 6.2. This is valid only on Windows.

Windows **NETBIOS**
NetBIOS. This is valid only on Windows.

Windows **SPX**
Sequenced packet exchange. This is valid only on Windows.

TCP
TCP/IP.

ALTER NAMELIST:

Use the MQSC command **ALTER NAMELIST** to alter a list of names. This list is most commonly a list of cluster names or queue names.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

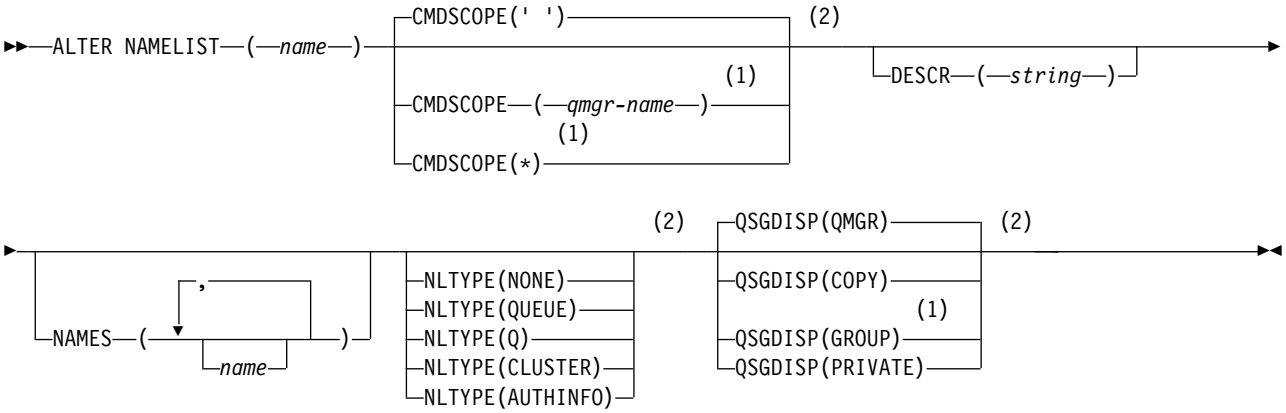
Parameters not specified in the **ALTER NAMELIST** command result in the existing values for those parameters being left unchanged.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes"
- "Parameter descriptions for ALTER NAMELIST"

Synonym: ALT NL

ALTER NAMELIST



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes

Successful completion of the command does not mean that the action completed. To check for true completion, see the ALTER NAMELIST step in Checking that async commands for distributed networks have finished.

Parameter descriptions for ALTER NAMELIST

(name)

Name of the list.

The name must not be the same as any other namelist name currently defined on this queue manager (unless **REPLACE** or **ALTER** is specified). See Rules for naming IBM MQ objects.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if **QSGDISP** is set to GROUP.

' ' The command runs on the queue manager on which it was entered.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of specifying * is the same as entering the command on every queue manager in the queue-sharing group.

DESCR(*string*)

Plain-text comment. It provides descriptive information about the namelist when an operator issues the **DISPLAY NAMELIST** command (see "DISPLAY NAMELIST" on page 848).

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

NAMES(*name, ...*)

List of names.

The names can be of any type, but must conform to the rules for naming IBM MQ objects, with a maximum length of 48 characters.

An empty list is valid: specify **NAMES()**. The maximum number of names in the list is 256.

z/OS **NLTYPE**

Indicates the type of names in the namelist.

This parameter is valid only on z/OS.

NONE
The names are of no particular type.

QUEUE or Q
A namelist that holds a list of queue names.

CLUSTER
A namelist that is associated with clustering, containing a list of the cluster names.

AUTHINFO
This namelist is associated with TLS and contains a list of authentication information object names.

Namelists used for clustering must have **NLTYPE(CLUSTER)** or **NLTYPE(NONE)**.

Namelists used for TLS must have **NLTYPE(AUTHINFO)**.

z/OS **QSGDISP**

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

QSGDISP	ALTER
COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters **QSGDISP(COPY)**. Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.
GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameters **QSGDISP(GROUP)**. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command. If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero: `DEFINE NAMELIST(name)` `REPLACE QSGDISP(COPY)` The **ALTER** for the group object takes effect regardless of whether the generated command with **QSGDISP(COPY)** fails.
PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with **QSGDISP(QMGR)** or **QSGDISP(COPY)**. Any object residing in the shared repository is unaffected.
QMGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters **QSGDISP(QMGR)**. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

ALTER PROCESS:

Use the MQSC command **ALTER PROCESS** to alter the parameters of an existing IBM MQ process definition.

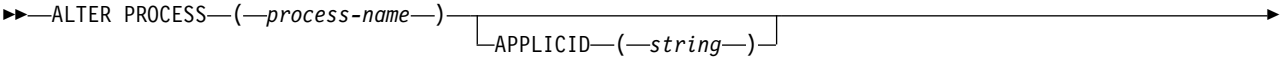
Using MQSC commands

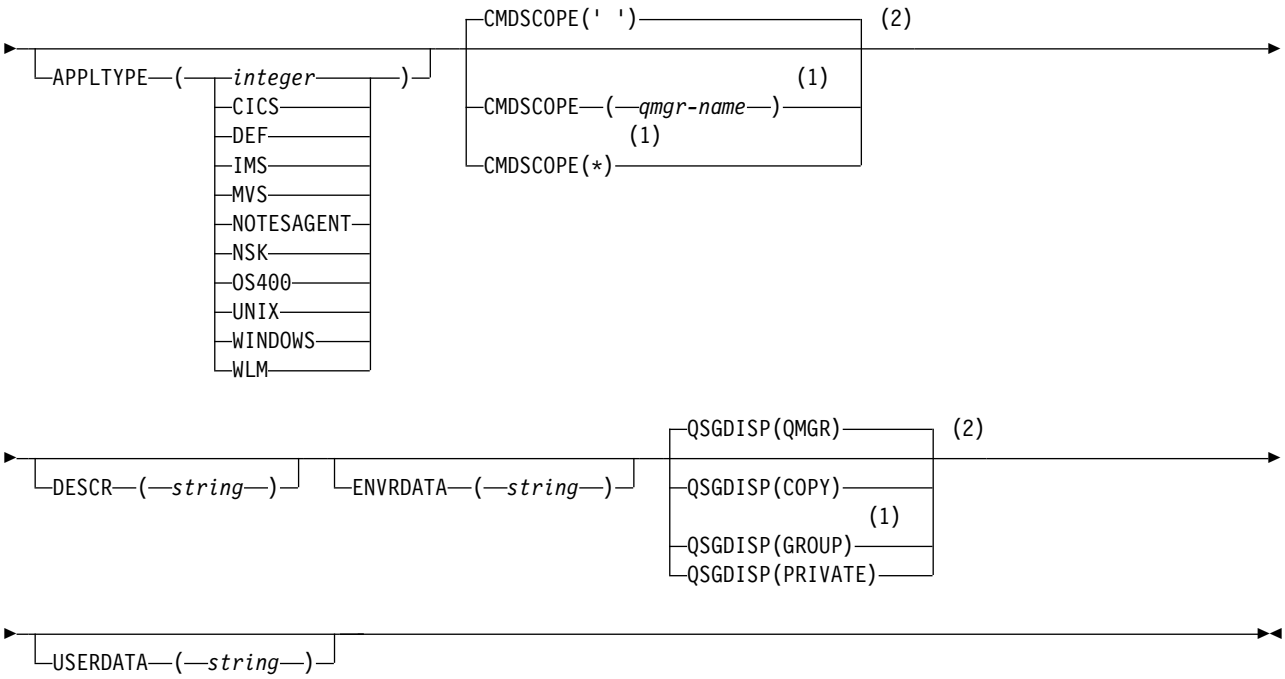
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources `2CR`. For an explanation of the source symbols, see Using commands on z/OS.

Synonym: `ALT PRO`

ALTER PROCESS





Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Parameter descriptions for ALTER PROCESS

process-name)
Name of the IBM MQ process definition (see Rules for naming IBM MQ objects). *process-name* is required.

The name must not be the same as any other process definition currently defined on this queue manager (unless **REPLACE** is specified).

APPLICID(*string*)
The name of the application to be started. The name might typically be a fully qualified file name of an executable object. Qualifying the file name is particularly important if you have multiple IBM MQ installations, to ensure the correct version of the application is run. The maximum length is 256 characters.

For a CICS application the name is a CICS transaction ID, and for an IMS™ application it is an IMS transaction ID.

z/OS On z/OS, for distributed queuing, it must be `"CSQX start"`.

APPLTYPE(*string*)
The type of application to be started. Valid application types are:

integer
A system-defined application type in the range zero through 65 535 or a user-defined application type in the range 65 536 through 999 999 999.

For certain values in the system range, a parameter from the following list can be specified instead of a numeric value:

CICS Represents a CICS transaction.

z/OS **IMS**
Represents an IMS transaction.

z/OS **MVS**
Represents a z/OS application (batch or TSO).

NOTESAGENT
Represents a Lotus Notes® agent.

NSS Client **NSK**
Represents an HP Integrity NonStop Server application.

IBM i **OS400**
Represents an IBM i application.

UNIX **UNIX**
Represents a UNIX application.

Windows **WINDOWS**
Represents a Windows application.

z/OS **WLM**
Represents a z/OS workload manager application.

DEF Specifying DEF causes the default application type for the platform at which the command is interpreted to be stored in the process definition. This default cannot be changed by the installation. If the platform supports clients, the default is interpreted as the default application type of the server.

Only use application types (other than user-defined types) that are supported on the platform at which the command runs:

- z/OS On z/OS: CICS, IMS, MVS, UNIX, WINDOWS, WLM, and DEF are supported
- IBM i On IBM i: OS400, CICS, and DEF are supported
- UNIX On UNIX: UNIX, WINDOWS, CICS, and DEF are supported
- Windows On Windows, WINDOWS, UNIX, CICS, and DEF are supported

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if **QSGDISP** is set to GROUP.

' ' The command runs on the queue manager on which it was entered.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

In a shared queue environment, you can provide a different queue manager name from the one you are using to enter the command. The command server must be enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect is the same as entering the command on every queue manager in the queue-sharing group.

DESCR(*string*)
Plain-text comment. It provides descriptive information about the object when an operator issues the **DISPLAY PROCESS** command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: Use characters from the coded character set identifier (CCSID) for this queue manager. Other characters might be translated incorrectly if the information is sent to another queue manager.

ENVRDATA(*string*)

A character string that contains environment information pertaining to the application to be started. The maximum length is 128 characters.

The meaning of **ENVRDATA** is determined by the trigger-monitor application. The trigger monitor provided by IBM MQ appends **ENVRDATA** to the parameter list passed to the started application. The parameter list consists of the MQTMC2 structure, followed by one blank, followed by **ENVRDATA** with trailing blanks removed.

Note:

1. z/OS On z/OS, **ENVRDATA** is not used by the trigger-monitor applications provided by IBM MQ.
2. z/OS On z/OS, if **APPLTYPE** is `WLM`, the default values for the ServiceName and ServiceStep fields in the work information header (MQWIH) can be supplied in **ENVRDATA**. The format must be:

   ```
   SERVICENAME=servname,SERVICESTEP=stepname
   ```

   where:

   SERVICENAME=
   is the first 12 characters of **ENVRDATA**.

   servname
   is a 32-character service name. It can contain embedded blanks or any other data, and have trailing blanks. It is copied to the MQWIH as is.

   SERVICESTEP=
   is the next 13 characters of **ENVRDATA**.

   stepname
   is a 1 - 8 character service step name. It is copied as-is to the MQWIH, and padded to eight characters with blanks.

   If the format is incorrect, the fields in the MQWIH are set to blanks.
3. UNIX On UNIX, **ENVRDATA** can be set to the ampersand character to make the started application run in the background.

z/OS **QSGDISP**

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

QSGDISP	ALTER
COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters **QSGDISP(COPY)**. Any object residing in the shared repository, or any object defined using a command that had the parameters **QSGDISP(QMGR)**, is not affected by this command.
GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameters **QSGDISP(GROUP)**. On the page set of the queue manager that executes the command, only a local copy of the object is altered by this command. If the command is successful, the following command is generated. `DEFINE PROCESS(process-name)` `REPLACE QSGDISP(COPY)` The command is sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero. The **ALTER** for the group object takes effect regardless of whether the generated command with **QSGDISP(COPY)** fails.
PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with **QSGDISP(QMGR)** or **QSGDISP(COPY)**. Any object residing in the shared repository is unaffected.
QMGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters **QSGDISP(QMGR)**. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

USERDATA(*string*)

A character string that contains user information pertaining to the application defined in the **APPLICID** that is to be started. The maximum length is 128 characters.

The meaning of **USERDATA** is determined by the trigger-monitor application. The trigger monitor provided by IBM MQ simply passes **USERDATA** to the started application as part of the parameter list. The parameter list consists of the MQTMC2 structure (containing **USERDATA**), followed by one blank, followed by **ENVRDATA** with trailing blanks removed.

For IBM MQ message channel agents, the format of this field is a channel name of up to 20 characters. See Managing objects for triggering for information about what **APPLICID** to provide to message channel agents.

Windows For Microsoft Windows, the character string must not contain double quotation marks if the process definition is going to be passed to **runmqtrm**.

ALTER PSID on z/OS: z/OS

Use the MQSC command **ALTER PSID** to change the expansion method for a page set.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

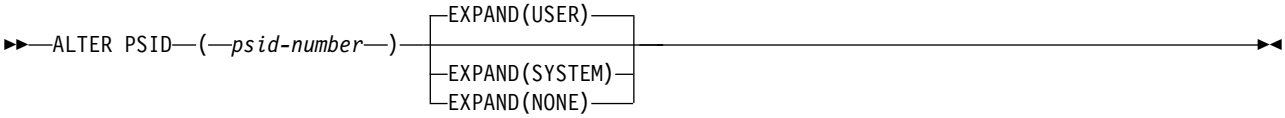
Parameters not specified in the **ALTER PSID** command result in the existing values for those parameters being left unchanged.

You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for ALTER PSID"

Synonym: ALT PSID

ALTER PSID



Parameter descriptions for ALTER PSID

(psid-number)
Identifier of the page set. This is required.

EXPAND
Controls how the queue manager should expand a page set when it becomes nearly full, and further pages are required in it.

USER The secondary extent size that was specified when the page set was defined is used. If no secondary extent size was specified, or if it was specified as zero, then no dynamic page set expansion can take place.

At restart, if a previously used page set has been replaced with a data set that is smaller, it is expanded until it reaches the size of the previously used data set. Only one extent is required to reach this size.

SYSTEM

A secondary extent size that is approximately 10 per cent of the current size of the page set is used. It might be rounded up depending on the characteristics of the DASD.

The secondary extent size that was specified when the page set was defined is ignored; dynamic expansion can occur if it was zero or not specified.

NONE
No further page set expansion is to take place.

Usage note

You can use **ALTER PSID** to reset an internal IBM MQ indicator that prevents the pageset from being expanded; for example, after the data set has been **ALTER**ed to **ADDVOLUMES**.

In this instance, although the **EXPAND** keyword must be specified with a value, you do not have to change the value from that already configured. For example, if **DISPLAY USAGE** shows pageset 3 configured with **EXPAND(SYSTEM)**, you issue the following command to allow IBM MQ to retry pageset expansion:

```
ALTER PSID(3) EXPAND(SYSTEM)
```

Related reference:

"DISPLAY USAGE on z/OS" on page 960
Use the MQSC command DISPLAY USAGE to display information about the current state of a page set, to display information about the log data sets, or to display information about the shared message data sets.

ALTER QMGR:

Use the MQSC command **ALTER QMGR** to alter the queue manager parameters for the local queue manager.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

Parameters not specified in the **ALTER QMGR** command result in the existing values for those parameters being left unchanged.

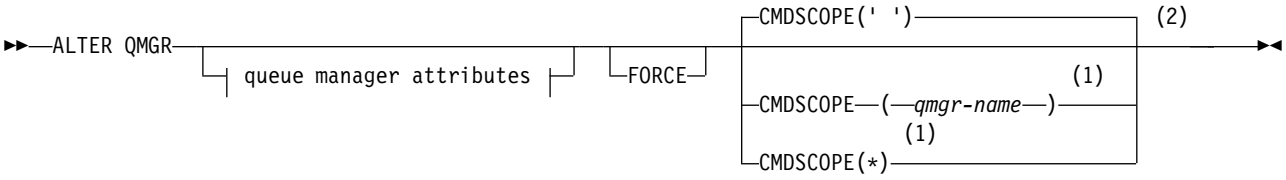
z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

This information is divided into three sections:

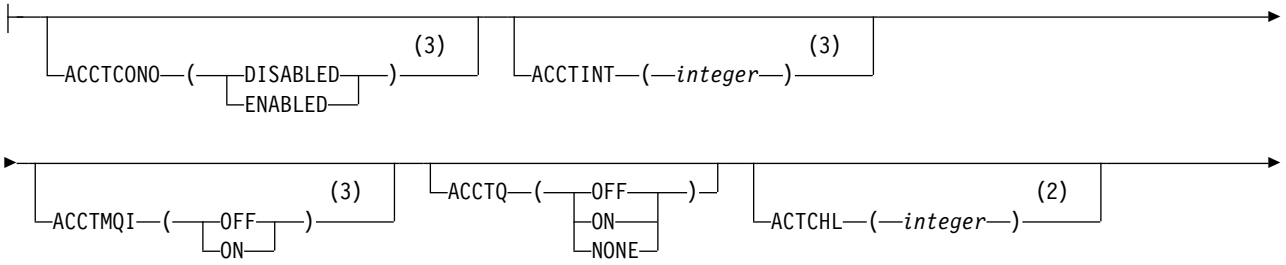
- "ALTER QMGR"
- "Parameter descriptions for ALTER QMGR" on page 462
- "Queue manager parameters" on page 463

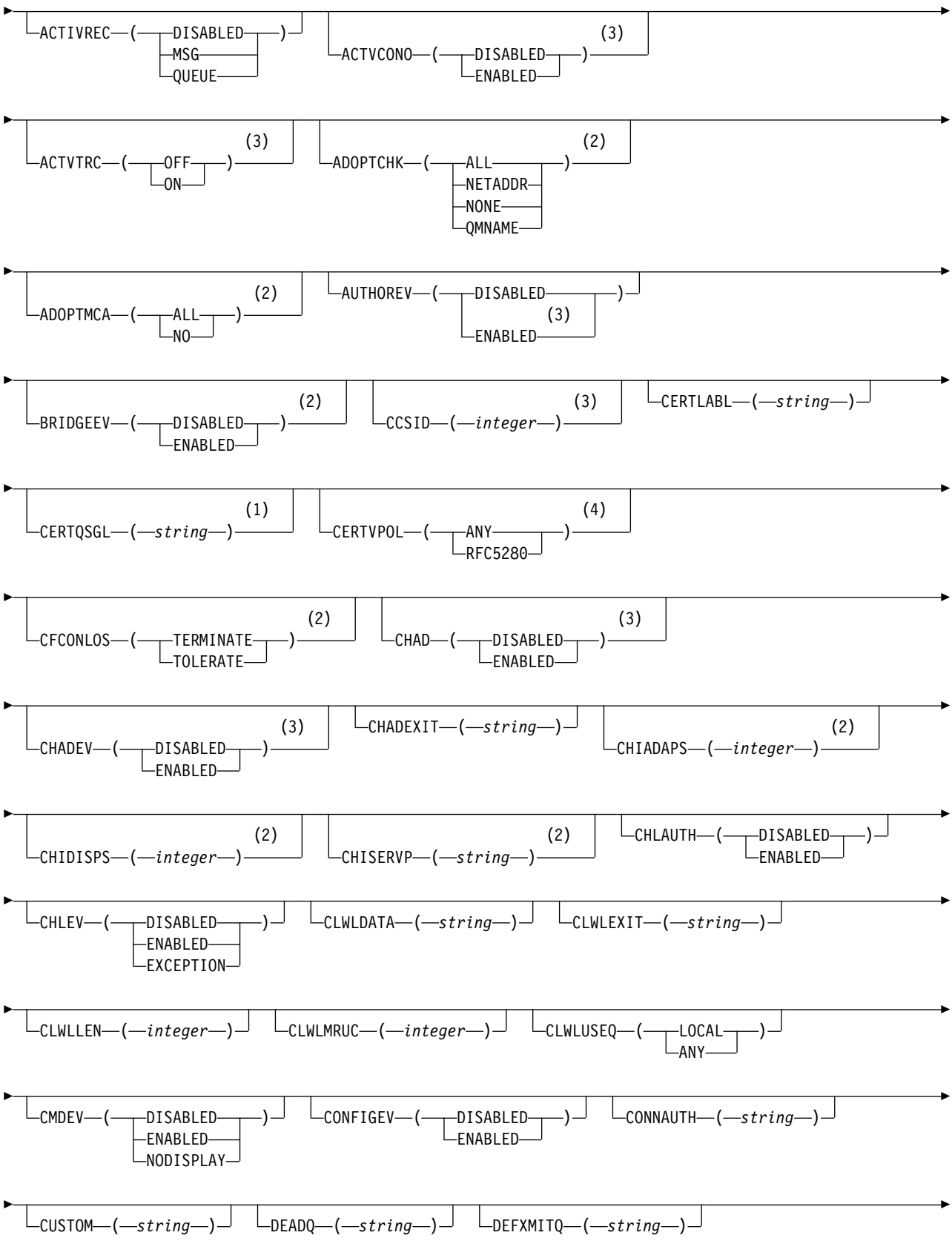
ALTER QMGR

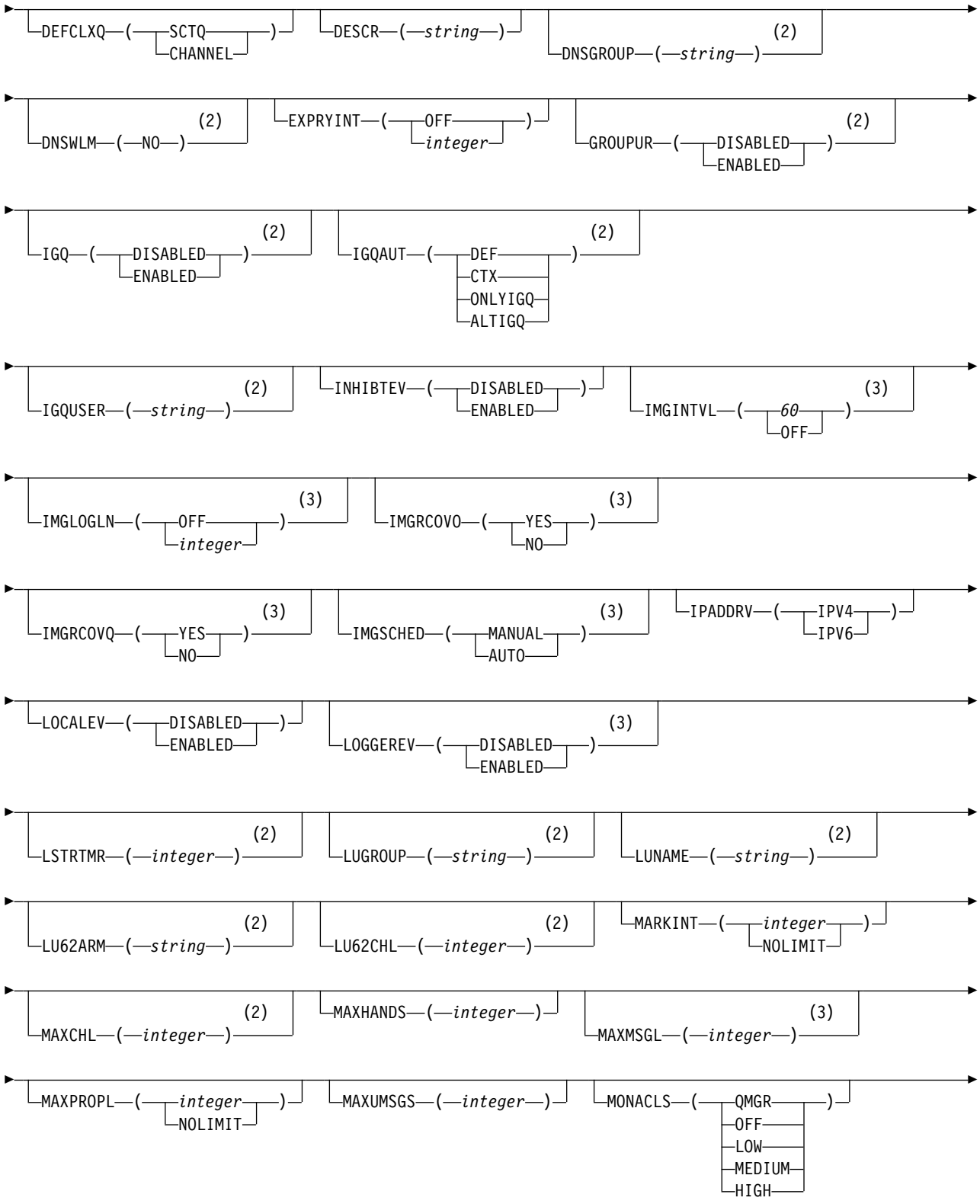
Synonym: ALT QMGR

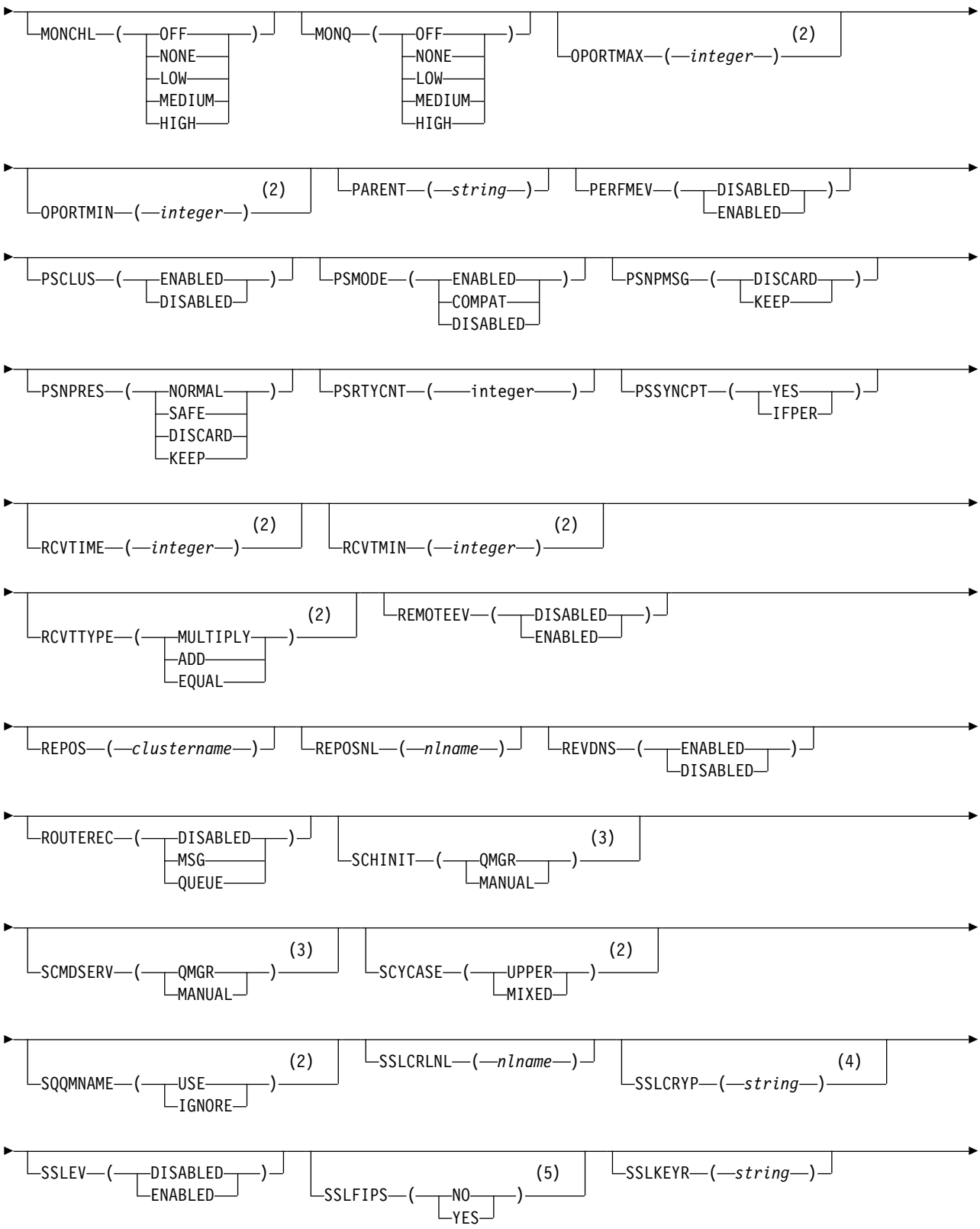


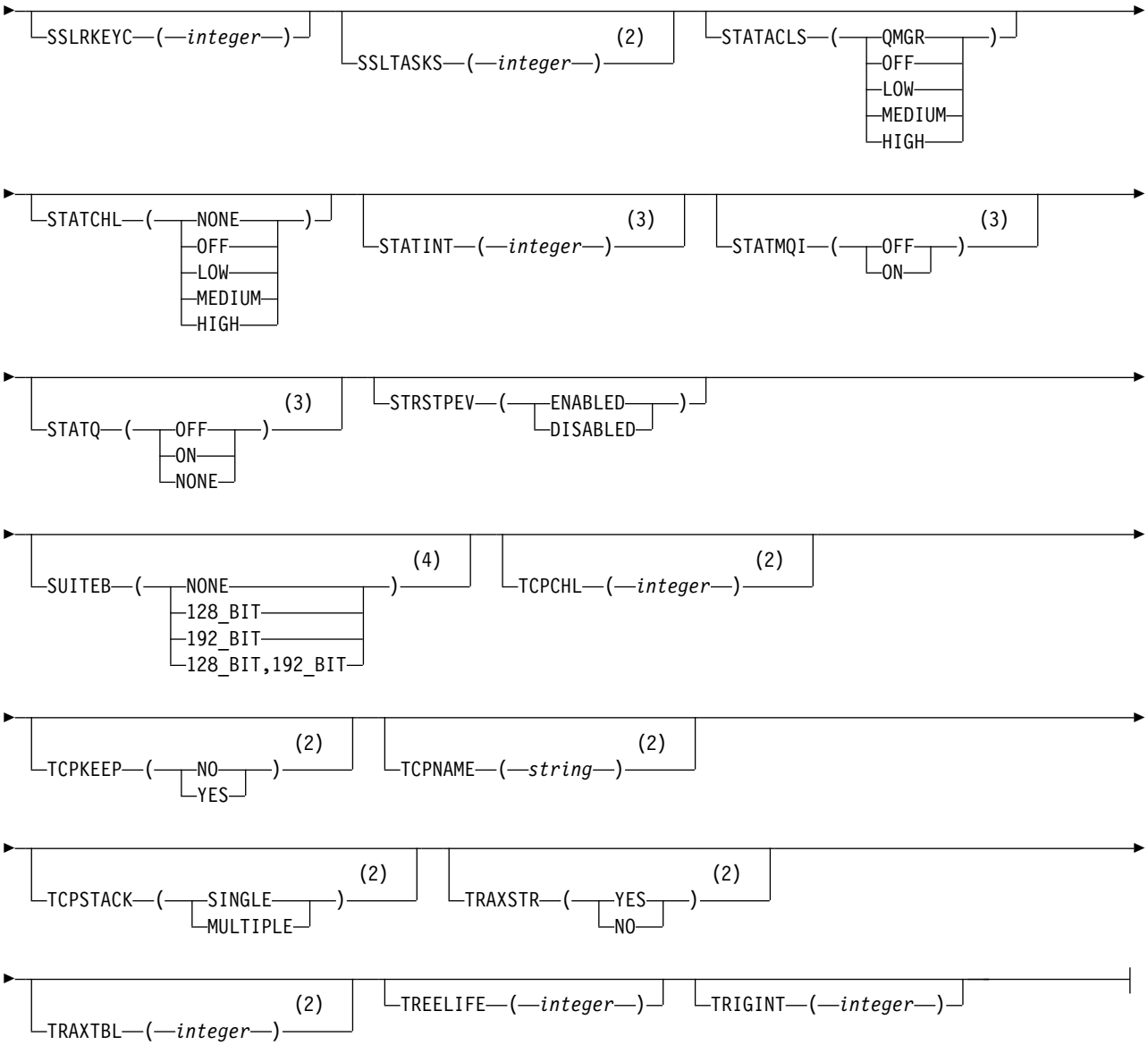
Queue manager attributes:




ACTIVREC
DISABLED
MSG
QUEUE
ACTVCONO
(3)
DISABLED
ENABLED
ACTVTRC
(3)
OFF
ON
ADOPTCHK
(2)
ALL
NETADDR
NONE
QMNAME
ADOPTMCA
(2)
ALL
NO
AUTHOREV
DISABLED
(3)
ENABLED
BRIDGEEV
(2)
DISABLED
ENABLED
CCSID
(3)
integer
CERTLABL
string
CERTQSGL
(1)
string
CERTVPOL
(4)
ANY
RFC5280
CFCONLOS
(2)
TERMINATE
TOLERATE
CHAD
(3)
DISABLED
ENABLED
CHADEV
(3)
DISABLED
ENABLED
CHADEXIT
string
CHIADAPS
(2)
integer
CHIDISPS
(2)
integer
CHISERVP
(2)
string
CHLAUTH
DISABLED
ENABLED
CHLEV
DISABLED
ENABLED
EXCEPTION
CLWLDATA
string
CLWLEXIT
string
CLWLLEN
integer
CLWLMRUC
integer
CLWLUSEQ
LOCAL
ANY
CMDEV
DISABLED
ENABLED
NODISPLAY
CONFIGEV
DISABLED
ENABLED
CONNAUTH
string
CUSTOM
string
DEADQ
string
DEFXMITQ
string


DEFCLXQ
SCTQ
CHANNEL
DESCR
string
DNSGROUP
(2)
DNSWLM
NO
(2)
EXPRYINT
OFF
integer
GROUPUR
DISABLED
ENABLED
(2)
IGQ
DISABLED
ENABLED
(2)
IGQAUT
DEF
CTX
ONLYIGQ
ALTIGQ
(2)
IGQUSER
string
(2)
INHIBTEV
DISABLED
ENABLED
IMGINTVL
60
OFF
(3)
IMGLOGLN
OFF
integer
(3)
IMGRCOVO
YES
NO
(3)
IMGRCOVQ
YES
NO
(3)
IMGSCHED
MANUAL
AUTO
(3)
IPADDRV
IPV4
IPV6
LOCALEV
DISABLED
ENABLED
LOGGEREV
DISABLED
ENABLED
(3)
LSTRTMR
integer
(2)
LUGROUP
string
(2)
LUNAME
string
(2)
LU62ARM
string
(2)
LU62CHL
integer
(2)
MARKINT
integer
NOLIMIT
MAXCHL
integer
(2)
MAXHANDS
integer
MAXMSGL
integer
(3)
MAXPROPL
integer
NOLIMIT
MAXUMSGS
integer
MONACLS
QMGR
OFF
LOW
MEDIUM
HIGH


MONCHL—(—OFF—)
NONE
LOW
MEDIUM
HIGH
MONQ—(—OFF—)
NONE
LOW
MEDIUM
HIGH
OPORTMAX—(—integer—)
(2)
OPORTMIN—(—integer—)
(2)
PARENT—(—string—)
PERFMEV—(—DISABLED—)
ENABLED
PSCLUS—(—ENABLED—)
DISABLED
PSMODE—(—ENABLED—)
COMPAT
DISABLED
PSNPMSG—(—DISCARD—)
KEEP
PSNPRES—(—NORMAL—)
SAFE
DISCARD
KEEP
PSRTYCNT—(—integer—)
PSSYNCPT—(—YES—)
IFPER
RCVTIME—(—integer—)
(2)
RCVTMIN—(—integer—)
(2)
RCVTTYPE—(—MULTIPLY—)
(2)
ADD
EQUAL
REMOTEEV—(—DISABLED—)
ENABLED
REPOS—(—clustername—)
REPOSNL—(—nlname—)
REVDNS—(—ENABLED—)
DISABLED
ROUTEREC—(—DISABLED—)
MSG
QUEUE
SCHINIT—(—QMGR—)
(3)
MANUAL
SCMDSERV—(—QMGR—)
(3)
MANUAL
SCYCASE—(—UPPER—)
(2)
MIXED
SQQMNAME—(—USE—)
(2)
IGNORE
SSLCRLNL—(—nlname—)
SSLCRYP—(—string—)
(4)
SSLEV—(—DISABLED—)
ENABLED
SSLFIPS—(—NO—)
(5)
YES
SSLKEYR—(—string—)



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Not valid on z/OS.

4 Valid only on UNIX, Linux, and Windows.

5 Not valid on IBM i.

Parameter descriptions for ALTER QMGR

The parameters you specify override the current values. Attributes that you do not specify are unchanged.

Note:

1. If you do not specify any parameters, the command completes successfully, but no queue manager options are changed.

2. Changes made using this command persist when the queue manager is stopped and restarted.

FORCE

Specify this parameter to force completion of the command if both of the following statements are true:

- The **DEFXMITQ** parameter is specified
- An application has a remote queue open, the resolution for which would be affected by this change

If **FORCE** is not specified in these circumstances, the command is unsuccessful.

Queue manager parameters

These parameters are the queue manager parameters for the **ALTER QMGR** command:

Multi **ACCTCONO**

Specifies whether applications can override the settings of the **ACCTQ** and **ACCTMQI** queue manager parameters:

DISABLED

Applications cannot override the settings of the **ACCTQ** and **ACCTMQI** parameters.

This is the queue manager's initial default value.

ENABLED

Applications can override the settings of the **ACCTQ** and **ACCTMQI** parameters by using the options field of the MQCNO structure of the MQCONNX API call.

Changes to this parameter are effective for connections to the queue manager that occur after the change.

This parameter is valid only on Multiplatforms.

Multi **ACCTINT(*integer*)**

The time interval, in seconds, at which intermediate accounting records are written.

Specify a value in the range 1 through 604800.

Changes to this parameter are effective for connections to the queue manager that occur after the change.

This parameter is valid only on Multiplatforms.

Multi **ACCTMQI**

Specifies whether accounting information for MQI data is to be collected:

OFF MQI accounting data collection is disabled.

This is the queue manager's initial default value.

ON MQI accounting data collection is enabled.

If queue manager attribute **ACCTCONO** is set to ENABLED, the value of this parameter can be overridden using the options field of the MQCNO structure.

Changes to this parameter are effective for connections to the queue manager that occur after the change.

This parameter is valid only on Multiplatforms.

ACCTQ

Specifies whether accounting data is to be collected for all queues.

z/OS On z/OS, the data collected is class 3 accounting data (thread-level and queue-level accounting).

OFF Accounting data collection is disabled for all queues which specify QMGR as the value for their ACCTQ parameter.

ON Accounting data collection is enabled for all queues which specify QMGR as the value of their ACCTQ parameter.

z/OS On z/OS systems, you must switch on class 3 accounting by the START TRACE command.

NONE
Accounting data collection for all queues is disabled regardless of the value of the ACCTQ parameter of the queue.

Changes to this parameter are effective only for connections to the queue manager occurring after the change to the parameter.

z/OS **ACTCHL(*integer*)**
The maximum number of channels that can be *active* at any time, unless the value is reduced below the number of currently active channels.

Specify a value from 1 through 9999 that is not greater than the value of MAXCHL. MAXCHL defines the maximum number of channels available.

If you change this value, you must also review the MAXCHL, LU62CHL, and TCPCHL values to ensure that there is no conflict of values

For an explanation of which channel states are considered active; see Channel states.

If the value of ACTCHL is reduced to less than its value when the channel initiator was initialized, channels continue to run until they stop. When the number of running channels falls below the value of ACTCHL, more channels can be started. Increasing the value of ACTCHL to more than its value when the channel initiator was initialized does not have immediate effect. The higher value of ACTCHL takes effect at the next channel initiator restart.

Sharing conversations do not contribute to the total for this parameter.

This parameter is valid only on z/OS.

ACTIVREC
Specifies whether activity reports are generated if requested in the message:

DISABLED
Activity reports are not generated.

MSG Activity reports are generated and sent to the reply queue specified by the originator in the message causing the report.

This is the queue manager's initial default value.

QUEUE
Activity reports are generated and sent to SYSTEM.ADMIN.ACTIVITY.QUEUE

See Activity recording.

Multi **ACTVCONO**
Specifies whether applications can override the settings of the **ACTVTRC** queue manager parameter:

DISABLED
Applications cannot override the settings of the **ACTVTRC** queue manager parameter.

This is the queue manager's initial default value.

ENABLED
Applications can override the settings of the **ACTVTRC** queue manager parameter by using the options field of the MQCNO structure of the MQCONNX API call.

Changes to this parameter are effective for connections to the queue manager that occur after the change.

This parameter is valid only on Multiplatforms.

Multi **ACTVTRC**
Specifies whether MQI application activity tracing information is to be collected. See Setting ACTVTRC to control collection of activity trace information.

OFF IBM MQ MQI application activity tracing information collection is not enabled.

This is the queue manager's initial default value.

ON IBM MQ MQI application activity tracing information collection is enabled.

If the queue manager attribute **ACTVCONO** is set to ENABLED, the value of this parameter can be overridden using the options field of the MQCNO structure.

Changes to this parameter are effective for connections to the queue manager that occur after the change.

This parameter is valid only on Multiplatforms.

z/OS **ADOPTCHK**
Specifies which elements are checked to determine whether an MCA is adopted. The check is made when a new inbound channel is detected with the same name as an already active MCA.

ALL Check the queue manager name and the network address. Perform this check to prevent your channels from being inadvertently or maliciously shut down.

This is the queue manager's initial default value.

NETADDR
Check the network address.

NONE
Do no checking.

QMNAME
Check the queue manager name.

Changes to this parameter take effect the next time that a channel attempts to adopt an MCA.

This parameter is valid only on z/OS.

z/OS **ADOPTMCA**
Specifies whether an orphaned instance of an MCA restarts immediately when a new inbound channel request matching the **ADOPTCHK** parameter is detected:

ALL Adopt all channel types.

This is the queue manager's initial default value.

NO Adoption of orphaned channels is not required.

Changes to this parameter take effect the next time that a channel attempts to adopt an MCA.

This parameter is valid only on z/OS.

AUTHOREV
Specifies whether authorization (Not Authorized) events are generated:

DISABLED
Authorization events are not generated.

This is the queue manager's initial default value.

ENABLED
Authorization events are generated.

z/OS This value is not supported on z/OS.

z/OS **BRIDGEEV**
Specifies whether IMS bridge events are generated.

DISABLED
IMS bridge events are not generated.

This is the queue manager's initial default value.

ENABLED
All IMS bridge events are generated.

This parameter is valid only on z/OS.

Multi **CCSID(*integer*)**
The coded character set identifier for the queue manager. The CCSID is the identifier used with all character string fields defined by the API. If the CCSID in the message descriptor is set to the value `MQCCSI_Q_MGR`, the value applies to application data in the body of a message. The value is set when the message is put to a queue.

Specify a value in the range 1 through 65535. The CCSID specifies a value that is defined for use on your platform, and use a character set that is appropriate to the platform.

If you use this parameter to change the CCSID, applications that are running when the change is applied continue to use the original CCSID. Therefore, stop and restart all running applications before you continue including the command server and channel programs. To stop and restart all running applications, stop and restart the queue manager after changing the parameter value.

This parameter is valid only on Multiplatforms. See Code page conversion for details of the supported CCSIDs for each platform.

z/OS To carry out the equivalent tasks on z/OS, use CSQ6SYSP to set your system parameters.

CERTLABL
Certificate label for this queue manager to use. The label identifies which personal certificate in the key repository has been selected.

The default and migrated queue manager values are:

- ULW On UNIX, Linux, and Windows: *ibmwebspheremqxxxx* where *xxxx* is the queue manager name folded to lower case.
- IBM i On IBM i:
  - If you specified SSLKEYR(*SYSTEM), the value is blank.

    Note that it is forbidden to use a nonblank queue manager CERTLABL with SSLKEYR(*SYSTEM). Attempting to do so results in an MQRCCF_Q_MGR_ATTR_CONFLICT error.
  - Otherwise, *ibmwebspheremqxxxx* where *xxxx* is the queue manager name folded to lower case.
- z/OS On z/OS: *ibmWebSphereMQXXXX* where *XXXX* is the queue manager name.

You should specify the preceding values. However, leaving **CERTLABL** as a blank value on the queue manager is interpreted by the system to mean the default values specified.

Important: You must run a REFRESH SECURITY TYPE(SSL) command if you make any changes to **CERTLABL** on the queue manager. However, you do not need to run the REFRESH SECURITY TYPE(SSL) command if you make any changes to **CERTLABL** on a channel.

z/OS **CERTQSGL**

Queue-sharing group (QSG) certificate label.

This parameter takes precedence over **CERTLABL** in the event that the queue manager is a member of a QSG.

The default value for this parameter is *ibmWebSphereMQXXXX* where *XXXX* is the queue-sharing group name.

This parameter is valid only on z/OS.

Multi **CERTVPOL**

Specifies which TLS certificate validation policy is used to validate digital certificates received from remote partner systems. This attribute can be used to control how strictly the certificate chain validation conforms to industry security standards.

ANY Apply each of the certificate validation policies supported by the secure sockets library and accept the certificate chain if any of the policies considers the certificate chain valid. This setting can be used for maximum backwards compatibility with older digital certificates which do not comply with the modern certificate standards.

RFC5280
Apply only the RFC 5280 compliant certificate validation policy. This setting provides stricter validation than the ANY setting, but rejects some older digital certificates.

For more information about certificate validation policies, see Certificate validation policies in IBM MQ.

Changes to the parameter take effect only after a **REFRESH SECURITY TYPE(SSL)** command is issued.

This parameter is valid only on Multiplatforms.

z/OS **CFCONLOS**

Specifies the action to be taken when the queue manager loses connectivity to the administration structure, or any CF structure with **CFCONLOS** set to ASQMGR.

TERMINATE
The queue manager terminates when connectivity to CF structures is lost.

TOLERATE
The queue manager tolerates loss of connectivity to CF structures without terminating.

All queue managers in the queue-sharing group must be at command level 710 or greater and OPMODE set to NEWFUNC for **TOLERATE** to be selected.

This parameter is valid only on z/OS.

Multi **CHAD**

Specifies whether receiver and server-connection channels can be defined automatically:

DISABLED
Auto-definition is not used.

This is the queue manager's initial default value.

ENABLED
Auto-definition is used.

Cluster-sender channels can always be defined automatically, regardless of the setting of this parameter.

This parameter is valid only on Multiplatforms.

Multi **CHADEV**

Specifies whether channel auto-definition events are generated.

DISABLED

Auto-definition events are not generated.

This is the queue manager's initial default value.

ENABLED

Auto-definition events are generated.

This parameter is valid only on Multiplatforms.

CHADEXIT(*string*)

Auto-definition exit name.

If this name is nonblank, the exit is called when an inbound request for an undefined receiver, server-connection, or cluster-sender channel is received. It is also called when starting a cluster-receiver channel.

The format and maximum length of the name depends on the environment:

- UNIX Linux On UNIX and Linux, it is of the form *libraryname(functionname)*. The maximum length is 128 characters.
- Windows On Windows, it is of the form *dllname(functionname)* where *dllname* is specified without the suffix `.DLL`. The maximum length is 128 characters.
- IBM i On IBM i, it is of the form:

  ```
  progname libname
  ```

  where *program name* occupies the first 10 characters and *libname* the second 10 characters (both blank-padded to the right if necessary). The maximum length of the string is 20 characters.
- z/OS On z/OS, it is a load module name, the maximum length is eight characters.

z/OS On z/OS, the **CHADEXIT** parameter applies only to cluster-sender and cluster-receiver channels.

z/OS **CHIADAPS(*integer*)**

The number of channel initiator adapter subtasks to use for processing IBM MQ calls.

Specify a value in the range 0 - 9999. Suggested settings are:
- Test system: 8
- Production system: 30

Changes to this parameter take effect when the channel initiator is restarted.

For more information about the relationship between `CHIADAPS`, `CHIDISPS` and `MAXCHL`, see Task 18: Tailor the channel initiator parameters.

This parameter is valid only on z/OS.

z/OS **CHIDISPS (*integer*)**

The number of dispatchers to use in the channel initiator.

Specify a value in the range 1 through 9999. Suggested settings are:
- Test system: 5
- Production system: 20

Changes to this parameter take effect when the channel initiator is restarted.

For more information about the relationship between CHIADAPS, CHIDISPS and MAXCHL, see Task 18: Tailor the channel initiator parameters.

This parameter is valid only on z/OS.

z/OS **CHISERVP**
This parameter is reserved for IBM use only; it is not for general use.

This parameter is valid only on z/OS.

CHLAUTH
Specifies whether the rules defined by channel authentication records are used. CHLAUTH rules can still be set and displayed regardless of the value of this attribute.

Changes to this parameter take effect the next time that an inbound channel attempts to start. Channels that are currently started are unaffected by changes to this parameter.

DISABLED
Channel authentication records are not checked.

ENABLED
Channel authentication records are checked.

CHLEV
Specifies whether channel events are generated.

DISABLED
Channel events are not generated. This is the queue manager's initial default value.

ENABLED
All channel events are generated.

EXCEPTION
All exception channel events are generated.

CLWLDATA(*string*)
Cluster workload exit data. The maximum length of the string is 32 characters.

This string is passed to the cluster workload exit when it is called.

CLWLEXIT(*string*)
Cluster workload exit name.

If this name is nonblank, the exit is called when a message is put to a cluster queue. The format and maximum length of the name depends on the environment:

- UNIX Linux On UNIX, and Linux, it is of the form *libraryname(functionname)*. The maximum length is 128 characters.
- Windows On Windows, it is of the form *dllname(functionname)*, where *dllname* is specified without the suffix .DLL. The maximum length is 128 characters.
- z/OS On z/OS, it is a load module name. The maximum length is eight characters.
- IBM i On IBM i, it is of the form:

  ```
  progname libname
  ```

  where *program name* occupies the first 10 characters and *libname* the second 10 characters (both blank-padded to the right if necessary). The maximum length is 20 characters.

CLWLLEN(*integer*)
The maximum number of bytes of message data that is passed to the cluster workload exit.

Specify a value in the range:

- ULW 0 - 999,999,999 on UNIX, Linux, and Windows.

- IBM i 0 - 999,999,999 on IBM i.
- z/OS 0 - 100 MB on z/OS systems.

CLWLMRUC(*integer*)
The maximum number of most recently used outbound cluster channels.

Specify a value in the range 1 through 999,999,999.

See CLWLMRUC queue manager attribute.

CLWLUSEQ
The attribute applies to queues with the queue attribute **CLWLUSEQ** set to QMGR. It specifies the behavior of an MQPUT operation when the target queue has a local instance and at least one remote cluster instance. It does not apply if the MQPUT originates from a cluster channel.

Specify either:

LOCAL
The local queue is the only target for MQPUT operations.

This is the queue manager's initial default value.

ANY The queue manager treats the local queue as another instance of the cluster queue for the purposes of workload distribution.

See CLWLUSEQ queue manager attribute.

CMDEV
Specifies whether command events are generated:

DISABLED
Command events are not generated.

This is the queue manager's initial default value.

ENABLED
Command events are generated for all successful commands.

NODISPLAY
Command events are generated for all successful commands, other than DISPLAY commands.

z/OS **CMDSCOPE**
Specifies how the command is run when the queue manager is a member of a queue-sharing group.

' The command is run on the queue manager on which it was entered.

qmgr-name
The command is run on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a different queue manager. You can do so if you are using a queue-sharing group environment, and if the command server is enabled. You can then specify a different queue manager to the one on which the command is entered.

*

The command is run on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of entering this value is the same as entering the command on every queue manager in the queue-sharing group.

This parameter is valid only on z/OS.

CONFIGEV
Specifies whether configuration events are generated:

ENABLED
Configuration events are generated. After setting this value, issue `REFRESH QMGR TYPE(CONFIGEV)` commands for all objects to bring the queue manager configuration up to date.

DISABLED
Configuration events are not generated.

This is the queue manager's initial default value.

CONNAUTH
The name of an authentication information object that is used to provide the location of user ID and password authentication. If **CONNAUTH** is blank, no user ID and password checking is done by the queue manager. The maximum length of the string is `MQ_AUTH_INFO_NAME_LENGTH`.

Only authentication information objects with type IDPWOS or IDPWLDAP can be specified; other types result in an error message when:

- 

The OAM reads the configuration on Multiplatforms.
- 

The security component reads the configuration on z/OS.

Changes to this configuration, or the object to which it refers, take effect when a **REFRESH SECURITY TYPE(CONNAUTH)** command is issued.

If you leave **CONNAUTH** blank, and attempt to connect to a channel that has one of the following options set in the **CHCKCLNT** field, the connection fails:

- 

`REQDADM`
- 

`REQUIRED`

CUSTOM(*string*)
The custom attribute for new features.

This attribute is reserved for the configuration of new features before named attributes are introduced. It can contain the values of zero or more attributes as pairs of attribute name and value, separated by at least one space. The attribute name-value pairs have the form `NAME(VALUE`. Escape a single quotation mark with another single quotation mark.

No values are defined for **Custom**.

DEADQ(*string*)
The local name of a dead-letter queue (or undelivered-message queue) on which messages that cannot be routed to their correct destination are put.

The queue named must be a local queue; see Rules for naming IBM MQ objects.

DEFCLXQ
The **DFTCLXQ** attribute controls which transmission queue is selected by default by cluster-sender channels to get messages from, to send the messages to cluster-receiver channels.

SCTQ

All cluster-sender channels send messages from `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. The `correlID` of messages placed on the transmission queue identifies which cluster-sender channel the message is destined for.

`SCTQ` is set when a queue manager is defined. This behavior is implicit in versions of IBM WebSphere MQ, earlier than Version 7.5. In earlier versions, the queue manager attribute **DEFCLXQ** was not present.

CHANNEL
Each cluster-sender channel sends messages from a different transmission queue. Each transmission queue is created as a permanent dynamic queue from the model queue `SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE`.

If the queue manager attribute, **DEFCLXQ**, is set to CHANNEL,the default configuration is changed to cluster-sender channels being associated with individual cluster transmission queues. The transmission queues are permanent-dynamic queues created from the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE. Each transmission queue is associated with one cluster-sender channel. As one cluster-sender channel services a cluster transmission queue, the transmission queue contains messages for only one queue manager in one cluster. You can configure clusters so that each queue manager in a cluster contains only one cluster queue. In this case, the message traffic from a queue manager to each cluster queue is transferred separately from messages to other queues.

DEFXMITQ(*string*)

Local name of the default transmission queue on which messages destined for a remote queue manager are put. The default transmission queue is used if there is no other suitable transmission queue defined.

The cluster transmission queue must not be used as the default transmission queue of the queue manager.

The queue named must be a local transmission queue; see Rules for naming IBM MQ objects.

DESCR(*string*)

Plain-text comment. It provides descriptive information about the queue manager.

It contains only displayable characters. The maximum length of the string is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

If the characters in the descriptive information are in the coded character set identifier (CCSID) for this queue manager they are translated correctly. They are translated when the descriptive information is sent to another queue manager. If they are not in the CCSID for this queue manager, they might be translated incorrectly.

z/OS **DNSGROUP(*string*)**

This parameter is no longer used. See WLM/DNS no longer supported.

z/OS **DNSWLM**

This parameter is no longer used. See WLM/DNS no longer supported.

NO This value is the only value accepted.

z/OS **EXPRYINT**

Specifies how often queues are scanned to discard expired messages:

OFF Queues are not scanned. No internal expiry processing is performed.

integer

The approximate interval in seconds at which queues are scanned. Each time that the expiry interval is reached, the queue manager looks for candidate queues that are worth scanning to discard expired messages.

The queue manager maintains information about the expired messages on each queue, and therefore whether a scan for expired messages is worthwhile. So, only a selection of queues is scanned at any time.

The value must be in the range 1 through 99999999. The minimum scan interval used is 5 seconds, even if you specify a lower value.

You must set the same **EXPRYINT** value for all queue managers within a queue-sharing group that support this attribute. Shared queues are scanned by only one queue manager in a queue-sharing group. This queue manager is either the first queue manager to restart, or the first queue manager for which **EXPRYINT** is set.

Changes to **EXPRYINT** take effect when the current interval expires. Changes also take effect if the new interval is less than the unexpired portion of the current interval. In this case, a scan is scheduled and the new interval value takes immediate effect.

This parameter is valid only on z/OS.

z/OS **GROUPUR**

This parameter controls whether CICS and XA client applications can establish transactions with a GROUP unit of recovery disposition.

The property can be enabled only when the queue manager is a member of a queue-sharing group.

ENABLED
CICS and XA client applications can establish transactions with a group unit of recovery disposition by specifying a queue-sharing group name when they connect.

DISABLED
CICS and XA client applications must connect using a queue manager name.

This parameter is valid only on z/OS.

z/OS **IGQ**

Specifies whether intra-group queuing is used.

The **IGQ** parameter is valid only on z/OS when the queue manager is a member of a queue-sharing group.

ENABLED
Message transfer between queue managers within a queue-sharing group uses the shared transmission queue, SYSTEM.QSG.TRANSMIT.QUEUE.

DISABLED
Message transfer between queue managers within a queue-sharing group uses non-shared transmission queues and channels. Queue managers that are not part of a queue-sharing group also use this mechanism.

If intra-group queuing is enabled, but the intra-group queuing agent is stopped, use the following command to restart it:

```
ALTER QMGR IGQ(ENABLED)
```

This parameter is valid only on z/OS.

z/OS **IGQAUT**

Specifies the type of authority checking and, therefore, the user IDs, to be used by the IGQ agent (IGQA). This parameter establishes the authority to put messages to a destination queue.

The **IGQAUT** parameter is valid only on z/OS when the queue manager is a member of a queue-sharing group.

DEF Indicates that the default user ID is used to establish authority to put messages to a destination queue.

For a one user ID check, the default user ID is the user ID of a queue manager within the queue-sharing group. The default user ID is the user ID of the queue manager that put the messages to the SYSTEM.QSG.TRANSMIT.QUEUE. This user ID is referred to as the QSGSEND user ID.

For two user ID checks, the default second user ID is the IGQ user ID.

CTX Indicates that the user ID from a *UserIdentifier* field is used to establish authority to put messages to a destination queue. The user ID is the *UserIdentifier* field in the message descriptor of a message on the SYSTEM.QSG.TRANSMIT.QUEUE.

For one user ID check, the QSGSEND user ID is used.

For two user ID checks, the QSGSEND user ID, the IGQ user ID and the alternate user ID are used. The alternate user ID is taken from the *UserIdentifier* field in the message descriptor of a message on the SYSTEM.QSG.TRANSMIT.QUEUE. The alternate user ID is referred to as ALT.

ONLYIGQ
Indicates that only the IGQ user ID is used to establish authority to put messages to a destination queue.

For all ID checks, the IGQ user ID is used.

ALTIGQ
Indicates that the IGQ user ID and the ALT user ID are used to establish authority to put messages to a destination queue.

For one user ID check, the IGQ user ID is used.

For two user ID checks, the IGQ user ID and the ALT user ID are used.

This parameter is valid only on z/OS.

z/OS **IGQUSER**
Nominates a user ID to be used by the IGQ agent (IGQA) to establish authority to put messages to a destination queue. The user ID is referred to as the IGQ user ID.

This parameter is valid only on z/OS when the queue manager is a member of a queue-sharing group. Possible values are:

Blanks
Indicates that the user ID of the receiving queue manager within the queue-sharing group is used.

Specific user ID
Indicates that the user ID specified in the **IGQUSER** parameter of the receiving queue manager is used.

Note:

1. As the receiving queue manager has authority to all queues it can access, security checking might not be performed for this user ID type.
2. As the value of blanks has a special meaning, you cannot use IGQUSER to specify a real user ID of blanks.

This parameter is valid only on z/OS.

Multi V 9.0.2 **IMGINTVL**
The target frequency with which the queue manager automatically writes media images, in minutes since the previous media image for the object.

Possible values are:

1 - 999 999 999
The time in minutes at which the queue manager automatically writes media images.

The default value is 60 minutes.

OFF Automatic media images are not written on a time interval basis.

This parameter is valid only on Multiplatforms.

Multi V 9.0.2 **IMGLOGLN**
The target size of recovery log, written before the queue manager automatically writes media images, in number of megabytes since the previous media image for the object. This limits the amount of log to be read when recovering an object.

Possible values are:

1 - 999 999 999
The target size of the recovery log in megabytes.

OFF Automatic media images are not written based on the size of log written.

OFF is the default value.

This parameter is valid only on Multiplatforms.

Multi V 9.0.2 **IMGRCOVO**

Specifies whether authentication information, channel, client connection, listener, namelist, process, alias queue, remote queue, and service objects are recoverable from a media image, if linear logging is being used.

Possible values are:

NO The "rcdmqimg (record media image)" on page 283 and"rcrmqobj (re-create object)" on page 289 commands are not permitted for these objects, and automatic media images, if enabled, are not written for these objects.

YES These objects are recoverable.

YES is the default value.

This parameter is valid only on Multiplatforms.

Multi V 9.0.2 **IMGRCOVQ**

Specifies the default **IMGRCOVQ** attribute for local and permanent dynamic queue objects, when used with this parameter.

Possible values are:

NO The **IMGRCOVQ** attribute for local and permanent dynamic queue objects is set to NO.

YES The **IMGRCOVQ** attribute for local and permanent dynamic queue objects is set to YES.

YES is the default value.

This parameter is valid only on Multiplatforms.

Multi V 9.0.2 **IMGSCHED**

Whether the queue manager automatically writes media images.

Possible values are:

AUTO
The queue manager attempts to automatically write a media image for an object, before **IMGINTVL** minutes have elapsed, or **IMGLOGLN** megabytes of recovery log have been written, since the previous media image for the object was taken.

The previous media image might have been taken manually or automatically, depending on the settings of **IMGINTVL** or **IMGLOGLN**.

MANUAL
Automatic media images are not written.

MANUAL is the default value.

This parameter is valid only on Multiplatforms.

INHIBTEV
Specifies whether inhibit events are generated. The events are generated for Inhibit Get and Inhibit Put)

ENABLED
Inhibit events are generated.

DISABLED
Inhibit events are not generated.

This is the queue manager's initial default value.

IPADDRV
Specifies which IP protocol is to be used for channel connections.

IPV4 The IPv4 IP address is to be used.

This is the queue manager's initial default value.

IPV6 The IPv6 IP address is to be used.

This parameter is used only in systems running IPv4 and IPv6. It applies to channels defined only with a **TRPTYPE** of TCP when either of the following two conditions is true:

- The **CONNAME** parameter of the channel contains a host name that resolves to both an IPv4 and an IPv6 address, and the **LOCLADDR** parameter is not specified.
- The value of the **CONNAME** and **LOCLADDR** parameters of the channel is a host name that resolves to both an IPv4 and IPv6 address.

LOCALEV
Specifies whether local error events are generated:

ENABLED
Local error events are generated.

DISABLED
Local error events are not generated.

This is the queue manager's initial default value.

Multi **LOGGEREV**
Specifies whether recovery log events are generated:

DISABLED
Logger events are not generated.

This is the queue manager's initial default value.

ENABLED
Logger events are generated.

This parameter is valid only on Multiplatforms.

z/OS **LSTRTMR(*integer*)**
The time interval, in seconds, between attempts by IBM MQ to restart a listener after an APPC or TCP/IP failure. When the listener is restarted on TCP/IP, it uses the same port and IP address as it used when it first started.

Specify a value in the range 5 through 9999.

Changes to this parameter take effect for listeners that are later started. Listeners that are currently started are unaffected by changes to this parameter.

This parameter is valid only on z/OS.

z/OS **LUGROUP(*string*)**
The generic LU name to be used by the LU 6.2 listener that handles inbound transmissions for the queue-sharing group. The maximum length of this parameter is eight characters.

If this name is blank, the listener cannot be used.

Changes to this parameter take effect for listeners that are later started. Listeners that are currently started are unaffected by changes to this parameter.

This parameter is valid only on z/OS.

z/OS **LUNAME(*string*)**
The name of the LU to use for outbound LU 6.2 transmissions. Set this parameter to be the same as the name of the LU to be used by the listener for inbound transmissions. The maximum length of this parameter is eight characters.

If this name is blank, the APPC/MVS default LU name is used. This name is variable, so LUNAME must always be set if you are using LU 6.2

Changes to this parameter take effect when the channel initiator is restarted.

This parameter is valid only on z/OS.

z/OS **LU62ARM(*string*)**
The suffix of the APPCPM member of SYS1.PARMLIB. This suffix nominates the LUADD for this channel initiator. When automatic restart manager (ARM) restarts the channel initiator, the z/OS command `SET APPC=` *xx* is issued.

If you do not provide a value for this parameter, no `SET APPC=` *xx* command is issued.

The maximum length of this parameter is two characters.

Changes to this parameter take effect when the channel initiator is restarted.

This parameter is valid only on z/OS.

z/OS **LU62CHL(*integer*)**
The maximum number of channels that can be current, or clients that can be connected, that use the LU 6.2 transmission protocol.

Specify a value 0- 9999 that is not greater than the value of `MAXCHL`. `MAXCHL` defines the maximum number of channels available. If you specify zero, the LU 6.2 transmission protocol is not used.

If you change this value, also review the `MAXCHL`, `LU62CHL`, and `ACTCHL` values. Ensure that there is no conflict of values and if necessary, raise the value of `MAXCHL` and `ACTCHL`.

If the value of this parameter is reduced, any current channels that exceed the new limit continue to run until they stop.

If the value of **`LU62CHL`** is non-zero when the channel initiator starts up, the value can be modified dynamically. If the value of **`LU62CHL`** is zero when the channel initiator starts up, a later ALTER command does not take effect. In this case, you should carry out an ALTER command, either before the channel initiator starts, or in CSQINP2 before you issue the **`START CHINIT`** command.

This parameter is valid only on z/OS.

MARKINT(*integer*)
The time interval, expressed in milliseconds, for which messages marked as browsed by a call to MQGET, with the get message option `MQGMO_MARK_BROWSE_CO_OP`, are expected to remain mark-browsed.

If messages are marked for more than approximately **`MARKINT`** milliseconds, the queue manager might automatically unmark messages. It might unmark messages that are marked as browsed for the cooperating set of handles.

This parameter does not affect the state of any message marked as browse by a call to MQGET with the get message option `MQGMO_MARK_BROWSE_HANDLE`.

Specify a value in the range 0 - 999,999,999.

The special value `NOLIMIT` indicates that the queue manager does not automatically unmark messages by this process.

z/OS **MAXCHL(*integer*)**
The maximum number of channels that can be *current* (including server-connection channels with connected clients).

Specify a value in the range 1- 9999. If you change this value, also review the **TCPCHL**, **LU62CHL**, and **ACTCHL** values to ensure that there is no conflict of values. If necessary, increase the number of active channels with the **ACTCHL** value. The values of **ACTCHL**, **LU62CHL**, and **TCPCHL** must not be greater than the maximum number of channels. Suggested settings are:
- Test system: 200
- Production system: 1000

For an explanation of which channel states are considered current; see Channel states.

If the value of this parameter is reduced, any current channels that exceed the new limit continue to run until they stop.

If the value of MAXCHL is reduced to less than its value when the channel initiator was initialized, channels continue to run until they stop. When the number of running channels falls below the value of MAXCHL, more channels can be started. Increasing the value of MAXCHL to more than its value when the channel initiator was initialized does not have immediate effect. The higher value of MAXCHL takes effect at the next channel initiator restart.

Sharing conversations do not contribute to the total for this parameter.

For more information about the relationship between **CHIADAPS**, **CHIDISPS**, and **MAXCHL**, see Task 18: Tailor the channel initiator parameters.

This parameter is valid only on z/OS.

MAXHANDS(*integer*)
The maximum number of open handles that any one connection can have at the same time.

This value is a value in the range 0 - 999,999,999.

MAXMSGL(*integer*)
The maximum length of messages allowed on queues for this queue manager.

This value is in the range 32 KB through 100 MB.

Ensure that you also consider the length of any message properties when deciding the value for the MAXMSGL parameter of a channel.

If you reduce the maximum message length for the queue manager, you must also reduce the maximum message length of the SYSTEM.DEFAULT.LOCAL.QUEUE definition. You must also reduce the maximum message length for all other queues defined on the queue manager. This change ensures that the limit of the queue manager is not less than the limit of any of the queues associated with it. If you do not change these lengths, and applications inquire only the **MAXMSGL** value of the queue, they might not work correctly.

Note that by adding the digital signature and key to the message, Advanced Message Security increases the length of the message.

MAXPROPL (*integer*)
The maximum length of property data in bytes that can be associated with a message.

This value is in the range 0 through 100 MB (104 857 600 bytes).

The special value NOLIMIT indicates that the size of the properties is not restricted, except by the upper limit.

MAXUMSGS(*integer*)
The maximum number of uncommitted messages within a sync point.

MAXUMSGS is a limit on the number of messages that can be retrieved, plus the number of messages that can be put, within any single sync point. The limit does not apply to messages that are put or retrieved outside sync point.

The number includes any trigger messages and report messages generated within the same unit of recovery.

If existing applications and queue manager processes are putting and getting a larger number of messages in sync point, reducing **MAXUMSGS** might cause problems.

z/OS An example of queue manager processes that might be affected is clustering on z/OS.

Specify a value in the range 1 through 999,999,999. The default value is 10000.

MAXUMSGS has no effect on MQ Telemetry. MQ Telemetry tries to batch requests to subscribe, unsubscribe, send, and receive messages from multiple clients into batches of work within a transaction.

MONACLS
Controls the collection of online monitoring data for auto-defined cluster-sender channels:

QMGR
Collection of online monitoring data is inherited from the setting of the **MONCHL** parameter of the queue manager.

This is the queue manager's initial default value.

OFF Monitoring for the channel is disabled.

LOW Unless **MONCHL** is NONE, monitoring is enabled with a low rate of data collection with a minimal effect on system performance. The data collected is not likely to be the most current.

MEDIUM
Unless **MONCHL** is NONE, monitoring is enabled with a moderate rate of data collection with limited effect on system performance.

HIGH Unless MONCHL is NONE, monitoring is enabled with a high rate of data collection with a likely effect on system performance. The data collected is the most current available.

A change to this parameter takes effect only on channels started after the change occurs. Any channel started before the change to the parameter continues with the value in force at the time that the channel started.

MONCHL
Controls the collection of online monitoring data for channels. The channels defined with **MONCHL(QMGR)** are affected by changing the QMGR **MONCHL** attribute.

OFF Online monitoring data collection is turned off for channels specifying a value of QMGR in their **MONCHL** parameter.

This is the queue manager's initial default value.

NONE
Online monitoring data collection is turned off for channels regardless of the setting of their **MONCHL** parameter.

LOW Online monitoring data collection is turned on, with a low ratio of data collection, for channels specifying a value of QMGR in their **MONCHL** parameter.

MEDIUM
Online monitoring data collection is turned on, with a moderate ratio of data collection, for channels specifying a value of QMGR in their **MONCHL** parameter.

HIGH Online monitoring data collection is turned on, with a high ratio of data collection, for channels specifying a value of QMGR in their **MONCHL** parameter.

A change to this parameter takes effect only on channels started after the change occurs. Any channel started before the change to the parameter continues with the value in force at the time that the channel started.

MONQ
Controls the collection of online monitoring data for queues.

OFF Online monitoring data collection is turned off for queues specifying a value of `QMGR` in their **MONQ** parameter.

This is the queue manager's initial default value.

NONE
Online monitoring data collection is turned off for queues regardless of the setting of their **MONQ** parameter.

LOW Online monitoring data collection is turned on for queues specifying a value of `QMGR` in their **MONQ** parameter.

MEDIUM
Online monitoring data collection is turned on for queues specifying a value of `QMGR` in their **MONQ** parameter.

HIGH Online monitoring data collection is turned on for queues specifying a value of `QMGR` in their **MONQ** parameter.

In contrast to **MONCHL**, there is no distinction between the values `LOW`, `MEDIUM`, and `HIGH`. These values all turn data collection on, but do not affect the rate of collection.

Changes to this parameter are effective only for queues opened after the parameter is changed.

z/OS **OPORTMAX(*integer*)**
The maximum value in the range of port numbers to be used when binding outgoing channels. When all the port numbers in the specified range are used, outgoing channels bind to any available port number.

Specify a value in the range 0 - 65535. A value of zero means that all outgoing channels bind to any available port number.

Specify a corresponding value for **OPORTMIN** to define a range of port numbers. Ensure that the value you specify for **OPORTMAX** is greater than or equal to the value you specify for **OPORTMIN**.

Changes to this parameter take effect for channels that are later started. Channels that are currently started are unaffected by changes to this parameter.

This parameter is valid only on z/OS.

z/OS **OPORTMIN(*integer*)**
The minimum value in the range of port numbers to be used when binding outgoing channels. When all the port numbers in the specified range are used, outgoing channels bind to any available port number.

Specify a value in the range 0 - 65535.

Specify a corresponding value for **OPORTMAX** to define a range of port numbers. Ensure that the value you specify for **OPORTMIN** is less than or equal to the value you specify for **OPORTMAX**.

Changes to this parameter take effect for channels that are later started. Channels that are currently started are unaffected by changes to this parameter.

This parameter is valid only on z/OS.

PARENT(*parentname*)
The name of the parent queue manager to which the local queue manager is to connect as its child in a hierarchy.

A blank value indicates that the queue manager has no parent queue manager.

If there is an existing parent queue manager it is disconnected.

IBM MQ hierarchical connections require that the queue manager attribute **PSMODE** is set to ENABLED.

The value of **PARENT** can be set to a blank value if **PSMODE** is set to DISABLED.

Before a queue manager can connect to a queue manager as its child in a hierarchy, channels must exist in both directions. The channels must exist between the parent queue manager and the child queue manager.

If a parent is already defined, the **ALTER QMGR PARENT** command disconnects from the original parent and sends a connection flow to the new parent queue manager.

Successful completion of the command does not mean that the action completed. To check that this command has completed, see the ALTER QMGR step in Checking that async commands for distributed networks have finished.

PERFMEV
Specifies whether performance-related events are generated:

ENABLED
Performance-related events are generated.

DISABLED
Performance-related events are not generated.

This is the queue manager's initial default value.

z/OS On IBM MQ for z/OS, all the queue managers in a queue-sharing group must have the same setting.

PSCLUS
Controls whether this queue manager participates in publish subscribe activity across any clusters in which it is a member. No clustered topic objects can exist in any cluster when modifying from ENABLED to DISABLED.

For more information about **PSCLUS**, see Inhibiting clustered publish/subscribe.

Note: To change a **PSCLUS** parameter status, the CHIN address space needs to be running.

ENABLED
This queue manager can define clustered topic objects, publish to subscribers on other queue managers, and register subscriptions that receive publications from other queue managers. All queue managers in the cluster running a version of IBM MQ that supports this option must specify **PSCLUS(ENABLED)** for the publish/subscribe activity to function as expected. ENABLED is the default value when a queue manager is created.

DISABLED
This queue manager cannot define clustered topic objects and ignores their definition on any other queue manager in the cluster.

Publications are not forwarded to subscribers elsewhere in the cluster, and subscriptions are not registered other than on the local queue manager.

To ensure that no publish/subscribe activity occurs in the cluster, all queue managers must specify **PSCLUS(DISABLED)**. As a minimum, full repositories must be consistent in enabling or disabling publish/subscribe participation.

PSMODE
Controls whether the publish/subscribe engine and the queued publish/subscribe interface are

running. It controls whether applications can publish or subscribe by using the application programming interface. It also controls whether the queues that are monitored by the queued publish/subscribe interface, are monitored.

Changing the **PSMODE** attribute can change the **PSMODE** status. Use one of the following commands to determine the current state of the publish/subscribe engine and the queued publish/subscribe interface:

- **DISPLAY PUBSUB**
- IBM i **DSPMQM** (on IBM i only)

COMPAT
The publish/subscribe engine is running. It is therefore possible to publish or subscribe by using the application programming interface.

The queued publish/subscribe interface is not running. Any publish/subscribe messages put to the queues that are monitored by the queued publish/subscribe interfaces are not acted upon.

Use this setting for compatibility with IBM Integration Bus (formerly known as WebSphere Message Broker) V6 or earlier versions that use this queue manager.

DISABLED
The publish/subscribe engine and the queued publish/subscribe interface are not running. It is therefore not possible to publish or subscribe by using the application programming interface. Any publish/subscribe messages put to the queues that are monitored by the queued publish/subscribe interfaces are not acted upon.

If a queue manager is in a publish/subscribe cluster or hierarchy, it might receive publish/subscribe messages from other queue managers in the cluster or hierarchy. Examples of such messages are publication messages or proxy subscriptions. While**PSMODE** is set to DISABLED those messages are not processed. For this reason, disable any queue manager in a publish/subscribe cluster or hierarchy only for as long as there is little build-up of messages.

ENABLED
The publish/subscribe engine and the queued publish/subscribe interface are running. It is therefore possible to publish or subscribe by using the application programming interface and the queues that are being monitored by the queued publish/subscribe interface.

This is the queue manager's initial default value.

Note: If a queue manager is in a publish/subscribe cluster or hierarchy, and you change **PSMODE** to ENABLED, you might have to run the command **REFRESH QMGR TYPE(PROXY)**. The command ensures that non-durable subscriptions are known across the cluster or hierarchy when **PSMODE** is set back to ENABLED. The circumstance in which you must run the command is as follows. If **PSMODE** is changed from ENABLED to DISABLED and back to ENABLED, and one or more non-durable subscriptions exist across all three stages.

PSNPMSG
If the queued publish/subscribe interface cannot process a non-persistent input message it might attempt to write the input message to the dead-letter queue. Whether it attempts to do so depends on the report options of the input message. The attempt to write the input message to the dead-letter queue might fail. In this case, the queued publish/subscribe interface might discard the input message. If MQRO_DISCARD_MSG is specified on the input message, the input message is discarded. If MQRO_DISCARD_MSG is not set, setting **PSNPMSG** to KEEP prevents the input message from being discarded. The default is to discard the input message.

Note: If you specify a value of IFPER for **PSSYNCPT**, you must not specify a value of KEEP for **PSNPMSG**.

DISCARD
Non-persistent input messages might be discarded if they cannot be processed.

KEEP Non-persistent input messages are not discarded if they cannot be processed. In this situation, the queued publish/subscribe interface continues to try to process this message again at appropriate intervals and does not continue processing subsequent messages.

PSNPRES
The **PSNPRES** attribute controls whether the queued publish/subscribe interface writes an undeliverable reply message to the dead-letter queue, or discards the message. The choice is necessary if the queued publish/subscribe interface cannot deliver a reply message to the reply-to queue.

For new queue managers, the initial value is NORMAL. If you specify a value of IFPER for **PSSYNCPT**, you must not specify a value of KEEP or SAFE for **PSNPRES**.

Multi For migrated queue managers on Multiplatforms, the value depends on DLQNonPersistentResponse and DiscardNonPersistentResponse.

NORMAL
Non-persistent responses which cannot be placed on the reply queue are put on the dead-letter queue. If they cannot be placed on the dead-letter queue then they are discarded.

SAFE Non-persistent responses which cannot be placed on the reply queue are put on the dead-letter queue. If the response cannot be sent and cannot be placed on the dead-letter queue, the queued publish/subscribe interface backs out of the current operation. It tries again at appropriate intervals, and does not continue processing subsequent messages.

DISCARD
Non-persistent responses which cannot be placed on the reply queue are discarded

KEEP

Non-persistent responses are not placed on the dead-letter queue or discarded. Instead the queued publish/subscribe interface backs out the current operation and then tries it again at appropriate intervals and does not continue processing subsequent messages.

PSRTYCNT
If the queued publish/subscribe interface fails to process a command message under sync point, the unit of work is backed out. The command tries to process the message a number of times again, before the publish/subscribe broker processes the command message according to its report options instead. This situation can arise for a number of reasons. For example, if a publish message cannot be delivered to a subscriber, and it is not possible to put the publication on the dead letter queue.

The initial value for this parameter on a new queue manager is 5.

Range is 0 - 999,999,999.

PSSYNCPT
Controls whether the queued publish/subscribe interface processes command messages (publishes or delete publication messages) under sync point.

YES All messages are processed under sync point.

IFPER Only persistent messages are part of the sync point

The initial value of the queue manager is IFPER.

z/OS **RCVTIME (*integer*)**
The approximate length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to the inactive state. This parameter applies only to message channels and not to MQI channels. This number can be qualified as follows:

- To specify that this number is a multiplier to apply to the negotiated **HBINT** value to determine how long a channel is to wait, set **RCVTTYPE** to MULTIPLY. Specify a **RCVTIME** value of zero or in the range 2 through 99. If you specify zero, the channel continues to wait indefinitely to receive data from its partner.
- To specify that **RCVTIME** is the number of seconds to add to the negotiated **HBINT** value to determine how long a channel is to wait, set **RCVTTYPE** to ADD. Specify an **RCVTIME** value in the range 1 through 999999.
- To specify that **RCVTIME** is a value, in seconds, that the channel is to wait, set **RCVTTYPE** to EQUAL. Specify an RCVTIME value in the range 0 - 999,999. If you specify zero, the channel continues to wait indefinitely to receive data from its partner.

Changes to this parameter take effect for channels that are later started. Channels that are currently started are unaffected by changes to this parameter.

For more information, see Checking that the other end of the channel is still available.

This parameter is valid only on z/OS.

z/OS **RCVTMIN(*integer*)**

The minimum length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to an inactive state. This parameter applies only to message channels (and not to MQI channels).

The TCP/IP channel wait time can be configured relative to the negotiated value of **HBINT**. If **RCVTTYPE** is MULTIPLY or ADD, the resultant value might be less than the value set in **RCVTMIN**. In this case, the TCP/IP channel wait time is set to **RCVTMIN**. If **RCVTTYPE** is EQUAL then **RCVTMIN** does not apply.

Specify a value, in seconds, between zero and 999999.

Changes to this parameter take effect for channels that are later started. Channels that are currently started are unaffected by changes to this parameter.

For more information, see Checking that the other end of the channel is still available.

This parameter is valid only on z/OS.

z/OS **RCVTTYPE**

The qualifier to apply to the value in **RCVTIME**.

MULTIPLY
Specifies that **RCVTIME** is a multiplier to be applied to the negotiated **HBINT** value to determine how long a channel waits.

ADD Specifies that **RCVTIME** is a value, in seconds, to be added to the negotiated **HBINT** value to determine how long a channel waits.

EQUAL
Specifies that **RCVTIME** is a value, in seconds, representing how long the channel waits.

Changes to this parameter take effect for channels that are later started. Channels that are currently started are unaffected by changes to this parameter.

For more information, see Checking that the other end of the channel is still available.

This parameter is valid only on z/OS.

REMOTEEV

Specifies whether remote error events are generated:

DISABLED
Remote error events are not generated.

This is the queue manager's initial default value.

ENABLED
Remote error events are generated.

z/OS If you are using the reduced function form of IBM MQ for z/OS supplied with WebSphere Application Server, only DISABLED is valid.

REPOS(*clustername*)
The name of a cluster for which this queue manager provides a repository manager service. The maximum length is 48 characters conforming to the rules for naming IBM MQ objects.

No more than one of the resultant values of **REPOS** and **REPOSNL** can be nonblank.

If you use the **REPOS** parameter to create a full repository queue manager, connect it to at least one other full repository queue manager in the cluster. Connect it using a cluster-sender channel. See the information in Components of a cluster for details about using cluster-sender channels with full repository queue managers.

Successful completion of the command does not mean that the action completed. To check for true completion, see the ALTER QMGR step in Checking that async commands for distributed networks have finished.

REPOSNL(*nlname*)
The name of a namelist of clusters for which this queue manager provides a repository manager service.

No more than one of the resultant values of **REPOS** and **REPOSNL** can be nonblank.

Both **REPOS** and **REPOSNL** might be blank, or **REPOS** might be blank and the namelist specified by **REPOSNL** is empty. In these cases, this queue manager does not have a full repository. It might be a client of other repository services that are defined in the cluster.

If you use the **REPOSNL** parameter to create a full repository queue manager, connect it to other full repository queue managers. Connect it to at least one other full repository queue manager in each cluster specified in the namelist using cluster-sender channels. See the information in Components of a cluster for details about using cluster-sender channels with full repository queue managers.

Successful completion of the command does not mean that the action completed. To check for true completion, see the ALTER QMGR step in Checking that async commands for distributed networks have finished.

REVDNS
Controls whether reverse lookup of the host name from a Domain Name Server (DNS) is done for the IP address from which a channel has connected. This attribute has an effect only on channels using a transport type (TRPTYPE) of TCP:

ENABLED
DNS host names are reverse looked-up for the IP addresses of inbound channels when this information is required. This setting is required for matching against CHLAUTH rules that contain host names, and to include the host name in error messages. The IP address is still included in messages that provide a connection identifier.

This is the initial default value for the queue manager.

DISABLED
DNS host names are not reverse looked-up for the IP addresses of inbound channels. With this setting any CHLAUTH rules using host names are not matched.

ROUTEREC
Specifies whether trace-route information is recorded if requested in the message. If this parameter is not set to DISABLED, it controls whether any reply generated is sent to SYSTEM.ADMIN.TRACE.ROUTE.QUEUE, or to the destination specified by the message itself. If **ROUTEREC** is not DISABLED, messages not yet at the final destination might have information added to them.

DISABLED
Trace-route information is not recorded.

MSG Trace-route information is recorded and sent to the destination specified by the originator of the message causing the trace route record.

This is the queue manager's initial default value.

QUEUE
Trace-route information is recorded and sent to `SYSTEM.ADMIN.TRACE.ROUTE.QUEUE`.

Multi **SCHINIT**

Specifies whether the channel initiator starts automatically when the queue manager starts.

QMGR
The channel initiator starts automatically when the queue manager starts.

MANUAL
The channel initiator does not start automatically.

This parameter is valid only on Multiplatforms.

Multi **SCMDSERV**

Specifies whether the command server starts automatically when the queue manager starts.

QMGR
The command server starts automatically when the queue manager starts.

MANUAL
The command server does not start automatically.

This parameter is valid only on Multiplatforms.

z/OS **SCYCASE**

Specifies whether the security profiles are uppercase or mixed case.

UPPER
The security profiles are uppercase only. However, `MXTOPIC` and `GMXTOPIC` are used for topic security, and can contain mixed-case profiles.

MIXED
The security profiles are mixed case. MQCMDS and MQCONN are used for command and connection security but they can contain only uppercase profiles.

Changes to **SCYCASE** become effective after you run the following command:

```
REFFRESH SECURITY(*) TYPE(CLASSES)
```

This parameter is valid only on z/OS.

z/OS **SQQMNAME**

The **SQQMNAME** attribute specifies whether a queue manager in a queue-sharing group opens a shared queue in the same group directly. The processing queue manager calls MQOPEN for a shared queue and sets the *ObjectQmgrName* parameter for the queue. If the shared queue is in the same queue-sharing group as the processing queue manager, the queue can be opened directly by the processing queue manager. Set the **SQQMNAME** attribute to control if the queue is opened directly, or by the *ObjectQmgrName* queue manager. The attribute will also be honored when opening a QALIAS with copy disposition, if the target queue is a shared queue in the same queue-sharing group as the processing queue manager. In this situation it is important that the QALIAS copy object on each queue manager in the queue-sharing group has the same target queue.

USE The *ObjectQmgrName* is used, and the appropriate transmission queue is opened.

IGNORE
The processing queue manager opens the shared queue directly. Setting the parameter to this value can reduce the traffic in your queue manager network.

This parameter is valid only on z/OS.

SSLCRLNL (*nlname*)

The name of a namelist of authentication information objects which are used to provide certificate revocation locations to allow enhanced TLS certificate checking.

If `SSLCRLNL` is blank, certificate revocation checking is not invoked unless one of the TLS certificates used contains an `AuthorityInfoAccess` or `CrlDistributionPoint` X.509 certificate extension.

Changes to `SSLCRLNL`, or to the names in a previously specified namelist, or to previously referenced authentication information objects become effective as follows:

- When a **`REFRESH SECURITY TYPE(SSL)`** command is issued.
- ULW On UNIX, Linux, and Windows:
  - When a new channel process is started
  - For channels that run as threads of the channel initiator, when the channel initiator is restarted
  - For channels that run as threads of the listener, when the listener is restarted
- IBM i On IBM i:
  - When a new channel process is started
  - For channels that run as threads of the channel initiator, when the channel initiator is restarted
  - For channels that run as threads of the listener, when the listener is restarted

  On IBM i queue managers, this parameter is ignored. However, it is used to determine which authentication information objects are written to the `AMQCLCHL.TAB` file.
- z/OS On z/OS, when the channel initiator is restarted.

Only authentication information objects with types of `LDAPCRL` or `OCSP` are allowed in the namelist referred to by **`SSLCRLNL`**. Any other type results in an error message when the list is processed and is subsequently ignored.

SSLCRYP(*string*)
Sets the name of the parameter string required to configure the cryptographic hardware present on the system.

All supported cryptographic hardware supports the PKCS #11 interface. Specify a string of the following format:

```
GSK_PKCS11= the PKCS #11 driver path and file name>
; the PKCS #11 token label> ;
the PKCS #11 token password> ; symmetric cipher setting>
;
```

The PKCS #11 driver path is an absolute path to the shared library providing support for the PKCS #11 card. The PKCS #11 driver file name is the name of the shared library. An example of the value required for the PKCS #11 driver path and file name is `/usr/lib/pkcs11/PKCS11_API.so`

To access symmetric cipher operations through GSKit, specify the symmetric cipher setting parameter. The value of this parameter is either:

SYMMETRIC_CIPHER_OFF
Do not access symmetric cipher operations.

SYMMETRIC_CIPHER_ON
Access symmetric cipher operations.

If the symmetric cipher setting parameter is not specified, it has the same effect as specifying `SYMMETRIC_CIPHER_OF F`.

The maximum length of the string is 256 characters.

If you specify a string that is not in the format listed, you get an error.

When the **SSLCRYP** value is changed, the cryptographic hardware parameters specified become the ones used for new TLS connection environments. The new information becomes effective:

- When a new channel process is started.
- For channels that run as threads of the channel initiator, when the channel initiator is restarted.
- For channels that run as threads of the listener, when the listener is restarted.
- When a **REFRESH SECURITY TYPE(SSL)** command is issued.

SSLEV
Specifies whether TLS events are generated.

DISABLED
TLS events are not generated.

This is the queue manager's initial default value.

ENABLED
All TLS events are generated.

ULW z/OS **SSLFIPS**
SSLFIPS specifies whether only FIPS-certified algorithms are to be used if cryptography is carried out in IBM MQ, rather than in cryptographic hardware. If cryptographic hardware is configured, the cryptographic modules used are those modules provided by the hardware product. These might, or might not, be FIPS-certified to a particular level. Whether the modules are FIPS-certified depends on the hardware product in use. For more information about FIPS, see the Federal Information Processing Standards (FIPS) manual.

NO If you set **SSLFIPS** to `NO`, you can use either FIPS certified or non-FIPS certified CipherSpecs.

If the queue manager runs without using cryptographic hardware, refer to the CipherSpecs listed in Specifying CipherSpecs.

This is the queue manager's initial default value.

YES Specifies that only FIPS-certified algorithms are to be used in the CipherSpecs allowed on all TLS connections from and to this queue manager.

For a listing of appropriate FIPS 140-2 certified CipherSpecs; see Specifying CipherSpecs.

Changes to **SSLFIPS** become effective as follows:

- Multi On UNIX, Linux, and Windows:
  - when a **REFRESH SECURITY TYPE(SSL)** command is issued
  - when a new channel process is started
  - for channels that run as threads of the channel initiator, when the channel initiator is restarted
  - for channels that run as threads of the listener, when the listener is restarted
  - for channels that run as threads of a process pooling process, when the process pooling process is started or restarted and first runs a TLS channel. If the process pooling process

has already run a TLS channel, and you want the change to become effective immediately, run the MQSC command **REFRESH SECURITY TYPE(SSL)**. The process pooling process is **amqrmppa**

- z/OS On z/OS, when the channel initiator is restarted.

This parameter is valid only on z/OS, UNIX, Linux, and Windows.

SSLKEYR(*string*)

The name of the Secure Sockets Layer key repository.

The maximum length of the string is 256 characters.

The format of the name depends on the environment:

- z/OS On z/OS, it is the name of a key ring.
- IBM i On IBM i, it is of the form *pathname*/*keyfile*, where *keyfile* is specified without the suffix `.kdb`, and identifies a GSKit key database file.
  - If you specify `*SYSTEM`, IBM MQ uses the system certificate store as the key repository for the queue manager. The queue manager is registered as a server application in the Digital Certificate Manager (DCM). You can assign any server/client certificate in the system store to the queue manager, because you registered it as a server application.
  - If you change the `SSLKEYR` parameter to a value other than `*SYSTEM`, IBM MQ unregisters the queue manager as an application with DCM.
  - The syntax of this parameter is validated to ensure that it contains a valid and absolute directory path.
- ULW On UNIX and Linux, it is of the form *pathname*/*keyfile*, and on Windows *pathname**keyfile*, where *keyfile* is specified without the suffix `.kdb` and identifies a Java keystore file.
  - The syntax of this parameter is validated to ensure that it contains a valid and absolute directory path.

If **SSLKEYR** is blank, channels using TLS do not start. If **SSLKEYR** is set to a value that does not correspond to a key ring or key database file, channels using TLS also do not start.

Changes to **SSLKEYR** become effective as follows:

- When a **REFRESH SECURITY TYPE(SSL)** command is issued.
- Multi on Multiplatforms:
  - when a new channel process is started
  - for channels that run as threads of the channel initiator, when the channel initiator is restarted
  - for channels that run as threads of the listener, when the listener is restarted
  - for channels that run as threads of a process pooling process, **amqrmppa**, when the process pooling process is started or restarted and first runs a TLS channel. If the process pooling process has already run a TLS channel, and you want the change to become effective immediately, run the MQSC command **REFRESH SECURITY TYPE(SSL)**
- z/OS On z/OS, when the channel initiator is restarted.

SSLRKEYC(*integer*)

The number of bytes to be sent and received within an TLS conversation before the secret key is renegotiated. The number of bytes includes control information.

`SSLRKEYC` is used only by TLS channels which initiate communication from the queue manager. For example, the sender channel initiates communication in a sender and receiver channel pairing.

If a value greater than zero is specified, the secret key is also renegotiated before message data is sent or received following a channel heartbeat. The count of bytes until the next secret key renegotiation is reset after each successful renegotiation.

Specify a value in the range 0 - 999,999,999. A value of zero means that the secret key is never renegotiated. If you specify an TLS secret key reset count in the range 1 - 32767 bytes (32 KB), TLS channels use a secret key reset count of 32 KB. The larger reset count value avoids the cost of excessive key resets which would occur for small TLS secret key reset values.

Attention: Non-zero values less than 4096 (4 KB) might cause channels to fail to start, or might cause inconsistencies in the values of **SSLKEYDA**, **SSLKEYTI**, and **SSLRKEYS**.

z/OS **SSLTASKS(*integer*)**
The number of server subtasks to use for processing TLS calls. To use TLS channels, you must have at least two of these tasks running.

This value is in the range 0 - 9999. To avoid problems with storage allocation, do not set the **SSLTASKS** parameter to a value greater than 50.

Changes to this parameter are effective when the channel initiator is restarted.

This parameter is valid only on z/OS.

STATACLS
Specifies whether statistics data is to be collected for auto-defined cluster-sender channels:

QMGR
Collection of statistics data is inherited from the setting of the **STATCHL** parameter of the queue manager.

This is the queue manager's initial default value.

OFF Statistics data collection for the channel is disabled.

LOW Unless STATCHL is NONE, statistics data collection is switched on with a low ratio of data collection with a minimal effect on system performance.

MEDIUM
Unless STATCHL is NONE, statistics data collection is switched on with a moderate ratio of data collection.

HIGH Unless STATCHL is NONE, statistics data collection is switched on with a high ratio of data collection.

A change to this parameter takes effect only on channels started after the change occurs. Any channel started before the change to the parameter continues with the value in force at the time that the channel started.

z/OS On z/OS systems, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying LOW, MEDIUM, or HIGH makes no difference to your results.

STATCHL
Specifies whether statistics data is to be collected for channels:

NONE
Statistics data collection is turned off for channels regardless of the setting of their **STATCHL** parameter.

OFF Statistics data collection is turned off for channels specifying a value of QMGR in their **STATCHL** parameter.

This is the queue manager's initial default value.

LOW Statistics data collection is turned on, with a low ratio of data collection, for channels specifying a value of QMGR in their **STATCHL** parameter.

MEDIUM
Statistics data collection is turned on, with a moderate ratio of data collection, for channels specifying a value of QMGR in their **STATCHL** parameter.

HIGH Statistics data collection is turned on, with a high ratio of data collection, for channels specifying a value of QMGR in their **STATCHL** parameter.

A change to this parameter takes effect only on channels started after the change occurs. Any channel started before the change to the parameter continues with the value in force at the time that the channel started.

z/OS On z/OS systems, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying LOW, MEDIUM, or HIGH makes no difference to your results.

Multi **STATINT(*integer*)**
The time interval, in seconds, at which statistics monitoring data is written to the monitoring queue.

Specify a value in the range 1 through 604800.

Changes to this parameter take immediate effect on the collection of monitoring and statistics data.

This parameter is valid only on Multiplatforms.

Multi **STATMQI**
Specifies whether statistics monitoring data is to be collected for the queue manager:

OFF Data collection for MQI statistics is disabled.

This is the queue manager's initial default value.

ON Data collection for MQI statistics is enabled.

Changes to this parameter take immediate effect on the collection of monitoring and statistics data.

This parameter is valid only on Multiplatforms.

Multi **STATQ**
Specifies whether statistics data is to be collected for queues:

NONE
Statistics data collection is turned off for queues regardless of the setting of their **STATQ** parameter.

OFF Statistics data collection is turned off for queues specifying a value of QMGR or OFF in their **STATQ** parameter. OFF is the default value.

ON Statistics data collection is turned on for queues specifying a value of QMGR or ON in their **STATQ** parameter.

Statistics messages are generated only for queues which are opened after statistics collection is enabled. You do not need to restart the queue manager for the new value of STATQ to take effect.

This parameter is valid only on Multiplatforms.

STRSTPEV
Specifies whether start and stop events are generated:

ENABLED
Start and stop events are generated.

This is the queue manager's initial default value.

DISABLED
Start and stop events are not generated.

SUITEB
Specifies whether Suite B-compliant cryptography is used and what strength is required.

NONE
Suite B is not used. NONE is the default

128_BIT
Suite B 128-bit level security is used.

192_BIT
Suite B 192-bit level security is used

128_BIT,192_BIT
Both Suite B 128-bit and 192-bit level security is used

z/OS **TCPCHL(*integer*)**
The maximum number of channels that can be current, or clients that can be connected, that use the TCP/IP transmission protocol.

The maximum number of sockets used is the sum of the values in **TCPCHL** and **CHIDISPS**. The z/OS UNIX System Services **MAXFILEPROC** parameter (specified in the BPXPRM*xx* member of SYS1.PARMLIB) controls how many sockets each task is allowed, and thus how many channels each dispatcher is allowed. In this case, the number of channels using TCP/IP is limited to the value of **MAXFILEPROC** multiplied by the value of **CHIDISPS**.

Specify a value 0-9999. The value must not be greater than the value of **MAXCHL**. **MAXCHL** defines the maximum number of channels available. TCP/IP might not support as many as 9999 channels. If so, the value you can specify is limited by the number of channels TCP/IP can support. If you specify zero, the TCP/IP transmission protocol is not used.

If you change this value, also review the **MAXCHL**, **LU62CHL**, and **ACTCHL** values to ensure that there is no conflict of values. If necessary, raise the value of **MAXCHL** and **ACTCHL**.

If the value of this parameter is reduced, any current channels that exceed the new limit continue to run until they stop.

Sharing conversations do not contribute to the total for this parameter.

If the value of **TCPCHL** is non-zero when the channel initiator starts up, the value can be modified dynamically. If the value of **TCPCHL** is zero when the channel initiator starts up, a later **ALTER** command does not take effect. In this case, you should carry out an **ALTER** command, either before the channel initiator starts, or in CSQINP2 before you issue the **START CHINIT** command.

This parameter is valid only on z/OS.

z/OS **TCPKEEP**
Specifies whether the **KEEPALIVE** facility is to be used to check that the other end of the connection is still available. If it is unavailable, the channel is closed.

NO The TCP **KEEPALIVE** facility is not to be used.

This is the queue manager's initial default value.

YES The TCP **KEEPALIVE** facility is to be used as specified in the TCP profile configuration data set. The interval is specified in the **KAINT** channel attribute.

Changes to this parameter take effect for channels that are later started. Channels that are currently started are unaffected by changes to this parameter.

This parameter is valid only on z/OS.

Using the **TCPKEEP** parameter is no longer required for 'modern' queue managers. The replacement is a combination of:

- using 'modern' client channels (**SHARECNV** <> 0)
- using receive timeout for message channels **RCVTIME**.

For more information, see the technote *Setting the TCP/IP KeepAlive interval to be used by IBM MQ*, at: http://www.ibm.com/support/docview.wss?uid=swg21216834

z/OS **TCPNAME(*string*)**

The name of either the only, or preferred, TCP/IP stack to be used, depending on the value of **TCPSTACK**. This name is the name of the z/OS UNIX System Services stack for TCP/IP, as specified in the **SUBFILESYSTYPE** NAME parameter in the `BPXPRMxx` member of `SYS1.PARMLIB`. **TCPNAME** is only applicable in CINET multiple stack environments. The queue manager's initial default value is TCPIP.

In INET single stack environments the channel initiator uses the only available TCP/IP stack.

The maximum length of this parameter is eight characters.

Changes to this parameter take effect when the channel initiator is restarted.

This parameter is valid only on z/OS.

z/OS **TCPSTACK**

Specifies whether the channel initiator can use only the TCP/IP stack specified in **TCPNAME**, or optionally bind to any selected TCP/IP stack defined. This parameter is only applicable in CINET multiple stack environments.

SINGLE
The channel initiator can use only the TCP/IP address space specified in **TCPNAME**.

MULTIPLE
The channel initiator can use any TCP/IP address space available to it.

Changes to this parameter take effect when the channel initiator is restarted.

This parameter is valid only on z/OS.

z/OS **TRAXSTR**

Specifies whether the channel initiator trace starts automatically:

YES Channel initiator trace is to start automatically.

NO Channel initiator trace is not to start automatically.

Changes to this parameter take effect when the channel initiator is restarted. If you want to start or stop channel initiator trace without restarting the channel initiator, use the **START TRACE** or **STOP TRACE** commands after starting the channel initiator.

This parameter is valid only on z/OS.

z/OS **TRAXTBL(*integer*)**

The size, in megabytes, of the trace data space of the channel initiator.

Specify a value in the range 2 through 2048.

Note:

1. Changes to this parameter take effect immediately; any existing trace table contents are lost.
2. The **CHINIT** trace is stored in a dataspace called `qmidCHIN.CSQXTRDS`. When you use large z/OS data spaces, ensure that sufficient auxiliary storage is available on your system to support any related z/OS paging activity. You might also need to increase the size of your `SYS1.DUMP` data sets.

This parameter is valid only on z/OS.

TREELIFE (*integer*)
The lifetime, in seconds of non-administrative topics.

Non-administrative topics are those topics created when an application publishes to, or subscribes on, a topic string that does not exist as an administrative node. When this non-administrative node no longer has any active subscriptions, this parameter determines how long the queue manager waits before removing that node. Only non-administrative topics that are in use by a durable subscription remain after the queue manager is recycled.

Specify a value in the range 0 through 604000. A value of 0 means that non-administrative topics are not removed by the queue manager.

TRIGINT(*integer*)
A time interval expressed in milliseconds.

The **TRIGINT** parameter is relevant only if the trigger type (**TRIGTYPE**) is set to FIRST (see "DEFINE QLOCAL" on page 683 for details). In this case trigger messages are normally generated only when a suitable message arrives on the queue, and the queue was previously empty. Under certain circumstances, however, an additional trigger message can be generated with FIRST triggering even if the queue was not empty. These additional trigger messages are not generated more often than every **TRIGINT** milliseconds; see Special case of trigger type FIRST.

Specify a value in the range 0 - 999,999,999.

Related information:

Working with queue managers

Working with dead-letter queues

z/OS Working with TLS on z/OS

ALTER queues:

Use the MQSC **ALTER** command to alter the parameters of a queue. A queue might be a local queue (**ALTER QLOCAL**), alias queue (**ALTER QALIAS**), model queue (**ALTER QMODEL**), a remote queue, a queue manager alias, or a reply-to queue alias (**ALTER QREMOTE**).

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

This section contains the following commands:

- "ALTER QALIAS" on page 517
- "ALTER QLOCAL on Multiplatforms" on page 518
- "ALTER QMODEL on z/OS" on page 521
- "ALTER QREMOTE on z/OS" on page 523

Parameters not specified in the **ALTER** queue commands result in the existing values for those parameters being left unchanged.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

Usage notes for ALTER queues

- Successful completion of the command does not mean that the action completed. To check for true completion, see the ALTER queues step in Checking that async commands for distributed networks have finished.

Parameter descriptions for ALTER QUEUE

The parameters that are relevant for each type of queue are tabulated in Table 90. Each parameter is described after the table.

Table 90. DEFINE and ALTER QUEUE parameters

Parameter	Local queue	Model queue	Alias queue	Remote queue
ACCTQ	✓	✓		
BOQNAME	✓	✓		
BOTHRESH	✓	✓		
z/OS CFSTRUCT	✓	✓		
CLCHNAME	✓			
CLUSNL	✓		✓	✓
CLUSTER	✓		✓	✓
CLWLPRTY	✓		✓	✓
CLWLRANK	✓		✓	✓
CLWLUSEQ	✓			
z/OS CMDSCOPE	✓	✓	✓	✓
CUSTOM	✓	✓	✓	✓
DEFBIND	✓		✓	✓
DEFPRESP	✓	✓	✓	✓
DEFPRTY	✓	✓	✓	✓
DEFPSIST	✓	✓	✓	✓
DEFREADA	✓	✓	✓	
DEFSOPT	✓	✓		
DEFTYPE	✓	✓		
DESCR	✓	✓	✓	✓

Table 90. DEFINE and ALTER QUEUE parameters (continued)

Parameter	Local queue	Model queue	Alias queue	Remote queue
DISTL	✓	✓		
FORCE	✓		✓	✓
GET	✓	✓	✓	
HARDENBO or **NOHARDENBO**	✓	✓		
V 9.0.2 **IMGRCOVQ**	✓	✓		
INDXTYPE	✓	✓		
INITQ	✓	✓		
LIKE	✓	✓	✓	✓
MAXDEPTH	✓	✓		
MAXMSGL	✓	✓		
MONQ	✓	✓		
MSGDLVSQ	✓	✓		
NOREPLACE	✓	✓	✓	✓
NPMCLASS	✓	✓		
PROCESS	✓	✓		
PROPCTL	✓	✓	✓	
PUT	✓	✓	✓	✓
queue-name	✓	✓	✓	✓
QDEPTHHI	✓	✓		
QDEPTHLO	✓	✓		
QDPHIEV	✓	✓		
QDPLOEV	✓	✓		
QDPMAXEV	✓	✓		
z/OS **QSGDISP**	✓	✓	✓	✓

Table 90. DEFINE and ALTER QUEUE parameters (continued)

Parameter	Local queue	Model queue	Alias queue	Remote queue
QSVCIEV	✓	✓		
QSVCINT	✓	✓		
REPLACE	✓	✓	✓	✓
RETINTVL	✓	✓		
RNAME				✓
RQMNAME				✓
SCOPE	✓		✓	✓
SHARE or **NOSHARE**	✓	✓		
STATQ	✓	✓		
z/OS **STGCLASS**	✓	✓		
TARGET			✓	
TARGQ			✓	
TARGTYPE			✓	
TRIGDATA	✓	✓		
TRIGDPTH	✓	✓		
TRIGGER or **NOTRIGGER**	✓	✓		
TRIGMPRI	✓	✓		
TRIGTYPE	✓	✓		
USAGE	✓	✓		
XMITQ				✓

queue-name

Local name of the queue, except the remote queue where it is the local definition of the remote queue.

The name must not be the same as any other queue name of any queue type currently defined on this queue manager, unless you specify **REPLACE** or **ALTER**. See Rules for naming IBM MQ objects.

ACCTQ

Specifies whether accounting data collection is to be enabled for the queue. On z/OS, the data collected is class 3 accounting data (thread-level and queue-level accounting). In order for

accounting data to be collected for this queue, accounting data for this connection must also be enabled. Turn on accounting data collection by setting either the **ACCTQ** queue manager attribute, or the options field in the MQCNO structure on the MQCONNX call.

QMGR The collection of accounting data is based on the setting of the **ACCTQ** parameter on the queue manager definition.

ON Accounting data collection is enabled for the queue unless the **ACCTQ** queue manager parameter has a value of NONE.

z/OS On z/OS systems, you must enable class 3 accounting using the **START TRACE** command.

OFF Accounting data collection is disabled for the queue.

BOQNAME(*queue-name*)
The excessive backout requeue name.

This parameter is supported only on local and model queues.

Use this parameter to set or change the back out queue name attribute of a local or model queue. Apart from allowing its value to be queried, the queue manager does nothing based on the value of this attribute. IBM MQ classes for JMS transfers a message that is backed out the maximum number of times to this queue. The maximum is specified by the **BOTHRESH** attribute.

BOTHRESH(*integer*)
The backout threshold.

This parameter is supported only on local and model queues.

Use this parameter to set or change the value of the back out threshold attribute of a local or model queue. Apart from allowing its value to be queried, the queue manager does nothing based on the value of this attribute. IBM MQ classes for JMS use the attribute to determine how many times back a message out. When the value is exceeded the message is transferred to the queue named by the **BOQNAME** attribute.

Specify a value in the range 0 - 999,999,999.

z/OS **CFSTRUCT(*structure-name*)**
Specifies the name of the coupling facility structure where you want messages stored when you use shared queues.

This parameter is supported only on z/OS for local and model queues.

The name:

- Cannot have more than 12 characters
- Must start with an uppercase letter (A - Z)
- Can include only the characters A - Z and 0 - 9

The name of the queue-sharing group to which the queue manager is connected is prefixed to the name you supply. The name of the queue-sharing group is always four characters, padded with @ symbols if necessary. For example, if you use a queue-sharing group named NY03 and you supply the name PRODUCT7, the resultant coupling facility structure name is NY03PRODUCT7. The administrative structure for the queue-sharing group (in this case NY03CSQ_ADMIN) cannot be used for storing messages.

For **ALTER QLOCAL**, **ALTER QMODEL**, **DEFINE QLOCAL** with **REPLACE**, and **DEFINE QMODEL** with **REPLACE** the following rules apply:

- On a local queue with **QSGDISP**(SHARED), **CFSTRUCT** cannot change.

If you change either the **CFSTRUCT** or **QSGDISP** value you must delete and redefine the queue. To preserve any of the messages on the queue you must offload the messages before you delete the queue. Reload the messages after you redefine the queue, or move the messages to another queue.

- On a model queue with **DEFTYPE**(SHAREDYN), **CFSTRUCT** cannot be blank.
- On a local queue with a **QSGDISP** other than SHARED, or a model queue with a **DEFTYPE** other than SHAREDYN, the value of **CFSTRUCT** does not matter.

For **DEFINE QLOCAL** with **NOREPLACE** and **DEFINE QMODEL** with **NOREPLACE**, the coupling facility structure:

- On a local queue with **QSGDISP**(SHARED) or a model queue with a **DEFTYPE**(SHAREDYN), **CFSTRUCT** cannot be blank.
- On a local queue with a **QSGDISP** other than SHARED, or a model queue with a **DEFTYPE** other than SHAREDYN, the value of **CFSTRUCT** does not matter.

Note: Before you can use the queue, the structure must be defined in the coupling facility Resource Management (CFRM) policy data set.

CLCHNAME(*channel name*)

This parameter is supported only on transmission queues.

CLCHNAME is the generic name of the cluster-sender channels that use this queue as a transmission queue. The attribute specifies which cluster-sender channels send messages to a cluster-receiver channel from this cluster transmission queue.

You can also set the transmission queue attribute **CLCHNAME** attribute to a cluster-sender channel manually. Messages that are destined for the queue manager connected by the cluster-sender channel are stored in the transmission queue that identifies the cluster-sender channel. They are not stored in the default cluster transmission queue. If you set the **CLCHNAME** attribute to blanks, the channel switches to the default cluster transmission queue when the channel restarts. The default queue is either SYSTEM.CLUSTER.TRANSMIT.*ChannelName* or SYSTEM.CLUSTER.TRANSMIT.QUEUE, depending on the value of the queue manager **DEFCLXQ** attribute.

By specifying asterisks, “" * "”, in **CLCHNAME**, you can associate a transmission queue with a set of cluster-sender channels. The asterisks can be at the beginning, end, or any number of places in the middle of the channel name string. **CLCHNAME** is limited to a length of 48 characters, MQ_OBJECT_NAME_LENGTH. A channel name is limited to 20 characters: MQ_CHANNEL_NAME_LENGTH. If you specify an asterisk you must also set the SHARE attribute so that multiple channels can concurrently access the transmission queue.

z/OS If you specify a “" * "” in **CLCHNAME**, to obtain a channel profile name, you must specify the channel profile name within quotation marks. If you do not specify the generic channel name within quotation marks you receive message CSQ9030E.

The default queue manager configuration is for all cluster-sender channels to send messages from a single transmission queue, SYSTEM.CLUSTER.TRANSMIT.QUEUE. The default configuration can be changed by modified by changing the queue manager attribute, **DEFCLXQ**. The default value of the attribute is SCTQ. You can change the value to CHANNEL. If you set the **DEFCLXQ** attribute to CHANNEL, each cluster-sender channel defaults to using a specific cluster transmission queue, SYSTEM.CLUSTER.TRANSMIT.*ChannelName*.

z/OS On z/OS, if this parameter is set, the queue:

- Must be shareable, by specifying the queue attribute SHARE.
- Must be indexed on the correlation ID by specifying INDXTYPE(CORRELID).
- Must not be a dynamic or a shared queue.

ULW z/OS **CLUSNL(*namelist name*)**

The name of the namelist that specifies a list of clusters to which the queue belongs.

This parameter is supported only on alias, local, and remote queues.

Changes to this parameter do not affect instances of the queue that are already open.

Only one of the resultant values of **CLUSNL** or **CLUSTER** can be nonblank; you cannot specify a value for both.

On local queues, this parameter cannot be set for the following queues:

- Transmission queues
- SYSTEM.CHANNEL.*xx* queues
- SYSTEM.CLUSTER.*xx* queues
- SYSTEM.COMMAND.*xx* queues
- z/OS On z/OS only, SYSTEM.QSG.*xx* queues

This parameter is valid only on the following platforms:

- UNIX, Linux, and Windows
- z/OS

ULW z/OS **CLUSTER(*cluster name*)**

The name of the cluster to which the queue belongs.

This parameter is supported only on alias, local, and remote queues.

The maximum length is 48 characters conforming to the rules for naming IBM MQ objects. Changes to this parameter do not affect instances of the queue that are already open.

Only one of the resultant values of **CLUSNL** or **CLUSTER** can be nonblank; you cannot specify a value for both.

On local queues, this parameter cannot be set for the following queues:

- Transmission queues
- SYSTEM.CHANNEL.*xx* queues
- SYSTEM.CLUSTER.*xx* queues
- SYSTEM.COMMAND.*xx* queues
- z/OS On z/OS only, SYSTEM.QSG.*xx* queues

This parameter is valid only on the following platforms:

- UNIX, Linux, and Windows
- z/OS

CLWLPRTY(*integer*)

Specifies the priority of the queue for the purposes of cluster workload distribution. This parameter is valid only for local, remote, and alias queues. The value must be in the range zero through 9 where zero is the lowest priority and 9 is the highest. For more information about this attribute, see CLWLPRTY queue attribute.

CLWLRANK (*integer*)

Specifies the rank of the queue for the purposes of cluster workload distribution. This parameter is valid only for local, remote, and alias queues. The value must be in the range zero through 9 where zero is the lowest rank and 9 is the highest. For more information about this attribute, see CLWLRANK queue attribute.

CLWLUSEQ

Specifies the behavior of an MQPUT operation when the target queue has a local instance and at

least one remote cluster instance. The parameter has no effect when the MQPUT originates from a cluster channel. This parameter is valid only for local queues.

QMGR The behavior is as specified by the **CLWLUSEQ** parameter of the queue manager definition.

ANY The queue manager is to treat the local queue as another instance of the cluster queue for the purposes of workload distribution.

LOCAL The local queue is the only target of the MQPUT operation.

z/OS **CMDSCOPE**

This parameter applies to z/OS only. It specifies where the command is run when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if **QSGDISP** is set to GROUP or SHARED.

'' The command runs on the queue manager on which it was entered.

QmgrName

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered. You can specify another name, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

CUSTOM(*string*)

The custom attribute for new features.

This attribute contains the values of attributes, as pairs of attribute name and value, separated by at least one space. The attribute name-value pairs have the form NAME(VALUE). Single quotation marks must be escaped with another single quotation mark.

CAPEXPRY(*integer*)

The maximum time, expressed in tenths of a second, until a message put using an object handle with this object in the resolution path, becomes eligible for expiry processing.

For more information on message expiry processing, see Enforcing lower expiration times.

integer

The value must be in the range one through to 999 999 999.

NOLIMIT

There is no limit on the expiry time of messages put using this object. This is the default value.

Specifying a value for **CAPEXPRY** that is not valid, does not cause the command to fail. Instead the default value is used.
Note that existing messages in the queue, prior to a change in **CAPEXPRY**, are not affected by the change (that is, their expiry time remains intact). Only new messages that are put into the queue after the change in **CAPEXPRY** have the new expiry time.

DEFBIND

Specifies the binding to be used when the application specifies MQOO_BIND_AS_Q_DEF on the MQOPEN call, and the queue is a cluster queue.

OPEN The queue handle is bound to a specific instance of the cluster queue when the queue is opened.

NOTFIXED
The queue handle is not bound to any instance of the cluster queue. The queue manager selects a specific queue instance when the message is put using MQPUT. It changes that selection later, if the need arises.

GROUP Allows an application to request that a group of messages is allocated to the same destination instance.

Multiple queues with the same name can be advertised in a queue manager cluster. An application can send all messages to a single instance, MQOO_BIND_ON_OPEN. It can allow a workload management algorithm to select the most suitable destination on a per message basis, MQOO_BIND_NOT_FIXED. It can allow an application to request that a group of messages be all allocated to the same destination instance. The workload balancing reselects a destination between groups of messages, without requiring an MQCLOSE and MQOPEN of the queue.

The MQPUT1 call always behaves as if NOTFIXED is specified.

This parameter is valid on all platforms.

DEFPRESP
Specifies the behavior to be used by applications when the put response type, within the MQPMO options, is set to MQPMO_RESPONSE_AS_Q_DEF.

SYNC Put operations to the queue specifying MQPMO_RESPONSE_AS_Q_DEF are issued as if MQPMO_SYNC_RESPONSE is specified instead.

ASYNC Put operations to the queue specifying MQPMO_RESPONSE_AS_Q_DEF are issued as if MQPMO_ASYNC_RESPONSE is specified instead; see MQPMO options (MQLONG).

DEFPRTY(*integer*)
The default priority of messages put on the queue. The value must be in the range 0 - 9. Zero is the lowest priority, through to the **MAXPRTY** queue manager parameter. The default value of **MAXPRTY** is 9.

DEFPSIST
Specifies the message persistence to be used when applications specify the MQPER_PERSISTENCE_AS_Q_DEF option.

NO Messages on this queue are lost across a restart of the queue manager.

YES Messages on this queue survive a restart of the queue manager.

z/OS On z/OS, N and Y are accepted as synonyms of NO and YES.

DEFREADA
Specifies the default read ahead behavior for non-persistent messages delivered to the client. Enabling read ahead can improve the performance of client applications consuming non-persistent messages.

NO Non-persistent messages are not read ahead unless the client application is configured to request read ahead.

YES Non-persistent messages are sent to the client before an application requests them. Non-persistent messages can be lost if the client ends abnormally or if the client does not delete all the messages it is sent.

DISABLED
Read ahead of non-persistent messages in not enabled for this queue. Messages are not sent ahead to the client regardless of whether read ahead is requested by the client application.

DEFSOPT
The default share option for applications opening this queue for input:

EXCL The open request is for exclusive input from the queue

SHARED The open request is for shared input from the queue

DEFTYPE

Queue definition type.

This parameter is supported only on model queues.

PERMDYN

A permanent dynamic queue is created when an application issues an MQOPEN MQI call with the name of this model queue specified in the object descriptor (MQOD).

z/OS On z/OS, the dynamic queue has a disposition of QMGR.

z/OS **SHAREDYN**

This option is available on z/OS only.

A permanent dynamic queue is created when an application issues an MQOPEN API call with the name of this model queue specified in the object descriptor (MQOD).

The dynamic queue has a disposition of SHARED.

TEMPDYN

A temporary dynamic queue is created when an application issues an MQOPEN API call with the name of this model queue specified in the object descriptor (MQOD).

z/OS On z/OS, the dynamic queue has a disposition of QMGR.

Do not specify this value for a model queue definition with a **DEFPSIST** parameter of YES.

If you specify this option, do not specify **INDXTYPE**(MSGTOKEN).

DESCR(*string*)

Plain-text comment. It provides descriptive information about the object when an operator issues the **DISPLAY QUEUE** command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: Use characters that are in the coded character set identifier (CCSID) of this queue manager. If you do not do so and if the information is sent to another queue manager, they might be translated incorrectly.

ULW **DISTL**

DISTL sets whether distribution lists are supported by the partner queue manager.

YES Distribution lists are supported by the partner queue manager.

NO Distribution lists are not supported by the partner queue manager.

Note: You do not normally change this parameter, because it is set by the MCA. However you can set this parameter when defining a transmission queue if the distribution list capability of the destination queue manager is known.

This parameter is valid only on UNIX, Linux, and Windows.

FORCE

This parameter applies only to the **ALTER** command on alias, local and remote queues.

Specify this parameter to force completion of the command in the following circumstances.

For an alias queue, if both of the following statements are true:

- The **TARGQ** parameter is specified
- An application has this alias queue open

For a local queue, if both of the following statements are true:

- The **NOSHARE** parameter is specified
- More than one application has the queue open for input

FORCE is also needed if both of the following statements are true:

- The **USAGE** parameter is changed
- Either one or more messages are on the queue, or one or more applications have the queue open

Do not change the **USAGE** parameter while there are messages on the queue; the format of messages changes when they are put on a transmission queue.

For a remote queue, if both of the following statements are true:

- The **XMITQ** parameter is changed
- One or more applications has this queue open as a remote queue

FORCE is also needed if both of the following statements are true:

- Any of the **RNAME**, **RQMNAME**, or **XMITQ** parameters are changed
- One or more applications has a queue open that resolved through this definition as a queue manager alias

Note: **FORCE** is not required if this definition is in use as a reply-to queue alias only.

If **FORCE** is not specified in the circumstances described, the command is unsuccessful.

GET Specifies whether applications are to be permitted to get messages from this queue:

ENABLED
Messages can be retrieved from the queue, by suitably authorized applications.

DISABLED
Applications cannot retrieve messages from the queue.

This parameter can also be changed using the MQSET API call.

HARDENBO &NOHARDENBO
Specifies whether hardening is used to ensure that the count of the number of times that a message is backed out is accurate.

This parameter is supported only on local and model queues.

HARDENBO
The count is hardened.

NOHARDENBO
The count is not hardened.

Note: z/OS This parameter affects only IBM MQ for z/OS. You can set this parameter on Multiplatforms, but it is ineffective.

Multi V 9.0.2 **IMGRCOVQ**
Specifies whether a local or permanent dynamic queue object is recoverable from a media image, if linear logging is being used. Possible values are:

YES These queue objects are recoverable.

NO The “rcdmqimg (record media image)” on page 283 and“rcrmqobj (re-create object)” on page 289 commands are not permitted for these objects, and automatic media images, if enabled, are not written for these objects.

QMGR
If you specify QMGR, and the **IMGRCOVQ** attribute for the queue manager specifies YES, these queue objects are recoverable.

If you specify QMGR and the **IMGRCOVQ** attribute for the queue manager specifies NO, the "rcdmqimg (record media image)" on page 283 and"rcrmqobj (re-create object)" on page 289 commands are not permitted for these objects, and automatic media images, if enabled, are not written for these objects.

QMGR is the default value.

This parameter is not valid on z/OS.

z/OS **INDXTYPE**

The type of index maintained by the queue manager to expedite MQGET operations on the queue. For shared queues, the type of index determines the type of MQGET operations that can be used.

This parameter is supported only on z/OS. On other platforms, all queues are automatically indexed.

This parameter is supported only on local and model queues.

Messages can be retrieved using a selection criterion only if an appropriate index type is maintained, as the following table shows:

Table 91. Index type required for different retrieval selection criteria

Retrieval selection criterion	**Index type required**	
	Shared queue	Other queue
None (sequential retrieval)	Any	Any
Message identifier	MSGID or NONE	Any
Correlation identifier	CORRELID	Any
Message and correlation identifiers	MSGID or CORRELID	Any
Group identifier	GROUPID	Any
Grouping	GROUPID	GROUPID
Message token	Not allowed	MSGTOKEN

where the value of **INDXTYPE** parameter has the following values:

NONE No index is maintained. Use NONE when messages are typically retrieved sequentially or use both the message identifier and the correlation identifier as a selection criterion on the MQGET call.

MSGID An index of message identifiers is maintained. Use MSGID when messages are typically retrieved using the message identifier as a selection criterion on the MQGET call with the correlation identifier set to NULL.

CORRELID
An index of correlation identifiers is maintained. Use CORRELID when messages are typically retrieved using the correlation identifier as a selection criterion on the MQGET call with the message identifier set to NULL.

GROUPID
An index of group identifiers is maintained. Use GROUPID when messages are retrieved using message grouping selection criteria.

Note:

1. You cannot set **INDXTYPE** to GROUPID if the queue is a transmission queue.
2. The queue must use a CF structure at CFLEVEL(3), to specify a shared queue with **INDXTYPE**(GROUPID).

z/OS **MSGTOKEN**

An index of message tokens is maintained. Use MSGTOKEN when the queue is a WLM-managed queue that you are using with the Workload Manager functions of z/OS.

Note: You cannot set **INDXTYPE** to MSGTOKEN if:

- The queue is a model queue with a definition type of SHAREDYN
- The queue is a temporary dynamic queue
- The queue is a transmission queue
- You specify **QSGDISP**(SHARED)

For queues that are not shared and do not use grouping or message tokens, the index type does not restrict the type of retrieval selection. However, the index is used to expedite **GET** operations on the queue, so choose the type that corresponds to the most common retrieval selection.

If you are altering or replacing an existing local queue, you can change the **INDXTYPE** parameter only in the cases indicated in the following table:

Table 92. Index type change permitted depending upon queue-sharing and presence of messages in the queue

Queue type		**NON-SHARED**			**SHARED**	
Queue state		Uncommitted activity	No uncommitted activity, messages present	No uncommitted activity, and empty	Open or messages present	Not open, and empty
Change **INDXTYPE** from:	To:	Change allowed?				
NONE	MSGID	No	Yes	Yes	No	Yes
NONE	CORRELID	No	Yes	Yes	No	Yes
NONE	MSGTOKEN	No	No	Yes	-	-
NONE	GROUPID	No	No	Yes	No	Yes
MSGID	NONE	No	Yes	Yes	No	Yes
MSGID	CORRELID	No	Yes	Yes	No	Yes
MSGID	MSGTOKEN	No	No	Yes	-	-
MSGID	GROUPID	No	No	Yes	No	Yes
CORRELID	NONE	No	Yes	Yes	No	Yes
CORRELID	MSGID	No	Yes	Yes	No	Yes
CORRELID	MSGTOKEN	No	No	Yes	-	-
CORRELID	GROUPID	No	No	Yes	No	Yes
MSGTOKEN	NONE	No	Yes	Yes	-	-
MSGTOKEN	MSGID	No	Yes	Yes	-	-
MSGTOKEN	CORRELID	No	Yes	Yes	-	-
MSGTOKEN	GROUPID	No	No	Yes	-	-
GROUPID	NONE	No	No	Yes	No	Yes
GROUPID	MSGID	No	No	Yes	No	Yes
GROUPID	CORRELID	No	No	Yes	No	Yes
GROUPID	MSGTOKEN	No	No	Yes	-	-

INITQ(*string*)
The local name of the initiation queue on this queue manager, to which trigger messages relating to this queue are written; see Rules for naming IBM MQ objects.

This parameter is supported only on local and model queues.

LIKE(*qtype-name*)
The name of a queue, with parameters that are used to model this definition.

If this field is not completed, the values of undefined parameter fields are taken from one of the following definitions. The choice depends on the queue type:

Table 93. Queue types and their corresponding definitions

Queue type	Definition
Alias queue	SYSTEM.DEFAULT.ALIAS.QUEUE
Local queue	SYSTEM.DEFAULT.LOCAL.QUEUE
Model queue	SYSTEM.DEFAULT.MODEL.QUEUE
Remote queue	SYSTEM.DEFAULT.REMOTE.QUEUE

For example, not completing this parameter is equivalent to defining the following value of **LIKE** for an alias queue:

```
LIKE(SYSTEM.DEFAULT.ALIAS.QUEUE)
```

If you require different default definitions for all queues, alter the default queue definitions instead of using the **LIKE** parameter.

z/OS On z/OS, the queue manager searches for an object with the name and queue type you specify with a disposition of QMGR, COPY, or SHARED. The disposition of the **LIKE** object is not copied to the object you are defining.

Note:

1. **QSGDISP**(GROUP) objects are not searched.
2. **LIKE** is ignored if **QSGDISP**(COPY) is specified.

ULW z/OS **MAXDEPTH(*integer*)**
The maximum number of messages allowed on the queue.

This parameter is supported only on local and model queues.

On the following platforms, specify a value in the range zero through 999999999:

- ULW UNIX, Linux, and Windows
- z/OS z/OS

On any other IBM MQ platform, specify a value in the range zero through 640000.

Other factors can still cause the queue to be treated as full, for example, if there is no further hard disk space available.

If this value is reduced, any messages that are already on the queue that exceed the new maximum remain intact.

MAXMSGL(*integer*)
The maximum length (in bytes) of messages on this queue.

This parameter is supported only on local and model queues.

ULW On UNIX, Linux, and Windows, specify a value in the range zero to the maximum message length for the queue manager. See the **MAXMSGL** parameter of the ALTER QMGR command, ALTER QMGR MAXMSGL.

z/OS On z/OS, specify a value in the range zero through 100 MB (104 857 600 bytes).

Message length includes the length of user data and the length of headers. For messages put on the transmission queue, there are additional transmission headers. Allow an additional 4000 bytes for all the message headers.

If this value is reduced, any messages that are already on the queue with length that exceeds the new maximum are not affected.

Applications can use this parameter to determine the size of buffer for retrieving messages from the queue. Therefore, the value can be reduced only if it is known that this reduction does not cause an application to operate incorrectly.

Note that by adding the digital signature and key to the message, Advanced Message Security increases the length of the message.

MONQ

Controls the collection of online monitoring data for queues.

This parameter is supported only on local and model queues.

QMGR Collect monitoring data according to the setting of the queue manager parameter **MONQ**.

OFF Online monitoring data collection is turned off for this queue.

LOW If the value of the **MONQ** parameter of the queue manager is not NONE, online monitoring data collection is turned on for this queue.

MEDIUM If the value of the **MONQ** parameter of the queue manager is not NONE, online monitoring data collection is turned on for this queue.

HIGH If the value of the **MONQ** parameter of the queue manager is not NONE, online monitoring data collection is turned on for this queue.

There is no distinction between the values LOW, MEDIUM, and HIGH. These values all turn data collection on, but do not affect the rate of collection.

When this parameter is used in an **ALTER** queue command, the change is effective only when the queue is next opened.

MSGDLVSQ

Message delivery sequence.

This parameter is supported only on local and model queues.

PRIORITY
Messages are delivered (in response to MQGET API calls) in first-in-first-out (FIFO) order within priority.

FIFO Messages are delivered (in response to MQGET API calls) in FIFO order. Priority is ignored for messages on this queue.

The message delivery sequence parameter can be changed from PRIORITY to FIFO while there are messages on the queue. The order of the messages already on the queue is not changed. Messages added to the queue later take the default priority of the queue, and so might be processed before some of the existing messages.

If the message delivery sequence is changed from FIFO to PRIORITY, the messages put on the queue while the queue was set to FIFO take the default priority.

Note: z/OS If **INDXTYPE**(GROUPID) is specified with **MSGDLVSQ**(PRIORITY), the priority in which groups are retrieved is based on the priority of the first message within each group. The priorities

0 and 1 are used by the queue manager to optimize the retrieval of messages in logical order. The first message in each group must not use these priorities. If it does, the message is stored as if it was priority two.

Multi **NPMCLASS**

The level of reliability to be assigned to non-persistent messages that are put to the queue:

NORMAL Non-persistent messages are lost after a failure, or queue manager shutdown. These messages are discarded on a queue manager restart.

HIGH The queue manager attempts to retain non-persistent messages on this queue over a queue manager restart or switch over.

z/OS You cannot set this parameter on z/OS.

PROCESS(*string*)

The local name of the IBM MQ process.

This parameter is supported only on local and model queues.

This parameter is the name of a process instance that identifies the application started by the queue manager when a trigger event occurs; see Rules for naming IBM MQ objects.

The process definition is not checked when the local queue is defined, but it must be available for a trigger event to occur.

If the queue is a transmission queue, the process definition contains the name of the channel to be started. This parameter is optional for transmission queues on the following platforms:

- IBM i IBM i
- ULW UNIX, Linux, and Windows
- z/OS z/OS

If you do not specify it, the channel name is taken from the value specified for the **TRIGDATA** parameter.

PROPCTL

Property control attribute. The attribute is optional. It is applicable to local, alias, and model queues.

PROPCTL options are as follows. The options do not affect message properties in the MQMD or MQMD extension.

ALL

Set ALL so that an application can read all the properties of the message either in MQRFH2 headers, or as properties of the message handle.

The ALL option enables applications that cannot be changed to access all the message properties from MQRFH2 headers. Applications that can be changed, can access all the properties of the message as properties of the message handle.

In some cases, the format of data in MQRFH2 headers in the received message might be different to the format in the message when it was sent.

COMPAT

Set COMPAT so that unmodified applications that expect JMS-related properties to be in an MQRFH2 header in the message data continue to work as before. Applications that can be changed, can access all the properties of the message as properties of the message handle.

If the message contains a property with a prefix of mcd., jms., usr., or mqext., all message properties are delivered to the application. If no message handle is supplied,

properties are returned in an MQRFH2 header. If a message handle is supplied, all properties are returned in the message handle.

If the message does not contain a property with one of those prefixes, and the application does not provide a message handle, no message properties are returned to the application. If a message handle is supplied, all properties are returned in the message handle.

In some cases, the format of data in MQRFH2 headers in the received message might be different to the format in the message when it was sent.

FORCE

Force all applications to read message properties from MQRFH2 headers.

Properties are always returned in the message data in an MQRFH2 header regardless of whether the application specifies a message handle.

A valid message handle supplied in the MsgHandle field of the MQGMO structure on the MQGET call is ignored. Properties of the message are not accessible using the message handle.

In some cases, the format of data in MQRFH2 headers in the received message might be different to the format in the message when it was sent.

NONE

If a message handle is supplied, all the properties are returned in the message handle.

All message properties are removed from the message body before it is delivered to the application.

PUT Specifies whether messages can be put on the queue.

ENABLED
Messages can be added to the queue (by suitably authorized applications).

DISABLED
Messages cannot be added to the queue.

This parameter can also be changed using the MQSET API call.

QDEPTHHI(*integer*)
The threshold against which the queue depth is compared to generate a Queue Depth High event.

This parameter is supported only on local and model queues.

z/OS For more information about the effect that shared queues on z/OS have on this event; see Shared queues and queue depth events on z/OS.

This event indicates that an application put a message on a queue resulting in the number of messages on the queue becoming greater than or equal to the queue depth high threshold. See the **QDPHIEV** parameter.

The value is expressed as a percentage of the maximum queue depth (**MAXDEPTH** parameter), and must be in the range zero through 100 and no less than **QDEPTHLO**.

QDEPTHLO(*integer*)
The threshold against which the queue depth is compared to generate a Queue Depth Low event.

This parameter is supported only on local and model queues.

z/OS For more information about the effect that shared queues on z/OS have on this event; see Shared queues and queue depth events on z/OS.

This event indicates that an application retrieved a message from a queue resulting in the number of messages on the queue becoming less than or equal to the queue depth low threshold. See the **QDPLOEV** parameter.

The value is expressed as a percentage of the maximum queue depth (**MAXDEPTH** parameter), and must be in the range zero through 100 and no greater than **QDEPTHHI**.

QDPHIEV

Controls whether Queue Depth High events are generated.

This parameter is supported only on local and model queues.

A Queue Depth High event indicates that an application put a message on a queue resulting in the number of messages on the queue becoming greater than or equal to the queue depth high threshold. See the **QDEPTHHI** parameter.

ENABLED
Queue Depth High events are generated.

DISABLED
Queue Depth High events are not generated.

Note: The value of this parameter can change implicitly.

z/OS On z/OS, shared queues affect the event.

For more information about this event, see Queue Depth High.

QDPLOEV

Controls whether Queue Depth Low events are generated.

This parameter is supported only on local and model queues.

A Queue Depth Low event indicates that an application retrieved a message from a queue resulting in the number of messages on the queue becoming less than or equal to the queue depth low threshold. See the **QDEPTHLO** parameter.

ENABLED
Queue Depth Low events are generated.

DISABLED
Queue Depth Low events are not generated.

Note: The value of this parameter can change implicitly.

z/OS On z/OS, shared queues affect the event.

For more information about this event, see Queue Depth Low.

QDPMAXEV

Controls whether Queue Full events are generated.

This parameter is supported only on local and model queues.

A Queue Full event indicates that a put to a queue was rejected because the queue is full. The queue depth reached its maximum value.

ENABLED
Queue Full events are generated.

DISABLED
Queue Full events are not generated.

Note: The value of this parameter can change implicitly.

z/OS On z/OS, shared queues affect the event.

For more information about this event, see Queue Full.

z/OS **QSGDISP**

This parameter applies to z/OS only.

Specifies the disposition of the object within the group.

Table 94. Action of **ALTER** *depending on different values of* **QSGDISP**.

QSGDISP	**ALTER**
COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters **QSGDISP**(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters **QSGDISP**(QMGR), is not affected by this command.
GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameters **QSGDISP**(GROUP). Any object residing on the page set of the queue manager that executes the command (except a local copy of the object), or any object defined using a command that had the parameters **QSGDISP**(SHARED), is not affected by this command. If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero: `DEFINE QUEUE(`*QNAME*`)` `REPLACE QSGDISP(COPY)` The ALTER for the group object takes effect regardless of whether the generated command with **QSGDISP**(COPY) fails.
PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with **QSGDISP**(QMGR) or **QSGDISP**(COPY). Any object residing in the shared repository is unaffected.
QMGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters **QSGDISP**(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.
SHARED	This value applies only to local queues. The object definition resides in the shared repository. The object was defined using a command that had the parameters **QSGDISP**(SHARED). Any object residing on the page set of the queue manager that executes the command, or any object defined using a command that had the parameters **QSGDISP**(GROUP), is not affected by this command. If the queue is clustered, a command is generated and sent to all active queue managers in the queue-sharing group to notify them of this clustered, shared queue.

QSVCIEV

Controls whether Service Interval High or Service Interval OK events are generated.

This parameter is supported only on local and model queues and is ineffective if it is specified on a shared queue.

A Service Interval High event is generated when a check indicates that no messages were retrieved from the queue for at least the time indicated by the **QSVCINT** parameter.

A Service Interval OK event is generated when a check indicates that messages were retrieved from the queue within the time indicated by the **QSVCINT** parameter.

Note: The value of this parameter can change implicitly. For more information, see the description of the Service Interval High and Service Interval OK events in Queue Service Interval High and Queue Service Interval OK.

HIGH Service Interval High events are generated

OK Service Interval OK events are generated

NONE No service interval events are generated

QSVCINT(*integer*)
The service interval used for comparison to generate Service Interval High and Service Interval OK events.

This parameter is supported only on local and model queues and is ineffective if it is specified on a shared queue.

See the **QSVCIEV** parameter.

The value is in units of milliseconds, and must be in the range zero through 999999999.

REPLACE & NOREPLACE
This option controls whether any existing definition is to be replaced with this one.

Note: z/OS On IBM MQ for z/OS, an existing definition is replaced only if it is of the same disposition. Any object with a different disposition is not changed.

REPLACE
If the object does exist, the effect is like issuing the **ALTER** command without the **FORCE** parameter and with all the other parameters specified. In particular, note that any messages that are on the existing queue are retained.

There is a difference between the **ALTER** command without the **FORCE** parameter, and the **DEFINE** command with the **REPLACE** parameter. The difference is that **ALTER** does not change unspecified parameters, but **DEFINE** with **REPLACE** sets all the parameters. If you use **REPLACE**, unspecified parameters are taken either from the object named on the **LIKE** parameter, or from the default definition, and the parameters of the object being replaced, if one exists, are ignored.

The command fails if both of the following statements are true:

- The command sets parameters that would require the use of the **FORCE** parameter if you were using the **ALTER** command
- The object is open

The **ALTER** command with the **FORCE** parameter succeeds in this situation.

ULW If **SCOPE(CELL)** is specified on UNIX, Linux, or Windows, and there is already a queue with the same name in the cell directory, the command fails, even if **REPLACE** is specified.

NOREPLACE
The definition must not replace any existing definition of the object.

RETINTVL(*integer*)
The number of hours from when the queue was defined, after which the queue is no longer needed. The value must be in the range 0 - 999,999,999.

This parameter is supported only on local and model queues.

The **CRDATE** and **CRTIME** can be displayed using the **DISPLAY QUEUE** command.

This information is available for use by an operator or a housekeeping application to delete queues that are no longer required.

Note: The queue manager does not delete queues based on this value, nor does it prevent queues from being deleted if their retention interval is not expired. It is the responsibility of the user to take any required action.

RNAME(*string*)
Name of remote queue. This parameter is the local name of the queue as defined on the queue manager specified by **RQMNAME**.

This parameter is supported only on remote queues.

- If this definition is used for a local definition of a remote queue, **RNAME** must not be blank when the open occurs.
- If this definition is used for a queue manager alias definition, **RNAME** must be blank when the open occurs.

  In a queue manager cluster, this definition applies only to the queue manager that made it. To advertise the alias to the whole cluster, add the **CLUSTER** attribute to the remote queue definition.
- If this definition is used for a reply-to queue alias, this name is the name of the queue that is to be the reply-to queue.

The name is not checked to ensure that it contains only those characters normally allowed for queue names; see Rules for naming IBM MQ objects.

RQMNAME(*string*)
The name of the remote queue manager on which the queue **RNAME** is defined.

This parameter is supported only on remote queues.

- If an application opens the local definition of a remote queue, **RQMNAME** must not be blank or the name of the local queue manager. When the open occurs, if **XMITQ** is blank there must be a local queue of this name, which is to be used as the transmission queue.
- If this definition is used for a queue manager alias, **RQMNAME** is the name of the queue manager that is being aliased. It can be the name of the local queue manager. Otherwise, if **XMITQ** is blank, when the open occurs there must be a local queue of this name, which is to be used as the transmission queue.
- If **RQMNAME** is used for a reply-to queue alias, **RQMNAME** is the name of the queue manager that is to be the reply-to queue manager.

The name is not checked to ensure that it contains only those characters normally allowed for IBM MQ object names; see Rules for naming IBM MQ objects.

ULW **SCOPE**
Specifies the scope of the queue definition.

This parameter is supported only on alias, local, and remote queues.

QMGR The queue definition has queue manager scope. This means that the definition of the queue does not extend beyond the queue manager that owns it. You can open a queue for output that is owned by another queue manager in either of two ways:

1. Specify the name of the owning queue manager.
2. Open a local definition of the queue on the other queue manager.

CELL The queue definition has cell scope. Cell scope means that the queue is known to all the queue managers in the cell. A queue with cell scope can be opened for output merely by specifying the name of the queue. The name of the queue manager that owns the queue need not be specified.

If there is already a queue with the same name in the cell directory, the command fails. The **REPLACE** option does not affect this situation.

This value is valid only if a name service supporting a cell directory is configured.

Restriction: The DCE name service is no longer supported.

This parameter is valid only on UNIX, Linux, and Windows.

SHARE and NOSHARE
Specifies whether multiple applications can get messages from this queue.

This parameter is supported only on local and model queues.

SHARE More than one application instance can get messages from the queue.

NOSHARE
Only a single application instance can get messages from the queue.

Multi **STATQ**
Specifies whether statistics data collection is enabled:

QMGR Statistics data collection is based on the setting of the **STATQ** parameter of the queue manager.

ON If the value of the **STATQ** parameter of the queue manager is not NONE, statistics data collection for the queue is enabled.

OFF Statistics data collection for the queue is disabled.

If this parameter is used in an **ALTER** queue command, the change is effective only for connections to the queue manager made after the change to the parameter.

This parameter is valid only on Multiplatforms.

z/OS **STGCLASS(*string*)**
The name of the storage class.

This parameter is supported only on local and model queues.

Note: You can change this parameter only if the queue is empty and closed.

This parameter is an installation-defined name. The first character of the name must be uppercase A through Z, and subsequent characters either uppercase A through Z or numeric 0 through 9.

This parameter is valid only on z/OS; see Storage classes.

If you specify **QSGDISP**(SHARED) or **DEFTYPE**(SHAREDYN), this parameter is ignored.

TARGET(*string*)
The name of the queue or topic object being aliased; See Rules for naming IBM MQ objects. The object can be a queue or a topic as defined by **TARGTYPE**. The maximum length is 48 characters.

This parameter is supported only on alias queues.

This object needs to be defined only when an application process opens the alias queue.

This parameter is a synonym of the parameter **TARGQ**; **TARGQ** is retained for compatibility. If you specify **TARGET**, you cannot also specify **TARGQ**.

TARGTYPE(*string*)
The type of object to which the alias resolves.

QUEUE The alias resolves to a queue.

TOPIC The alias resolves to a topic.

TRIGDATA(*string*)
The data that is inserted in the trigger message. The maximum length of the string is 64 bytes.

This parameter is supported only on local and model queues.

For a transmission queue, you can use this parameter to specify the name of the channel to be started.

This parameter can also be changed using the MQSET API call.

TRIGDPTH(*integer*)
The number of messages that have to be on the queue before a trigger message is written, if **TRIGTYPE** is DEPTH. The value must be in the range 1 - 999,999,999.

This parameter is supported only on local and model queues.

This parameter can also be changed using the MQSET API call.

TRIGGER & NOTRIGGER
Specifies whether trigger messages are written to the initiation queue, named by the **INITQ** parameter, to trigger the application, named by the **PROCESS** parameter:

TRIGGER
Triggering is active, and trigger messages are written to the initiation queue.

NOTRIGGER
Triggering is not active, and trigger messages are not written to the initiation queue.

This parameter is supported only on local and model queues.

This parameter can also be changed using the MQSET API call.

TRIGMPRI(*integer*)
The message priority number that triggers this queue. The value must be in the range zero through to the **MAXPRTY** queue manager parameter; see "DISPLAY QMGR" on page 860 for details.

This parameter can also be changed using the MQSET API call.

TRIGTYPE
Specifies whether and under what conditions a trigger message is written to the initiation queue. The initiation queue is (named by the **INITQ** parameter.

This parameter is supported only on local and model queues.

FIRST Whenever the first message of priority equal to or greater than the priority specified by the **TRIGMPRI** parameter of the queue arrives on the queue.

EVERY Every time a message arrives on the queue with priority equal to or greater than the priority specified by the **TRIGMPRI** parameter of the queue.

DEPTH When the number of messages with priority equal to or greater than the priority specified by **TRIGMPRI** is equal to the number indicated by the **TRIGDPTH** parameter.

NONE No trigger messages are written.

This parameter can also be changed using the MQSET API call.

USAGE
Queue usage.

This parameter is supported only on local and model queues.

NORMAL The queue is not a transmission queue.

XMITQ The queue is a transmission queue, which is used to hold messages that are destined for a remote queue manager. When an application puts a message to a remote queue, the message is stored on the appropriate transmission queue. It stays there, awaiting transmission to the remote queue manager.

If you specify this option, do not specify values for **CLUSTER** and **CLUSNL**.

z/OS Additionally, on z/OS, do not specify **INDXTYPE**(MSGTOKEN) or **INDXTYPE**(GROUPID).

XMITQ(*string*)

The name of the transmission queue to be used for forwarding messages to the remote queue. **XMITQ** is used with either remote queue or queue manager alias definitions.

This parameter is supported only on remote queues.

If **XMITQ** is blank, a queue with the same name as **RQMNAME** is used as the transmission queue.

This parameter is ignored if the definition is being used as a queue manager alias and **RQMNAME** is the name of the local queue manager.

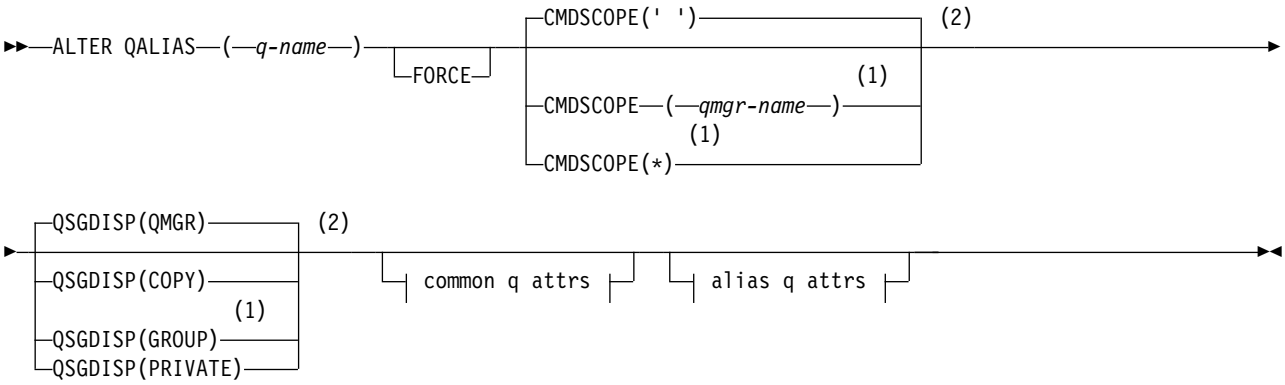
It is also ignored if the definition is used as a reply-to queue alias definition.

ALTER QALIAS:

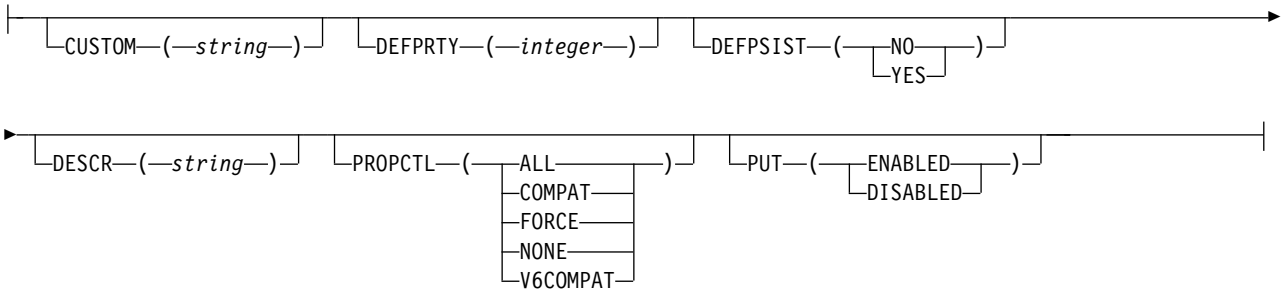
Use the MQSC command **ALTER QALIAS** to alter the parameters of an alias queue.

Synonym: ALT QA

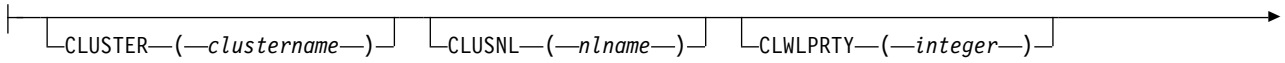
ALTER QALIAS

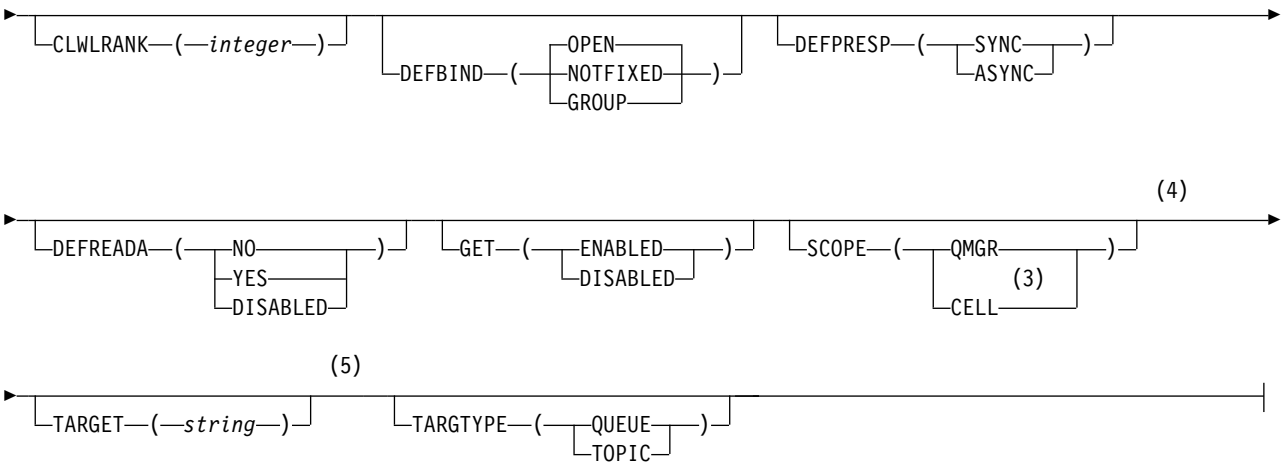


Common q attrs:



Alias q attrs:





Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Valid only on Windows, UNIX, and Linux systems.

4 Not valid on z/OS.

5 The TARGQ parameter is available for compatibility with previous releases. It is a synonym of TARGET; you cannot specify both parameters.

The parameters are described in "ALTER queues" on page 494.

Related information:

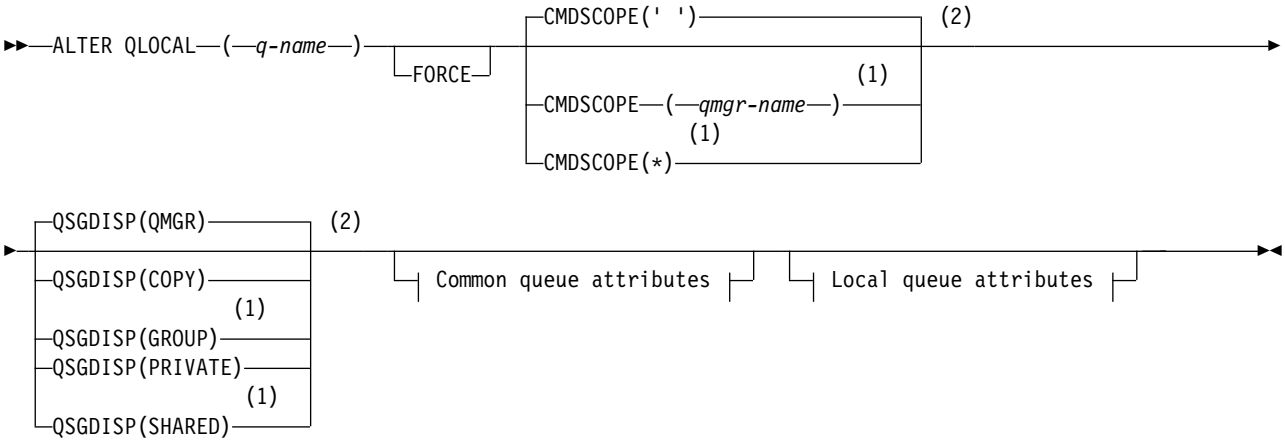
Working with alias queues

ALTER QLOCAL on Multiplatforms: Multi

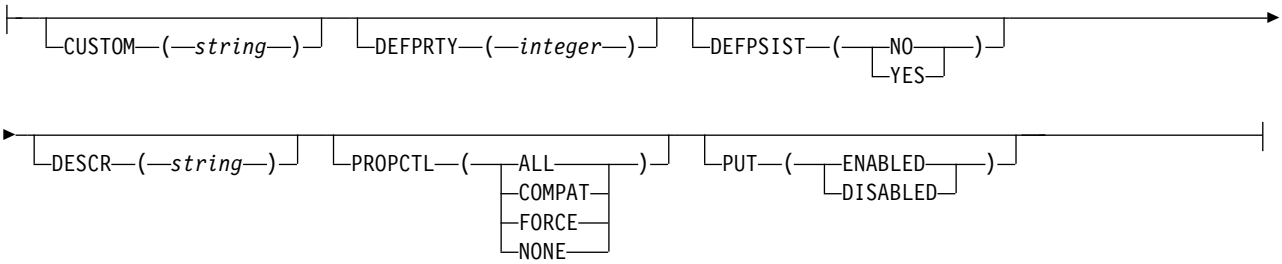
Use the MQSC command **ALTER QLOCAL** to alter the parameters of a local queue.

Synonym: ALT QL

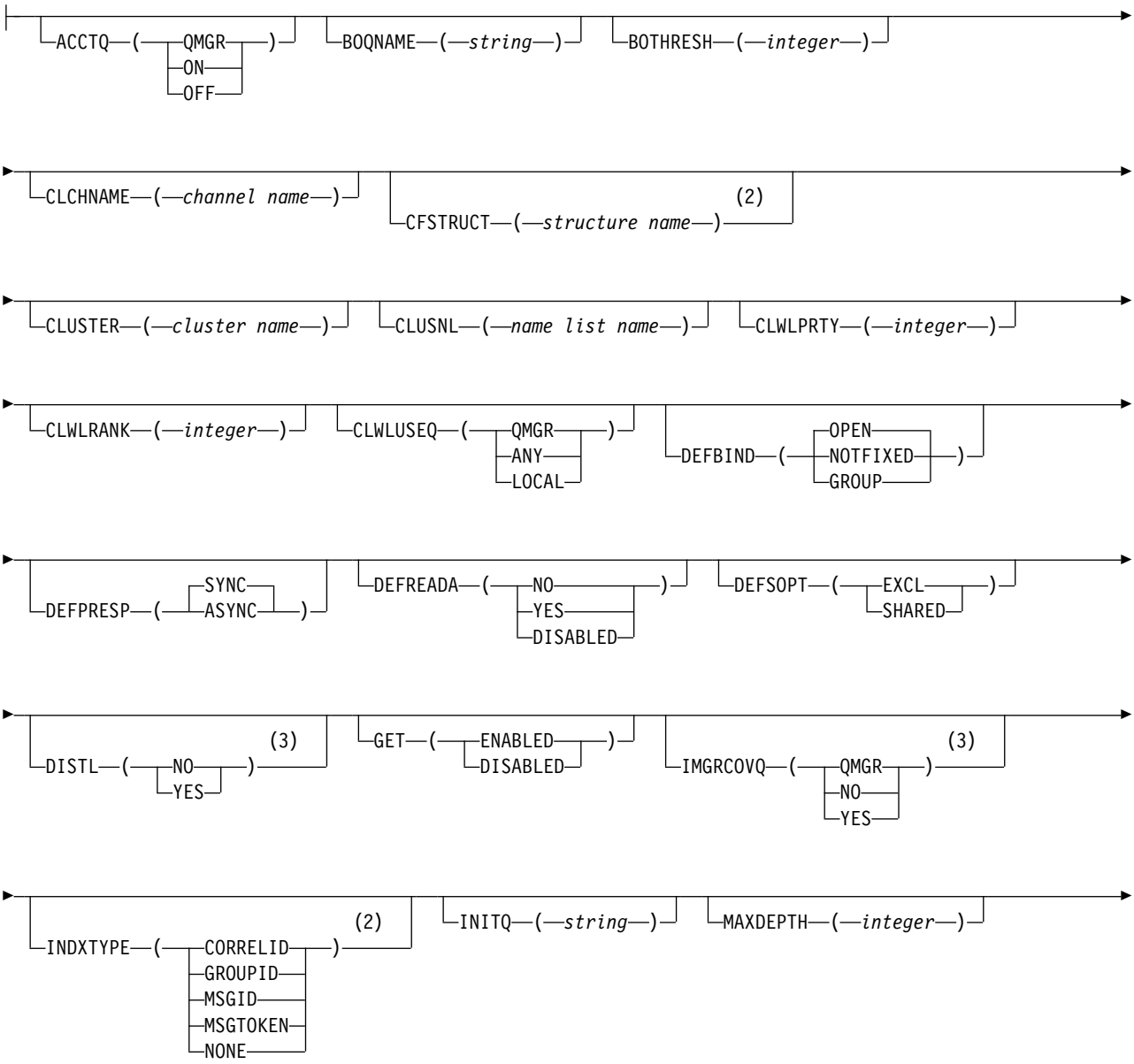
ALTER QLOCAL

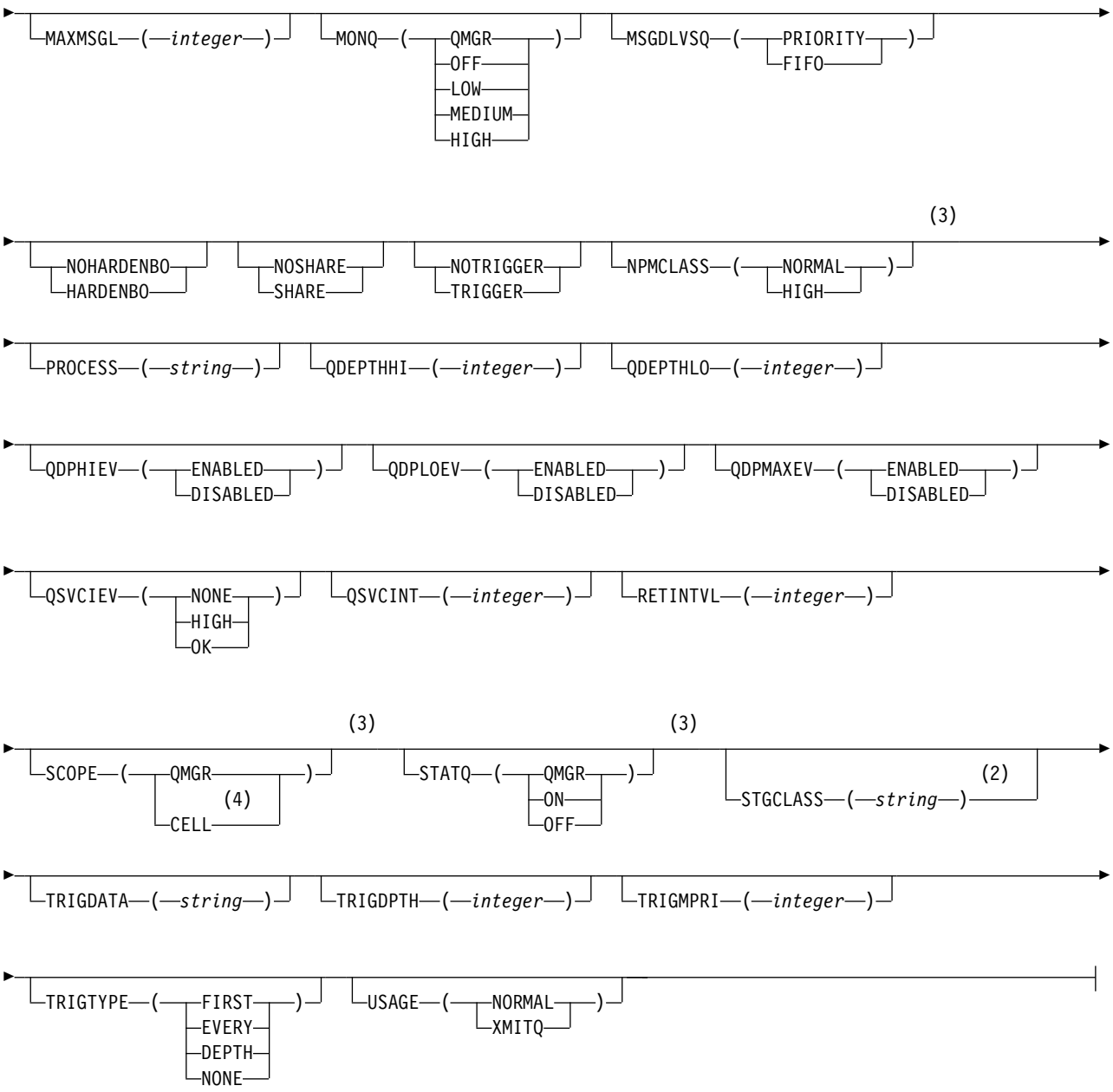


Common queue attributes:



Local queue attributes:





Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Not valid on z/OS.

4 Valid on Multiplatforms.

The parameters are described in "ALTER queues" on page 494.

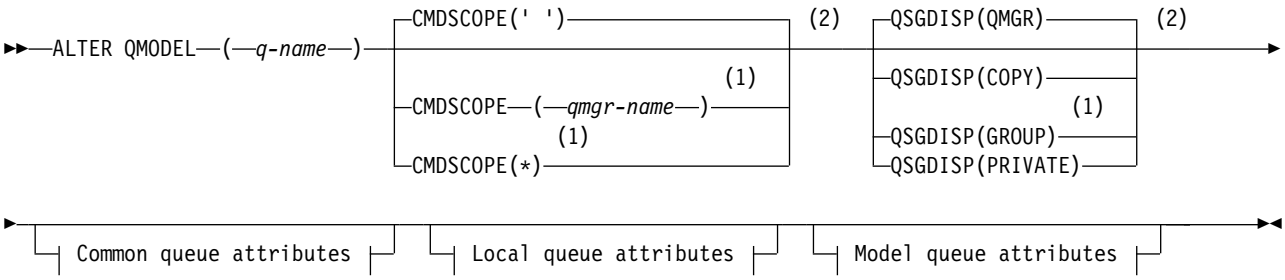
Related information:
Changing local queue attributes

ALTER QMODEL on z/OS: z/OS

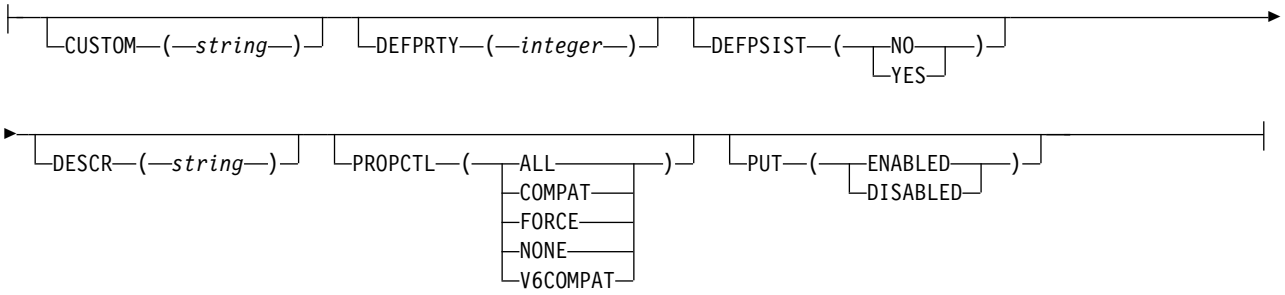
Use the MQSC command **ALTER QMODEL** to alter the parameters of a model queue.

Synonym: ALT QM

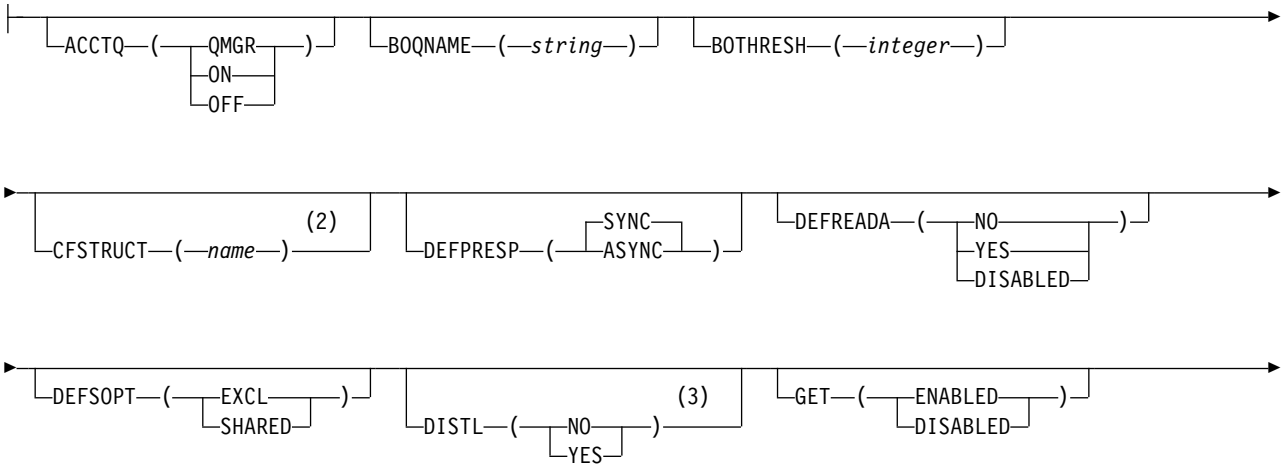
ALTER QMODEL

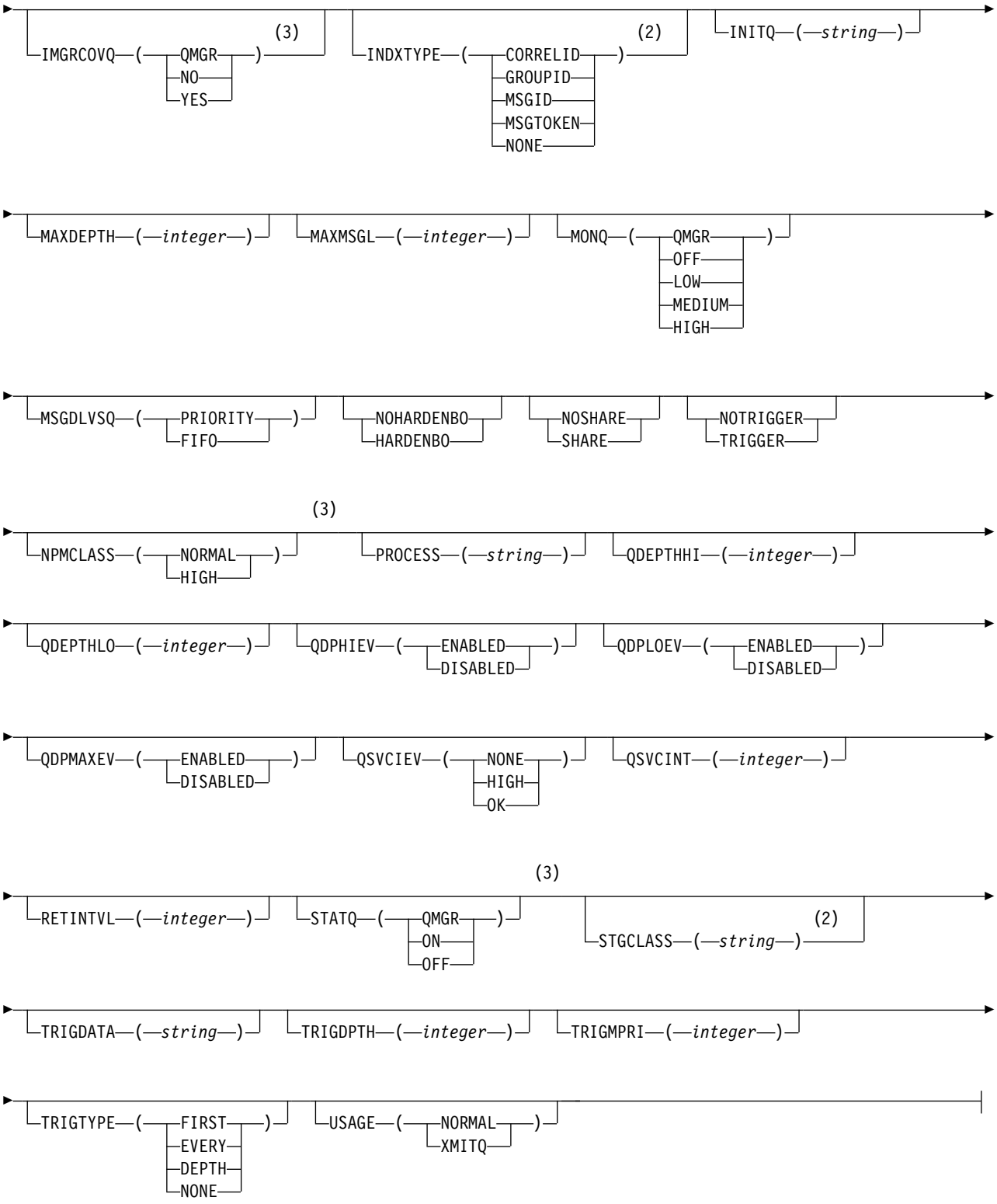


Common queue attributes:



Local queue attributes:





Model queue attributes:

Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Not valid on z/OS.

The parameters are described in "ALTER queues" on page 494.

Related information:

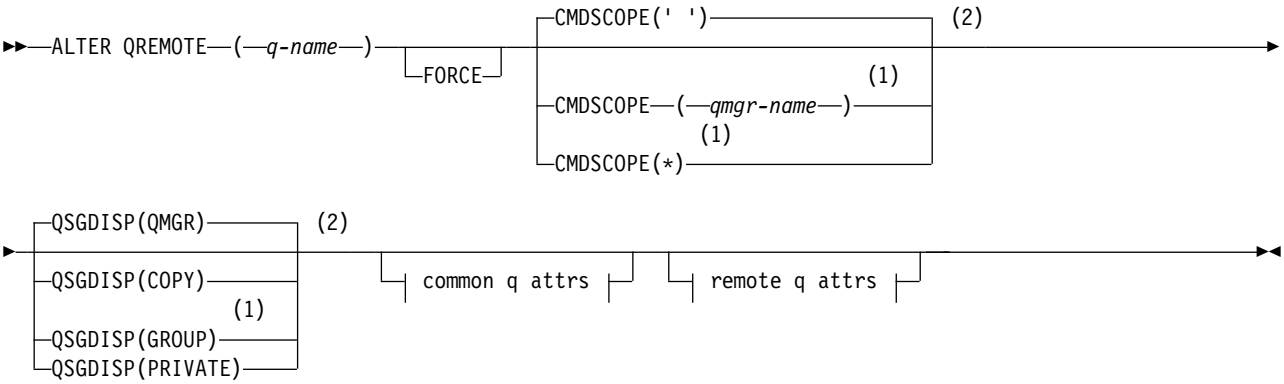
Working with model queues

ALTER QREMOTE on z/OS: z/OS

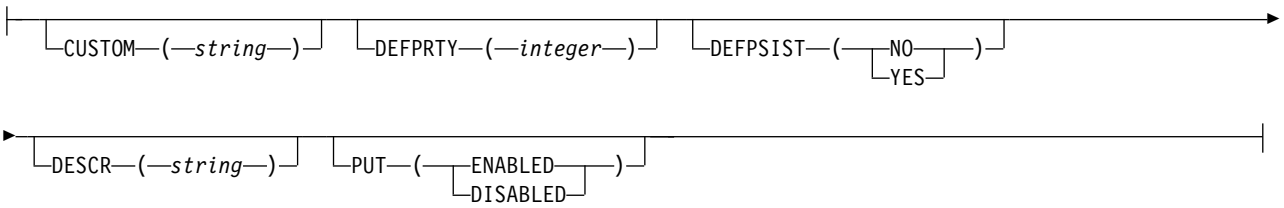
Use the MQSC command **ALTER QREMOTE** to alter the parameters of a local definition of a remote queue, a queue manager alias, or a reply-to queue alias.

Synonym: ALT QR

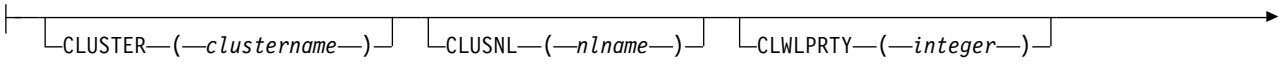
ALTER QREMOTE

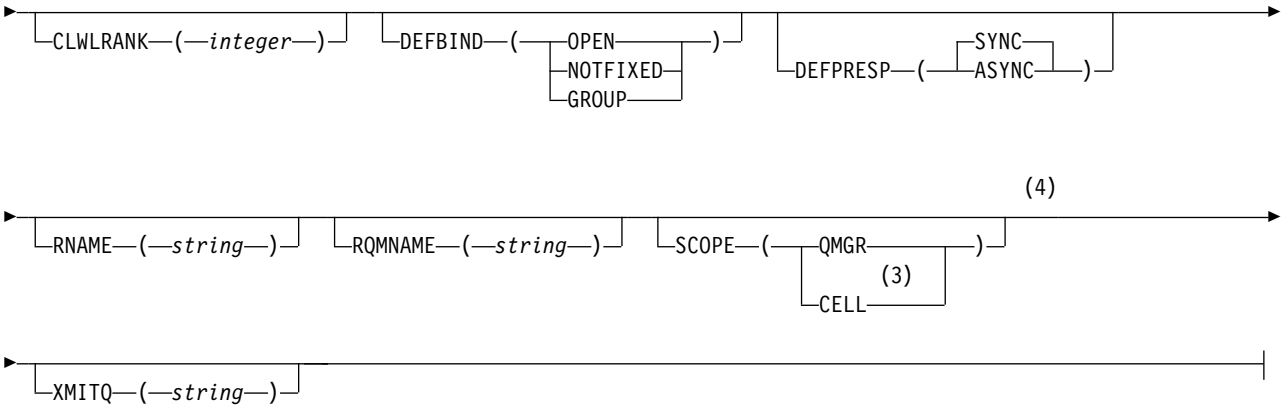


Common q attrs:



Remote q attrs:





Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Valid only on UNIX, Linux, and Windows.

4 Not valid on z/OS.

The parameters are described in "ALTER queues" on page 494.

ALTER SECURITY on z/OS: z/OS

Use the MQSC command **ALTER SECURITY** to define system-wide security options.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

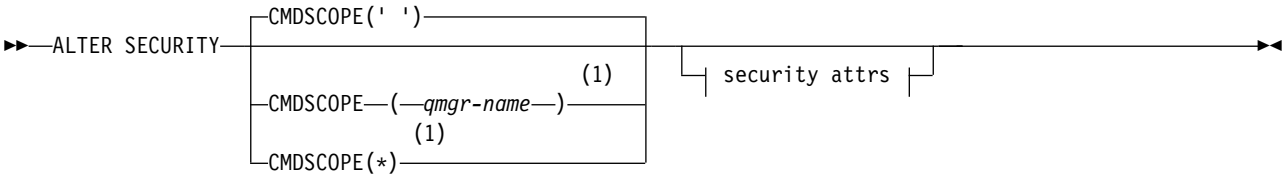
Parameters not specified in the **ALTER SECURITY** command result in the existing values for those parameters being left unchanged.

You can issue this command from sources 12CR. For an explanation of the source symbols, see Using commands on z/OS.

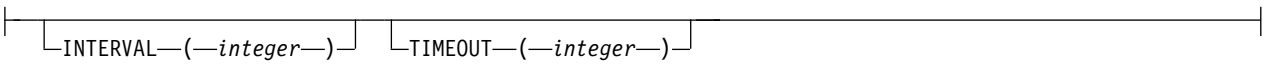
- Syntax diagram
- "Parameter descriptions for ALTER SECURITY" on page 525

Synonym: ALT SEC

ALTER SECURITY



Security attrs:



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

Parameter descriptions for ALTER SECURITY

The parameters you specify override the current parameter values. Attributes that you do not specify are unchanged.

Note: If you do not specify any parameters, the command completes successfully, but no security options are changed.

CMDSCOPE
This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE cannot be used for commands issued from the first initialization input data set CSQINP1.

' ' The command runs on the queue manager on which it was entered.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

INTERVAL(*integer*)
The interval between checks for user IDs and their associated resources to determine whether the **TIMEOUT** has expired. The value is in minutes, in the range zero through 10080 (one week). If **INTERVAL** is specified as zero, no user timeouts occur.

TIMEOUT(*integer*)
How long security information about an unused user ID and associated resources is retained by IBM MQ. The value specifies a number of minutes in the range zero through 10080 (one week). If **TIMEOUT** is specified as zero, and **INTERVAL** is nonzero, all such information is discarded by the queue manager every **INTERVAL** number of minutes.

The length of time that an unused user ID and associated resources are retained by IBM MQ depends on the value of **INTERVAL**. The user ID times out at a time between **TIMEOUT** and **TIMEOUT** plus **INTERVAL**.

When the **TIMEOUT** and **INTERVAL** parameters are changed, the previous timer request is canceled and a new timer request is scheduled immediately, using the new **TIMEOUT** value. When the timer request is actioned, a new value for **INTERVAL** is set.

Related information:
User ID timeouts

ALTER SERVICE on Multiplatforms: Multi

Use the MQSC command **ALTER SERVICE** to alter the parameters of an existing IBM MQ service definition.

Using MQSC commands

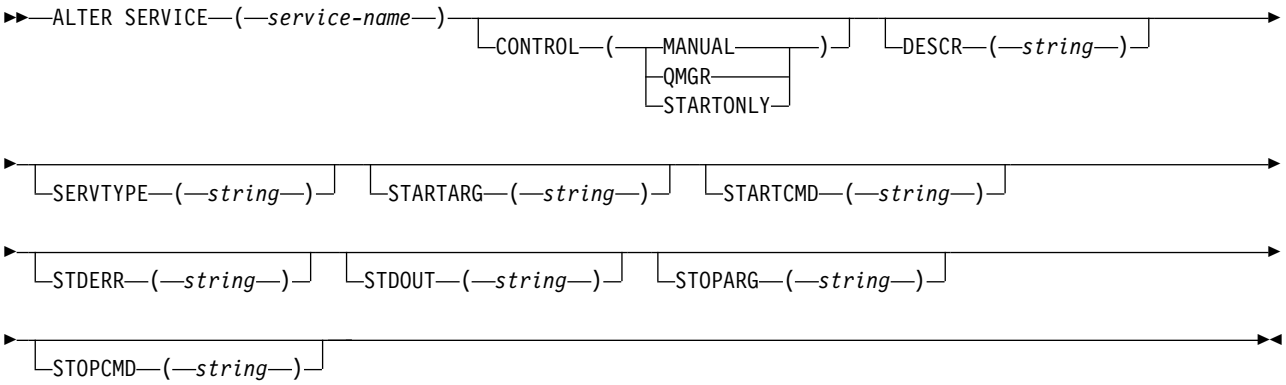
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

Parameters not specified in the **ALTER SERVICE** command result in the existing values for those parameters being left unchanged.

- Syntax diagram
- "Parameter descriptions for ALTER SERVICE"

Synonym:

ALTER SERVICE



Parameter descriptions for ALTER SERVICE

The parameter descriptions apply to the **ALTER SERVICE** and **DEFINE SERVICE** commands, with the following exceptions:

- The **LIKE** parameter applies only to the **DEFINE SERVICE** command.
- The **NOREPLACE** and **REPLACE** parameter applies only to the **DEFINE SERVICE** command.

(service-name)
Name of the IBM MQ service definition (see Rules for naming IBM MQ objects).

The name must not be the same as any other service definition currently defined on this queue manager (unless **REPLACE** is specified).

CONTROL(*string*)
Specifies how the service is to be started and stopped:

MANUAL
The service is not to be started automatically or stopped automatically. It is to be controlled by use of the **START SERVICE** and **STOP SERVICE** commands.

QMGR
The service being defined is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY
The service is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

DESCR(*string*)
Plain-text comment. It provides descriptive information about the service when an operator issues the **DISPLAY SERVICE** command (see "DISPLAY SERVICE on Multiplatforms" on page 912).

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

LIKE(*service-name*)
The name of a service the parameters of which are used to model this definition.

This parameter applies only to the **DEFINE SERVICE** command.

If this field is not completed, and you do not complete the parameter fields related to the command, the values are taken from the default definition for services on this queue manager. Not completing this parameter is equivalent to specifying:

```
LIKE(SYSTEM.DEFAULT.SERVICE)
```

A default service is provided but it can be altered by the installation of the default values required. See Rules for naming IBM MQ objects.

REPLACE and NOREPLACE
Whether the existing definition is to be replaced with this one.

This parameter applies only to the **DEFINE SERVICE** command.

REPLACE
The definition must replace any existing definition of the same name. If a definition does not exist, one is created.

NOREPLACE
The definition should not replace any existing definition of the same name.

SERVTYPE
Specifies the mode in which the service is to run:

COMMAND
A command service object. Multiple instances of a command service object can be executed concurrently. You cannot monitor the status of command service objects.

SERVER
A server service object. Only one instance of a server service object can be executed at a time. The status of server service objects can be monitored using the **DISPLAY SVSTATUS** command.

STARTARG(*string*)
Specifies the arguments to be passed to the user program at queue manager startup.

STARTCMD(*string*)
Specifies the name of the program which is to run. You must specify a fully qualified path name to the executable program.

STDERR(*string*)
Specifies the path to a file to which the standard error (stderr) of the service program is

redirected. If the file does not exist when the service program is started, the file is created. If this value is blank then any data written to stderr by the service program is discarded.

STDOUT(*string*)
Specifies the path to a file to which the standard output (stdout) of the service program is redirected. If the file does not exist when the service program is started, the file is created. If this value is blank then any data written to stdout by the service program is discarded.

STOPARG(*string*)
Specifies the arguments to be passed to the stop program when instructed to stop the service.

STOPCMD(*string*)
Specifies the name of the executable program to run when the service is requested to stop. You must specify a fully qualified path name to the executable program.

Replaceable inserts can be used for any of the **STARTCMD**, **STARTARG**, **STOPCMD**, **STOPARG**, **STDOUT** or **STDERR** strings, for more information, see Replaceable inserts on service definitions.

Related reference:

"DEFINE SERVICE on Multiplatforms" on page 691
Use the MQSC command **DEFINE SERVICE** to define a new IBM MQ service definition, and set its parameters.

"DISPLAY SVSTATUS on Multiplatforms" on page 931
Use the MQSC command **DISPLAY SVSTATUS** to display status information for one or more services. Only services with a **SERVTYPE** of SERVER are displayed.

"START SERVICE on Multiplatforms" on page 1039
Use the MQSC command **START SERVICE** to start a service. The identified service definition is started within the queue manager and inherits the environment and security variables of the queue manager.

"STOP SERVICE on Multiplatforms" on page 1057
Use the MQSC command **STOP SERVICE** to stop a service.

Related information:

Working with services

Examples of using service objects

ALTER SMDS on z/OS: z/OS

Use the MQSC command **ALTER SMDS** to alter the parameters of existing IBM MQ definitions relating to one or more shared message data sets associated with a specific application structure. It is only supported when the CFSTRUCT definition is using the option OFFLOAD(SMDS).

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

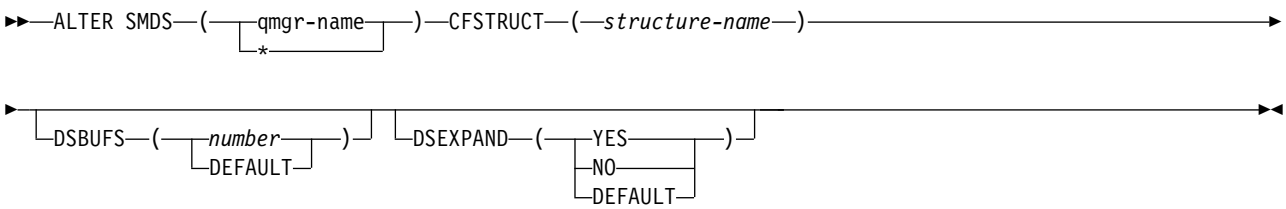
Parameters not specified in the **ALTER SMDS** command result in the existing values for those parameters being left unchanged.

You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for ALTER SMDS" on page 529

Synonym:

ALTER SMDS



Parameter descriptions for ALTER SMDS

SMDS(*qmgr-name*|*)
Specify the queue manager for which the shared message data set properties are to be modified, or an asterisk to modify the properties for all data sets associated with the specified CFSTRUCT.

CFSTRUCT(*structure-name*)
Specify the coupling facility application structure for which the properties of one or more shared message data sets are to be modified.

DSBUFS(number|DEFAULT)
Specify an override value for the number of buffers to be allocated in the specified queue manager or queue managers for accessing shared message data sets for this structure, as a number in the range 1 to 9999, or specify DEFAULT to cancel a previous override and resume using the **DSBUFS** value from the CFSTRUCT definition. The size of each buffer is equal to the logical block size. SMDS buffers are allocated in memory objects residing in z/OS 64-bit storage (above the bar).

When this parameter is altered, any affected queue managers which are already connected to the structure dynamically increase or decrease the number of data set buffers being used for this structure to match the new value. If the specified target value cannot be reached, the affected queue manager replaces the specified **DSBUFS** parameter with the actual new number of buffers. If the queue manager is not active, the change will come into effect when the queue manager is restarted.

DSEXPAND(YES|NO|DEFAULT)
Specify an override value to be used by the specified queue manager or queue managers to control expansion of shared message data sets for this structure.

This parameter controls whether the queue manager should expand a shared message data set when it becomes nearly full, and further blocks are required in the data set.

YES Expansion is supported.

Each time expansion is required, the data set is expanded by the secondary allocation specified when the data set was defined. If no secondary allocation was specified, or it was specified as zero, then a secondary allocation amount of approximately 10% of the existing size is used.

NO No automatic data set expansion is to take place.

DEFAULT
Cancels a previous override.

If you used DEFAULT to cancel a previous override it resumes using the **DSEXPAND** value from the CFSTRUCT definition.

If an expansion attempt fails, the **DSEXPAND** override for the affected queue manager is automatically changed to NO to prevent further expansion attempts, but it can be changed back to YES using the **ALTER SMDS** command to enable further expansion attempts.

When this parameter is altered, any affected queue managers which are already connected to the structure immediately start using the new parameter value.

ALTER STGCLASS on z/OS: z/OS

Use the MQSC command **ALTER STGCLASS** to alter the characteristics of a storage class.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

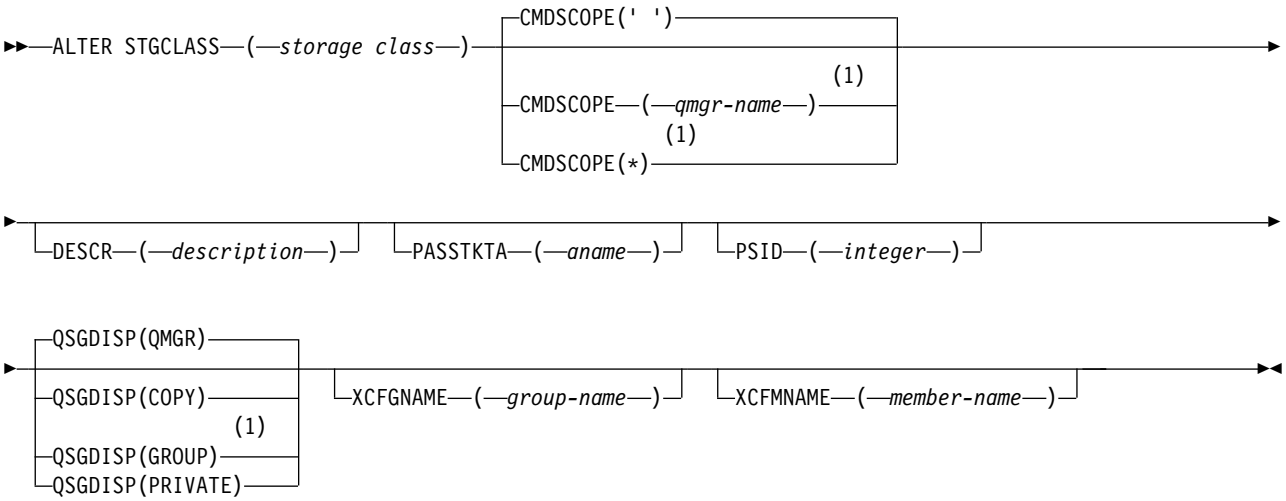
Parameters not specified in the **ALTER STGCLASS** command result in the existing values for those parameters being left unchanged.

You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for ALTER STGCLASS"

Synonym: ALT STC

ALTER STGCLASS



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

Parameter descriptions for ALTER STGCLASS

(storage-class)
Name of the storage class.

This name is one to 8 characters. The first character is in the range A through Z; subsequent characters are A through Z or 0 through 9.

Note: Exceptionally, certain all numeric storage class names are allowed, but are reserved for the use of IBM service personnel.

The storage class must not be the same as any other storage class currently defined on this queue manager.

CMDSCOPE

This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if **QSGDISP** is set to GROUP.

' ' The command runs on the queue manager on which it was entered.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

DESCR(*description*)

Plain-text comment. It provides descriptive information about the object when an operator issues the **DISPLAY STGCLASS** command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager

PASSTKTA(*application name*)

The application name that is passed to RACF when authenticating the PassTicket specified in the MQIIH header.

PSID(*integer*)

The page set identifier that this storage class is to be associated with.

Note: No check is made that the page set has been defined; an error is raised only when you try to put a message to a queue that specifies this storage class (MQRC_PAGESET_ERROR).

The string consists of two numeric characters, in the range 00 through 99. See "DEFINE PSID on z/OS" on page 656.

QSGDISP

Specifies the disposition of the object in the group.

QSGDISP	ALTER
COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters **QSGDISP(COPY)**. Any object residing in the shared repository, or any object defined using a command that had the parameters **QSGDISP(QMGR)**, is not affected by this command.

QSGDISP	ALTER
GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameters **QSGDISP(GROUP)**. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command. If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero: `DEFINE STGCLASS(storage-class)` `REPLACE QSGDISP(COPY)` The ALTER for the group object takes effect regardless of whether the generated command with **QSGDISP(COPY)** fails.
PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with **QSGDISP(QMGR)** or **QSGDISP(COPY)**. Any object residing in the shared repository is unaffected.
QMGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters **QSGDISP(QMGR)**. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

XCFGNAME(*group name*)
If you are using the IMS bridge, this name is the name of the XCF group to which the IMS system belongs. (This name is the group name specified in the IMS parameter list.)

This name is 1 - 8 characters. The first character is in the range A through Z; subsequent characters are A through Z or 0 - 9.

XCFMNAME(*member name*)
If you are using the IMS bridge, this name is the XCF member name of the IMS system within the XCF group specified in XCFGNAME. (This name is the member name specified in the IMS parameter list.)

This name is 1 - 16 characters. The first character is in the range A through Z; subsequent characters are A through Z or 0 - 9.

ALTER SUB:

Use the MQSC command **ALTER SUB** to alter the characteristics of an existing subscription.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

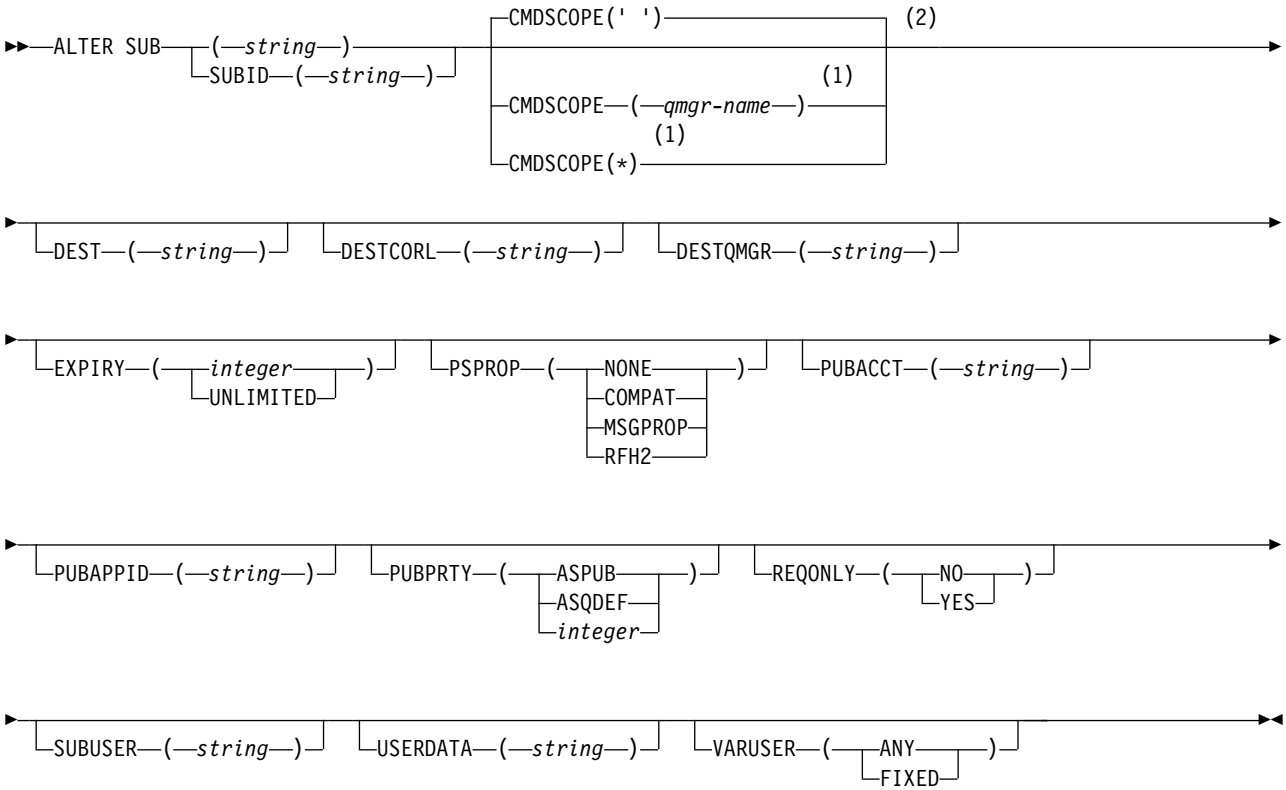
Parameters not specified in the **ALTER SUB** command result in the existing values for those parameters being left unchanged.

z/OS You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for ALTER SUB" on page 533
- "Parameter descriptions for ALTER SUB" on page 534

Synonym: **ALT SUB**

ALTER SUB



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes for ALTER SUB

1. The following forms are valid for the command:

   ```
   ALT SUB(xyz)
   ALT SUB SUBID(123)
   ALT SUB(xyz) SUBID(123)
   ```

2. Although permitted on the **DEFINE** command, you cannot alter the following fields using **DEFINE SUB (REPLACE)**:
   - **TOPICOBJ**
   - **TOPICSTR**
   - **WSCHEMA**
   - **SELECTOR**
   - **SUBSCOPE**
   - **DESTCLAS**
3. At the time the **ALT SUB** command processes, no check is performed that the named **DEST** or **DESTQMGR** exists. These names are used at publishing time as the *ObjectName* and *ObjectQMgrName* for an MQOPEN call. These names are resolved according to the IBM MQ name resolution rules.
4. V 9.0.2 V 9.0.0.1 Subscriptions with a **SUBTYPE** of PROXY cannot be modified. Attempts to modify a proxy subscription by using the PCF interface return MQRCCF_SUBSCRIPTION_IN_USE. MQSC reports the following message:

```
AMQ8469: IBM MQ subscription SYSTEM.PROXY in use.
```

Parameter descriptions for ALTER SUB

(string)
A mandatory parameter. Specifies the unique name for this subscription, see **SUBNAME** property.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of setting this value is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

DEST(*string*)
The destination for messages published to this subscription; this parameter is the name of a queue.

DESTCORL(*string*)
The **CorrelId** used for messages published to this subscription.

A blank value (default) results in a system generated correlation identifier being used.

If set to ' 00000000000000000000000000000000000000000000000000000000 ' (48 zeros) the **CorrelId** set by the publishing application will be maintained in the copy of the message delivered to the subscription, unless messages are propagated across a publish/subscribe hierarchy.

DESTQMGR(*string*)
The destination queue manager for messages published to this subscription. You must define the channels to the remote queue manager, for example, the XMITQ, and a sender channel. If you do not, messages do not arrive at the destination.

EXPIRY
The time to expiry of the subscription object from the creation date and time.

(*integer*)
The time to expiry, in tenths of a second, from the creation date and time.

UNLIMITED
There is no expiry time. This is the default option supplied with the product.

PSPROP
The manner in which publish subscribe related message properties are added to messages sent to this subscription.

NONE Do not add publish subscribe properties to the message.

COMPAT Publish/subscribe properties are added within an MQRFH version 1 header unless the message was published in PCF format.

MSGPROP
Publish/subscribe properties are added as message properties.

RFH2 Publish/subscribe properties are added within an MQRFH version 2 header.

PUBACCT(*string*)
Accounting token passed by the subscriber, for propagation into messages published to this subscription in the `AccountingToken` field of the MQMD.

PUBAPPID(*string*)
Identity data passed by the subscriber, for propagation into messages published to this subscription in the `ApplIdentityData` field of the MQMD.

PUBPRTY
The priority of the message sent to this subscription.

ASPUB Priority of the message sent to this subscription is taken from the priority supplied in the published message.

ASQDEF Priority of the message sent to this subscription is taken from the default priority of the queue defined as a destination.

(integer)
An integer providing an explicit priority for messages published to this subscription.

REQONLY
Indicates whether the subscriber polls for updates using the MQSUBRQ API call, or whether all publications are delivered to this subscription.

NO All publications on the topic are delivered to this subscription.

YES Publications are only delivered to this subscription in response to an MQSUBRQ API call.

This parameter is equivalent to the subscribe option MQSO_PUBLICATIONS_ON_REQUEST.

SUBLEVEL(*integer*)
The level within the subscription hierarchy at which this subscription is made. The range is zero through 9.

SUBUSER(*string*)
Specifies the user ID that is used for security checks that are performed to ensure that publications can be put to the destination queue associated with the subscription. This ID is either the user ID associated with the creator of the subscription or, if subscription takeover is permitted, the user ID that last took over the subscription. The length of this parameter must not exceed 12 characters.

USERDATA(*string*)
Specifies the user data associated with the subscription. The string is a variable length value that can be retrieved by the application on an MQSUB API call and passed in a message sent to this subscription as a message property. The **USERDATA** is stored in the RFH2 header in the mqps folder with the key Sud.

V 9.0.2 V 9.0.0.2 An IBM MQ classes for JMS application can retrieve the subscription user data from the message by using the constant JMS_IBM_SUBSCRIPTION_USER_DATA. For more information, see Retrieval of user subscription data.

VARUSER
Specifies whether a user other than the subscription creator can connect to and take over ownership of the subscription.

ANY Any user can connect to and takeover ownership of the subscription.

FIXED Takeover by another USERID is not permitted.

Related information:
Changing local subscription attributes

ALTER TOPIC:

Use the MQSC **ALTER TOPIC** command to alter the parameters of an existing IBM MQ topic object.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

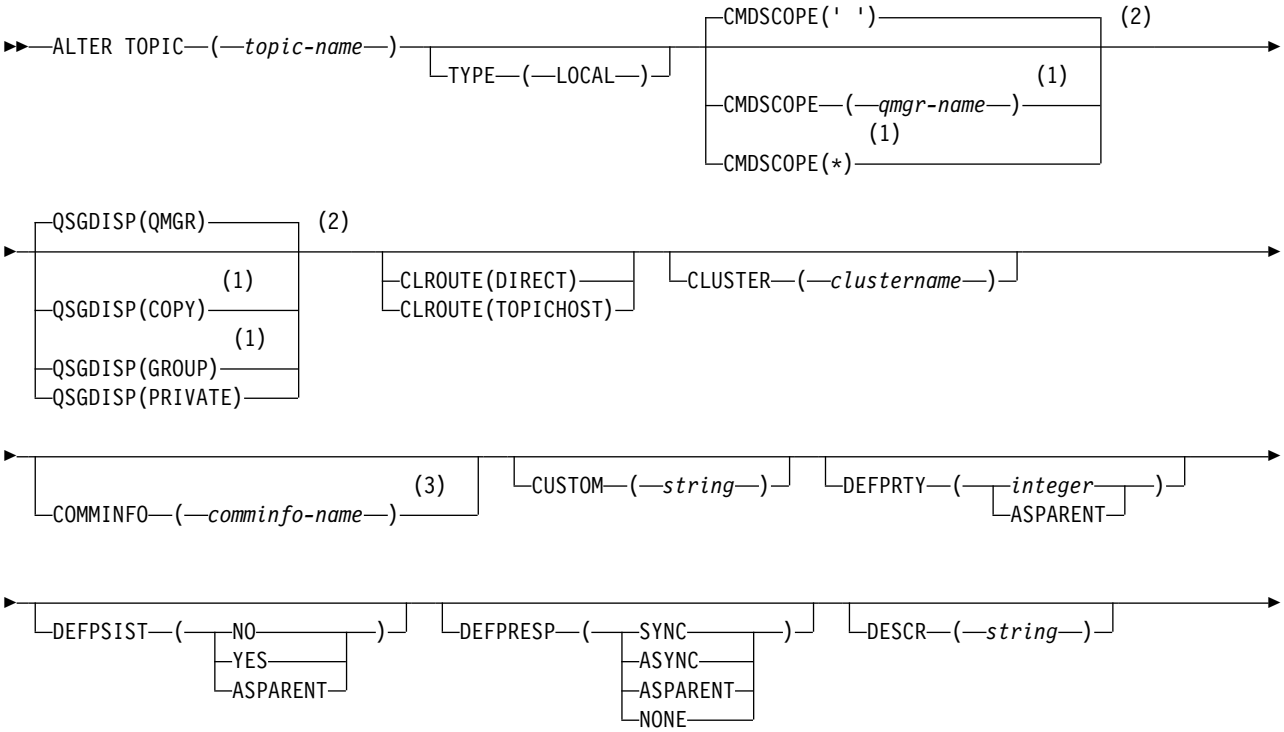
Parameters not specified in the **ALTER TOPIC** command result in the existing values for those parameters being left unchanged.

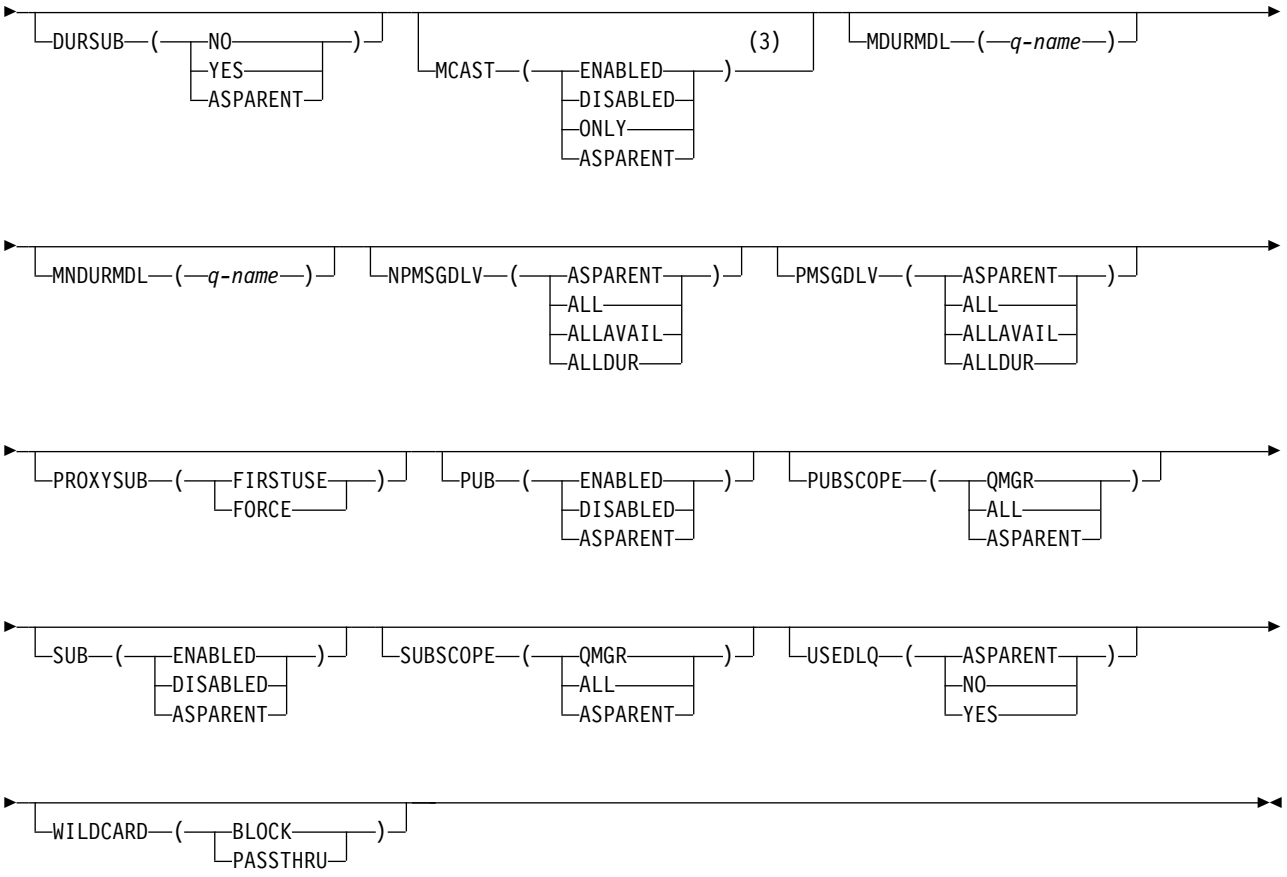
- Syntax diagram
- "Usage notes for ALTER TOPIC" on page 537
- "Parameter descriptions for ALTER TOPIC" on page 537

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

Synonym: ALT TOPIC

ALTER TOPIC





Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Not valid on z/OS.

Usage notes for ALTER TOPIC

- Successful completion of the command does not mean that the action completed. To check for true completion, see the ALTER TOPIC step in Checking that async commands for distributed networks have finished.

Parameter descriptions for ALTER TOPIC

(topic-name)
Name of the IBM MQ topic definition (see Rules for naming IBM MQ objects). The maximum length is 48 characters.

The name must not be the same as any other topic definition currently defined on this queue manager (unless REPLACE is specified).

CLROUTE
The routing behavior to use for topics in the cluster defined by the **CLUSTER** parameter.

DIRECT When you configure a direct routed clustered topic on a queue manager, all queue managers in the cluster become aware of all other queue managers in the cluster. When performing publish and subscribe operations, each queue manager can connect direct to any other queue manager in the cluster.

TOPICHOST
When you use topic host routing, all queue managers in the cluster become aware of the cluster queue managers that host the routed topic definition (that is, the queue managers on which you have defined the topic object). When performing publish and subscribe operations, queue managers in the cluster connect only to these topic host queue managers, and not directly to each other. The topic host queue managers are responsible for routing publications from queue managers on which publications are published to queue managers with matching subscriptions.

After a topic object has been clustered (through setting the **CLUSTER** property) you cannot change the value of the **CLROUTE** property. The object must be un-clustered (**CLUSTER** set to `' '`) before you can change the value. Un-clustering a topic converts the topic definition to a local topic, which results in a period during which publications are not delivered to subscriptions on remote queue managers; this should be considered when performing this change. See The effect of defining a non-cluster topic with the same name as a cluster topic from another queue manager. If you try to change the value of the **CLROUTE** property while it is clustered, the system generates an `MQRCCF_CLROUTE_NOT_ALTERABLE` exception.

See also Routing for publish/subscribe clusters: Notes® on behavior and Designing publish/subscribe clusters.

CLUSTER
The name of the cluster to which this topic belongs.Setting this parameter to a cluster that this queue manager is a member of makes all queue managers in the cluster aware of this topic. Any publication to this topic or a topic string below it put to any queue manager in the cluster is propagated to subscriptions on any other queue manager in the cluster. For more details, see Distributed publish/subscribe networks.

`' '` If no topic object above this topic in the topic tree has set this parameter to a cluster name, then this topic does not belong to a cluster. Publications and subscriptions for this topic are not propagated to publish/subscribe cluster-connected queue managers. If a topic node higher in the topic tree has a cluster name set, publications and subscriptions to this topic are also propagated throughout the cluster.

string The topic belongs to this cluster. It is not recommended that this is set to a different cluster from a topic object above this topic object in the topic tree. Other queue managers in the cluster will honor this object's definition unless a local definition of the same name exists on those queue managers.

To prevent all subscriptions and publications being propagated throughout a cluster, leave this parameter blank on the system topics SYSTEM.BASE.TOPIC and SYSTEM.DEFAULT.TOPIC, except in special circumstances, for example to support migration.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if **QSGDISP** is set to `GROUP`.

`' '` The command runs on the queue manager on which it was entered.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

`*` The command runs on the local queue manager and is also passed to every active queue

manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

COMMINFO(*comminfo-name*)
The name of the communication information object associated with this topic object.

CUSTOM(*string*)
The custom attribute for new features.

This attribute contains the values of attributes, as pairs of attribute name and value, separated by at least one space. The attribute name-value pairs have the form `NAME(VALUE)`. Single quotation marks must be escaped with another single quotation mark.

CAPEXPRY(*integer*)
The maximum time, expressed in tenths of a second, until a message published to a topic which inherits properties from this object, remains in the system until it becomes eligible for expiry processing.

For more information on message expiry processing, see Enforcing lower expiration times.

integer
The value must be in the range one through to 999 999 999.

NOLIMIT
There is no limit on the expiry time of messages put to this topic.

ASPARENT
The maximum message expiry time is based on the setting of the closest parent administrative topic object in the topic tree. This is the default value.

Specifying a value for `CAPEXPRY` that is not valid, does not cause the command to fail. Instead, the default value is used.

DEFPRTY(*integer*)
The default priority of messages published to the topic.

(*integer*)
The value must be in the range zero (the lowest priority), through to the **MAXPRTY** queue manager parameter (**MAXPRTY** is 9).

ASPARENT
The default priority is based on the setting of the closest parent administrative topic object in the topic tree.

DEFPSIST
Specifies the message persistence to be used when applications specify the MQPER_PERSISTENCE_AS_TOPIC_DEF option.

ASPARENT
The default persistence is based on the setting of the closest parent administrative topic object in the topic tree.

NO Messages on this queue are lost during a restart of the queue manager.

YES Messages on this queue survive a restart of the queue manager.

On z/OS, `N` and `Y` are accepted as synonyms of `NO` and `YES`.

DEFPRESP
Specifies the put response to be used when applications specify the MQPMO_RESPONSE_AS_DEF option.

ASPARENT
The default put response is based on the setting of the closest parent administrative topic object in the topic tree.

SYNC Put operations to the queue that specify MQPMO_RESPONSE_AS_Q_DEF are issued as if MQPMO_SYNC_RESPONSE had been specified instead. Fields in the MQMD and MQPMO are returned by the queue manager to the application.

ASYNC Put operations to the queue that specify MQPMO_RESPONSE_AS_Q_DEF are always issued as if MQPMO_ASYNC_RESPONSE had been specified instead. Some fields in the MQMD and MQPMO are not returned by the queue manager to the application. However, an improvement in performance might be seen for messages put in a transaction and any non-persistent messages

DESCR(*string*)
Plain-text comment. It provides descriptive information about the object when an operator issues the **DISPLAY TOPIC** command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

DURSUB
Specifies whether applications are permitted to make durable subscriptions on this topic.

ASPARENT
Whether durable subscriptions can be made on this topic is based on the setting of the closest parent administrative topic object in the topic tree.

NO Durable subscriptions cannot be made on this topic.

YES Durable subscriptions can be made on this topic.

MCAST
Specifies whether multicast is allowable in the topic tree. The values are:

ASPARENT
The multicast attribute of the topic is inherited from the parent.

DISABLED
No multicast traffic is allowed at this node.

ENABLED
Multicast traffic is allowed at this node.

ONLY Only subscriptions from a multicast capable client are allowed.

MDURMDL(*string*)
The name of the model queue to be used for durable subscriptions that request that the queue manager manages the destination of its publications (see Rules for naming IBM MQ objects). The maximum length is 48 characters.

If **MDURMDL** is blank, it operates in the same way as ASPARENT values on other attributes. The name of the model queue to be used is based on the closest parent administrative topic object in the topic tree with a value set for **MDURMDL**.

If you use **MDURMDL** to specify a model queue for a clustered topic, you must ensure that the queue is defined on every queue manager in the cluster where a durable subscription using this topic can be made.

The dynamic queue created from this model has a prefix of SYSTEM.MANAGED.DURABLE

MNDURMDL(*string*)
The name of the model queue to be used for non-durable subscriptions that request that the queue manager manages the destination of its publications (see Rules for naming IBM MQ objects). The maximum length is 48 characters.

If **MNDURMDL** is blank, it operates in the same way as ASPARENT values on other attributes. The name of the model queue to be used is based on the closest parent administrative topic object in the topic tree with a value set for **MNDURMDL**.

If you use **MNDURMDL** to specify a model queue for a clustered topic, you must ensure that the queue is defined on every queue manager in the cluster where a non-durable subscription using this topic can be made.

The dynamic queue created from this model has a prefix of SYSTEM.MANAGED.NDURABLE.

NPMSGDLV
The delivery mechanism for non-persistent messages published to this topic:

ASPARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

ALL Non-persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT call fails.

ALLAVAIL
Non-persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

ALLDUR Non-persistent messages must be delivered to all durable subscribers. Failure to deliver a non-persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no subscribers receive the message and the MQPUT calls fails.

PMSGDLV
The delivery mechanism for persistent messages published to this topic:

ASPARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

ALL Persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT call fails.

ALLAVAIL
Persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

ALLDUR Persistent messages must be delivered to all durable subscribers. Failure to deliver a persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no subscribers receive the message and the MQPUT calls fails.

PROXYSUB
Controls when a proxy subscription is sent for this topic, or topic strings below this topic, to neighboring queue managers when in a publish/subscribe cluster or hierarchy. For more details, see Subscription performance in publish/subscribe networks.

FIRSTUSE
For each unique topic string at or below this topic object, a proxy subscription is asynchronously sent to all neighboring queue managers when a local subscription is created or a proxy subscription is received that is propagated to further directly connected queue managers in a hierarchy.

FORCE A wildcard proxy subscription that matches all topic strings at and below this point in the topic tree is sent to neighboring queue managers even if no local subscriptions exist.

Note: The proxy subscription is sent when this value is set on **DEFINE** or **ALTER**. When set on a clustered topic, all queue managers in the cluster issue the wildcard proxy subscription to all other queue managers in the cluster.

PUB Controls whether messages can be published to this topic.

ASPARENT
Whether messages can be published to the topic is based on the setting of the closest parent administrative topic object in the topic tree.

ENABLED
Messages can be published to the topic (by suitably authorized applications).

DISABLED
Messages cannot be published to the topic.

See also Special handling for the **PUB** parameter.

PUBSCOPE
Determines whether this queue manager propagates publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster.

Note: You can restrict the behavior on a publication-by-publication basis, using MQPMO_SCOPE_QMGR on the Put Message options.

ASPARENT
Whether this queue manager propagates publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster is based on the setting of the first parent administrative node found in the topic tree that relates to this topic.

QMGR Publications for this topic are not propagated to connected queue managers.

ALL Publications for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

z/OS **QSGDISP**
This parameter applies to z/OS only.

Specifies the disposition of the object within the group.

QSGDISP	ALTER
COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters **QSGDISP(COPY)**. Any object residing in the shared repository, or any object defined using a command that had the parameters **QSGDISP(QMGR)**, is not affected by this command.

QSGDISP	ALTER
GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameters **QSGDISP(GROUP)**. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command. If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero: `DEFINE TOPIC(name)` `REPLACE QSGDISP(COPY)` The ALTER for the group object takes effect regardless of whether the generated command with **QSGDISP(COPY)** fails.
PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with **QSGDISP(QMGR)** or **QSGDISP(COPY)**. Any object residing in the shared repository is unaffected.
QMGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters **QSGDISP(QMGR)**. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

SUB Controls whether applications are to be permitted to subscribe to this topic.

ASPARENT
Whether applications can subscribe to the topic is based on the setting of the closest parent administrative topic object in the topic tree.

ENABLED
Subscriptions can be made to the topic (by suitably authorized applications).

DISABLED
Applications cannot subscribe to the topic.

SUBSCOPE
Determines whether this queue manager subscribes to publications in this queue manager or in the network of connected queue managers. If subscribing to all queue managers, the queue manager propagates subscriptions to them as part of a hierarchy or as part of a publish/subscribe cluster.

Note: You can restrict the behavior on a subscription-by-subscription basis, using **MQPMO_SCOPE_QMGR** on the Subscription Descriptor or **SUBSCOPE(QMGR)** on **DEFINE SUB**. Individual subscribers can override the **SUBSCOPE** setting of ALL by specifying the **MQSO_SCOPE_QMGR** subscription option when creating a subscription.

ASPARENT
Whether this queue manager subscribes to publications in the same way as the setting of the first parent administrative node found in the topic tree relating to this topic.

QMGR Only publications that are published on this queue manager reach the subscriber.

ALL A publication made on this queue manager or on another queue manager reaches the subscriber. Subscriptions for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

TOPICSTR(*string* **)**
The topic string represented by this topic object definition. This parameter is required and cannot contain the empty string.

The topic string must not be the same as any other topic string already represented by a topic object definition.

The maximum length of the string is 10,240 characters.

TYPE (topic-type)
If this parameter is used it must follow immediately after the *topic-name* parameter on all platforms z/OS except z/OS.

LOCAL
A local topic object.

USEDLQ
Determines whether the dead-letter queue is used when publication messages cannot be delivered to their correct subscriber queue.

ASPARENT
Determines whether to use the dead-letter queue using the setting of the closest administrative topic object in the topic tree.

NO Publication messages that cannot be delivered to their correct subscriber queue are treated as a failure to put the message. The MQPUT of an application to a topic fails in accordance with the settings of NPMSGDLV and PMSGDLV.

YES When the DEADQ queue manager attribute provides the name of a dead-letter queue, then it is used. If the queue manager does not provide the name of a dead-letter queue, then the behavior is as for NO.

WILDCARD
The behavior of wildcard subscriptions with respect to this topic.

PASSTHRU
Subscriptions made to a wildcarded topic less specific than the topic string at this topic object receive publications made to this topic and to topic strings more specific than this topic.

BLOCK
Subscriptions made to a wildcarded topic less specific than the topic string at this topic object do not receive publications made to this topic or to topic strings more specific than this topic.

The value of this attribute is used when subscriptions are defined. If you alter this attribute, the set of topics covered by existing subscriptions is not affected by the modification. This scenario applies also if the topology is changed when topic objects are created or deleted; the set of topics matching subscriptions created following the modification of the WILDCARD attribute is created using the modified topology. If you want to force the matching set of topics to be re-evaluated for existing subscriptions, you must restart the queue manager.

Related information:
Changing administrative topic attributes

ALTER TRACE on z/OS: z/OS

Use the MQSC command ALTER TRACE to change the trace events being traced for a particular active queue manager trace. ALTER TRACE stops the specified trace, and restarts it with the altered parameters.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

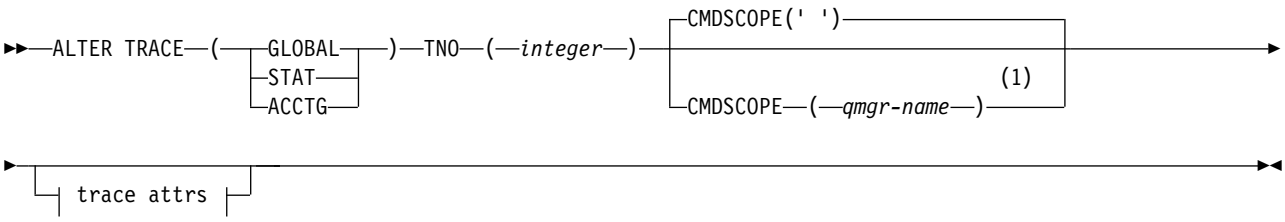
Parameters not specified in the ALTER TRACE command result in the existing values for those parameters being left unchanged.

You can issue this command from sources 12CR. For an explanation of the source symbols, see Using commands on z/OS.

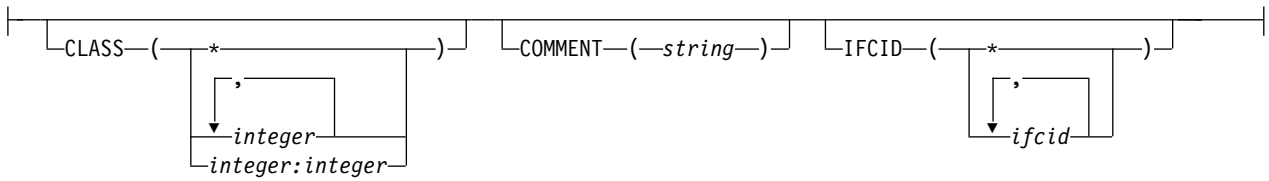
- Syntax diagram
- "Usage notes"
- "Parameter descriptions for ALTER TRACE" on page 546
- "Trace parameters" on page 546

Synonym: ALT TRACE

ALTER TRACE



Trace attrs:



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

Usage notes

Channel initiator traces cannot be altered.

Parameter descriptions for ALTER TRACE

Specify one of the following trace types:

GLOBAL
Service data from the entire queue manager (the synonym is G)

STAT Statistical data (the synonym is S)

ACCTG
Accounting data (the synonym is A)

And:

TNO(*integer* **)**
The number of the trace to be altered (1 through 32). You can specify only one trace number.

CMDSCOPE
This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE cannot be used for commands issued from the first initialization input data set CSQINP1.

' ' The command runs on the queue manager on which it was entered.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

Trace parameters

CLASS(*integer* **)**
The new trace class. See "START TRACE on z/OS" on page 1041 for a list of allowed classes. A range of classes can be specified as *m:n* (for example, CLASS(01:03)). CLASS(*) activates all classes.

COMMENT(*string* **)**
A comment that is reproduced in the trace output record (except in the resident trace tables).

string is any character string. If it includes blanks, commas, or special characters, it must be enclosed between single quotation marks (').

IFCID(*ifcid* **)**
Reserved for IBM Service.

ARCHIVE LOG on z/OS: z/OS

Use the MQSC command ARCHIVE LOG as part of your backup procedure. It takes a copy of the current active log (or both logs if you are using dual logging).

Using MQSC commands

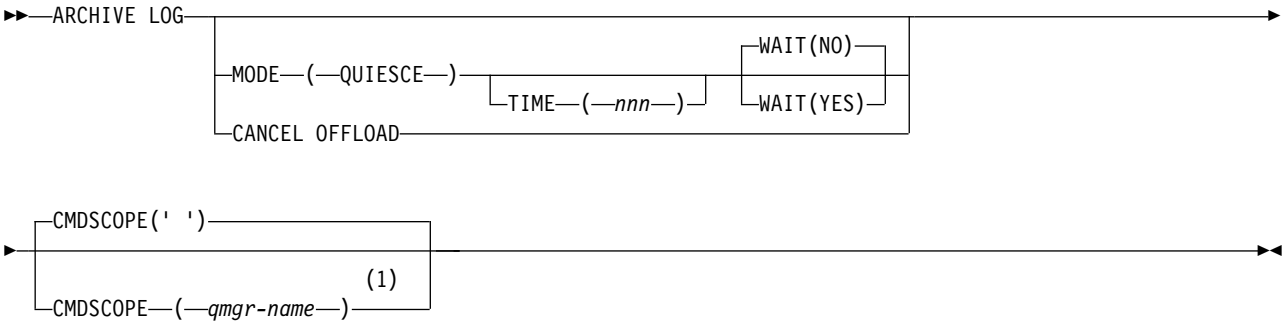
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 12CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for ARCHIVE LOG"
- "Parameter descriptions for ARCHIVE LOG" on page 548

Synonym: ARC LOG

ARCHIVE LOG



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

Usage notes for ARCHIVE LOG

In detail, ARCHIVE LOG:

1. Truncates the current active log data sets.
2. Continues logging, switching to the next active log data set.
3. Starts a task to offload the data sets.
4. Archives previous active log data sets not yet archived.

If the MODE(QUIESCE) parameter is used, the ARCHIVE LOG command quiesces (suspends) all user update activity on the current active log before the offload process. Once a system-wide point of consistency is reached (that is, when all currently active update users have reached a commit point), the current active log data set is immediately truncated, and the offload process is initiated. The resulting point of consistency is captured in the current active log before it is offloaded.

Normally, control returns to the user immediately, and the quiescing is done asynchronously. However, if the WAIT(YES) parameter is used, the quiescing is done synchronously, and control does not return to the user until it has finished.

- You cannot issue an ARCHIVE LOG command while a previous ARCHIVE LOG command is in progress.

- You cannot issue an ARCHIVE LOG command when the active log data set is the last available active log data set, because it would use all the available active log data set space, and IBM MQ would halt all processing until an offload had been completed.
- You can issue an ARCHIVE LOG command without the MODE(QUIESCE) option when a STOP QMGR MODE(QUIESCE) is in progress, but not when a STOP QMGR MODE (FORCE) is in progress.
- You can issue a DISPLAY LOG command to discover whether an ARCHIVE LOG command is active. If an ARCHIVE LOG command is active, the DISPLAY command returns message CSQV400I.
- You can issue an ARCHIVE LOG command even if archiving is not being used (that is, OFFLOAD is set to NO in the CSQ6LOGP system parameter macro), or dynamically using the SET LOG command. In this case, the current active log data sets are truncated and logging continues using the next active log data set, but there is no offloading to archive data sets.

Parameter descriptions for ARCHIVE LOG

All the parameters are optional. If none are specified, the current active log data sets are switched and offloaded immediately.

CANCEL OFFLOAD
Cancels any offloading currently in progress and restarts the offload process. The process starts with the oldest active log data set and proceeds through all the active data sets that need offloading.

Use this command only if the offload task does not appear to be working, or if you want to restart a previous offload attempt that failed.

CMDSCOPE
This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE cannot be used for commands issued from the first initialization input data set CSQINP1.

' '
The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

MODE(QUIESCE)
Stops any new update activity on the queue manager, and brings all existing users to a point of consistency after a commit. When this state is reached, or the number of active users is zero, the current active log is archived.

The time that the queue manager waits to reach such a state is limited to the value specified by QUIESCE in the CSQ6ARVP system parameter macro. The value of QUIESCE can be overridden by the TIME parameter of this command. If activity has not quiesced in that time, the command fails; no offload is done, and logging continues with the current active log data set.

TIME(*nnn*)
Overrides the quiesce time period specified by the QUIESCE value of the CSQ6ARVP system parameter macro.

nnn is the time, in seconds, in the range 001 through 999.

To specify the TIME parameter, you must also specify MODE(QUIESCE).

If you specify the TIME parameter, you must specify an appropriate value for the quiesce period. If you make the period too short or too long, one of the following problems might occur:

- The quiesce might not be complete
- IBM MQ lock contention might develop
- A timeout might interrupt the quiesce

WAIT Specifies whether IBM MQ is to wait until the quiesce process has finished before returning to the issuer of the ARCHIVE LOG command.

To specify the WAIT parameter, you must also specify MODE(QUIESCE).

NO Specifies that control is returned to the issuer when the quiesce process starts. (The synonym is **N**.) This makes the quiesce process asynchronous to the issuer; you can issue further MQSC commands when the ARCHIVE LOG command returns control to you. This is the default.

YES Specifies that control is returned to the issuer when the quiesce process finishes. (The synonym is **Y**.) This makes the quiesce process synchronous to the issuer; further MQSC commands are not processed until the ARCHIVE LOG command finishes.

BACKUP CFSTRUCT on z/OS: z/OS

Use the MQSC command BACKUP CFSTRUCT to initiate a CF application structure backup.

Using MQSC commands

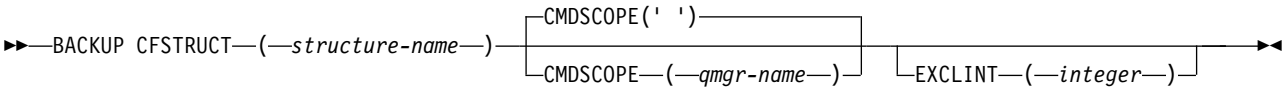
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for BACKUP CFSTRUCT"
- "Keyword and parameter descriptions for BACKUP CFSTRUCT" on page 550

Synonym: None

BACKUP CFSTRUCT



Usage notes for BACKUP CFSTRUCT

1. This command is valid only on z/OS when the queue manager is a member of a queue-sharing group.
2. Only persistent shared queue messages are backed up. Non-persistent messages are not backed up and cannot be recovered
3. You can concurrently run separate backups for different application structures on different queue managers within the queue-sharing group. You can also concurrently run separate backups for different application structures on the same queue manager.
4. This command fails if the specified CF structure is defined with either a CFLEVEL less than 3, or with RECOVER set to NO.

5. The command fails if a specified application structure is currently in the process of being backed up by another queue manager within the queue-sharing group.

Keyword and parameter descriptions for BACKUP CFSTRUCT

structure-name
The name of the coupling facility (CF) application structure to be backed up. An asterisk (*) on its own specifies all recoverable CF structures. A trailing asterisk (*) matches all recoverable structure names with the specified stem followed by zero or more characters. The value (CSQ*) matches all recoverable CF structures with the specified stem (CSQ) followed by zero or more characters.

CMDSCOPE
This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

' '
The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and the command server is enabled.

EXCLINT(*integer*)
Specifies a value that defines a number of seconds that are used as an exclusion time. The backup excludes backing-up activity during this exclusion time. The exclusion time starts immediately before the back up starts. For example, if EXCLINT(30) is specified, the backup does not include the last 30 seconds worth of activity for this application-structure before back up started.

The value must be in the range 30 through 600. The default value is 30.

CLEAR QLOCAL:

Use the MQSC command CLEAR QLOCAL to clear the messages from a local queue.

Using MQSC commands

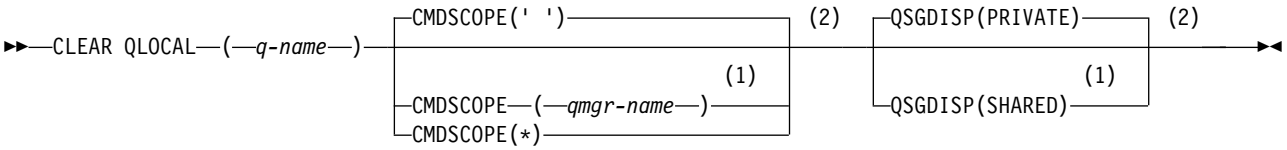
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for CLEAR QLOCAL" on page 551

Synonym: CLEAR QL

CLEAR QLOCAL



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Parameter descriptions for CLEAR QLOCAL

You must specify which local queue you want to clear.

The command fails if either:

- The queue has uncommitted messages that have been put on the queue under syncpoint
- The queue is currently open by an application (with any open options)

If an application has this queue open, or has a queue open that eventually resolves to this queue, the command fails. The command also fails if this queue is a transmission queue, and any queue that is, or resolves to, a remote queue that references this transmission queue, is open.

(q-name)
The name of the local queue to be cleared. The name must be defined to the local queue manager.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to SHARED.

' '
The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

*
The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

z/OS **QSGDISP**
Specifies whether the queue definition is shared. This parameter applies to z/OS only.

PRIVATE
Clear only the private queue named *q-name*. The queue is private if it was defined using a command that had the parameters QSGDISP(COPY) or QSGDISP(QMGR). This is the default value.

SHARED
Clear only the shared queue named *q-name*. The queue is shared if it was defined using a command that had the parameters QSGDISP(SHARED).

Related information:
Clearing a local queue

CLEAR TOPICSTR:

Use the MQSC command CLEAR TOPICSTR to clear the retained message which is stored for the specified topic string.

Using MQSC commands

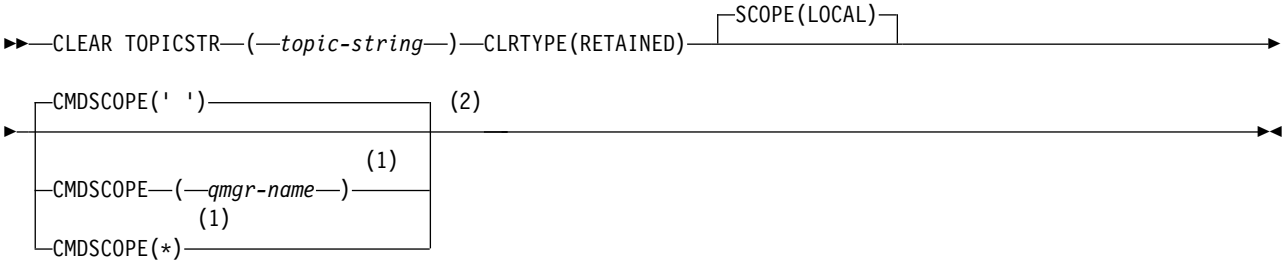
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- Usage notes for CLEAR TOPICSTR
- Parameter descriptions for CLEAR TOPICSTR

Synonym: None.

CLEAR TOPICSTR



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes for CLEAR TOPICSTR

1. If the topic string specified has no retained message the command will complete successfully. You can find out whether a topic string has a retained message by using the DISPLAY TPSTATUS command. The RETAINED field shows whether there is a retained message.
2. The topic-string input parameter on this command must match the topic you want to act on. You are advised to keep the character strings in your topic strings as characters that can be used from location issuing the command. If you issue commands using MQSC, you will have fewer characters available to you than if you are using an application submitting PCF messages, such as the IBM MQ Explorer.
3. You might need to use CLEAR TOPICSTR to remove a retained publication from a publish/subscribe cluster. For example:
   - If you accidentally configure a retained publication, and then need to remove it from all cluster queue managers, you issue this command on all members of the cluster.
   - In a direct routed publish/subscribe cluster, if you move a publishing application to a new queue manager and the previous queue manager holds no subscriptions for the affected topic string, you need to ensure that the previous queue manager does not resend the old retained publication to

other members of the cluster. To do this, wait until the application has published on the new queue manager, then issue this command on the previous queue manager to remove the retained publication held there.

See also Design considerations for retained publications in publish/subscribe clusters

Parameter descriptions for CLEAR TOPICSTR

You must specify which topic string you want to remove the retained publication from.

(topic-string)
The topic string to be cleared. This string can represent several topics to be cleared by using wildcards as shown in the following table:

Special Character	Behavior
#	Wildcard, multiple topic level
+	Wildcard, single topic level

Note: the '+' and '#' are not treated as wildcards if they are mixed in with other characters (including themselves) within a topic level. In the following string, the '#' and '+' characters are treated as ordinary characters.

```
level0/level1/#+/level3/level#
```

To illustrate the effect of wildcards, the following example is used.

Clearing the following topic:

```
/a/b/#/z
```

clears the following topics:

```
/a/b/z
/a/b/c/z
/a/b/c/y/z
```

CLRTYPE
This is a mandatory parameter.

The value must be:

RETAINED
Remove the retained publication from the specified topic string.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the name of the local queue manager, if the shared queue object definition has its queue-sharing group disposition attribute QSGDISP set to SHARED.

' '
The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

*
The command runs on the local queue manager and is also passed to every active queue

manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

SCOPE

The scope of the deletion of retained messages.

The value can be:

LOCAL

The retained message is removed from the specified topic string at the local queue manager only. This is the default value.

DEFINE AUTHINFO:

Use the MQSC command **DEFINE AUTHINFO** to define an authentication information object. These objects contain the definitions required to perform certificate revocation checking using OCSP or Certificate Revocation Lists (CRLs) on LDAP servers, and the definitions required to enable user ID and password checking.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

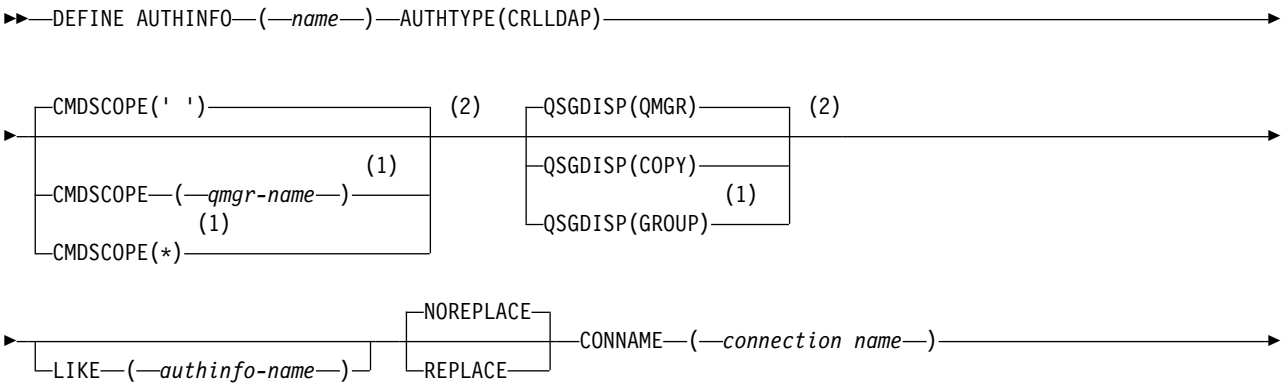
- "Usage notes for DEFINE AUTHINFO" on page 557
- "Parameter descriptions for DEFINE AUTHINFO" on page 557
- Syntax diagram for TYPE(CRLLDAP)
- Syntax diagram for TYPE(OCSP)
- Syntax diagram for TYPE(IDPWOS)
- Syntax diagram for TYPE(IDPWLDAP)

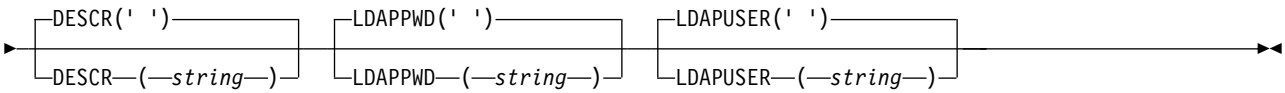
Synonym: DEF AUTHINFO

Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

Syntax diagram for TYPE(CRLLDAP)

DEFINE AUTHINFO





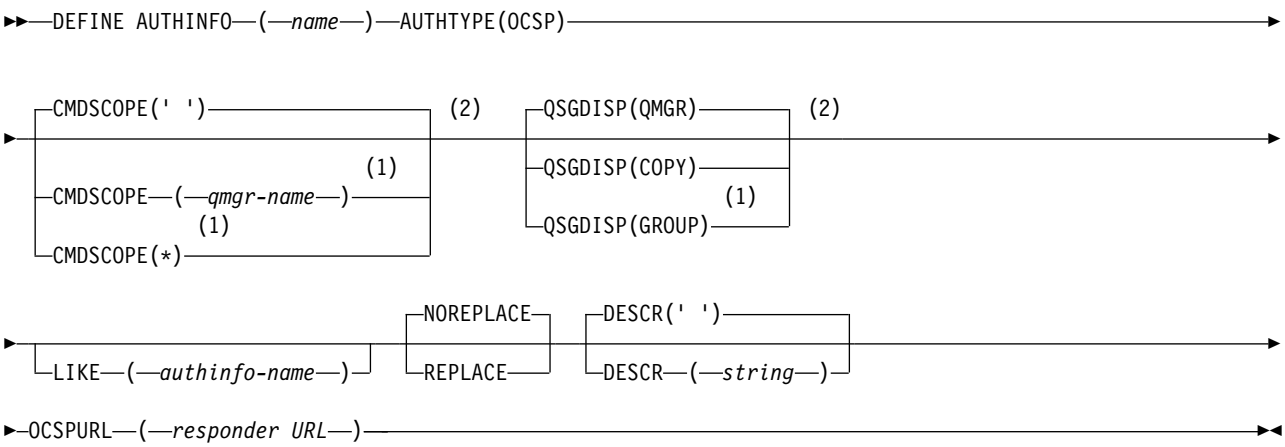
Notes:

1 Valid only when the queue manager is a member of a queue-sharing group. You can use queue-sharing groups only on IBM MQ for z/OS.

2 Valid only on z/OS.

Syntax diagram for TYPE(OCSP)

DEFINE AUTHINFO



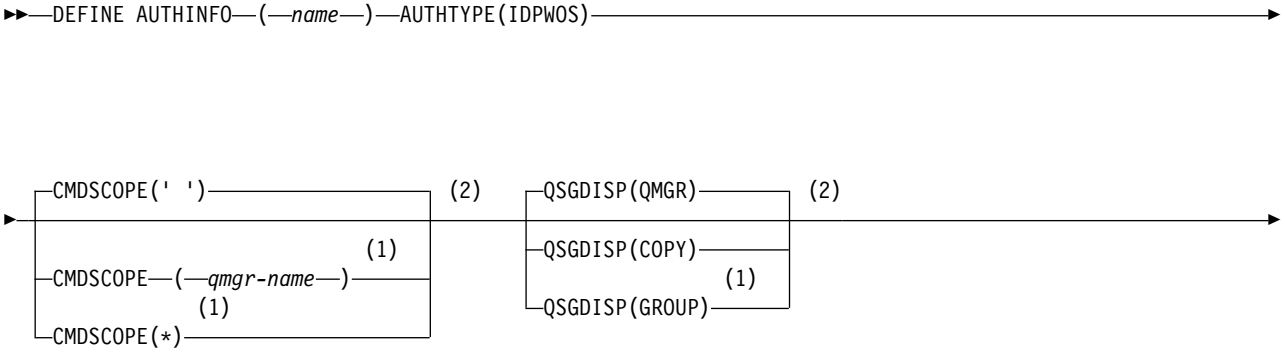
Notes:

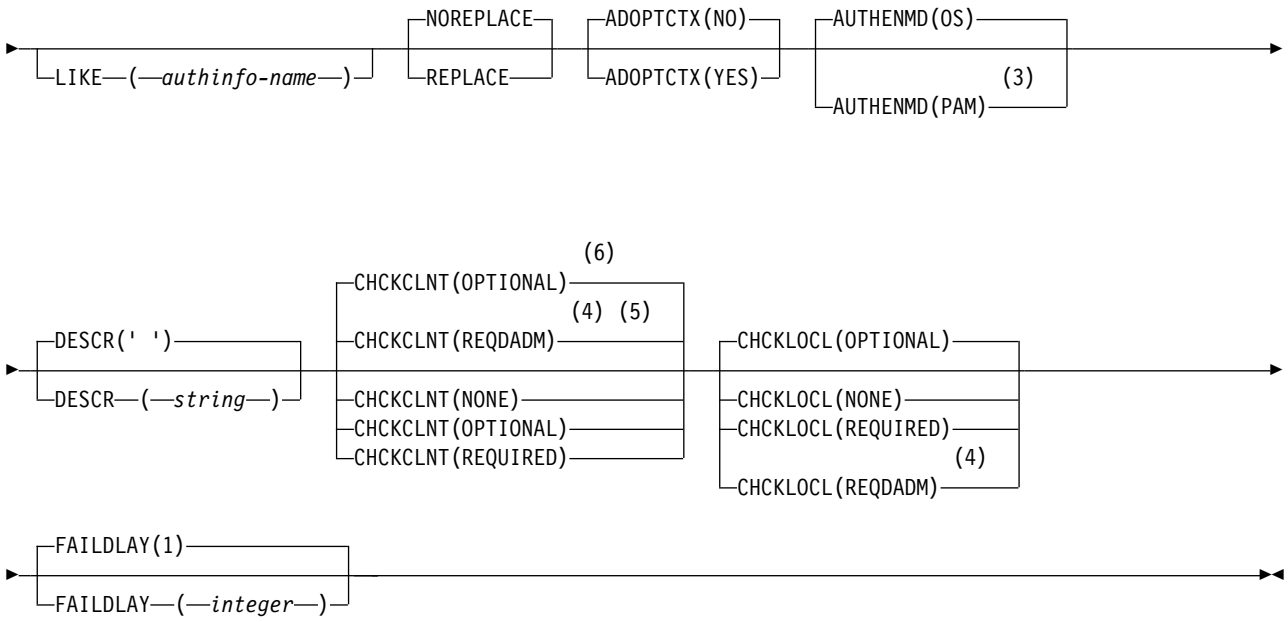
1 Valid only when the queue manager is a member of a queue-sharing group. You can use queue-sharing groups only on IBM MQ for z/OS.

2 Valid only on z/OS.

Syntax diagram for TYPE(IDPWOS)

DEFINE AUTHINFO





Notes:

1 Valid only when the queue manager is a member of a queue-sharing group. You can use queue-sharing groups only on IBM MQ for z/OS.

2 Valid only on z/OS.

3 Not valid on z/OS and PAM value can be set only on UNIX.

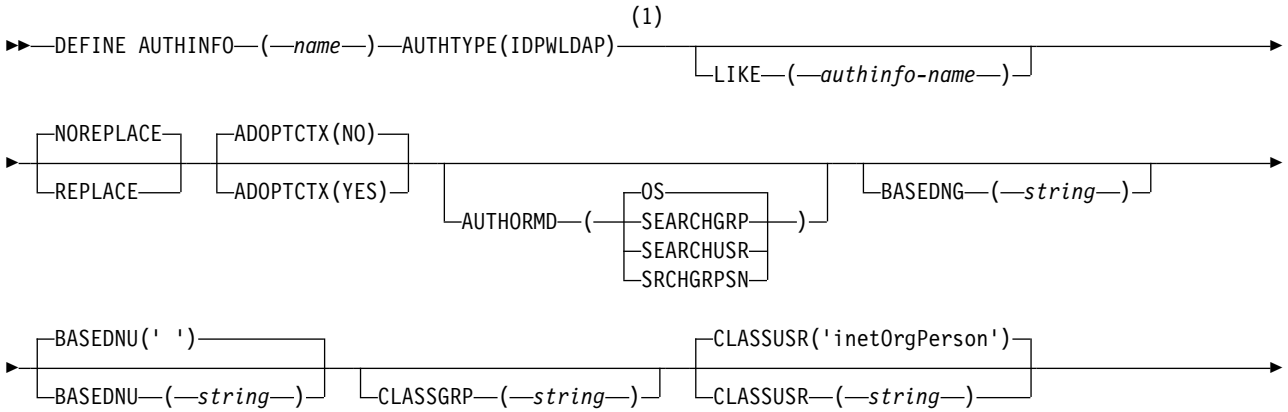
4 Not valid on IBM MQ for z/OS.

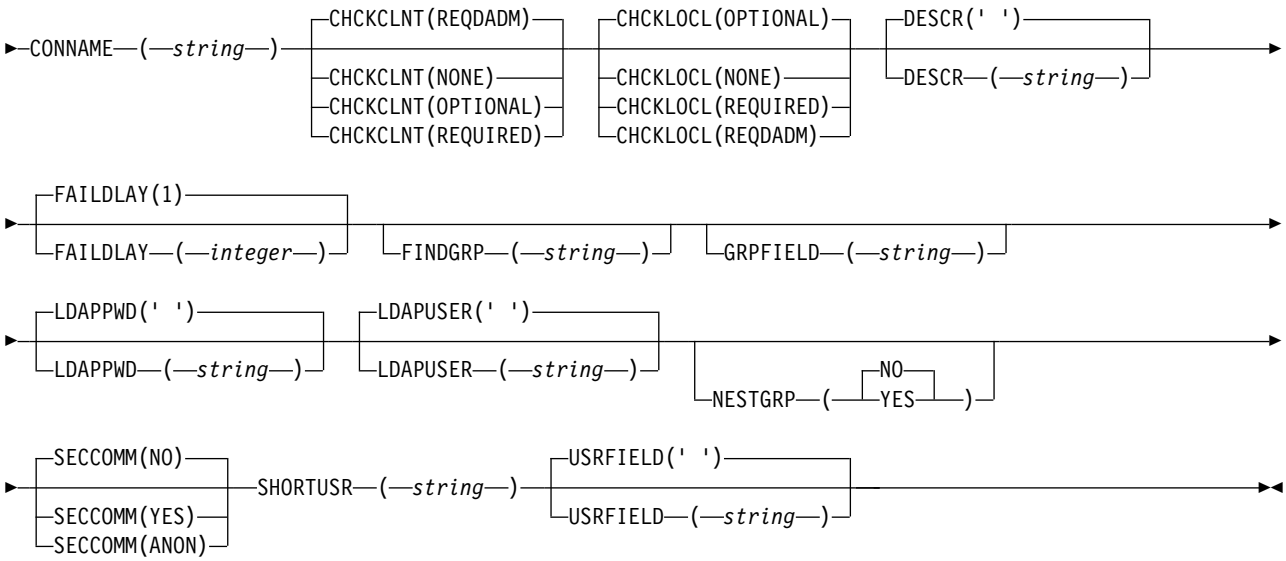
5 Default for platforms other than z/OS.

6 Default for z/OS.

Syntax diagram for TYPE(IDPWLDAP)

DEFINE AUTHINFO





Notes:

1 Not valid on IBM MQ for z/OS.

Usage notes for DEFINE AUTHINFO

IBM i On IBM i, authentication information objects of AUTHTYPE(CRLLDAP) and AUTHTYPE(OCSP) are only used for channels of type CLNTCONN through use of the AMQCLCHL.TAB. Certificates are defined by Digital Certificate Manager for each certificate authority, and are verified against the LDAP servers.

Attention: After running the DEFINE AUTHINFO command, you must restart the queue manager. If you do not restart the queue manager, the setmqaut command does not return the correct result.

Parameter descriptions for DEFINE AUTHINFO

name Name of the authentication information object. This parameter is required.

The name must not be the same as any other authentication information object name currently defined on this queue manager (unless **REPLACE** or **ALTER** is specified). See Rules for naming IBM MQ objects.

ADOPTCTX

Whether to use the presented credentials as the context for this application. This means that they are used for authorization checks, shown on administrative displays, and appear in messages.

YES The user ID presented in the MQCSP structure, which has been successfully validated by password, is adopted as the context to use for this application. Therefore, this user ID will be the credentials checked for authorization to use IBM MQ resources.

If the user ID presented is an LDAP user ID, and authorization checks are done using operating system user IDs, the SHORTUSR associated with the user entry in LDAP will be adopted as the credentials for authorization checks to be done against.

NO Authentication will be performed on the user ID and password presented in the MQCSP structure, but then the credentials will not be adopted for further use. Authorization will be performed using the user ID the application is running under.

This attribute is only valid for an **AUTHTYPE** of IDPWOS and IDPWLDAP.

AUTHENMD
Authentication method. Whether to use the operating system or Pluggable Authentication Method (PAM) to authenticate user passwords.

OS UNIX Use the traditional UNIX password verification method.

UNIX Linux **PAM**
Use the PAM to authenticate the user password.

You can set the PAM value only on UNIX and Linux.

Changes to this attribute are effective only after you run the REFRESH SECURITY TYPE(CONNAUTH) command.

This attribute is valid only for an **AUTHTYPE** of IDPWOS.

AUTHORMD
Authorization Method.

OS Use operating system groups to determine permissions associated with a user.

This is how IBM MQ has previously worked, and is the default value.

SEARCHGRP
A group entry in the LDAP repository contains an attribute listing the Distinguished Name of all the users belonging to that group. Membership is indicated by the attribute defined in FINDGRP. This value is typically *member* or *uniqueMember*.

SEARCHUSR
A user entry in the LDAP repository contains an attribute listing the Distinguished Name of all the groups to which the specified user belongs. The attribute to query is defined by the FINDGRP value, typically *memberOf*.

V 9.0.5 **SRCHGRPSN**
A group entry in the LDAP repository contains an attribute listing the short user name of all the users belonging to that group. The attribute in the user record that contains the short user name is specified by SHORTUSR.

Membership is indicated by the attribute defined in FINDGRP. This value is typically *memberUid*.

Note: This authorization method should only be used if all user short names are distinct.

Many LDAP servers use an attribute of the group object to determine group membership and you should, therefore, set this value to SEARCHGRP.

Microsoft Active Directory typically stores group memberships as a user attribute. The IBM Tivoli Directory Server supports both methods.

In general, retrieving memberships through a user attribute will be faster than searching for groups that list the user as a member.

AUTHTYPE
The type of authentication information.

CRLLDAP
Certificate Revocation List checking is done using LDAP servers.

IDPWLDAP
Connection authentication user ID and password checking is done using an LDAP server.

Attention: z/OS This option is not available on IBM MQ for z/OS

IDPWOS
Connection authentication user ID and password checking is done using the operating system.

OCSP Certificate revocation checking is done using OCSP.

An authentication information object with **AUTHTYPE(OCSP)** does not apply for use on queue managers on the following platforms:

- IBM i
- z/OS

However, it can be specified on those platforms to be copied to the client channel definition table (CCDT) for client use.

This parameter is required.

You cannot define an authentication information object as LIKE one with a different **AUTHTYPE**. You cannot alter the **AUTHTYPE** of an authentication information object after you have created it.

BASEDNG
Base DN for groups

In order to be able to find group names, this parameter must be set with the base DN to search for groups in the LDAP server.

BASEDNU(*base DN*)
In order to be able to find the short user name attribute (see SHORTUSR) this parameter must be set with the base DN to search for users within the LDAP server.

This attribute is valid only for an **AUTHTYPE** of IDPWLDAP.

CHCKCLNT
This attribute determines the authentication requirements for client applications, and is valid only for an **AUTHTYPE** of IDPWOS or IDPWLDAP. The possible values are:

NONE No user ID and password checks are made. If any user ID or password is supplied by a client application, the credentials are ignored.

OPTIONAL
Client applications are not required to provide a user ID and password.

Any applications that do provide a user ID and password in the MQCSP structure have them authenticated by the queue manager against the password store indicated by the **AUTHTYPE**.

The connection is only allowed to continue if the user ID and password are valid.

This option might be useful during migration, for example.

REQUIRED
All client applications must provide a user ID and password in the MQCSP structure. This user ID and password is authenticated by the queue manager against the password store indicated by the **AUTHTYPE**.

The connection will only be allowed to continue if the user ID and password are valid.

REQDADM
All client applications using a privileged user ID must provide a user ID and password in the MQCSP structure. Any locally bound applications using a non-privileged user ID are not required to provide a user ID and password and are treated as with the OPTIONAL setting.

Any provided user ID and password are authenticated by the queue manager against the password store indicated by the **AUTHTYPE**. The connection is only allowed to continue if the user ID and password are valid.

Note: The REQDADM value for the **CHCKCLNT** attribute is irrelevant if the authentication type is LDAP. This is because there is no concept of privileged user ID when using LDAP user accounts. LDAP user accounts and groups must be assigned permission explicitly.

z/OS (This setting is not allowed on z/OS systems.)

Important:

1. This attribute can be overridden by the **CHCKCLNT** attribute of the CHLAUTH rule that matches the client connection. The CONNAUTH *AUTHINFO CHCKCLNT* attribute on the queue manager therefore determines the default client checking behavior for client connections that do not match a CHLAUTH rule, or where the CHLAUTH rule matched has **CHCKCLNT** ASQMGR.
2. If you select NONE and the client connection matches a CHLAUTH record with **CHCKCLNT** REQUIRED (or REQDADM on platforms other than z/OS), the connection fails. You receive the following message:
   - Multi AMQ9793 on Multiplatforms.
   - z/OS CSQX793E on z/OS.
3. This parameter is valid only with **TYPE(USERMAP)**, **TYPE(ADDRESSMAP)** and TYPE (SSLPEERMAP), and only when **USERSRC** is not set to NOACCESS.
4. This parameter applies only to inbound connections that are server-connection channels.

CHCKLOCL

This attribute determines the authentication requirements for locally bound applications, and is valid only for an **AUTHTYPE** of IDPWOS or IDPWLDAP.

IBM MQ Appliance For information about use of this attribute on IBM MQ Appliance, see Control commands on the IBM MQ Appliance in the IBM MQ Appliance documentation. The possible values are:

NONE No user ID and password checks are made. If any user ID or password is supplied by a locally bound application, the credentials are ignored.

OPTIONAL

Locally bound applications are not required to provide a user ID and password.

Any applications that do provide a user ID and password in the MQCSP structure have them authenticated by the queue manager against the password store indicated by the **AUTHTYPE**.

The connection is only allowed to continue if the user ID and password are valid.

This option might be useful during migration, for example.

REQUIRED

All locally bound applications must provide a user ID and password in the MQCSP structure. This user ID and password will be authenticated by the queue manager against the password store indicated by the **AUTHTYPE**. The connection will only be allowed to continue if the user ID and password are valid.

z/OS If your user ID has UPDATE access to the BATCH profile in the MQCONN class, you can treat **CHCKLOCL(REQUIRED)** as if it is **CHCKLOCL(OPTIONAL)**. That is, you do not have to supply a password, but if you do, the password must be the correct one.
See Using **CHCKLOCL** on locally bound applications.

REQDADM

All locally bound applications using a privileged user ID must provide a user ID and password in the MQCSP structure. Any locally bound applications using a non-privileged user ID are not required to provide a user ID and password and are treated as with the OPTIONAL setting.

Any provided user ID and password will be authenticated by the queue manager against the password store indicated by the **AUTHTYPE**. The connection will only be allowed to continue if the user ID and password are valid.

z/OS (This setting is not allowed on z/OS systems.)

CLASSGRP

The LDAP object class used for group records in the LDAP repository.

If the value is blank, `groupOfNames` is used.

Other commonly used values include `groupOfUniqueNames` or `group`.

CLASSUSR(*LDAP class name*)

The LDAP object class used for user records in the LDAP repository.

If blank, the value defaults to *inetOrgPerson*, which is generally the value needed.
For Microsoft Active Directory, the value you require is often *user*.
This attribute is valid only for an **AUTHTYPE** of *IDPWLDAP*.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command runs on the queue manager on which it was entered.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of `*` is the same as entering the command on every queue manager in the queue-sharing group.

CONNAME(*connection name*)

The host name, IPv4 dotted decimal address, or IPv6 hexadecimal notation of the host on which the LDAP server is running, with an optional port number.

This parameter is valid only for **AUTHTYPE(CRLLDAP)**, when it is mandatory.

If you specify the connection name as an IPv6 address, only systems with an IPv6 stack are able to resolve this address. If the AUTHINFO object is part of the CRL namelist of the queue manager, ensure that any clients using the client channel table generated by the queue manager can resolve the connection name.

z/OS On z/OS, if a **CONNAME** is to resolve to an IPv6 network address, a level of z/OS that supports IPv6 for connection to an LDAP server is required.

The syntax for **CONNAME** is the same as for channels. For example,

```
conname(' hostname (nnn)')
```

where *nnn* is the port number.

The maximum length for the field depends on your platform:

- ULW On UNIX, Linux, and Windows, the maximum length is 264 characters.
- IBM i On IBM i, the maximum length is 264 characters.

- z/OS On z/OS, the maximim length is 48 characters.

This attribute is valid only for an **AUTHTYPE** of CRLLDAP and IDPWLDAP, when the attribute is mandatory.

When used with an **AUTHTYPE** of IDPWLDAP, this can be a comma separated list of connection names.

DESCR(*string*)
Plain-text comment. It provides descriptive information about the authentication information object when an operator issues the **DISPLAY AUTHINFO** command (see "DISPLAY AUTHINFO" on page 738).

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

FAILDLAY(*delay time*)
When a user ID and password are provided for connection authentication, and the authentication fails due to the user ID or password being incorrect, this is the delay, in seconds, before the failure is returned to the application.

This can aid in avoiding busy loops from an application that simply retries, continuously, after receiving a failure.

The value must be in the range 0 - 60 seconds. The default value is 1.

This attribute is only valid for an **AUTHTYPE** of IDPWOS and IDPWLDAP.

FINDGRP
Name of the attribute used within an LDAP entry to determine group membership.

When **AUTHORMD** = SEARCHGRP, the **FINDGRP** attribute is typically set to member or uniqueMember.

When **AUTHORMD** = SEARCHUSR, the **FINDGRP** attribute is typically set to memberOf.

V 9.0.5 When **AUTHORMD** = SRCHGRPSN, the **FINDGRP** attribute is typically set to memberUid.

When the **FINDGRP** attribute is left blank:

- If **AUTHORMD** = SEARCHGRP, the **FINDGRP** attribute defaults to memberOf.
- If **AUTHORMD** = SEARCHUSR, the **FINDGRP** attribute defaults to member.
- V 9.0.5 If **AUTHORMD** = SRCHGRPSN, the **FINDGRP** attribute defaults to memberUid.

GRPFIELD
LDAP attribute that represents a simple name for the group.

If the value is blank, commands like **setmqaut** must use a qualified name for the group. The value can either be a full DN, or a single attribute.

LDAPPWD(*LDAP password*)
The password associated with the Distinguished Name of the user who is accessing the LDAP server. Its maximum size is 32 characters.

This attribute is valid only for an **AUTHTYPE** of CRLLDAP and IDPWLDAP.

z/OS On z/OS, the **LDAPPWD** used for accessing the LDAP server might not be the one defined in the **AUTHINFO** object. If more than one **AUTHINFO** object is placed in the namelist referred to by the QMGR parameter **SSLCRLNL**, the **LDAPPWD** in the first **AUTHINFO** object is used for accessing all LDAP servers.

LDAPUSER(*LDAP user*)

The Distinguished Name of the user who is accessing the LDAP server. (See the SSLPEER parameter for more information about distinguished names.)

This attribute is valid only for an **AUTHTYPE** of CRLLDAP and IDPWLDAP.

The maximum size for the user name is as follows:

- Multi 1024 characters on Multiplatforms
- z/OS 256 characters on z/OS

z/OS On z/OS, the **LDAPUSER** used for accessing the LDAP Server might not be the one defined in the **AUTHINFO** object. If more than one **AUTHINFO** object is placed in the namelist referred to by the QMGR parameter **SSLCRLNL**, the **LDAPUSER** in the first **AUTHINFO** object is used for accessing all LDAP servers.

Multi On Multiplatforms, the maximum accepted line length is defined to be BUFSIZ, which can be found in stdio.h.

LIKE(*authinfo-name*)

The name of an authentication information object, with parameters that are used to model this definition.

z/OS On z/OS, the queue manager searches for an object with the name you specify and a disposition of QMGR or COPY. The disposition of the LIKE object is not copied to the object you are defining.

Note:

1. **QSGDISP (GROUP)** objects are not searched.
2. LIKE is ignored if **QSGDISP(COPY)** is specified. However, the group object defined is used as a LIKE object.

NESTGRP

Group nesting.

NO Only the initially discovered groups are considered for authorization.

YES
The group list is searched recursively to enumerate all the groups to which a user belongs.

The group's Distinguished Name is used when searching the group list recursively, regardless of the authorization method selected in AUTHORMD.

OCSPURL(*Responder URL*)

The URL of the OCSP responder used to check for certificate revocation. This value must be an HTTP URL containing the host name and port number of the OCSP responder. If the OCSP responder is using port 80, which is the default for HTTP, then the port number can be omitted. HTTP URLs are defined in RFC 1738.

This field is case sensitive. It must start with the string http:// in lowercase. The rest of the URL might be case sensitive, depending on the OCSP server implementation. To preserve case, use single quotation marks to specify the OCSPURL parameter value, for example:

```
OCSPURL ('http://ocsp.example.ibm.com')
```

This parameter is applicable only for **AUTHTYPE(OCSP)**, when it is mandatory.

z/OS **QSGDISP**

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

QSGDISP	DEFINE
COPY	The object is defined on the page set of the queue manager that executes the command using the **QSGDISP(GROUP)** object of the same name as the LIKE object.
GROUP	The object definition resides in the shared repository. GROUP is allowed only if the queue manager is in a queue-sharing group. If the definition is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to make or refresh local copies on page set zero: `DEFINE AUTHINFO(name)` `REPLACE QSGDISP(COPY)` The DEFINE for the group object takes effect regardless of whether the generated command with **QSGDISP(COPY)** fails.
PRIVATE	Not permitted.
QMGR	The object is defined on the page set of the queue manager that executes the command.

REPLACE and NOREPLACE

Whether the existing definition (and on z/OS, with the same disposition) is to be replaced with this one. This parameter is optional. Any object with a different disposition is not changed.

REPLACE

The definition must replace any existing definition of the same name. If a definition does not exist, one is created.

NOREPLACE

The definition must not replace any existing definition of the same name.

SECCOMM

Whether connectivity to the LDAP server should be done securely using TLS

YES Connectivity to the LDAP server is made securely using TLS.

The certificate used is the default certificate for the queue manager, named in **CERTLABL** on the queue manager object, or if that is blank, the one described in Digital certificate labels, understanding the requirements.

The certificate is located in the key repository specified in **SSLKEYR** on the queue manager object. A cipherspec will be negotiated that is supported by both IBM MQ and the LDAP server.

If the queue manager is configured to use **SSLFIPS(YES)** or **SUITEB** cipher specs, then this is taken account of in the connection to the LDAP server as well.

ANON

Connectivity to the LDAP server is made securely using TLS just as for **SECCOMM(YES)** with one difference.

No certificate is sent to the LDAP server; the connection will be made anonymously. To use this setting, ensure that the key repository specified in **SSLKEYR**, on the queue manager object, does not contain a certificate marked as the default.

NO Connectivity to the LDAP server does not use TLS.

This attribute is valid only for an **AUTHTYPE** of IDPWLDAP.

SHORTUSR(*LDAP field name*)

A field in the user record to be used as a short user name in IBM MQ.

This field must contain values of 12 characters or less. This short user name is used for the following purposes:

- If LDAP authentication is enabled, but LDAP authorization is not enabled, this is used as an operating system user ID for authorization checks. In this case, the attribute must represent an operating system user ID.
- If LDAP authentication and authorization are both enabled, this is used as the user ID carried with the message in order for the LDAP user name to be rediscovered when the user ID inside the message needs to be used.

  For example, on another queue manager, or when writing report messages. In this case, the attribute does not need to represent an operating system user ID, but must be a unique string. An employee serial number is an example of a good attribute for this purpose.

This attribute is valid only for an **AUTHTYPE** of IDPWLDAP and is mandatory.

USRFIELD(*LDAP field name* **)**

If the user ID provided by an application for authentication does not contain a qualifier for the field in the LDAP user record, that is, it does not contain an equals (=) sign, this attribute identifies the field in the LDAP user record that is used to interpret the provided user ID.

This field can be blank. If this is the case, any unqualified user IDs use the **SHORTUSR** parameter to interpret the provided user ID.

The contents of this field will be concatenated with an ' = ' sign, together with the value provided by the application, to form the full user ID to be located in an LDAP user record. For example, the application provides a user of fred and this field has the value cn, then the LDAP repository will be searched for cn=fred.

This attribute is valid only for an **AUTHTYPE** of **IDPWLDAP**.

DEFINE BUFFPOOL on z/OS: z/OS

Use the MQSC command DEFINE BUFFPOOL to define a buffer pool that is used for holding messages in main storage.

Using MQSC commands

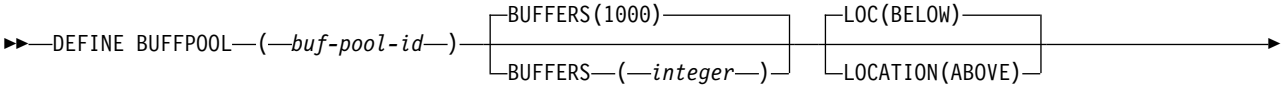
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

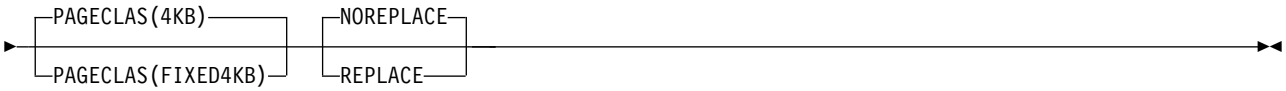
You can issue this command from source 1. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes" on page 566
- "Parameter descriptions for DEFINE BUFFPOOL" on page 566

Synonym: DEF BP

DEFINE BUFFPOOL





Usage notes

1. Specify DEFINE BUFFPOOL commands in a data set identified by the CSQINP1 DD concatenation in the queue manager started task procedure.
2. Use the DISPLAY USAGE TYPE(PAGESET) command to display buffer pool information (see "DISPLAY USAGE on z/OS" on page 960).
3. Use the ALTER BUFFPOOL command to dynamically change the settings of a predefined buffer pool (see "ALTER BUFFPOOL on z/OS" on page 380).

Parameter descriptions for DEFINE BUFFPOOL

If more than one DEFINE BUFFPOOL command is issued for the same buffer pool, only the last one is processed.

(buf-pool-id)
Buffer pool identifier.

CD If IBM MQ Version 8.0 new functions are enabled with OPMODE, this parameter is an integer in the range zero through 99. Otherwise, this parameter is an integer in the range zero through 15.

BUFFERS(*integer* **)**
This parameter is required and is the number of 4096 byte buffers to be used in this buffer pool.

If the value of the **LOCATION** parameter is BELOW, the minimum value of buffers is 100 and the maximum value is 500,000. If the value of the **LOCATION** parameter is ABOVE, then valid values are in the range of 100 to 999999999 (nine nines). The storage used for buffers in a buffer pool with **LOCATION ABOVE** is obtained in multiples of 4MB. Therefore specifying a **BUFFERS** value which is a multiple of 1024 will make the most efficient use of storage.
See Buffers and buffer pools for guidance on the number of buffers you can define in each buffer pool.
When defining a buffer pool care should be taken to ensure that there is sufficient storage available for it either above or below the bar. For more information, see Address space storage.

LOCATION(LOC)(*BELOW* or *ABOVE*)
LOCATION and **LOC** are synonyms and either, but not both, can be used.

The **LOCATION** or **LOC** parameter specifies where the memory used by the specified buffer pool is located.

This memory location can be either ABOVE (64 bit) or BELOW (31 bit) the bar. Valid values for this parameter are BELOW or ABOVE, with BELOW being the default.

CD ABOVE can only be specified if IBM MQ Version 8.0 new functions are enabled with **OPMODE**. BELOW can be specified regardless of the value of **OPMODE** and has the same effect as not specifying the **LOCATION** parameter.

When altering a buffer pool, you should take care to make sure that there is sufficient storage available if increasing the number of buffers, or changing the **LOCATION** value. Switching the location of the buffer pool can be a CPU and I/O intensive task. You should perform this task when the queue manager is not being heavily used.

For more information, see Address space storage.

PAGECLAS(*4KB* **or** *FIXED4KB* **)**

Optional parameter that describes the type of virtual storage pages used for backing the buffers in the buffer pool.

This attribute applies to all buffers in the buffer pool, including any that are added later as a result of using the ALTER BUFFPOOL command. The default value is 4KB, which means that pageable 4KB pages are used to back the buffers in the pool.

4KB is the only valid value if the buffer pool has its location attribute set to BELOW. If the buffer pool has its LOCATION attribute set to ABOVE, it is also possible to specify FIXED4KB. This means that fixed 4KB pages, which are permanently in real storage and will never be paged out to auxiliary storage, are used to back the buffers in the buffer pool.

CD FIXED4KB can only be specified if IBM MQ Version 8.0 new functions are enabled with OPMODE, whereas 4KB can be specified regardless of the value of OPMODE.

The PAGECLAS attribute of a buffer pool can be altered at any time. However, the alteration only takes place when the buffer pool switches location from above the bar, to below the bar, or the other way round. Otherwise, the value is stored in the log of the queue manager and is applied when the queue manager next restarts.

When you specify PAGECLAS(FIXED4KB) the whole buffer pool is backed by page-fixed 4KB pages, so ensure that there is sufficient real storage available on the LPAR. Otherwise, the queue manager might not start, or other address spaces might be impacted; for more information, see Address space storage.

See IBM MQ Support Pac MP16: IBM MQ for z/OS - Capacity planning & tuning for advice on when to use the FIXED4KB value of the PAGECLAS attribute.

REPLACE/NOREPLACE

Optional attribute describing whether this definition of a buffer pool overrides any definition that might already be contained in the log of the queue manager.

REPLACE

This definition of the buffer pool overrides the definition stored in the log of the queue manager, if there is one. If the definition in the log of the queue manager is different from this definition, the differences are discarded and message CSQP064I is issued.

NOREPLACE

This is the default value, and provides the same behavior as with previous releases of IBM MQ. If there is a definition of the buffer pool in the log of the queue manager that is used, and this definition is ignored.

Attention: The queue manager records the current buffer pool settings in checkpoint log records. These buffer pool settings are automatically restored when a queue manager is later restarted. This restoration occurs after processing of the CSQINP1 data set. Therefore, if you have used **ALTER BUFFPOOL** since the buffer pool was last defined, any **DEFINE BUFFPOOL** command in CSQINP1 has been ignored at restart, unless the **REPLACE** attribute has been specified.

CD

Switching from Version 8.0 new function mode to compatibility mode

When you switch from `OPMODE=(NEWFUNC,800)` or `OPMODE=(NEWFUNC,900)` to `OPMODE=(COMPAT,800)` or `OPMODE=(COMPAT,900)` the following occurs:

1. Any buffer pools with an ID greater than 15 are marked as suspended. This means that these buffer pools cannot be used, deleted, or altered until Version 8.0 new functions are enabled again. Information about the buffer pools is kept in check point log records until Version 8.0 new functions are enabled again.

   Any page set that uses a suspended buffer pool is also suspended. Information about the suspended page set is also kept in check point records.

While suspended, any object definitions or messages in the page set are unavailable. An attempt to use a queue or topic which uses the suspended page set results in an MQRC_PAGESET_ERROR message

While suspended, a page set can be associated with a different buffer pool by using the FORMAT function of the utility program CSQUTIL, specifying TYPE(REPLACE). You can then issue a **DEFINE PSID** command to bring the page set back into use with a different buffer pool.

Note: All units of recovery that involved the suspended page set, except units that are indoubt, will have been backed out by the queue manager when the page set was last used. Indoubt units of recovery can be resolved when the page set is again in use by the queue manager.

2. Any buffer pools with an ID of 15 or less that have their LOCATION attribute set to ABOVE, will have the LOCATION attribute switched to BELOW, and their PAGECLAS attribute set to 4KB.

DEFINE CFSTRUCT on z/OS: z/OS

Use the MQSC command DEFINE CFSTRUCT to define queue manager CF level capability, message offload environment, and backup and recovery parameters for a coupling facility application structure.

Using MQSC commands

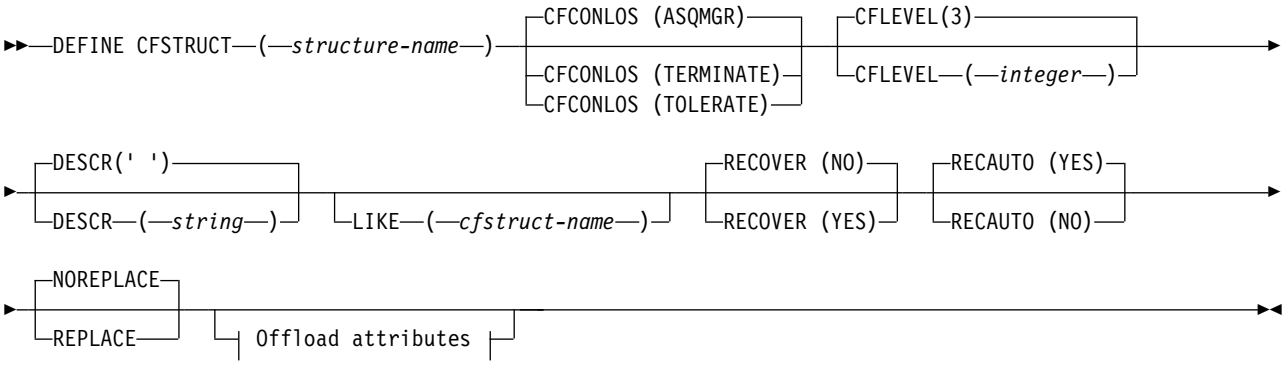
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

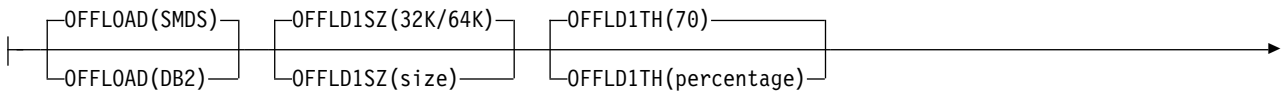
- Syntax diagram
- "Usage notes for DEFINE CFSTRUCT" on page 569
- "Parameter descriptions for DEFINE CFSTRUCT" on page 569

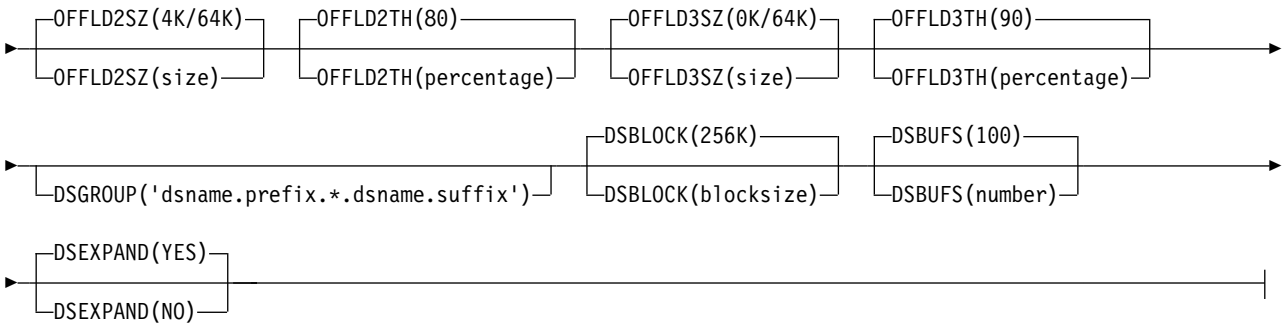
Synonym: DEF CFSTRUCT

DEFINE CFSTRUCT



Offload attributes:





Usage notes for DEFINE CFSTRUCT

1. This command is valid only on z/OS when the queue manager is a member of a queue-sharing group.
2. This command cannot specify the CF administration structure (CSQ_ADMIN).
3. Before any newly defined CF structure can be used by any queues, the structure must be defined in the Coupling Facility Resource Management (CFRM) policy data set.
4. Only CF structures with RECOVER(YES) defined can be backed up and recovered.

Parameter descriptions for DEFINE CFSTRUCT

(structure-name)

Name of the coupling facility application structure that has queue manager CF level capability and backup and recovery parameters you want to define. This parameter is required.

The name:

- Cannot have more than 12 characters.
- Must start with an uppercase letter (A through Z).
- Can include only the characters A through Z and 0 through 9.

The name of the queue-sharing group to which the queue manager is connected is prefixed to the name you supply. The name of the queue-sharing group is always four characters, padded with @ symbols if necessary. For example, if you use a queue-sharing group named `NY03` and you supply the name `PRODUCT7`, the resultant coupling facility structure name is `NY03PRODUCT7`. The administrative structure for the queue-sharing group (in this case `NY03CSQ_ADMIN`) cannot be used for storing messages.

CFCONLOS

This parameter specifies the action to be taken when a queue manager loses connectivity to the CF structure. The value can be:

ASQMGR

The action taken is based on the setting of the CFCONLOS queue manager attribute.

TERMINATE

The queue manager ends when connectivity to the structure is lost.

TOLERATE

The queue manager tolerates loss of connectivity to the structure without terminating.

This parameter is only valid from CFLEVEL(5).

CFLEVEL(*integer*)

Specifies the functional capability level for this CF application structure. Value can be one of the following:

1 A CF structure that can be "auto-created" by a queue manager at command level 520.

2 A CF structure at command level 520 that can only be created or deleted by a queue manager at command level 530 or greater.

3

A CF structure at command level 530. This CFLEVEL is required if you want to use persistent messages on shared queues (if RECOVER(YES) is set), or for message grouping (when a local queue is defined with INDXTYPE(GROUPID)), or both.

You can only increase the value of CFLEVEL to 3 if all the queue managers in the queue-sharing group are at command level 530 or greater - this is to ensure that there are no latent command level 520 connections to queues referencing the structure.

You can only decrease the value of CFLEVEL from 3 if all the queues that reference the CF structure are both empty (have no messages or uncommitted activity) and closed.

4

This CFLEVEL supports all the CFLEVEL(3) functions. CFLEVEL(4) allows queues defined with CF structures at this level to have messages with a length greater than 63 KB.

Only a queue manager with a command level of 600 or above can connect to a CF structure at CFLEVEL(4).

You can only increase the value of CFLEVEL to 4 if all the queue managers in the queue-sharing group are at command level 600 or greater.

You can only decrease the value of CFLEVEL from 4 if all the queues that reference the CF structure are both empty (have no messages or uncommitted activity) and closed.

5

This CFLEVEL supports all functions for CFLEVEL(4). In addition, CFLEVEL(5) enables the following new functions. If altering an existing CFSTRUCT to CFLEVEL(5), you must review other attributes as indicated:

- queues defined with CF structures at this level can have message data offloaded to either shared message data sets (SMDS), or Db2, under control of the OFFLOAD attribute. The offload threshold and size parameters (such as OFFLD1TH, and OFFLD1SZ) determine whether any particular messages are offloaded given its size and current CF structure utilization. If using SMDS offload, the DSGROUP, DSBUFS, DSEXPAND and DSBLOCK attributes are respected.
- structures at CFLEVEL(5) allow the queue manager to tolerate a loss of connectivity to the CF structure. The CFCONLOS attribute determines queue manager behavior when a loss of connectivity is detected, and the RECAUTO attribute controls subsequent automatic structure recovery behavior.
- messages containing IBM MQ message properties are stored in a different format on shared queues in a CFLEVEL(5) structure. This format leads to internal processing optimizations. Additional application migration capabilities are also available and these are enabled via the queue PROPCTL attribute.

Only a queue manager with a command level of 710 or above can connect to a CF structure at CFLEVEL(5).

Note:

You can only increase the value of CFLEVEL to 5 if all the queue managers in the queue-sharing group are at command level 710 or greater and have IBM WebSphere MQ Version 7.1.0 new functions enabled with OPMODE

You can decrease the value of CFLEVEL from 5 if all the queues that reference the CF structure are both empty, that is the queues, and CF structure have no messages or uncommitted activity, and are closed.

DESCR(*string*)

Plain-text comment that provides descriptive information about the object when an operator issues the DISPLAY CFSTRUCT command.

The string should contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

LIKE(*cfstruct-name*)

The name of a CFSTRUCT object, with attributes used to model this definition.

The initial values of all attributes are copied from the object, except any DSGROUP attribute is ignored because each structure requires its own unique value.

OFFLOAD

Specify whether offloaded message data is to be stored in a group of shared message data sets or in Db2.

SMDS

Offload messages from coupling facility to shared message data set (SMDS). This value is the default assumption when a new structure is defined with CFLEVEL(5).

Db2 Offload messages from coupling facility to Db2. This value is the default assumption when an existing structure is increased to CFLEVEL(5) using DEFINE with the REPLACE option.

Offloading messages using Db2 has significant performance impact. If you want to use the offload rules as a means of increasing capacity, the SMDS option should be specified or assumed.

This parameter is only valid from CFLEVEL(5). At CFLEVEL(4) any message offloading is always to Db2, and only applies to messages greater than the maximum coupling facility entry size.

Note:

If you change the offload technique (from Db2 to SMDS or the other way) then all new messages will be written using the new method but any existing large messages stored using the previous technique can still be retrieved. The relevant Db2 message table or shared message data sets will continue to be used until the queue managers have detected that there are no further messages stored in the old format.

If SMDS is specified or assumed, then the DSGROUP parameter is also required. It can be specified either on the same command or on a previous DEFINE or ALTER command for the same structure.

OFFLD1TH(percentage) OFFLD1SZ(size)
OFFLD2TH(percentage) OFFLD2SZ(size)
OFFLD3TH(percentage) OFFLD3SZ(size)

Specify rules for when messages smaller than the maximum coupling facility entry size are to be offloaded to external storage (shared message data sets or Db2 tables) instead of being stored in the application structure. These rules can be used to increase the effective capacity of the structure. The offloaded message still requires an entry in the coupling facility containing

message control information, and a descriptor referring to the offloaded message data, but the amount of structure space required is less than the amount that would be needed to store the whole message.

If the message data is very small (of the order of 100 bytes) it might fit into the same coupling facility entry as the message control information, without needing additional data elements. In this case, no space can be saved, so any offload rules are ignored and the message data is not offloaded. The actual number varies, depending whether more than the default headers are used, or if message properties are being stored.

Messages exceeding the maximum coupling facility entry size (63.75 KB including control information) are always offloaded as they cannot be stored in a coupling facility entry. Messages where the message body exceeds 63 KB are also offloaded to ensure that enough space is available for the control information. Additional rules to request offloading of smaller messages can be specified using these pairs of keywords. Each rule indicates that when the usage of the structure (in either elements or entries) exceeds the specified threshold percentage value, the message data will be offloaded if the total size of the coupling facility entry required to store the whole message (including message data, headers and descriptors) exceeds the specified size value. The minimal set of headers and descriptors require approximately 400 bytes, however this could be greater if other headers or properties are added. This figure would also be greater if an MQMD version greater than 1 is used.

percentage
The usage threshold percentage value is an integer in the range 0 (meaning this rule always applies) to 100 (meaning this rule only applies when the structure is full). For example, OFFLD1TH(75) OFFLD1SZ(32K) means that when the structure is over 75% full, messages greater than 32 kilobytes in size are offloaded.

size The message size value should be specified as an integer followed by K, giving the number of kilobytes in the range **0K** to **64K**. As messages exceeding 63.75 KB are always offloaded, the value 64K is allowed as a simple way to indicate that the rule is not being used.

In general, the smaller the numbers, the more messages are offloaded.

A message is offloaded if any offload rule matches. The normal convention is that a later rule would be for a higher usage level and a smaller message size than an earlier one, but no check is made for consistency or redundancy between the rules.

When structure ALTER processing is active, the number of used elements or entries can temporarily exceed the reported total number, giving a percentage exceeding 100, because the new elements or entries are made available during ALTER processing but the total is only updated when the ALTER completes. At such times, a rule specifying 100 for the threshold may temporarily take effect. If a rule is not intended to be used at all, it should specify 64K for the size.

The default values assumed for the offload rules when defining a new structure at CFLEVEL(5) or upgrading an existing structure to CFLEVEL(5) depend on the OFFLOAD method option. For OFFLOAD(SMDS), the default rules specify increasing amounts of offloading as the structure becomes full. This increases the effective structure capacity with minimal performance impact. For OFFLOAD(Db2), the default rules have the same threshold values as for SMDS but the size values are set to 64K so that the rules never apply and messages are offloaded only if they are too large to be stored in the structure, as for CFLEVEL(4).

For OFFLOAD(SMDS) the defaults are:

- OFFLD1TH(70) OFFLD1SZ(32K)
- OFFLD2TH(80) OFFLD2SZ(4K)
- OFFLD3TH(90) OFFLD3SZ(0K)

For OFFLOAD(Db2) the defaults are:

- OFFLD1TH(70) OFFLD1SZ(64K)
- OFFLD2TH(80) OFFLD2SZ(64K)
- OFFLD3TH(90) OFFLD3SZ(64K)

If the OFFLOAD method option is changed from Db2 to SMDS or back when the current offload rules all match the default values for the old method, the offload rules are switched to the default values for the new method. However, if any of the rules have been changed, the current values are kept when switching method.

These parameters are only valid from CFLEVEL(5). At CFLEVEL(4) any message offloading is always to Db2, and only applies to messages greater than the maximum coupling facility entry size.

DSGROUP

For OFFLOAD(SMDS), specify the generic data set name to be used for the group of shared message data sets associated with this structure (one for each queue manager), with exactly one asterisk indicating where the queue manager name should be inserted to form the specific data set name.

dsname.prefix.*.dsname.suffix
The value must be a valid data set name when the asterisk is replaced by a queue manager name of up to four characters.

The entire parameter value must be enclosed in quotation marks.

This parameter cannot be changed after any data sets have been activated for the structure.

If SMDS is specified or assumed, then the DSGROUP parameter must also be specified.

This parameter is only valid from CFLEVEL(5).

DSBLOCK

For OFFLOAD(SMDS), specify the logical block size, which is the unit in which shared message data set space is allocated to individual queues.

8K
16K
32K
64K
128K
256K
512K
1M Each message is written starting at the next page within the current block and is allocated further blocks as needed. A larger size decreases space management requirements and reduces I/O for large messages, but increases buffer space requirements and disk space requirements for small queues.

This parameter cannot be changed after any data sets have been activated for the structure.

This parameter is only valid from CFLEVEL(5).

DSBUFS

For OFFLOAD(SMDS), specify the number of buffers to be allocated in each queue manager for accessing shared message data sets, as a number in the range 1 - 9999. The size of each buffer is equal to the logical block size. SMDS buffers are allocated in memory objects residing in z/OS 64-bit storage (above the bar).

number
This parameter can be overridden for individual queue managers using the DSBUFS parameter on ALTER SMDS.

When this parameter is altered, any queue managers which are already connected to the structure (and which do not have an individual DSBUFS override value) dynamically increase or decrease the number of data set buffers being used for this structure to match the new value. If the specified target value cannot be reached, the affected queue manager adjusts the DSBUFS parameter associated with its own individual SMDS definition (as for the ALTER SMDS command) to match the actual new number of buffers.

This parameter is only valid from CFLEVEL(5).

DSEXPAND

For OFFLOAD(SMDS), this parameter controls whether the queue manager should expand a shared message data set when it becomes nearly full, and further blocks are required in the data set.

YES Expansion is supported.

Each time expansion is required, the data set is expanded by the secondary allocation specified when the data set was defined. If no secondary allocation was specified, or it was specified as zero, then a secondary allocation amount of approximately 10% of the existing size is used

NO No automatic data set expansion is to take place.

This parameter can be overridden for individual queue managers using the DSEXPAND parameter on ALTER SMDS.

If an expansion attempt fails, the DSEXPAND override for the affected queue manager is automatically changed to NO to prevent further expansion attempts, but it can be changed back to YES using the ALTER SMDS command to enable further expansion attempts.

When this parameter is altered, any queue managers which are already connected to the structure (and which do not have an individual DSEXPAND override value) immediately start using the new parameter value.

This parameter is only valid from CFLEVEL(5).

RECOVER

Specifies whether CF recovery is supported for the application structure. Values are:

NO CF application structure recovery is not supported. (The synonym is **N**.)

YES CF application structure recovery is supported. (The synonym is **Y**.)

You can only set RECOVER(YES) if the structure has a CFLEVEL of 3 or higher. Set RECOVER(YES) if you intend to use persistent messages.

You can only change RECOVER(NO) to RECOVER(YES) if all the queue managers in the queue-sharing group are at command level 530 or greater; this is to ensure that there are no latent command level 520 connections to queues referencing the CFSTRUCT.

You can only change RECOVER(YES) to RECOVER(NO) if all the queues that reference the CF structure are both empty (have no messages or uncommitted activity) and closed.

RECAUTO

Specifies the automatic recovery action to be taken when a queue manager detects that the structure is failed or when a queue manager loses connectivity to the structure and no systems in the SysPlex have connectivity to the Coupling Facility that the structure is allocated in. Values can be:

YES The structure and associated shared message data sets which also need recovery will be automatically recovered (The synonym is **Y**.)

NO The structure will not be automatically recovered. (The synonym is **N**.)

This parameter has no effect for structures defined with RECOVER(NO).

This parameter is only valid from a CFLEVEL(5)

REPLACE and NOREPLACE
Defines whether the existing definition is to be replaced with this one. This parameter is optional.

REPLACE
The definition should replace any existing definition of the same name. If a definition does not exist, one is created. If you use the REPLACE option, all queues that use this CF structure must be empty and closed.

NOREPLACE
The definition should not replace any existing definition of the same name.

DEFINE CHANNEL:

Use the MQSC command **DEFINE CHANNEL** to define a new channel, and set its parameters.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

Synonym: DEF CHL

- "Usage notes"
- "Parameter descriptions for DEFINE CHANNEL"

Usage notes

- For CLUSSDR channels, you can specify the REPLACE option only for manually created channels.
- Successful completion of the command does not mean that the action completed. To check for true completion, see the DEFINE CHANNEL step in Checking that async commands for distributed networks have finished.

Parameter descriptions for DEFINE CHANNEL

The following table shows the parameters that are relevant for each type of channel:

SDR "Sender channel" on page 610

SVR "Server channel" on page 613

RCVR "Receiver channel" on page 617

RQSTR "Requester channel" on page 619

CLNTCONN
"Client-connection channel" on page 622

SVRCONN
"Server-connection channel" on page 625

CLUSSDR
"Cluster-sender channel" on page 627

CLUSRCVR
"Cluster-receiver channel" on page 630

Multi V 9.0.0 **AMQP**

"AMQP channel" on page 634

There is a description of each parameter after the table. Parameters are optional unless the description states that they are required.

Table 95. DEFINE and ALTER CHANNEL parameters

Parameter	SDR	SVR	RCVR	RQSTR	CLNTCONN	SVRCONN	CLUSSDR	CLUSRCVR	AMQP V 9.0.0
AFFINITY					✓				
AMQPKA									✓ 9.0.0
BACKLOG									
BATCHHB	✓	✓					✓	✓	
BATCHINT	✓	✓					✓	✓	
BATCHLIM	✓	✓					✓	✓	
BATCHSZ	✓	✓	✓	✓			✓	✓	
CERTLABL	✓	✓	✓	✓	✓	✓		✓	✓ 9.0.0
channel-name	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
CHLTYPE	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
CLNTWGHT					✓				
CLUSNL							✓	✓	
CLUSTER							✓	✓	
CLWLPRTY							✓	✓	
CLWLRANK							✓	✓	
CLWLWGHT							✓	✓	
z/OS CMDSCOPE	✓	✓	✓	✓	✓	✓	✓	✓	
COMPHDR	✓	✓	✓	✓	✓	✓	✓	✓	
COMPMSG	✓	✓	✓	✓	✓	✓	✓	✓	
CONNAME	✓	✓		✓	✓		✓	✓	
CONVERT	✓	✓					✓	✓	
DEFCDISP	✓	✓	✓	✓		✓			

Table 95. DEFINE and ALTER CHANNEL parameters (continued)

Parameter	SDR	SVR	RCVR	RQSTR	CLNTCONN	SVRCONN	CLUSSDR	CLUSRCVR	AMQP 9.0.0
DEFRECON					✓				
DESCR	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
DISCINT	✓	✓				✓	✓	✓	
HBINT	✓	✓	✓	✓	✓	✓	✓	✓	
JAASCFG									
KAINT	✓	✓	✓	✓	✓	✓	✓	✓	
LIKE	✓	✓	✓	✓	✓	✓	✓	✓	
LOCLADDR	✓	✓		✓	✓		✓	✓	✓ 9.0.0
LONGRTY	✓	✓					✓	✓	
LONGTMR	✓	✓					✓	✓	
MAXINST						✓			✓ 9.0.0
MAXINSTC						✓			
MAXMSGL	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
MCANAME	✓	✓		✓			✓	✓	
MCATYPE	✓	✓		✓			✓	✓	
MCAUSER			✓	✓		✓		✓	✓ 9.0.0
MODENAME	✓	✓		✓	✓		✓	✓	
MONCHL	✓	✓	✓	✓		✓	✓	✓	
MRDATA			✓	✓				✓	
MREXIT			✓	✓				✓	
MRRTY			✓	✓				✓	
MRTMR			✓	✓				✓	
MSGDATA	✓	✓	✓	✓			✓	✓	
MSGEXIT	✓	✓	✓	✓			✓	✓	

Table 95. DEFINE and ALTER CHANNEL parameters (continued)

Parameter	SDR	SVR	RCVR	RQSTR	CLNTCONN	SVRCONN	CLUSSDR	CLUSRCVR	AMQP 9.0.0
NETPRTY								✓	
NPMSPEED	✓	✓	✓	✓			✓	✓	
PASSWORD	✓	✓		✓	✓		✓		
PORT									✓ 9.0.0
PROPCTL	✓	✓					✓	✓	
PUTAUT			✓	✓		✓		✓	
QMNAME					✓				
z/OS QSGDISP	✓	✓	✓	✓	✓	✓	✓	✓	
RCVDATA	✓	✓	✓	✓	✓	✓	✓	✓	
RCVEXIT	✓	✓	✓	✓	✓	✓	✓	✓	
REPLACE	✓	✓	✓	✓	✓	✓	✓	✓	
SCYDATA	✓	✓	✓	✓	✓	✓	✓	✓	
SCYEXIT	✓	✓	✓	✓	✓	✓	✓	✓	
SENDDATA	✓	✓	✓	✓	✓	✓	✓	✓	
SENDEXIT	✓	✓	✓	✓	✓	✓	✓	✓	
SEQWRAP	✓	✓	✓	✓			✓	✓	
SHARECNV					✓	✓			
SHORTRTY	✓	✓					✓	✓	
SHORTTMR	✓	✓					✓	✓	
SSLCAUTH		✓	✓	✓		✓		✓	
SSLCIPH	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
SSLKEYP									
SSLPEER	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
STATCHL	✓	✓	✓	✓			✓	✓	

Table 95. DEFINE and ALTER CHANNEL parameters (continued)

Parameter	SDR	SVR	RCVR	RQSTR	CLNTCONN	SVRCONN	CLUSSDR	CLUSRCVR	AMQP 9.0.0
TPNAME	✓	✓		✓	✓	✓	✓	✓	
TPROOT									✓ 9.0.0
TRPTYPE	✓	✓	✓	✓	✓	✓	✓	✓	
USECLTID									✓ 9.0.0
USEDLQ	✓	✓	✓	✓			✓	✓	
USERID	✓	✓		✓	✓		✓		
XMITQ	✓	✓							

AFFINITY

Use the channel affinity attribute when client applications connect multiple times using the same queue manager name. With the attribute, you can choose whether the client uses the same client channel definition for each connection. This attribute is intended to be used when multiple applicable channel definitions are available.

PREFERRED
: The first connection in a process reading a client channel definition table (CCDT) creates a list of applicable definitions. The list is based on the weightings, with any applicable **CLNTWGHT(0)** definitions first and in alphabetic order. Each connection in the process attempts to connect using the first definition in the list. If a connection is unsuccessful the next definition is used. Unsuccessful non- **CLNTWGHT(0)** definitions are moved to the end of the list. **CLNTWGHT(0)** definitions remain at the start of the list and are selected first for each connection. For C, C++ and .NET (including fully managed .NET) clients the list is updated if the CCDT was modified since the list was created. Each client process with the same host name creates the same list.

NONE
: The first connection in a process reading a CCDT creates a list of applicable definitions. All connections in a process select an applicable definition based on the weighting with any applicable **CLNTWGHT(0)** definitions selected first in alphabetic order. For C, C++ and .NET (including fully managed .NET) clients the list is updated if the CCDT was modified since the list was created.

For example, suppose that we had the following definitions in the CCDT:

```
CHLNAME(A) QMNAME (QM1) CLNTWGHT(3)
CHLNAME(B) QMNAME (QM1) CLNTWGHT(4)
CHLNAME(C) QMNAME (QM1) CLNTWGHT(4)
```

The first connection in a process creates its own ordered list based on the weightings. So it might, for example, create the ordered list `CHLNAME(B), CHLNAME(A), CHLNAME(C)`.

For **AFFINITY(PREFFERED)**, each connection in the process attempts to connect using `CHLNAME(B)`. If a connection is unsuccessful the definition is moved to the end of the list which now becomes `CHLNAME(A), CHLNAME(C), CHLNAME(B)`. Each connection in the process then attempts to connect using `CHLNAME(A)`.

For **AFFINITY(NONE)**, each connection in the process attempts to connect using one of the three definitions selected at random based on the weightings.

If sharing conversations is enabled with a non-zero channel weighting and **AFFINITY(NONE)**, multiple connections do not have to share an existing channel instance. They can connect to the same queue manager name using different applicable definitions rather than sharing an existing channel instance.

Multi V 9.0.0 **AMQPKA(*integer*)**

The keep alive time for an AMQP channel in seconds. If the AMQP client has not sent any frames within the keep alive interval, then the connection is closed with an `amqp:resource-limit-exceeded` AMQP error condition.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of `AMQP`

BATCHHB(*integer*)

Specifies whether batch heartbeats are to be used. The value is the length of the heartbeat in milliseconds.

Batch heartbeats allow a sending channel to verify that the receiving channel is still active just before committing a batch of messages. If the receiving channel is not active, the batch can be backed out rather than becoming in-doubt, as would otherwise be the case. By backing out the batch, the messages remain available for processing so they could, for example, be redirected to another channel.

If the sending channel received a communication from the receiving channel within the batch heartbeat interval, the receiving channel is assumed to be still active. If not, a 'heartbeat' is sent to the receiving channel to check.

The value must be in the range 0 - 999999. A value of zero indicates that batch heart beats are not used.

This parameter is valid for channels with a channel type (**CHLTYPE**) of only `SDR`, `SVR`, `CLUSSDR`, and `CLUSRCVR`.

BATCHINT(*integer*)

The minimum amount of time, in milliseconds, that a channel keeps a batch open.

The batch is terminated when one of the following conditions is met:

- **BATCHSZ** messages are sent.
- **BATCHLIM** kilobytes are sent.
- The transmission queue is empty and **BATCHINT** is exceeded.

The value must be in the range 0 - 999999999. Zero means that the batch is terminated as soon as the transmission queue becomes empty, or the **BATCHSZ** limit is reached.

This parameter is valid for channels with a channel type (**CHLTYPE**) of only `SDR`, `SVR`, `CLUSSDR`, and `CLUSRCVR`.

BATCHLIM(*integer*)

The limit, in kilobytes, of the amount of data that can be sent through a channel before taking a sync point. A sync point is taken after the message that caused the limit to be reached flows across the channel. A value of zero in this attribute means that no data limit is applied to batches over this channel.

The batch is terminated when one of the following conditions is met:

- **BATCHSZ** messages are sent.
- **BATCHLIM** kilobytes are sent.
- The transmission queue is empty and **BATCHINT** is exceeded.

This parameter is valid for channels with a channel type (**CHLTYPE**) of only `SDR`, `SVR`, `CLUSSDR`, and `CLUSRCVR`.

The value must be in the range 0 - 999999. The default value is 5000.

This parameter is supported on all platforms.

BATCHSZ(*integer*)

The maximum number of messages that can be sent through a channel before taking a sync point.

The maximum batch size used is the lowest of the following values:

- The **BATCHSZ** of the sending channel.
- The **BATCHSZ** of the receiving channel.
- z/OS On z/OS, three less than the maximum number of uncommitted messages allowed at the sending queue manager (or one if this value is zero or less).
- Multi On Multiplatforms, the maximum number of uncommitted messages allowed at the sending queue manager (or one if this value is zero or less).
- z/OS On z/OS, three less than the maximum number of uncommitted messages allowed at the receiving queue manager (or one if this value is zero or less).
- Multi On Multiplatforms, the maximum number of uncommitted messages allowed at the receiving queue manager (or one if this value is zero or less).

While non-persistent messages sent over an **NPMSPEED(FAST)** channel are delivered to a queue immediately (without waiting for a complete batch), the messages still contribute to the batch size for a channel and, therefore, cause confirm flows to occur when **BATCHSZ** messages have flowed.

If the batch flows are causing a performance impact when moving only non-persistent messages, and **NPMSPEED** is set to FAST, you should consider setting the **BATCHSZ** to the maximum permissible value of 9999, and **BATCHLIM** to zero.

Additionally, setting **BATCHINT** to a high value, for example, 999999999 keeps each batch "open" for longer, even if there are no new messages waiting on the transmission queue.

The above settings minimize the frequency of confirm flows, but be aware that if any persistent messages are moved over a channel with these settings, there will be significant delays in the delivery of those persistent messages only.

The maximum number of uncommitted messages is specified by the **MAXUMSGS** parameter of the **ALTER QMGR** command. This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, RCVR, RQSTR, CLUSSDR, or CLUSRCVR.

The value must be in the range 1 - 9999.

CERTLABL

Certificate label for this channel to use.

The label identifies which personal certificate in the key repository is sent to the remote peer. If this attribute is blank, the certificate is determined by the queue manager **CERTLABL** parameter.

Note that inbound channels (including receiver, cluster-receiver, unqualified server, and server-connection channels) only send the configured certificate if the IBM MQ version of the remote peer fully supports certificate label configuration, and the channel is using a TLS CipherSpec. See Interoperability of Elliptic Curve and RSA CipherSpecs for further information.

In all other cases, the queue manager **CERTLABL** parameter determines the certificate sent. In particular, the following only ever receive the certificate configured by the **CERTLABL** parameter of the queue manager, regardless of the channel-specific label setting:

- All current Java and JMS clients.
- Versions of IBM MQ prior to Version 8.0.

You do not need to run the **REFRESH SECURITY TYPE(SSL)** command if you make any changes to **CERTLABL** on a channel. However, you must run a **REFRESH SECURITY TYPE(SSL)** command if you make any changes to **CERTLABL** on the queue manager.

Note: It is an error to inquire, or set, this attribute for cluster-sender channels. If you attempt to do so, you receive the error MQRCCF_WRONG_CHANNEL_TYPE. However, the attribute is present in cluster-sender channel objects (including MQCD structures) and a channel auto-definition (CHAD) exit might set it programmatically if required.

(channel-name)
The name of the new channel definition.

This parameter is required on all types of channel.

Multi On CLUSSDR channels, this parameter can take a different form to the other channel types. If your convention for naming CLUSSDR channels includes the name of the queue manager, you can define a CLUSSDR channel using the +QMNAME+ construction. After connection to the matching CLUSRCVR channel, IBM MQ substitutes the correct repository queue manager name in place of +QMNAME+ in the CLUSSDR channel definition. See Components of a cluster.

The name must not be the same as any existing channel defined on this queue manager, unless REPLACE or ALTER is specified.

z/OS On z/OS, CLNTCONN channel names can duplicate others.

The maximum length of the string is 20 characters, and the string must contain only valid characters; see Rules for naming IBM MQ objects.

CHLTYPE
Channel type. This parameter is required.

Multi On Multiplatforms, it must follow immediately after the *(channel-name)* parameter.

SDR Sender channel

SVR Server channel

RCVR Receiver channel

RQSTR Requester channel

CLNTCONN
Client-connection channel

SVRCONN
Server-connection channel

CLUSSDR
CLUSSDR channel.

CLUSRCVR
Cluster-receiver channel.

V 9.0.0 **AMQP**
AMQP channel

Note: If you are using the REPLACE option, you cannot change the channel type.

CLNTWGHT
Set the client channel weighting attribute to select a client channel definition at random based on its weighting when more than one suitable definition is available. Specify a value in the range 0 - 99.

The special value 0 indicates that no random load balancing is performed and applicable definitions are selected in alphabetic order. To enable random load balancing the value can be in the range 1 - 99, where 1 is the lowest weighting and 99 is the highest.

If a client application issues MQCONN with a queue manager name of **name* a client channel definition can be selected at random. The chosen definition is randomly selected based on the weighting. Any applicable **CLNTWGHT(0)** definitions selected are selected first in alphabetic order. Randomness in the selection of client connection definitions is not guaranteed.

For example, suppose that we had the following two definitions in the CCDT:

```
CHLNAME(TO.QM1) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address1) CLNTWGHT(2)
CHLNAME(TO.QM2) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address2) CLNTWGHT(4)
```

A client MQCONN with queue manager name *GRP1 would choose one of the two definitions based on the weighting of the channel definition. (A random integer 1 - 6 would be generated. If the integer was in the range 1 through 2, address1 would be used otherwise address2 would be used). If this connection was unsuccessful the client would then use the other definition.

The CCDT might contain applicable definitions with both zero and non-zero weighting. In this situation, the definitions with zero weighting are chosen first and in alphabetic order. If these connections are unsuccessful the definitions with non-zero weighting are chosen based on their weighting.

For example, suppose that we had the following four definitions in the CCDT:

```
CHLNAME(TO.QM1) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address1) CLNTWGHT(1)
CHLNAME(TO.QM2) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address2) CLNTWGHT(2)
CHLNAME(TO.QM3) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address3) CLNTWGHT(0)
CHLNAME(TO.QM4) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address4) CLNTWGHT(0)
```

A client MQCONN with queue manager name *GRP1 would first choose definition TO.QM3. If this connection was unsuccessful the client would then choose definition TO.QM4. If this connection was also unsuccessful the client would then randomly choose one of the remaining two definitions based on their weighting.

CLNTWGHT is supported for all transport protocols.

CLUSNL(*nlname*)
The name of the namelist that specifies a list of clusters to which the channel belongs.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of CLUSSDR and CLUSRCVR channels. Only one of the resultant values of **CLUSTER** or **CLUSNL** can be nonblank, the other must be blank.

CLUSTER(*clustername*)
The name of the cluster to which the channel belongs. The maximum length is 48 characters conforming to the rules for naming IBM MQ objects.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of CLUSSDR and CLUSRCVR channels. Only one of the resultant values of **CLUSTER** or **CLUSNL** can be nonblank, the other must be blank.

CLWLPRTY(*integer*)
Specifies the priority of the channel for the purposes of cluster workload distribution. The value must be in the range 0 - 9 where 0 is the lowest priority and 9 is the highest.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of CLUSSDR and CLUSRCVR channels.

For more information about this attribute, see **CLWLPRTY** channel attribute.

CLWLRANK(*integer*)
Specifies the rank of the channel for the purposes of cluster workload distribution. The value must be in the range 0 - 9 where 0 is the lowest rank and 9 is the highest.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of CLUSSDR and CLUSRCVR channels.

For more information about this attribute, see CLWLRANK channel attribute.

CLWLWGHT(*integer*)
Specifies the weighting to be applied to a channel so that the proportion of messages sent down the channel can be controlled by workload management. The value must be in the range 1 - 99 where 1 is the lowest rank and 99 is the highest.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of CLUSSDR and CLUSRCVR channels.

For more information about this attribute, see CLWLWGHT channel attribute.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must either be left blank, or if **QSGDISP** is set to GROUP, the local queue manager name.

' ' The command runs on the queue manager on which it was entered.

QmgrName
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which the command was entered. To do so, you must be using a shared queue environment, and the command server must be enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

COMPHDR
The list of header data compression techniques supported by the channel.

For SDR, SVR, CLUSSDR, CLUSRCVR, and CLNTCONN channels, the values must be specified in order of preference. The first compression technique in the list that is supported by the remote end of the channel is used.
The mutually supported compression techniques of the channel are passed to the message exit of the sending channel. The message exit can alter the compression technique on a per message basis. Compression alters the data passed to send and receive exits.

NONE No header data compression is performed.

SYSTEM Header data compression is performed.

COMPMSG
The list of message data compression techniques supported by the channel.

For SDR, SVR, CLUSSDR, CLUSRCVR, and CLNTCONN channels, the values must be specified in order of preference. The first compression technique in the list that is supported by the remote end of the channel is used.
The mutually supported compression techniques of the channel are passed to the message exit of the sending channel. The message exit can alter the compression technique on a per message basis. Compression alters the data passed to send and receive exits.

NONE No message data compression is performed.

RLE Message data compression is performed using run-length encoding.

ZLIBFAST
Message data compression is performed using ZLIB encoding with speed prioritized.

z/OS On z/OS systems with zEDC Express facility enabled, compression can be offloaded to zEDC Express.

ZLIBHIGH
Message data compression is performed using ZLIB encoding with compression prioritized.

ANY Any compression technique supported by the queue manager can be used. This value is only valid for RCVR, RQSTR, and SVRCONN channels.

CONNAME(*string* <, *string* >)
Connection name.

For CLUSRCVR channels, **CONNAME** relates to the local queue manager, and for other channels it relates to the target queue manager.

z/OS On z/OS, **CONNAME** is mandatory for CLUSRCVR channels. In addition, whether you specify **CONNAME**, or the name is generated for you, the **CONNAME** produced must be a valid connection name for the local queue manager, otherwise the full repository is not able to make a connection back to the local queue manager.

z/OS On z/OS, the maximum length of the string is 48 characters.

Multi On Multiplatforms, the maximum length of the string is 264 characters

A workaround to the 48 character limit might be one of the following suggestions:

- Set up your DNS servers so that you use, for example, host name of `myserver` instead of `myserver.location.company.com`, ensuring you can use the short host name.
- Use IP addresses.

Specify **CONNAME** as a comma-separated list of names of machines for the stated **TRPTYPE**. Typically only one machine name is required. You can provide multiple machine names to configure multiple connections with the same properties. The connections are usually tried in the order they are specified in the connection list until a connection is successfully established. The order is modified for clients if the **CLNTWGHT** attribute is provided. If no connection is successful, the channel attempts the connection again, as determined by the attributes of the channel. With client channels, a connection-list provides an alternative to using queue manager groups to configure multiple connections. With message channels, a connection list is used to configure connections to the alternative addresses of a multi-instance queue manager.

CONNAME is required for channels with a channel type (**CHLTYPE**) of SDR, RQSTR, CLNTCONN, and CLUSSDR. It is optional for SVR channels, and for CLUSRCVR channels of **TRPTYPE(TCP)**, and is not valid for RCVR or SVRCONN channels.

Providing multiple connection names in a list was first supported in IBM WebSphere MQ Version 7.0.1. It changes the syntax of the **CONNAME** parameter. Earlier clients and queue managers connect using the first connection name in the list, and do not read the rest of the connection names in the list. In order for the earlier clients and queue managers to parse the new syntax, you must specify a port number on the first connection name in the list. Specifying a port number avoids problems when connecting to the channel from a client or queue manager that is running at a level earlier than IBM WebSphere MQ Version 7.0.1.

Multi On Multiplatforms, the TCP/IP connection name parameter of a cluster-receiver channel is optional. If you leave the connection name blank, IBM MQ generates a connection name for you, assuming the default port and using the current IP address of the system. You can override the default port number, but still use the current IP address of the system. For each connection name leave the IP name blank, and provide the port number in parentheses; for example:

```
(1415)
```

The generated **CONNAME** is always in the dotted decimal (IPv4) or hexadecimal (IPv6) form, rather than in the form of an alphanumeric DNS host name.

Tip: If you are using any of the special characters in your connection name (for example, parentheses) you must enclose the string in single quotation marks.

The value you specify depends on the transport type (**TRPTYPE**) to be used:

LU62

- z/OS On z/OS, there are two forms in which to specify the value:

  Logical unit name
  The logical unit information for the queue manager, comprising the logical unit name, TP name, and optional mode name. Logical unit name can be specified in one of three forms:

Table 96. Forms of logical unit name

Form	Example
luname	IGY12355
luname/TPname	IGY12345/APING
luname/TPname/modename	IGY12345/APINGD/#INTER

  For the first form, the TP name and mode name must be specified for the **TPNAME** and **MODENAME** parameters; otherwise these parameters must be blank.

  Note: For CLNTCONN channels, only the first form is allowed.

  Symbolic name
  The symbolic destination name for the logical unit information for the queue manager, as defined in the side information data set. The **TPNAME** and **MODENAME** parameters must be blank.

  Note: For CLUSRCVR channels, the side information is on the other queue managers in the cluster. Alternatively, it can be a name that a channel auto-definition exit can resolve into the appropriate logical unit information for the local queue manager.

  The specified or implied LU name can be that of a VTAM generic resources group.

- Multi On IBM i, UNIX, Linux, and Windows, **CONNAME** is the name of the CPI-C communications side object. Alternatively, if the **TPNAME** is not blank, **CONNAME** is the fully qualified name of the partner logical unit. See Configuration parameters for an LU 6.2 connection.

NetBIOS
A unique NetBIOS name (limited to 16 characters).

SPX The 4-byte network address, the 6-byte node address, and the 2-byte socket number. These values must be entered in hexadecimal, with a period separating the network and node addresses. The socket number must be enclosed in brackets, for example:

```
CONNAME('0a0b0c0d.804abcde23a1(5e86)')
```

TCP Either the host name, or the network address of the remote machine (or the local machine for CLUSRCVR channels). This address can be followed by an optional port number, enclosed in parentheses.

If the **CONNAME** is a host name, the host name is resolved to an IP address.

The IP stack used for communication depends on the value specified for **CONNAME** and the value specified for **LOCLADDR**. See **LOCLADDR** for information about how this value is resolved.

z/OS On z/OS, the connection name can include the IP_name of an z/OS dynamic DNS group or a Network Dispatcher input port. Do not include the IP_name or input port for channels with a channel type (**CHLTYPE**) of CLUSSDR.

On all platforms,, you do not always need to specify the network address of your queue manager. If you define a channel with a channel type (**CHLTYPE**) of CLUSRCVR that is using TCP/IP, IBM MQ generates a **CONNAME** for you. It assumes the default port and uses the current IPv4 address of the system. If the system does not have an IPv4 address, the current IPv6 address of the system is used.

Note: If you are using clustering between IPv6-only and IPv4-only queue managers, do not specify an IPv6 network address as the **CONNAME** for CLUSRCVR channels. A queue manager that is capable only of IPv4 communication is unable to start a CLUSSDR channel definition that specifies the **CONNAME** in IPv6 hexadecimal form. Consider, instead, using host names in a heterogeneous IP environment.

CONVERT

Specifies whether the sending message channel agent attempts conversion of the application message data, if the receiving message channel agent cannot perform this conversion.

NO No conversion by sender

YES Conversion by sender

z/OS On z/OS, N and Y are accepted as synonyms of NO and YES.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, CLUSSDR, or CLUSRCVR.

DEFCDISP

Specifies the default channel disposition of the channel.

PRIVATE
The intended disposition of the channel is as a private channel.

FIXSHARED
The intended disposition of the channel is as a shared channel associated with a specific queue manager.

SHARED The intended disposition of the channel is as a shared channel.

This parameter does not apply to channels with a channel type (**CHLTYPE**) of CLNTCONN, CLUSSDR, or CLUSRCVR.

DEFRECON

Specifies whether a client connection automatically reconnects a client application if its connection breaks.

NO Unless overridden by **MQCONNX**, the client is not reconnected automatically.

YES Unless overridden by **MQCONNX**, the client reconnects automatically.

QMGR Unless overridden by **MQCONNX**, the client reconnects automatically, but only to the same queue manager. The QMGR option has the same effect as MQCNO_RECONNECT_Q_MGR.

DISABLED
Reconnection is disabled, even if requested by the client program using the **MQCONNX** MQI call.

Table 97. Automatic reconnection depends on the values set in the application and in the channel definition

DEFRECON	**Reconnection options set in the application**			
	MQCNO_RECONNECT	MQCNO_RECONNECT_Q_MGR	MQCNO_RECONNECT_AS_DEF	MQCNO_RECONNECT_DISABLED
NO	YES	QMGR	NO	NO
YES	YES	QMGR	YES	NO
QMGR	YES	QMGR	QMGR	NO
DISABLED	NO	NO	NO	NO

DESCR(*string*)
Plain-text comment. It provides descriptive information about the channel when an operator issues the **DISPLAY CHANNEL** command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If the information is sent to another queue manager they might be translated incorrectly. The characters must be in the coded character set identifier (CCSID) of the local queue manager.

DISCINT(*integer*)
The minimum time in seconds for which the channel waits for a message to arrive on the transmission queue. The waiting period starts after a batch ends. After the end of the waiting period, if there are no more messages, the channel is ended. A value of zero causes the message channel agent to wait indefinitely.

The value must be in the range 0 - 999 999.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SVRCONN, SDR, SVR, CLUSSDR, CLUSRCVR.

For SVRCONN channels using the TCP protocol, **DISCINT** has a different interpretation. It is the minimum time in seconds for which the SVRCONN instance remains active without any communication from its partner client. A value of zero disables this disconnect processing. The SVRCONN inactivity interval applies only between IBM MQ API calls from a client, so no client is disconnected during an extended MQGET with wait call. This attribute is ignored for SVRCONN channels using protocols other than TCP.

HBINT(*integer*)

HBINT specifies the approximate time between heartbeat flows sent by a message channel agent (MCA). The flows are sent when there are no messages on the transmission queue.

Heartbeat flows unblock the receiving MCA, which is waiting for messages to arrive or for the disconnect interval to expire. When the receiving MCA is unblocked, it can disconnect the channel without waiting for the disconnect interval to expire. Heartbeat flows also free any storage buffers that are allocated for large messages. They also close any queues that are left open at the receiving end of the channel.

The value is in seconds and must be in the range 0 - 999999. A value of zero means that no heartbeat flows are to be sent. The default value is 300. To be most useful, the value needs to be less than the disconnect interval value.

For SVRCONN and CLNTCONN channels, heartbeats can flow from both the server side as well as the client side independently. If no data is transferred across the channel during the heartbeat interval, the CLNTCONN MQI agent sends a heartbeat flow. The SVRCONN MQI agent responds to it with another heartbeat flow. The flows happen irrespective of the state of the channel. For example, irrespective of whether it is inactive while making an API call, or is inactive waiting for client user input. The SVRCONN MQI agent is also capable of initiating a heartbeat to the client, again irrespective of the state of the channel. The SVRCONN and CLNTCONN MQI agents are

prevented from heart beating to each other at the same time. The server heartbeat is flowed if no data is transferred across the channel for the heartbeat interval plus 5 seconds.

For server-connection and client-connection channels working in the channel mode before IBM WebSphere MQ Version 7.0, heartbeats flow only when a server MCA is waiting for an MQGET command with the WAIT option specified, which it has issued on behalf of a client application.

For more information, see Heartbeat interval (HBINT).

KAINT(*integer*)
The value passed to the communications stack for keepalive timing for this channel.

For this attribute to be effective, TCP/IP keepalive must be enabled both in the queue manager and in TCP/IP.

z/OS On z/OS, enable TCP/IP keepalive in the queue manager by issuing the **ALTER QMGR TCPKEEP(YES)** command. If the **TCPKEEP** queue manager parameter is NO, the value is ignored, and the keepalive facility is not used.

Multi On Multiplatforms, TCP/IP keepalive is enabled when the **KEEPALIVE=YES** parameter is specified in the TCP stanza. Modify the TCP stanza in the distributed queuing configuration file, qm.ini, or through the IBM MQ Explorer.

Keepalive must also be enabled within TCP/IP itself. Refer to your TCP/IP documentation for information about configuring keepalive:

- AIX On AIX, use the **no** command.
- HP-UX On HP-UX, use the **ndd** command.
- Windows On Windows, edit the registry.
- z/OS On z/OS, update your TCP/IP PROFILE data set and add or change the **INTERVAL** parameter in the TCPCONFIG section.

z/OS Although the **KAINT** parameter is available on all platforms, its setting is implemented only on z/OS.

Multi On Multiplatforms, you can access and modify the parameter, but there is no functional implementation of the parameter, it is only stored and forwarded. This functionality is useful in a clustered environment where a value set in a cluster-receiver channel definition on AIX, for example, flows to (and is implemented by) z/OS queue managers that are in, or join, the cluster. On Multiplatforms, if you need the functionality provided by the **KAINT** parameter, use the Heartbeat Interval (**HBINT**) parameter, as described in **HBINT**.

(*integer*)
The KeepAlive interval to be used, in seconds, in the range 1 through 99999.

0 The value used is that specified by the **INTERVAL** statement in the TCP profile configuration data set.

AUTO The KeepAlive interval is calculated based upon the negotiated heartbeat value as follows:

- If the negotiated **HBINT** is greater than zero, keepalive interval is set to that value plus 60 seconds.
- If the negotiated **HBINT** is zero, the keepalive value used is that specified by the **INTERVAL** statement in the TCP/IP PROFILE configuration data set.

If AUTO is specified for **KAINT**, and it is a server-connection channel, the **TCP INTERVAL** value is used instead for the keepalive interval.

In this case, **KAINT** is zero in **DISPLAY CHSTATUS**; it would be non-zero if an integer had been coded instead of AUTO.

This parameter is valid for all channel types. It is ignored for channels with a **TRPTYPE** other than TCP or SPX.

LIKE(*channel-name*)
The name of a channel. The parameters of this channel are used to model this definition.

If you do not set **LIKE**, and do not set a parameter field related to the command, its value is taken from one of the default channels. The default values depend upon the channel type:

SYSTEM.DEF.SENDER
Sender channel

SYSTEM.DEF.SERVER
Server channel

SYSTEM.DEF.RECEIVER
Receiver channel

SYSTEM.DEF.REQUESTER
Requester channel

SYSTEM.DEF.SVRCONN
Server-connection channel

SYSTEM.DEF.CLNTCONN
Client-connection channel

SYSTEM.DEF.CLUSSDR
CLUSSDR channel

SYSTEM.DEF.CLUSRCVR
Cluster-receiver channel

V 9.0.0 **SYSTEM.DEF.AMQP**
AMQP channel

This parameter is equivalent to defining the following object for a SDR channel, and similarly for other channel types:

```
LIKE(SYSTEM.DEF.SENDER)
```

These default channel definitions can be altered by the installation to the default values required.

z/OS On z/OS, the queue manager searches page set zero for an object with the name you specify and a disposition of QMGR or COPY. The disposition of the **LIKE** object is not copied to the object and channel type you are defining.

Note:

1. **QSGDISP(GROUP)** objects are not searched.
2. **LIKE** is ignored if **QSGDISP(COPY)** is specified. However, the group object defined is used as a **LIKE** object.

LOCLADDR(*string*)
LOCLADDR is the local communications address for the channel. For channels other than AMQP channels, use this parameter if you want a channel to use a particular IP address, port, or port range for outbound communications. **LOCLADDR** might be useful in recovery scenarios where a channel is restarted on a different TCP/IP stack. **LOCLADDR** is also useful to force a channel to use an IPv4 or IPv6 stack on a dual-stack system. You can also use **LOCLADDR** to force a channel to use a dual-mode stack on a single-stack system.

Note: AMQP channels do not support the same format of **LOCLADDR** as other IBM MQ channels. For the format supported by AMQ, see the next parameter **AMQP: LOCLADDR**.

For channels other than AMQP channels, the **LOCLADDR** parameter is valid only for channels with a transport type (**TRPTYPE**) of TCP. If **TRPTYPE** is not TCP, the data is ignored and no error message is issued.

The value is the optional IP address, and optional port or port range used for outbound TCP/IP communications. The format for this information is as follows:

```
LOCLADDR([ip-addr][(low-port[,high-port])][,[ip-addr][(low-port[,high-port])]])
```

The maximum length of **LOCLADDR**, including multiple addresses, is MQ_LOCAL_ADDRESS_LENGTH.

If you omit **LOCLADDR**, a local address is automatically allocated.

Note, that you can set **LOCLADDR** for a C client using the Client Channel Definition Table (CCDT).

All the parameters are optional. Omitting the ip-addr part of the address is useful to enable the configuration of a fixed port number for an IP firewall. Omitting the port number is useful to select a particular network adapter without having the identify a unique local port number. The TCP/IP stack generates a unique port number.

Specify *[,[ip-addr][(low-port[,high-port])]]* multiple times for each additional local address. Use multiple local addresses if you want to specify a specific subset of local network adapters. You can also use *[,[ip-addr][(low-port[,high-port])]]* to represent a particular local network address on different servers that are part of a multi-instance queue manager configuration.

ip-addr
ip-addr is specified in one of three forms:

IPv4 dotted decimal
For example, 192.0.2.1

IPv6 hexadecimal notation
For example, 2001:DB8:0:0:0:0:0:0

Alphanumeric host name form
For example WWW.EXAMPLE.COM

low-port and high-port
low-port and high-port are port numbers enclosed in parentheses.

The following table shows how the **LOCLADDR** parameter can be used:

Table 98. Examples of how the **LOCLADDR** *parameter can be used*

LOCLADDR	**Meaning**
9.20.4.98	Channel binds to this address locally
9.20.4.98, 9.20.4.99	Channel binds to either IP address. The address might be two network adapters on one server, or a different network adapter on two different servers in a multi-instance configuration.
9.20.4.98(1000)	Channel binds to this address and port 1000 locally
9.20.4.98(1000,2000)	Channel binds to this address and uses a port in the range 1000 - 2000 locally
(1000)	Channel binds to port 1000 locally
(1000,2000)	Channel binds to port in range 1000 - 2000 locally

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, RQSTR, CLNTCONN, CLUSSDR, OR CLUSRCVR.

On CLUSSDR channels, the IP address and port to which the outbound channel binds, is a combination of fields. It is a concatenation of the IP address, as defined in the **LOCLADDR** parameter, and the port range from the cluster cache. If there is no port range in the cache, the port range defined in the **LOCLADDR** parameter is used.

z/OS This port range does not apply to z/OS systems.

Even though this parameter is similar in form to **CONNAME**, it must not be confused with it. The **LOCLADDR** parameter specifies the characteristics of the local communications, whereas the **CONNAME** parameter specifies how to reach a remote queue manager.

When a channel is started, the values specified for **CONNAME** and **LOCLADDR** determine the IP stack to be used for communication; see Table 3 and Local Address (**LOCLADDR**).

If the TCP/IP stack for the local address is not installed or configured, the channel does not start and an exception message is generated.

z/OS For example, on z/OS systems, the message is "CSQO015E: Command issued but no reply received." The message indicates that the connect() request specifies an interface address that is not known on the default IP stack. To direct the connect() request to the alternative stack, specify the **LOCLADDR** parameter in the channel definition as either an interface on the alternative stack, or a DNS host name. The same specification also works for listeners that might not use the default stack. To find the value to code for **LOCLADDR**, run the **NETSTAT HOME** command on the IP stacks that you want to use as alternatives.

Table 99. How the IP stack to be used for communication is determined

Protocols supported	CONNAME	Action of channel	
IPv4 only	IPv4 address [1]		Channel binds to IPv4 stack
	IPv6 address [2]		Channel fails to resolve **CONNAME**
	IPv4 and 6 host name [3]		Channel binds to IPv4 stack
	IPv4 address	IPv4 address	Channel binds to IPv4 stack
	IPv6 address	IPv4 address	Channel fails to resolve **CONNAME**
	IPv4 and 6 host name	IPv4 address	Channel binds to IPv4 stack
	Any address [4]	IPv6 address	Channel fails to resolve **LOCLADDR**
	IPv4 address	IPv4 and 6 host name	Channel binds to IPv4 stack
	IPv6 address	IPv4 and 6 host name	Channel fails to resolve **CONNAME**
	IPv4 and 6 host name	IPv4 and 6 host name	Channel binds to IPv4 stack
IPv4 and IPv6	IPv4 address		Channel binds to IPv4 stack
	IPv6 address		Channel binds to IPv6 stack
	IPv4 and 6 host name		Channel binds to stack determined by **IPADDRV**
	IPv4 address	IPv4 address	Channel binds to IPv4 stack
	IPv6 address	IPv4 address	Channel fails to resolve **CONNAME**
	IPv4 and 6 host name	IPv4 address	Channel binds to IPv4 stack
	IPv4 address	IPv6 address	Channel maps **CONNAME** to IPv6 [5]
	IPv6 address	IPv6 address	Channel binds IPv6 stack
	IPv4 and 6 host name	IPv6 address	Channel binds IPv6 stack
	IPv4 address	IPv4 and 6 host name	Channel binds to IPv4 stack
	IPv6 address	IPv4 and 6 host name	Channel binds to IPv6 stack
	IPv4 and 6 host name	IPv4 and 6 host name	Channel binds to stack determined by **IPADDRV**

Table 99. How the IP stack to be used for communication is determined (continued)

Protocols supported	CONNAME	Action of channel	
IPv6 only	IPv4 address		Channel maps **CONNAME** to IPv6 [5]
	IPv6 address		Channel binds to IPv6 stack
	IPv4 and 6 host name		Channel binds to IPv6 stack
	Any address	IPv4 address	Channel fails to resolve **LOCLADDR**
	IPv4 address	IPv6 address	Channel maps **CONNAME** to IPv6 [5]
	IPv6 address	IPv6 address	Channel binds to IPv6 stack
	IPv4 and 6 host name	IPv6 address	Channel binds to IPv6 stack
	IPv4 address	IPv4 and 6 host name	Channel maps **CONNAME** to IPv6 [5]
	IPv6 address	IPv4 and 6 host name	Channel binds to IPv6 stack
	IPv4 and 6 host name	IPv4 and 6 host name	Channel binds to IPv6 stack

Notes:

1. IPv4 address. An IPv4 host name that resolves only to an IPv4 network address or a specific dotted notation IPv4 address, for example `1.2.3.4`. This note applies to all occurrences of ' IPv4 address' in this table.
2. IPv6 address. An IPv6 host name that resolves only to an IPv6 network address or a specific hexadecimal notation IPv6 address, for example `4321:54bc`. This note applies to all occurrences of ' IPv6 address' in this table.
3. IPv4 and 6 host name. A host name that resolves to both IPv4 and IPv6 network addresses. This note applies to all occurrences of ' IPv4 and 6 host name' in this table.
4. Any address. IPv4 address, IPv6 address, or IPv4 and 6 host name. This note applies to all occurrences of 'Any address' in this table.
5. Maps IPv4 **CONNAME** to IPv4 mapped IPv6 address. IPv6 stack implementations that do not support IPv4 mapped IPv6 addressing fail to resolve the **CONNAME**. Mapped addresses might require protocol translators in order to be used. The use of mapped addresses is not recommended.

AMQP: LOCLADDR(*ip-addr*)

Note: For the format of **LOCLADDR** that other IBM MQ channels use, see the previous parameter **LOCLADDR**.

For AMQP channels, **LOCLADDR** is the local communications address for the channel. Use this parameter if you want to force the client to use a particular IP address. **LOCLADDR** is also useful to force a channel to use an IPv4 or IPv6 address if a choice is available, or to use a particular network adapter on a system with multiple network adapters.

The maximum length of **LOCLADDR** is `MQ_LOCAL_ADDRESS_LENGTH`.

If you omit **LOCLADDR**, a local address is automatically allocated.

ip-addr

`ip-addr` is a single network address, specified in one of three forms:

IPv4 dotted decimal
For example, `192.0.2.1`

IPv6 hexadecimal notation
For example, `2001:DB8:0:0:0:0:0:0`

Alphanumeric host name form
For example, `WWW.EXAMPLE.COM`

If an IP address is entered, only the address format is validated. The IP address itself is not validated.

LONGRTY(*integer*)

The **LONGRTY** parameter specifies the maximum number of further attempts that are made by a

SDR, SVR, or CLUSSDR channel to connect to a remote queue manager. The interval between attempts is specified by **LONGTMR**. The **LONGRTY** parameter takes effect if the count specified by **SHORTRTY** is exhausted.

If this count is exhausted without success, an error is logged to the operator, and the channel stops. In this circumstance, the channel must be restarted with a command. It is not started automatically by the channel initiator.

The **LONGRTY** value must be in the range 0 - 9999999.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, CLUSSDR, or CLUSRCVR.

A channel attempts to reconnect if it fails to connect initially, whether it is started automatically by the channel initiator or by an explicit command. It also tries to connect again if the connection fails after the channel successfully connecting. If the cause of the failure is such that more attempts are unlikely to be successful, they are not attempted.

LONGTMR(*integer*)

For **LONGRTY**, **LONGTMR** is the maximum number of seconds to wait before reattempting connection to the remote queue manager.

The time is approximate; zero means that another connection attempt is made as soon as possible.

The interval between attempting to reconnect might be extended if the channel has to wait to become active.

The **LONGTMR** value must be in the range 0 - 9999999.

Note: For implementation reasons, the maximum **LONGTMR** value is 999,999; values exceeding this maximum are treated as 999,999. Similarly, the minimum interval between attempting to reconnect is 2 seconds. Values less than this minimum are treated as 2 seconds.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, CLUSSDR, or CLUSRCVR.

MAXINST(*integer*)

The maximum number of simultaneous instances of an individual SVRCONN channel or AMQP channel that can be started.

The value must be in the range 0 - 999999999.

A value of zero prevents all client access on this channel.

New instances of SVRCONN channels cannot start if the number of running instances equals or exceeds the value of this parameter. If **MAXINST** is changed to less than the number of instances of the SVRCONN channel that are currently running, the number of running instances is not affected.

V 9.0.0 If an AMQP client attempts to connect to an AMQP channel, and the number of connected clients has reached **MAXINST**, the channel closes the connection with a close frame. The close frame contains the following message: `amqp:resource-limit-exceeded`. If a client connects with an ID that is already connected (that is, it performs a client-takeover), and the client is permitted to take over the connection, the takeover will succeed regardless of whether the number of connected clients has reached **MAXINST**.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SVRCONN or AMQP.

MAXINSTC(*integer*)

The maximum number of simultaneous individual SVRCONN channels that can be started from a single client. In this context, connections that originate from the same remote network address are regarded as coming from the same client.

The value must be in the range 0 - 999999999.

A value of zero prevents all client access on this channel.

If you reduce the value of **MAXINSTC** to less than the number of instances of the SVRCONN channel that is currently running from an individual client, the running instances are not affected. New SVRCONN instances from that client cannot start until the client is running fewer instances than the value of **MAXINSTC**.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SVRCONN.

MAXMSGL(*integer*)
Specifies the maximum message length that can be transmitted on the channel. This parameter is compared with the value for the partner and the actual maximum used is the lower of the two values. The value is ineffective if the MQCB function is being executed and the channel type (**CHLTYPE**) is SVRCONN.

The value zero means the maximum message length for the queue manager; see ALTER QMGR MAXMSGL.

On Multiplatforms, specify a value in the range zero to the maximum message length for the queue manager.

On z/OS, specify a value in the range 0 - 104857600 bytes (100 MB).

Note that by adding the digital signature and key to the message, Advanced Message Security increases the length of the message.

MCANAME(*string*)
Message channel agent name.

This parameter is reserved, and if specified must be set to blanks (maximum length 20 characters).

MCATYPE
Specifies whether the message-channel-agent program on an outbound message channel runs as a thread or a process.

PROCESS
The message channel agent runs as a separate process.

THREAD The message channel agent runs as a separate thread

In situations where a threaded listener is required to service many incoming requests, resources can become strained. In this case, use multiple listener processes and target incoming requests at specific listeners though the port number specified on the listener.

On Multiplatforms, this parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, RQSTR, CLUSSDR, or CLUSRCVR.

On z/OS, this parameter is supported only for channels with a channel type of CLUSRCVR. When specified in a CLUSRCVR definition, **MCATYPE** is used by a remote machine to determine the corresponding CLUSSDR definition.

MCAUSER(*string*)
Message channel agent user identifier.

Note: An alternative way of providing a user ID for a channel to run under is to use channel authentication records. With channel authentication records, different connections can use the same channel while using different credentials. If both **MCAUSER** on the channel is set and channel authentication records are used to apply to the same channel, the channel authentication records take precedence. The **MCAUSER** on the channel definition is only used if the channel authentication record uses **USERSRC(CHANNEL)**. For more details, see Channel authentication records

This parameter interacts with **PUTAUT**, see **PUTAUT**.

If **MCAUSER** is nonblank, a user identifier is used by the message channel agent for authorization to access IBM MQ resources. If **PUTAUT** is DEF, authorization includes authorization to put the message to the destination queue for RCVR or RQSTR channels.

If it is blank, the message channel agent uses its default user identifier.

The default user identifier is derived from the user ID that started the receiving channel. The possible values are:

z/OS **z/OS**
The user ID assigned to the channel-initiator started task by the z/OS started-procedures table.

TCP/IP, Multiplatforms
The user ID from the inetd.conf entry, or the user that started the listener.

SNA, Multiplatforms
The user ID from the SNA server entry. In the absence of the user ID from the SNA server entry, the user from the incoming attach request, or the user that started the listener.

NetBIOS or SPX
The user ID that started the listener.

The maximum length of the string is:

- Windows 64 characters on Windows. On Windows, you can optionally qualify a user identifier with the domain name in the format user@domain.
- 12 characters on other platforms.

This parameter is not valid for channels with a channel type (**CHLTYPE**) of SDR, SVR, CLNTCONN, CLUSSDR.

MODENAME(*string*)
LU 6.2 mode name (maximum length 8 characters).

This parameter is valid only for channels with a transport type (**TRPTYPE**) of LU62. If **TRPTYPE** is not LU62, the data is ignored and no error message is issued.

If specified, this parameter must be set to the SNA mode name unless the **CONNAME** contains a side-object name. If **CONNAME** is a a side-object name it must be set to blanks. The actual name is then taken from the CPI-C Communications Side Object, or APPC side information data set, see Configuration parameters for an LU 6.2 connection.

This parameter is not valid for channels with a channel type (**CHLTYPE**) of RCVR or SVRCONN.

MONCHL Controls the collection of online monitoring data for channels:

QMGR Collect monitoring data according to the setting of the queue manager parameter **MONCHL**.

OFF Monitoring data collection is turned off for this channel.

LOW If the value of the queue manager **MONCHL** parameter is not NONE, online monitoring data is turned on. Data us collected at a low rate for this channel.

MEDIUM If the value of the queue manager **MONCHL** parameter is not NONE, online monitoring data is turned on. Data us collected at a medium rate for this channel.

HIGH If the value of the queue manager **MONCHL** parameter is not NONE, online monitoring data is turned on. Data us collected at a high rate for this channel.

Changes to this parameter take effect only on channels started after the change occurs.

For cluster channels, the value of this parameter is not replicated in the repository and, therefore, not used in the auto-definition of CLUSSDR channels. For auto-defined CLUSSDR channels, the value of this parameter is taken from the queue manager attribute **MONACLS**. This value might then be overridden in the channel auto-definition exit.

MRDATA(*string*)
Channel message-retry exit user data. The maximum length is 32 characters.

This parameter is passed to the channel message-retry exit when it is called.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of RCVR, RQSTR, or CLUSRCVR.

MREXIT(*string*)
Channel message-retry exit name.

The format and maximum length of the name is the same as for **MSGEXIT**, however you can specify only one message-retry exit.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of RCVR, RQSTR, or CLUSRCVR.

MRRTY(*integer*)
The number of times the channel tries again before it decides it cannot deliver the message.

This parameter controls the action of the MCA only if the message-retry exit name is blank. If the exit name is not blank, the value of **MRRTY** is passed to the exit to use. The number of attempts to redeliver the message is controlled by the exit, and not by this parameter.

The value must be in the range 0 - 999999999. A value of zero means that no attempts to redeliver the message are tried.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of RCVR, RQSTR, or CLUSRCVR.

MRTMR(*integer*)
The minimum interval of time that must pass before the channel can try the MQPUT operation again. The time interval is in milliseconds.

This parameter controls the action of the MCA only if the message-retry exit name is blank. If the exit name is not blank, the value of **MRTMR** is passed to the exit to use. The number of attempts to redeliver the message is controlled by the exit, and not by this parameter.

The value must be in the range 0 - 999999999. A value of zero means that if the value of **MRRTY** is greater than zero, the channel reattempts delivery as soon as possible.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of RCVR, RQSTR, or CLUSRCVR.

MSGDATA(*string*)
User data for the channel message exit. The maximum length is 32 characters.

This data is passed to the channel message exit when it is called.

ULW On UNIX, Linux, and Windows, you can specify data for more than one exit program by specifying multiple strings separated by commas. The total length of the field must not exceed 999 characters.

IBM i On IBM i, you can specify up to 10 strings, each of length 32 characters. The first string of data is passed to the first message exit specified, the second string to the second exit, and so on.

z/OS On z/OS, you can specify up to eight strings, each of length 32 characters. The first string of data is passed to the first message exit specified, the second string to the second exit, and so on.

On other platforms, you can specify only one string of message exit data for each channel.

Note: This parameter is accepted but ignored for SVRCONN and CLNTCONN channels.

MSGEXIT(*string*)

Channel message exit name.

If **MSGEXIT** is nonblank the exit is called at the following times:

- Immediately after a SDR or SVR channel retrieves a message from the transmission queue.
- Immediately before a RQSTR channel puts a message on destination queue.
- When the channel is initialized or ended.

The exit is passed the entire application message and transmission queue header for modification.

MSGEXIT is accepted and ignored by CLNTCONN and SVRCONN channels. CLNTCONN or SVRCONN channels do not call message exits.

The format and maximum length of the exit name depends on the platform; see Table 100.

If the **MSGEXIT**, **MREXIT**, **SCYEXIT**, **SENDEXIT**, and **RCVEXIT** parameters are all left blank, the channel user exit is not invoked. If any of these parameters is nonblank, the channel exit program is called. You can enter text string for these parameters. The maximum length of the string is 128 characters.

Table 100. Message exit format and length

Platform	Exit name format	Maximum length	Comment
UNIX and Linux	*libraryname* (*functionname*)	128	You can specify the name of more than one exit program. Specify multiple strings separated by commas. However, the total number of characters specified must not exceed 999.
Windows	*dllname* (*functionname*)	128	1. You can specify the name of more than one exit program. Specify multiple strings separated by commas. However, the total number of characters specified must not exceed 999. 2. *dllname* is specified without the suffix (.DLL).
IBM i	*progname libname*	20	1. You can specify the names of up to 10 exit programs by specifying multiple strings separated by commas. 2. *program name* occupies the first 10 characters and *libname* the second 10 characters. If necessary, both fields are padded to the right with blanks.
z/OS	*loadModuleName*	8	1. You can specify the names of up to eight exit programs by specifying multiple strings separated by commas. 2. 128 characters are allowed for exit names for CLNTCONN channels, subject to a maximum total length including commas of 999.

NETPRTY(*integer*)
The priority for the network connection. Distributed queuing chooses the path with the highest priority if there are multiple paths available. The value must be in the range 0 - 9; 0 is the lowest priority.

This parameter is valid only for CLUSRCVR channels.

NPMSPEED
The class of service for nonpersistent messages on this channel:

FAST Fast delivery for nonpersistent messages; messages might be lost if the channel is lost. Messages are retrieved using MQGMO_SYNCPOINT_IF_PERSISTENT and so are not included in the batch unit of work.

NORMAL Normal delivery for nonpersistent messages.

If the value of **NPMSPEED** differs between the sender and receiver, or either one does not support it, NORMAL is used.

Notes:

1. If the active recovery logs for IBM MQ for z/OS are switching and archiving more frequently than expected, given that the messages being sent across a channel are non-persistent, setting NPMSPEED(FAST) on both the sending and receiving ends of the channel can minimize the SYSTEM.CHANNEL.SYNCQ updates.
2. If you are seeing high CPU usage relating to updates to the SYSTEM.CHANNEL.SYNCQ, setting NPMSPEED(FAST) can significantly reduce the CPU usage.

This parameter is valid only for channels with a **CHLTYPE** of SDR, SVR, RCVR, RQSTR, CLUSSDR, or CLUSRCVR.

PASSWORD(*string*)
Password used by the message channel agent when attempting to initiate a secure LU 6.2 session with a remote message channel agent. The maximum length is 12 characters.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, RQSTR, CLNTCONN, or CLUSSDR.

z/OS On z/OS, it is supported only for channels with a channel type (**CHLTYPE**) of CLNTCONN.

Although the maximum length of the parameter is 12 characters, only the first 10 characters are used.

V 9.0.0 **PORT(*integer*)**
The port number used to connect an AMQP channel. The default port for AMQP 1.0 connections is 5672. If you are already using port 5672, you can specify a different port.

PROPCTL
Property control attribute; see **PROPCTL** channel options.

PROPCTL specifies what happens to message properties when a message is sent to another queue manager; see

This parameter is applicable to SDR, SVR, CLUSSDR, and CLUSRCVR channels.

This parameter is optional.

Permitted values are:

COMPAT COMPAT allows applications which expect JMS-related properties to be in an MQRFH2 header in the message data to continue to work unmodified.

Table 101. Results for message properties

Message properties	Result
The message contains a property with a prefix of `mcd.`, `jms.`, `usr.` or `mqext.`	If the **Support** value is `MQPD_SUPPORT_OPTIONAL`, all optional message properties are placed in one or more MQRFH2 headers. This rule does not apply to properties in the message descriptor or extension, which remain in the same place. Optional message properties are moved into the message data before the message it sent to the remote queue manager.
The message does not contain a property with a prefix of `mcd.`, `jms.`, `usr.` or `mqext.`	All message properties, except properties in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.
The message contains a property where the **Support** field of the property descriptor is not set to `MQPD_SUPPORT_OPTIONAL`	The message is rejected with reason `MQRC_UNSUPPORTED_PROPERTY` and treated in accordance with its report options.
The message contains one or more properties where the **Support** field of the property descriptor is set to `MQPD_SUPPORT_OPTIONAL`. Other fields of the property descriptor are set to non-default values.	The properties with non-default values are removed from the message before the message is sent to the remote queue manager.
The MQRFH2 folder that would contain the message property needs to be assigned with the `content='properties'` attribute	The properties are removed to prevent MQRFH2 headers with unsupported syntax flowing to a Version 6 or prior queue manager.

NONE All properties of the message, except properties in the message descriptor or extension, are removed from the message. The properties are removed before the message is sent to the remote queue manager.

If the message contains a property where the **Support** field of the property descriptor is not set to `MQPD_SUPPORT_OPTIONAL` then the message is rejected with reason `MQRC_UNSUPPORTED_PROPERTY`. The error is reported in accordance with the report options set in the message header.

ALL All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except properties in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

PUTAUT

PUTAUT specifies which user identifiers are used to establish authority for a channel. It specifies the user identifier to put messages to the destination queue using a message channel, or to run an MQI call using an MQI channel.

DEF The default user ID is used.

z/OS On z/OS, DEF might involve using both the user ID received from the network and that derived from **MCAUSER**.

CTX The user ID from the *UserIdentifier* field of the message descriptor is used.

z/OS On z/OS, CTX might involve also using the user ID received from the network or that derived from **MCAUSER**, or both.

z/OS **ONLYMCA**
The user ID derived from MCAUSER is used. Any user ID received from the network is not used. This value is supported only on z/OS.

z/OS **ALTMCA**
The user ID from the *UserIdentifier* field of the message descriptor is used. Any user ID received from the network is not used. This value is supported only on z/OS.

z/OS On z/OS, the user IDs that are checked, and how many user IDs are checked, depends on the setting of the MQADMIN RACF class hlq.RESLEVEL profile. Depending on the level of access the user ID of the channel initiator has to hlq.RESLEVEL, zero, one, or two user IDs are checked. To see how many user IDs are checked, see RESLEVEL and the channel initiator connection. For more information about which user IDs are checked, see User IDs used by the channel initiator.

z/OS On z/OS, this parameter is valid only for channels with a channel type (**CHLTYPE**) of RCVR, RQSTR, CLUSRCVR, or SVRCONN. CTX and ALTMCA are not valid for SVRCONN channels.

Multi On Multiplatforms, this parameter is valid only for channels with a channel type (**CHLTYPE**) of RCVR, RQSTR, or CLUSRCVR.

QMNAME(*string*)
Queue manager name.

For CLNTCONN channels, **QMNAME** is the name of a queue manager to which an IBM MQ MQI client application can request connection. **QMNAME** is not necessarily the same as the name of the queue manager on which the channel is defined; see Queue manager groups in the CCDT.

For channels of other types, the **QMNAME** parameter is not valid.

z/OS **QSGDISP**
This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

Table 102. Object dispositions for **QSGDISP** *options*

QSGDISP	**DEFINE**
COPY	The object is defined on the page set of the queue manager that executes the command using the **QSGDISP(GROUP)** object of the same name as the **LIKE** object.
GROUP	The object definition resides in the shared repository but only if the queue manager is in a queue-sharing group. If the definition is successful, the following command is generated. The command is sent to all active queue managers in the queue-sharing group to make or refresh local copies on page set zero: `DEFINE CHANNEL(channel-name) CHLTYPE(type)` `REPLACE QSGDISP(COPY)` The **DEFINE** command for the group object takes effect regardless of whether the generated command with **QSGDISP(COPY)** fails.
PRIVATE	Not permitted.
QMGR	The object is defined on the page set of the queue manager that executes the command.

RCVDATA(*string*)
Channel receive exit user data (maximum length 32 characters).

This parameter is passed to the channel receive exit when it is called.

ULW On UNIX, Linux, and Windows, you can specify data for more than one exit program by specifying multiple strings separated by commas. The total length of the field must not exceed 999 characters.

IBM i On IBM i, you can specify up to 10 strings, each of length 32 characters. The first string of data is passed to the first receive exit specified, the second string to the second exit, and so on.

z/OS On z/OS, you can specify up to eight strings, each of length 32 characters. The first string of data is passed to the first receive exit specified, the second string to the second exit, and so on.

On other platforms, you can specify only one string of receive exit data for each channel.

RCVEXIT(*string*)
Channel receive exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately before the received network data is processed.

  The exit is given the complete transmission buffer as received. The contents of the buffer can be modified as required.
- At initialization and termination of the channel.

ULW On UNIX, Linux, and Windows, you can specify the name of more than one exit program by specifying multiple strings separated by commas. However, the total number of characters specified must not exceed 999.

IBM i On IBM i, you can specify the names of up to 10 exit programs by specifying multiple strings separated by commas.

z/OS On z/OS, you can specify the names of up to eight exit programs by specifying multiple strings separated by commas.

On other platforms, you can specify only one receive exit name for each channel.

The format and maximum length of the name is the same as for **MSGEXIT**.

REPLACE and NOREPLACE
Replace the existing definition with this one, or not. This parameter is optional.

z/OS On z/OS it must have the same disposition. Any object with a different disposition is not changed.

REPLACE
The definition replaces any existing definition of the same name. If a definition does not exist, one is created. **REPLACE** does not alter the channel status.

NOREPLACE
The definition does not replace any existing definition of the same name.

SCYDATA(*string*)
Channel security exit user data (maximum length 32 characters).

This parameter is passed to the channel security exit when it is called.

SCYEXIT(*string*)
Channel security exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately after establishing a channel.

  Before any messages are transferred, the exit is able to instigate security flows to validate connection authorization.
- Upon receipt of a response to a security message flow.

  Any security message flows received from the remote processor on the remote queue manager are given to the exit.
- At initialization and termination of the channel.

The format and maximum length of the name is the same as for **MSGEXIT** but only one name is allowed.

SENDDATA(*string*)
Channel send exit user data. The maximum length is 32 characters.

This parameter is passed to the channel send exit when it is called.

ULW On UNIX, Linux, and Windows, you can specify data for more than one exit program by specifying multiple strings separated by commas. The total length of the field must not exceed 999 characters.

IBM i On IBM i, you can specify up to 10 strings, each of length 32 characters. The first string of data is passed to the first send exit specified, the second string to the second exit, and so on.

z/OS On z/OS, you can specify up to eight strings, each of length 32 characters. The first string of data is passed to the first send exit specified, the second string to the second exit, and so on.

On other platforms, you can specify only one string of send exit data for each channel.

SENDEXIT(*string*)
Channel send exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately before data is sent out on the network.

  The exit is given the complete transmission buffer before it is transmitted. The contents of the buffer can be modified as required.
- At initialization and termination of the channel.

ULW On UNIX, Linux, and Windows, you can specify the name of more than one exit program by specifying multiple strings separated by commas. However, the total number of characters specified must not exceed 999.

IBM i On IBM i, you can specify the names of up to 10 exit programs by specifying multiple strings separated by commas.

z/OS On z/OS, you can specify the names of up to eight exit programs by specifying multiple strings separated by commas.

On other platforms, you can specify only one send exit name for each channel.

The format and maximum length of the name is the same as for **MSGEXIT**.

SEQWRAP(*integer*)
When this value is reached, sequence numbers wrap to start again at 1.

This value is nonnegotiable and must match in both the local and remote channel definitions.

The value must be in the range 100 - 999999999.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, RCVR, RQSTR, CLUSSDR, or CLUSRCVR.

SHARECNV(*integer*)
Specifies the maximum number of conversations that can be sharing each TCP/IP channel instance. A **SHARECNV** value of:

1 Specifies no sharing of conversations over a TCP/IP channel instance. Client heart beating is available whether in an MQGET call or not. Read ahead and client asynchronous consumption are also available, and channel quiescing is more controllable.

0 Specifies no sharing of conversations over a TCP/IP channel instance.

The value must be in the range zero through 999999999.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of CLNTCONN or SVRCONN. If the CLNTCONN **SHARECNV** value does not match the SVRCONN **SHARECNV** value, the lower of the two values is used. This parameter is ignored for channels with a transport type (**TRPTYPE**) other than TCP.

All the conversations on a socket are received by the same thread.

High **SHARECNV** limits have the advantage of reducing queue manager thread usage. If many conversations sharing a socket are all busy, there is a possibility of delays. The conversations contend with one another to use the receiving thread. In this situation, a lower **SHARECNV** value is better.

The number of shared conversations does not contribute to the **MAXINST** or **MAXINSTC** totals.

Note: You should restart the client for this change to take effect.

SHORTRTY(*integer*)

SHORTRTY specifies the maximum number of attempts that are made by a SDR, SVR, or CLUSSDR channel to connect to the remote queue manager, at intervals specified by **SHORTTMR**. After the number of attempts is exhausted, the channel tries to reconnect using to the schedule defined by **LONGRTY**.

The value must be in the range 0 - 999999999.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, CLUSSDR, or CLUSRCVR.

A channel attempts to reconnect if it fails to connect initially, whether it is started automatically by the channel initiator or by an explicit command. It also tries to connect again if the connection fails after the channel successfully connecting. If the cause of the failure is such that more attempts are unlikely to be successful, they are not attempted.

SHORTTMR(*integer*)

For **SHORTRTY**, **SHORTTMR** is the maximum number of seconds to wait before reattempting connection to the remote queue manager.

The time is approximate. From IBM MQ Version 8.0, zero means that another connection attempt is made as soon as possible.

The interval between attempting to reconnect might be extended if the channel has to wait to become active.

The value must be in the range 0 - 999999999.

Note: For implementation reasons, the maximum **SHORTTMR** value is 999,999; values exceeding this maximum are treated as 999,999. From IBM MQ Version 8.0, if SHORTTMR is set to 1 then the minimum interval between attempting to connect is 2 seconds.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR, SVR, CLUSSDR, or CLUSRCVR.

SSLCAUTH

SSLCAUTH defines whether IBM MQ requires a certificate from the TLS client. The TLS client is the initiating end of the channel. **SSLCAUTH** is applied to the TLS server, to determine the behavior required of the client. The TLS server is the end of the channel that receives the initiation flow.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of RCVR, SVRCONN, CLUSRCVR, SVR, OR RQSTR.

The parameter is used only for channels with **SSLCIPH** specified. If **SSLCIPH** is blank, the data is ignored and no error message is issued.

REQUIRED

IBM MQ requires and validates a certificate from the TLS client.

OPTIONAL
The peer TLS client system might still send a certificate. If it does, the contents of this certificate are validated as normal.

SSLCIPH*(string)*

SSLCIPH specifies the CipherSpec that is used on the channel. The maximum length is 32 characters. This parameter is valid on all channel types which use transport type **TRPTYPE(TCP)**. If the **SSLCIPH** parameter is blank, no attempt is made to use TLS on the channel.

Specify the name of the CipherSpec you are using. The CipherSpecs that can be used with IBM MQ SSL support are shown in the following table. The **SSLCIPH** values must specify the same CipherSpec on both ends of the channel.

Note: IBM i z/OS On IBM MQ for z/OS, you can also specify the two digit hexadecimal code of a CipherSpec, whether or not it appears in the table. On IBM i, you can also specify the two digit hexadecimal code of a CipherSpec, whether or not it appears in the table. Also, on IBM i, installation of AC3 is a prerequisite for the use of SSL.

Platform support[1]	CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[2]	Suite B
ULW z/OS	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	SHA-1	AES	128	Yes	No
ULW z/OS	TLS_RSA_WITH_AES_256_CBC_SHA [3]	TLS 1.0	SHA-1	AES	256	Yes	No
All	ECDHE_ECDSA_AES_128_CBC_SHA256	TLS 1.2	SHA-256	AES	128	Yes	No
All	ECDHE_ECDSA_AES_256_CBC_SHA384 [3]	TLS 1.2	SHA-384	AES	256	Yes	No
Multi	ECDHE_ECDSA_AES_128_GCM_SHA256 [4]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	128 bit
Multi	ECDHE_ECDSA_AES_256_GCM_SHA384[3] [4]	TLS 1.2	AEAD AES-128 GCM	AES	256	Yes	192 bit
All	ECDHE_RSA_AES_128_CBC_SHA256	TLS 1.2	SHA-256	AES	128	Yes	No
All	ECDHE_RSA_AES_256_CBC_SHA384 [3]	TLS 1.2	SHA-384	AES	256	Yes	No
Multi	ECDHE_RSA_AES_128_GCM_SHA256 [4]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	No
Multi	ECDHE_RSA_AES_256_GCM_SHA384 [3] [4]	TLS 1.2	AEAD AES-128 GCM	AES	SHA384	Yes	No
[5] IBM i	ECDHE_ECDSA_RC4_128_SHA256	TLS 1.2	AEAD AES-128 GCM	AES	SHA256	Yes	No
IBM i	ECDHE_ECDSA_3DES_EDE_CBC_SHA256	TLS 1.2	AEAD AES-128 GCM	3DES	SHA256	Yes	No
IBM i	ECDHE_ECDSA_NULL_SHA256	TLS 1.2	AEAD AES-128 GCM	ECDSA	SHA256	Yes	No
IBM i	ECDHE_ECDSA_AES_256_GCM_SHA384 [3] [4]	TLS 1.2	AEAD AES-128 GCM	AES	SHA384	Yes	No

Platform support[1]	CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[2]	Suite B
ULW z/OS	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	SHA-256	AES	128	Yes	No
ULW z/OS	TLS_RSA_WITH_AES_256_CBC_SHA256 [3]	TLS 1.2	SHA-256	AES	256	Yes	No
Multi	TLS_RSA_WITH_AES_128_GCM_SHA256 [4]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	No
Multi	TLS_RSA_WITH_AES_256_GCM_SHA384 [3] [4]	TLS 1.2	AEAD AES-128 GCM	AES	256	Yes	No

Notes:

1. If no specific platform is noted, the CipherSpec is available on all platforms. For a list of platforms covered by each platform icon, see Release and platform icons in the product documentation.
2. Specifies whether the CipherSpec is FIPS-certified on a FIPS-certified platform. See Federal Information Processing Standards (FIPS) for an explanation of FIPS.
3. This CipherSpec cannot be used to secure a connection from the IBM MQ Explorer to a queue manager unless the appropriate unrestricted policy files are applied to the JRE used by the Explorer.
4. Following a recommendation by NIST, GCM CipherSpecs now have a restriction which means that after 2^22 TLS records are sent, using the same session key, the connection will be terminated with message AMQ9288.

   To prevent this error from happening: avoid using GCM Ciphers, enable secret key reset, or start your IBM MQ queue manager with the environment variable GSK_ENFORCE_GCM_RESTRICTION=GSK_FALSE set.
   Important: The GCM restriction is active, regardless of the FIPS mode being used.
5. IBM i The CipherSpecs listed as supported on IBM i, apply to Versions 7.2 and 7.3 of IBM i.

When you request a personal certificate, you specify a key size for the public and private key pair. The key size that is used during the SSL handshake can depend on the size stored in the certificate and on the CipherSpec:

- ULW z/OS On z/OS, UNIX, Linux, and Windows, when a CipherSpec name includes `_EXPORT`, the maximum handshake key size is 512 bits. If either of the certificates exchanged during the SSL handshake has a key size greater than 512 bits, a temporary 512-bit key is generated for use during the handshake.
- ULW On UNIX, Linux, and Windows, when a CipherSpec name includes `_EXPORT1024`, the handshake key size is 1024 bits.
- Otherwise the handshake key size is the size stored in the certificate.

SSLPEER ***(string)***

Specifies the certificate filter used by the peer queue manager or client at the other end of the channel. The filter is used to compare with the distinguished name of the certificate. A *distinguished name* is the identifier of the TLS certificate. If the distinguished name in the certificate received from the peer does not match the **SSLPEER** filter, the channel does not start.

Note: An alternative way of restricting connections into channels by matching against the TLS Subject distinguished name, is to use channel authentication records. With channel authentication records, different TLS subject distinguished name patterns can be applied to the same channel. Both **SSLPEER** and a channel authentication record can be applied to the same channel. If so, the inbound certificate must match both patterns in order to connect. For more information, see Channel authentication records.

SSLPEER is optional. If it is not specified, the distinguished name of the peer is not checked at channel startup. The distinguished name from the certificate is still written into the **SSLPEER** definition held in memory, and passed to the security exit. If **SSLCIPH** is blank, the data is ignored and no error message is issued.

This parameter is valid for all channel types.

The **SSLPEER** value is specified in the standard form used to specify a distinguished name. For example:

```
SSLPEER('SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN="H1_C_FR1",O=IBM,C=GB')
```

You can use a semi-colon as a separator instead of a comma.

The possible attribute types supported are:

Table 103. Attribute types supported by SSLPEER

Attribute	Description
SERIALNUMBER	Certificate serial number
MAIL	Email address
E	Email address (Deprecated in preference to MAIL)
UID or USERID	User identifier
CN	Common Name
T	Title
OU	Organizational Unit name
DC	Domain component
O	Organization name
STREET	Street / First line of address
L	Locality name
ST (or SP or S)	State or Province name
PC	Postal code / zipcode
C	Country
UNSTRUCTUREDNAME	Host name
UNSTRUCTUREDADDRESS	IP address
DNQ	Distinguished name qualifier

IBM MQ accepts only uppercase letters for the attribute types.

If any of the unsupported attribute types are specified in the **SSLPEER** string, an error is output either when the attribute is defined, or at run time. When the error is output depends on which platform you are running on. An error implies that the **SSLPEER** string does not match the distinguished name of the flowed certificate.

If the distinguished name of the flowed certificate contains multiple organizational unit (OU) attributes, and **SSLPEER** specifies that these attributes are to be compared, they must be defined in descending hierarchical order. For example, if the distinguished name of the flowed certificate contains the OUs `OU=Large Unit, OU=Medium Unit, OU=Small Unit`, specifying the following **SSLPEER** values works:

```
('OU=Large Unit,OU=Medium Unit')
('OU=*,OU=Medium Unit,OU=Small Unit')
('OU=*,OU=Medium Unit')
```

but specifying the following **SSLPEER** values fails:

```
('OU=Medium Unit,OU=Small Unit')
('OU=Large Unit,OU=Small Unit')
('OU=Medium Unit')
('OU=Small Unit, Medium Unit, Large Unit')
```

As indicated in these examples, attributes at the low end of the hierarchy can be omitted. For example, `('OU=Large Unit,OU=Medium Unit')` is equivalent to `('OU=Large Unit,OU=Medium Unit,OU=*')`

If two DNs are equal in all respects except for their domain component (DC)) values, almost the same matching rules apply as for OUs. The exception is that with DC values, the left-most DC is the lowest-level and most specific, and the comparison ordering differs accordingly.

Any or all the attribute values can be generic, either an asterisk `*` on its own, or a stem with initiating or trailing asterisks. Asterisks allow the **SSLPEER** to match any distinguished name value, or any value starting with the stem for that attribute. You can specify an asterisk at the beginning or end of any attribute value in the DN on the certificate. If you do so, you can still check for an exact match with **SSLPEER**. Specify `\*` to check for an exact match. For example, if you have an attribute of `CN='Test*'` in the DN of the certificate, you use the following command to check for an exact match:

```
SSLPEER('CN=Test\*')
```

Multi The maximum length of the parameter is 1024 bytes on Multiplatforms.

z/OS The maximum length of the parameter is 256 bytes on z/OS.

Channel authentication records provide greater flexibility when using SSLPEER and support 1024 bytes on all platforms.

STATCHL

Controls the collection of statistics data for channels:

QMGR The value of the **STATCHL** parameter of the queue manager is inherited by the channel.

OFF Statistics data collection is turned off for this channel.

LOW If the value of the **STATCHL** parameter of the queue manager is not `NONE`, statistics data collection is turned on. Data is collected at a low rate for this channel.

MEDIUM If the value of the **STATCHL** parameter of the queue manager is not `NONE`, statistics data collection is turned on.Data is collected at a medium rate for this channel.

HIGH If the value of the **STATCHL** parameter of the queue manager is not `NONE`, statistics data collection is turned on.Data is collected at a high rate for this channel.

Changes to this parameter take effect only on channels started after the change occurs.

z/OS On z/OS systems, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying `LOW`, `MEDIUM`, or `HIGH` makes no difference to your results.

For cluster channels, the value of this parameter is not replicated in the repository and therefore is not used in the auto-definition of `CLUSSDR` channels. For auto-defined `CLUSSDR` channels, the value of this parameter is taken from the attribute **STATACLS** of the queue manager. This value might then be overridden in the channel auto-definition exit.

TPNAME(*string*)

LU 6.2 transaction program name (maximum length 64 characters).

This parameter is valid only for channels with a transport type (**TRPTYPE**) of `LU62`.

Set this parameter to the SNA transaction program name, unless the **CONNAME** contains a side-object name in which case set it to blanks. The actual name is taken instead from the CPI-C Communications Side Object, or the APPC side information data set. See Configuration parameters for an LU 6.2 connection

Windows z/OS On Windows SNA Server, and in the side object on z/OS, the TPNAME is wrapped to uppercase.

This parameter is not valid for channels with a channel type (**CHLTYPE**) of RCVR.

V 9.0.0 **TPROOT**

The topic root for an AMQP channel. The default value for TPROOT is SYSTEM.BASE.TOPIC. With this value, the topic string an AMQP client uses to publish or subscribe has no prefix, and the client can exchange messages with other IBM MQ publish/subscribe applications. Alternatively, AMQP clients can publish and subscribe under a different topic prefix, specified in the TPROOT attribute.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of AMQP.

TRPTYPE

Transport type to be used:

LU62 SNA LU 6.2

NETBIOS

Windows Supported on Windows, and DOS.

z/OS Also used on z/OS for defining client-connection channels that connect to servers on the platforms supporting NetBIOS.

SPX Sequenced packet exchange

Windows Supported on Windows, and DOS.

z/OS Also used on z/OS for defining client-connection channels that connect to servers on the platforms supporting SPX.

TCP Transmission Control Protocol - part of the TCP/IP protocol suite.

If you do not enter a value for this parameter, the value specified in the SYSTEM.DEF.*channel-type* definition is used. If the channel is initiated from the other end, no check is made that the correct transport type is specified.

Multi On Multiplatforms, if the SYSTEM.DEF.*channel-type* definition does not exist, you must specify a value.

z/OS On z/OS, if the SYSTEM.DEF.*channel-type* definition does not exist, the default is LU62.

Multi V 9.0.0 **USECLTID**

Specifies that the client ID should be used for authorization checks for an AMQP channel, instead of the MCAUSER attribute value.

NO The MCA user ID should be used for authorization checks.

YES The client ID should be used for authorization checks.

USEDLQ

Determines whether the dead-letter queue is used when messages cannot be delivered by channels.

NO Messages that cannot be delivered by a channel are treated as a failure. The channel either discards the message, or the channel ends, in accordance with the **NPMSPEED** setting.

YES When the **DEADQ** queue manager attribute provides the name of a dead-letter queue, then it is used, else the behavior is as for NO. YES is the default value.

USERID(*string*)

Task user identifier. The maximum length is 12 characters.

This parameter is used by the message channel agent when attempting to initiate a secure LU 6.2 session with a remote message channel agent.

Multi On Multiplatforms, this parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, RQSTR, CLNTCONN, or CLUSSDR.

z/OS On z/OS, this parameter is supported only for CLNTCONN channels.

Although the maximum length of the parameter is 12 characters, only the first 10 characters are used.

At the receiving end, if passwords are encrypted and the LU 6.2 software is using a different encryption method, the channel does not start. The error is diagnosed as invalid security details. You can avoid invalid security details by modifying the receiving SNA configuration to either:

- Turn off password substitution, or
- Define a security user ID and password.

XMITQ(*string*)

Transmission queue name.

The name of the queue from which messages are retrieved. See Rules for naming IBM MQ objects.

This parameter is valid only for channels with a channel type (**CHLTYPE**) of SDR or SVR. For these channel types, this parameter is required.

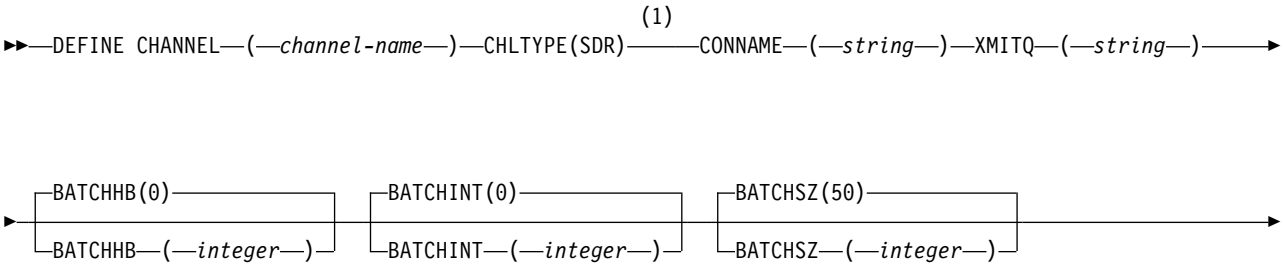
There is a separate syntax diagram for each type of channel.

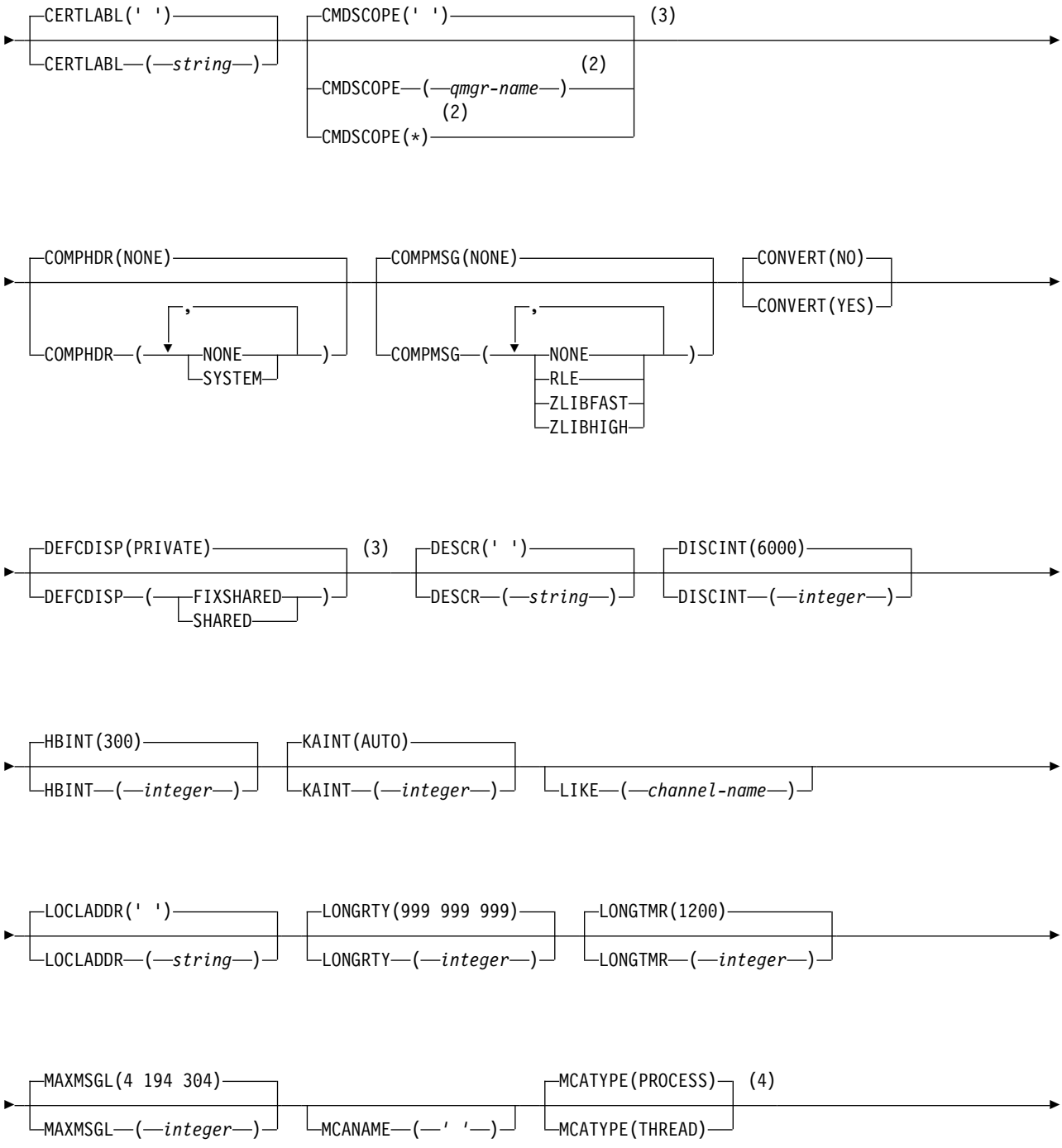
Sender channel:

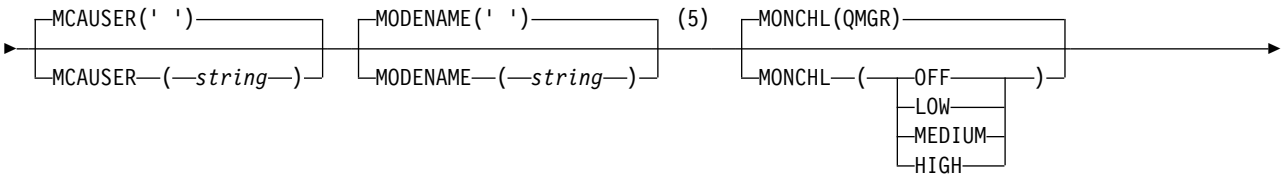
Syntax diagram for a sender channel when using the DEFINE CHANNEL command.

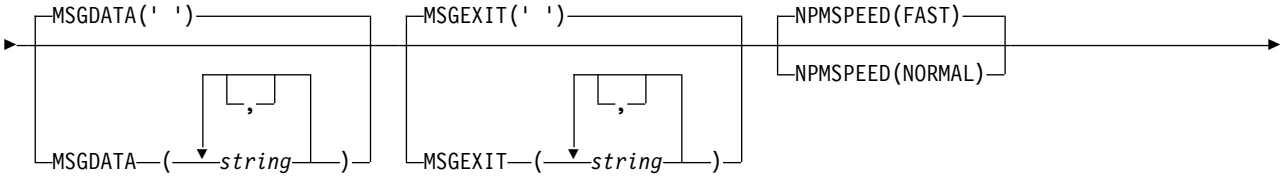
Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See “How to read railroad diagrams” on page 187.

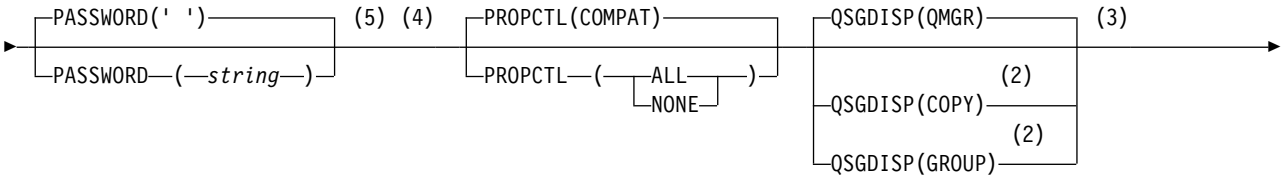
DEFINE CHANNEL

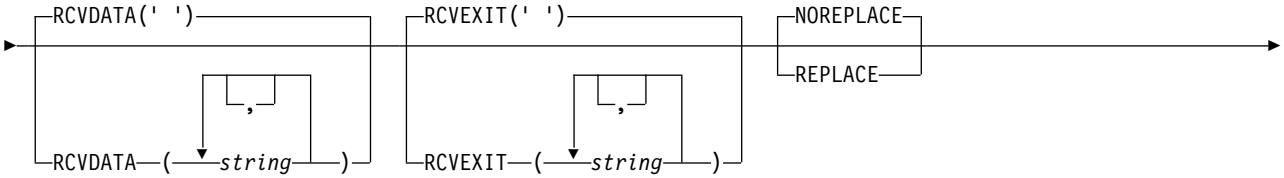


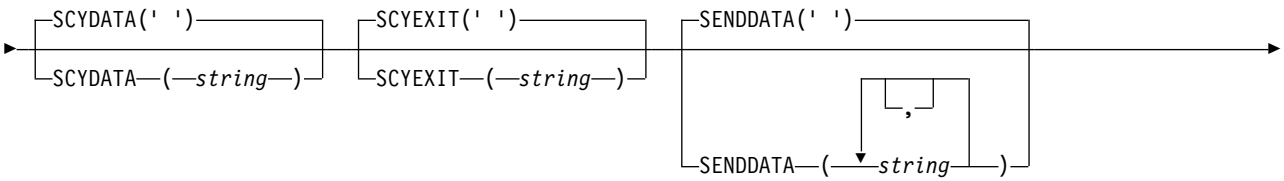

CERTLABL(' ')
CERTLABL—(—string—)
CMDSCOPE(' ')
(3)
CMDSCOPE—(—qmgr-name—)
(2)
CMDSCOPE(*)
(2)
COMPHDR(NONE)
COMPHDR—(—,—NONE—)
SYSTEM
COMPMSG(NONE)
COMPMSG—(—,—NONE—)
RLE
ZLIBFAST
ZLIBHIGH
CONVERT(NO)
CONVERT(YES)
DEFCDISP(PRIVATE)
(3)
DEFCDISP—(—FIXSHARED—)
SHARED
DESCR(' ')
DESCR—(—string—)
DISCINT(6000)
DISCINT—(—integer—)
HBINT(300)
HBINT—(—integer—)
KAINT(AUTO)
KAINT—(—integer—)
LIKE—(—channel-name—)
LOCLADDR(' ')
LOCLADDR—(—string—)
LONGRTY(999 999 999)
LONGRTY—(—integer—)
LONGTMR(1200)
LONGTMR—(—integer—)
MAXMSGL(4 194 304)
MAXMSGL—(—integer—)
MCANAME—(—' '—)
MCATYPE(PROCESS)
(4)
MCATYPE(THREAD)

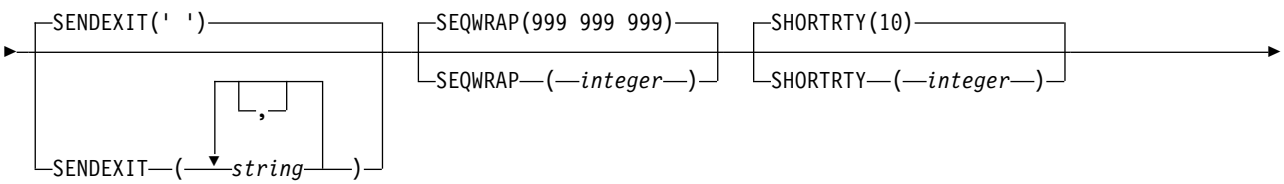

MCAUSER(' ')
MCAUSER(string)
MODENAME(' ')
MODENAME(string)
(5)
MONCHL(QMGR)
MONCHL(OFF)
LOW
MEDIUM
HIGH

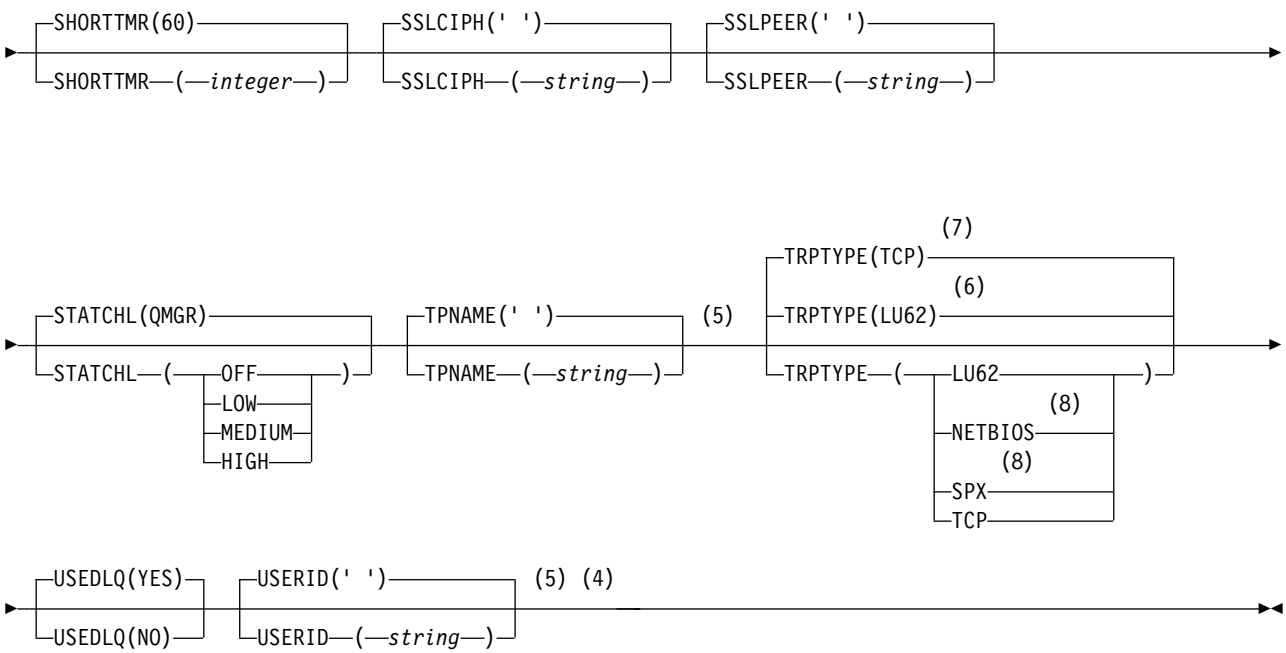

MSGDATA(' ')
MSGDATA(string)
MSGEXIT(' ')
MSGEXIT(string)
NPMSPEED(FAST)
NPMSPEED(NORMAL)


PASSWORD(' ')
PASSWORD(string)
(5) (4)
PROPCTL(COMPAT)
PROPCTL(ALL)
NONE
QSGDISP(QMGR)
(3)
QSGDISP(COPY)
(2)
QSGDISP(GROUP)
(2)


RCVDATA(' ')
RCVDATA(string)
RCVEXIT(' ')
RCVEXIT(string)
NOREPLACE
REPLACE


SCYDATA(' ')
SCYDATA(string)
SCYEXIT(' ')
SCYEXIT(string)
SENDDATA(' ')
SENDDATA(string)


SENDEXIT(' ')
SENDEXIT(string)
SEQWRAP(999 999 999)
SEQWRAP(integer)
SHORTRTY(10)
SHORTRTY(integer)



Notes:

1 This parameter must follow immediately after the channel name z/OS except on z/OS.

2 Valid only on IBM MQ for z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Not valid on z/OS.

5 Valid only if TRPTYPE is LU62.

6 Default for z/OS.

7 Default for Multiplatforms.

8 Valid only on Windows.

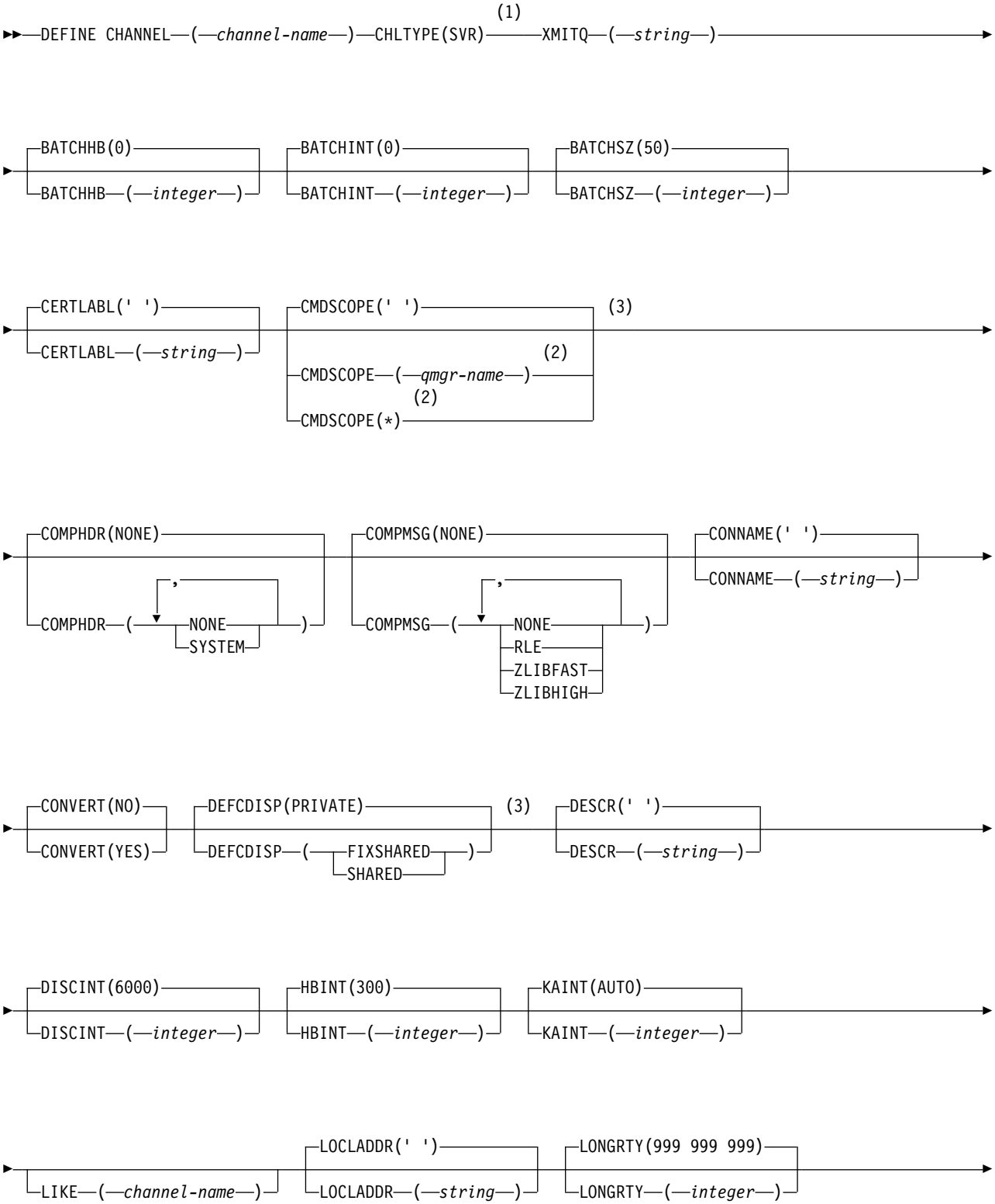
The parameters are described in "DEFINE CHANNEL" on page 575.

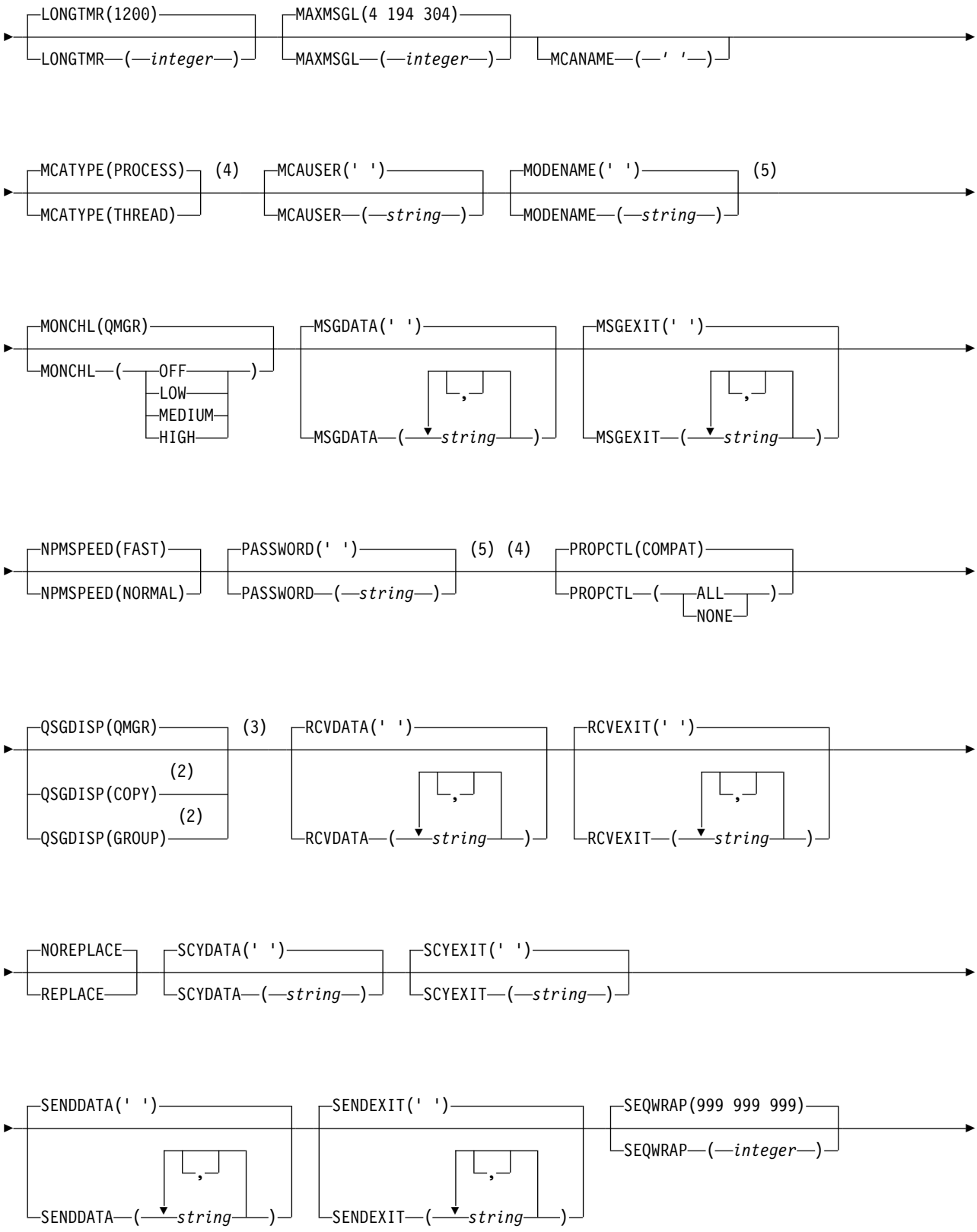
Server channel:

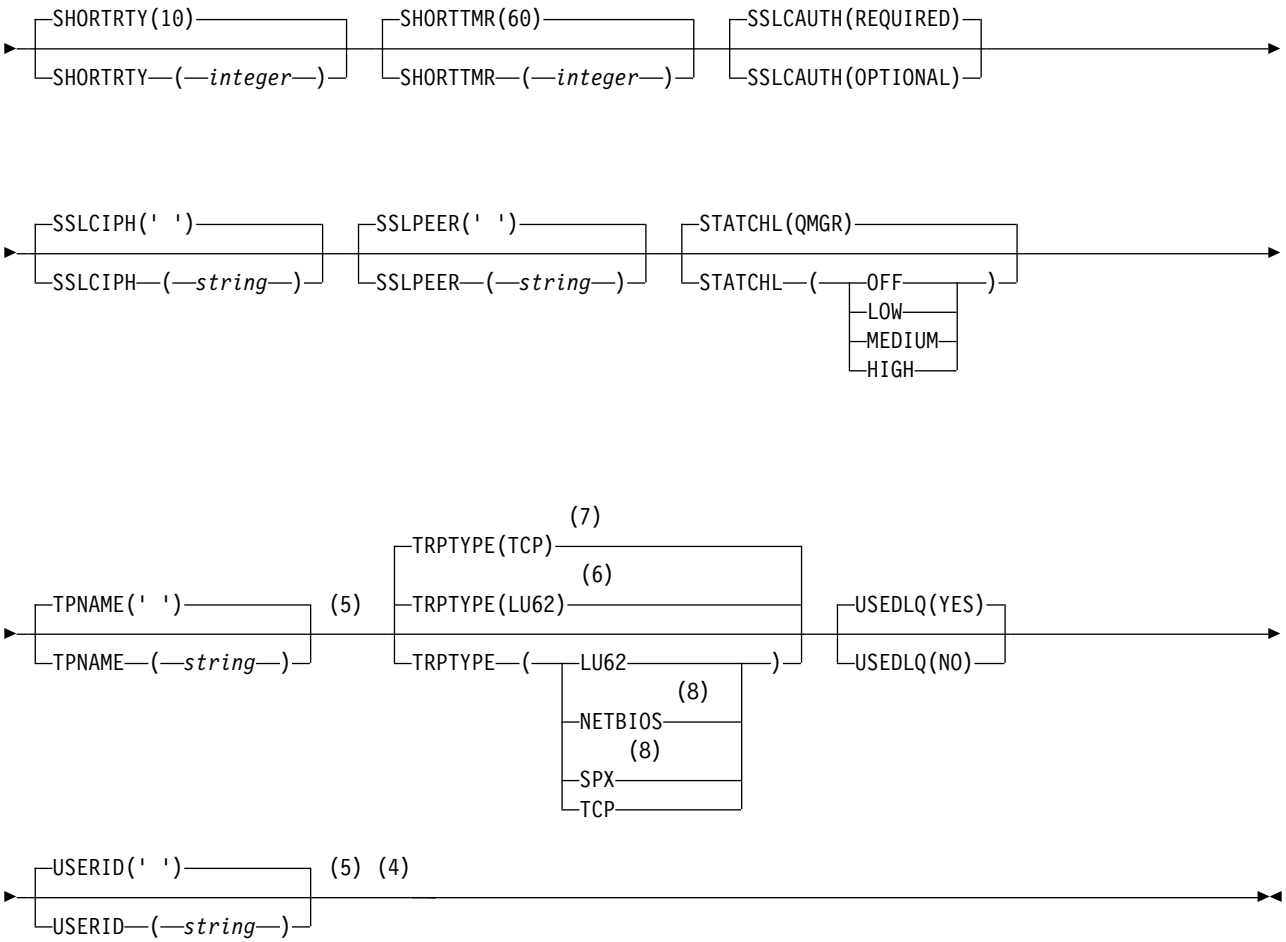
Syntax diagram for a server channel when using the DEFINE CHANNEL command.

Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

DEFINE CHANNEL




LONGTMR(1200)
LONGTMR—(—integer—)
MAXMSGL(4 194 304)
MAXMSGL—(—integer—)
MCANAME—(—' '—)
MCATYPE(PROCESS)
MCATYPE(THREAD)
(4)
MCAUSER(' ')
MCAUSER—(—string—)
MODENAME(' ')
MODENAME—(—string—)
(5)
MONCHL(QMGR)
MONCHL—(—OFF—)
LOW
MEDIUM
HIGH
MSGDATA(' ')
MSGDATA—(—string—)
MSGEXIT(' ')
MSGEXIT—(—string—)
NPMSPEED(FAST)
NPMSPEED(NORMAL)
PASSWORD(' ')
PASSWORD—(—string—)
(5) (4)
PROPCTL(COMPAT)
PROPCTL—(—ALL—)
NONE
QSGDISP(QMGR)
(3)
(2)
QSGDISP(COPY)
(2)
QSGDISP(GROUP)
RCVDATA(' ')
RCVDATA—(—string—)
RCVEXIT(' ')
RCVEXIT—(—string—)
NOREPLACE
REPLACE
SCYDATA(' ')
SCYDATA—(—string—)
SCYEXIT(' ')
SCYEXIT—(—string—)
SENDDATA(' ')
SENDDATA—(—string—)
SENDEXIT(' ')
SENDEXIT—(—string—)
SEQWRAP(999 999 999)
SEQWRAP—(—integer—)



Notes:

1 This parameter must follow immediately after the channel name z/OS except on z/OS.

2 Valid only on IBM MQ for z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Not valid on z/OS.

5 Valid only if TRPTYPE is LU62.

6 Default for z/OS.

7 Default for Multiplatforms.

8 Valid only on Windows.

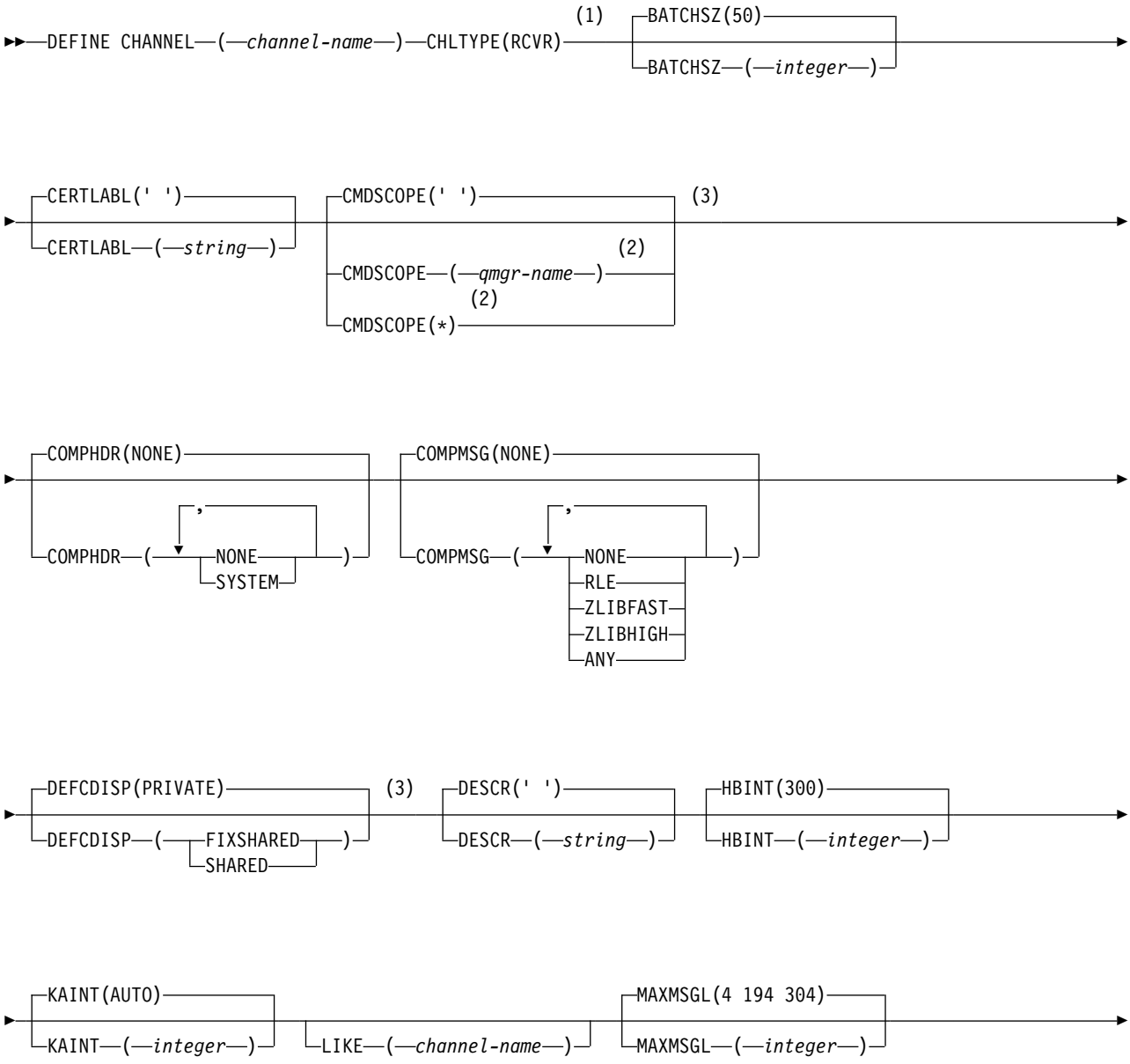
The parameters are described in “DEFINE CHANNEL” on page 575.

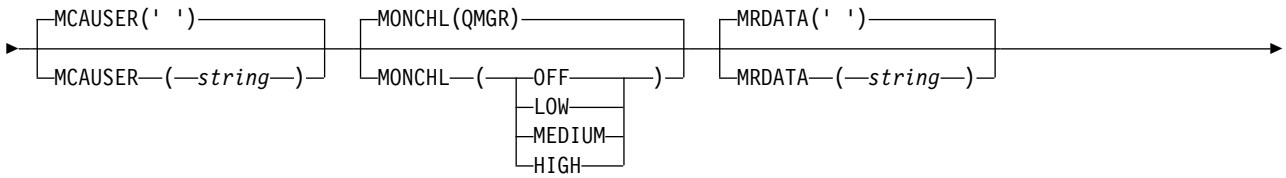
Receiver channel:

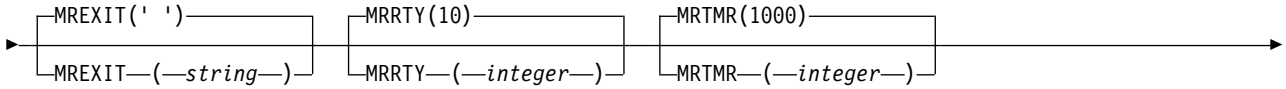
Syntax diagram for a receiver channel when using the DEFINE CHANNEL command.

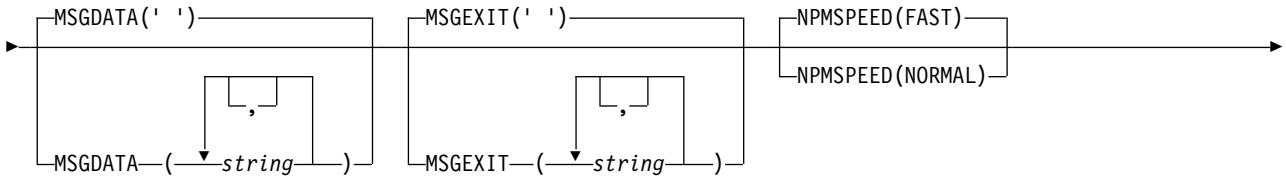
Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

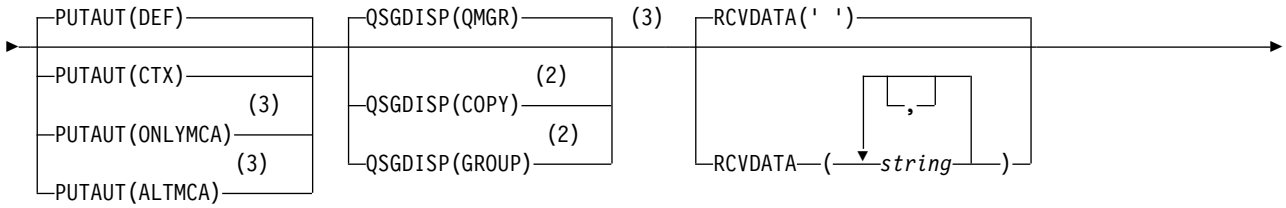
DEFINE CHANNEL

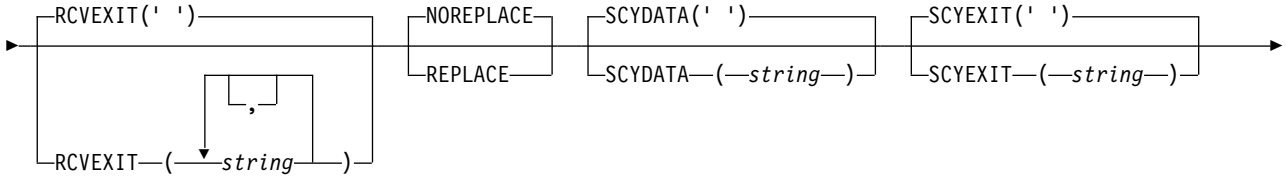


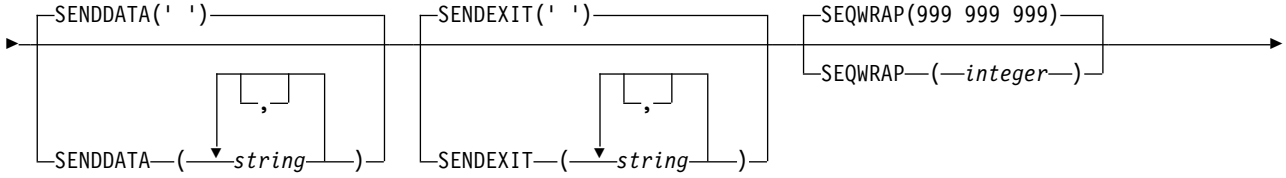

MCAUSER(' ')
MCAUSER(string)
MONCHL(QMGR)
MONCHL(OFF)
LOW
MEDIUM
HIGH
MRDATA(' ')
MRDATA(string)

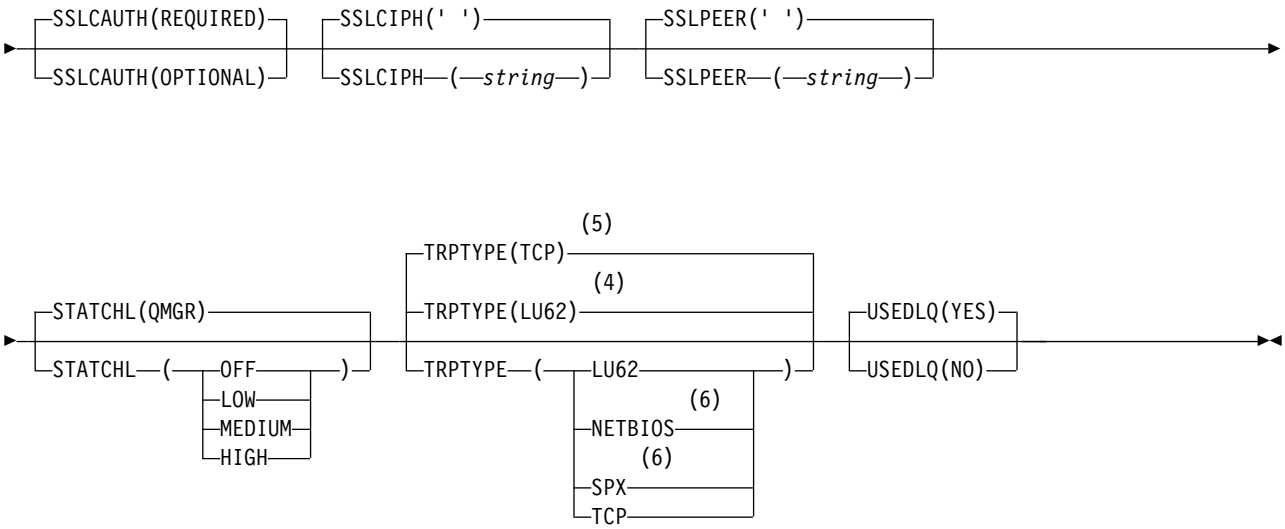

MREXIT(' ')
MREXIT(string)
MRRTY(10)
MRRTY(integer)
MRTMR(1000)
MRTMR(integer)


MSGDATA(' ')
MSGDATA(string)
MSGEXIT(' ')
MSGEXIT(string)
NPMSPEED(FAST)
NPMSPEED(NORMAL)


PUTAUT(DEF)
PUTAUT(CTX)
PUTAUT(ONLYMCA)
(3)
PUTAUT(ALTMCA)
(3)
QSGDISP(QMGR)
(3)
QSGDISP(COPY)
(2)
QSGDISP(GROUP)
(2)
RCVDATA(' ')
RCVDATA(string)


RCVEXIT(' ')
RCVEXIT(string)
NOREPLACE
REPLACE
SCYDATA(' ')
SCYDATA(string)
SCYEXIT(' ')
SCYEXIT(string)


SENDDATA(' ')
SENDDATA(string)
SENDEXIT(' ')
SENDEXIT(string)
SEQWRAP(999 999 999)
SEQWRAP(integer)



Notes:

1 This parameter must follow immediately after the channel name z/OS except on z/OS.

2 Valid only on IBM MQ for z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Default for z/OS.

5 Default for Multiplatforms.

6 Valid only on Windows.

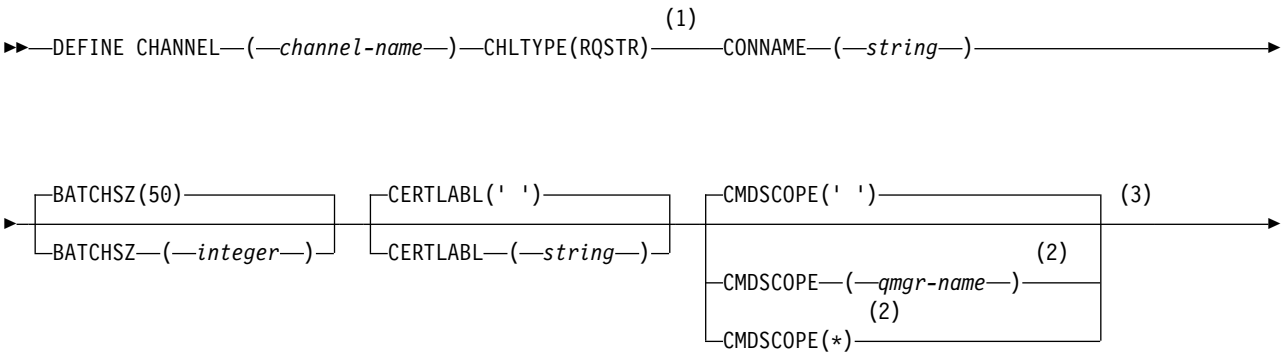
The parameters are described in "DEFINE CHANNEL" on page 575.

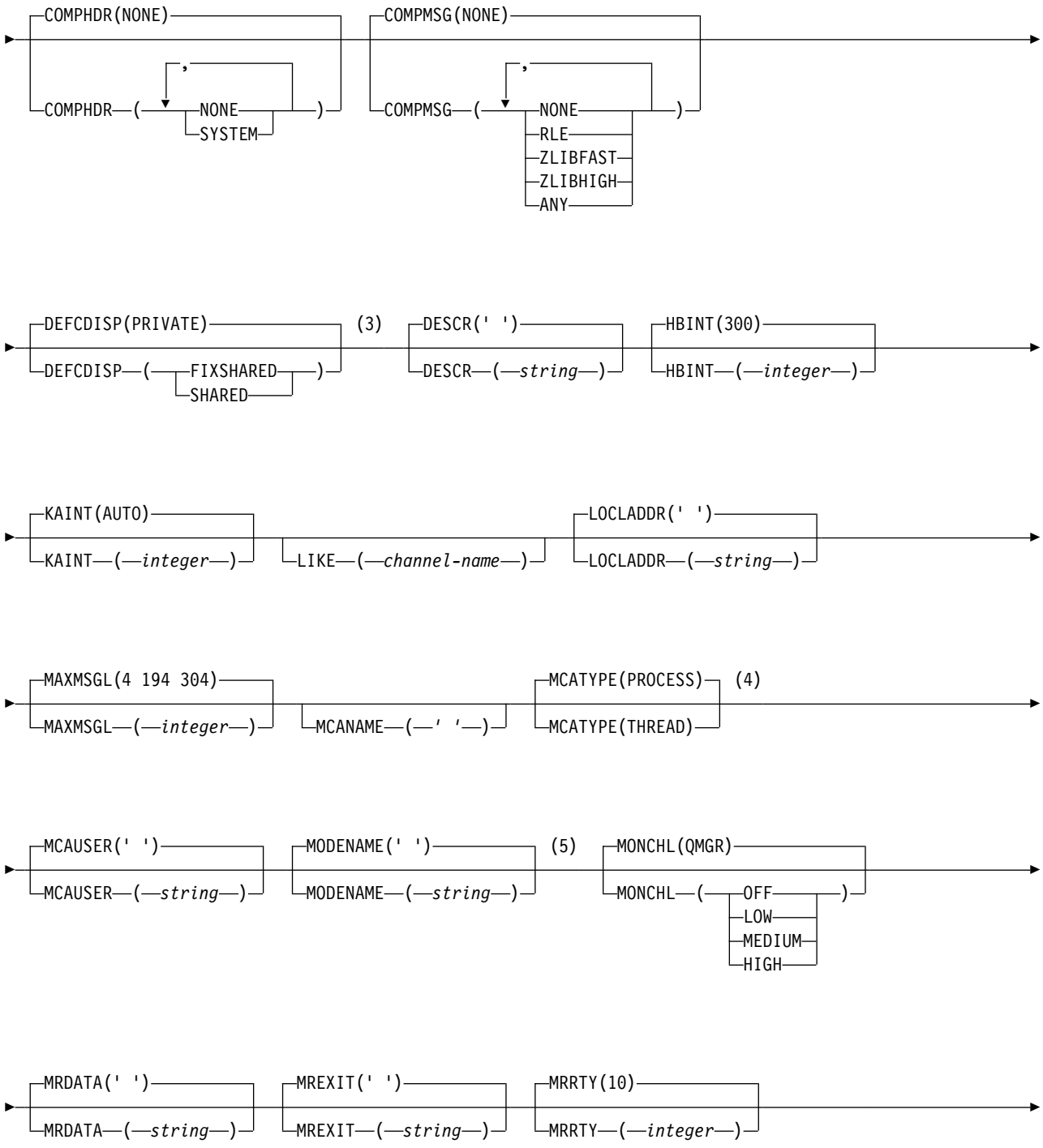
Requester channel:

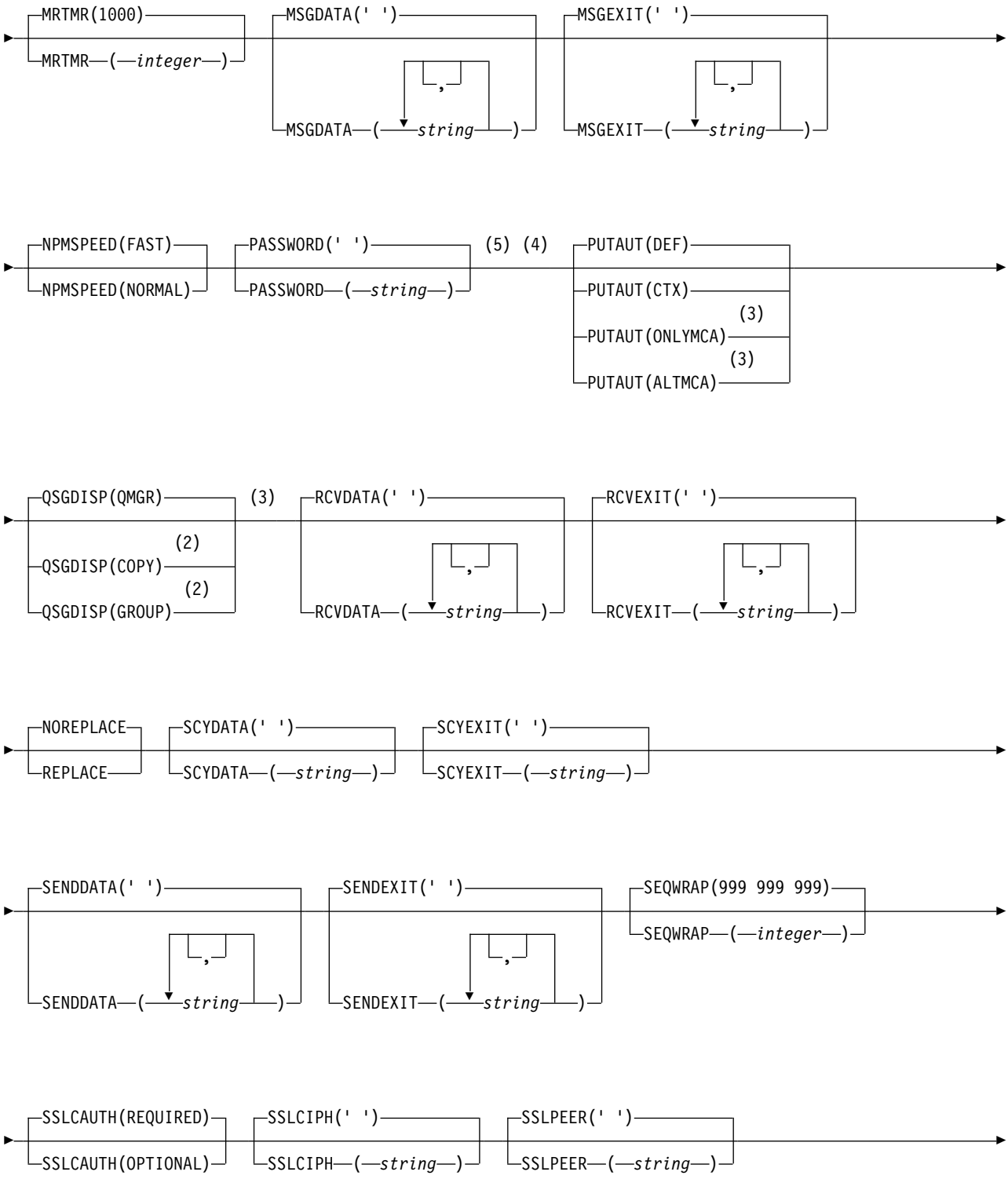
Syntax diagram for a requester channel when using the DEFINE CHANNEL command.

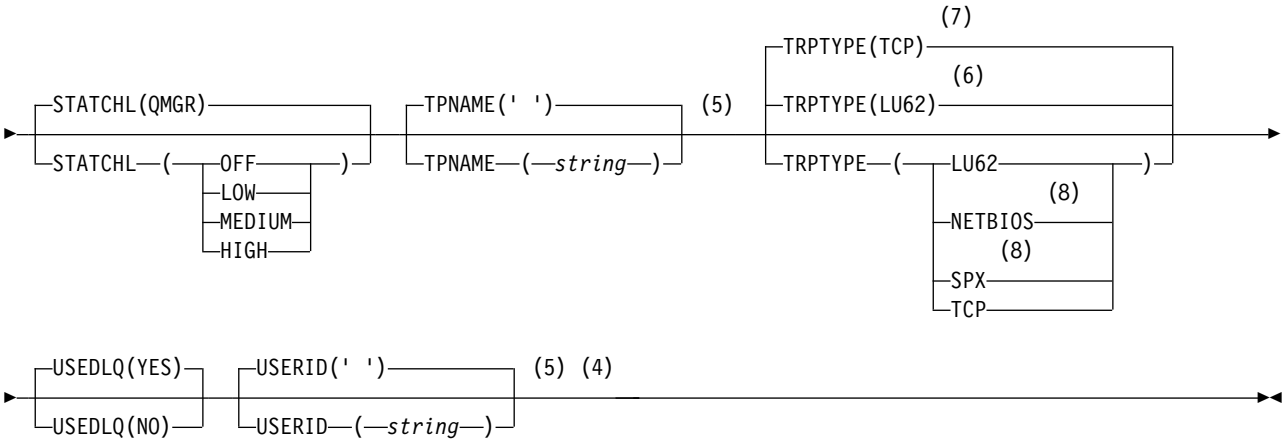
Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

DEFINE CHANNEL




COMPHDR(NONE)
COMPHDR
(
,
NONE
SYSTEM
)
COMPMSG(NONE)
COMPMSG
(
,
NONE
RLE
ZLIBFAST
ZLIBHIGH
ANY
)
DEFCDISP(PRIVATE)
DEFCDISP
(
FIXSHARED
SHARED
)
(3)
DESCR(' ')
DESCR
(string)
HBINT(300)
HBINT
(integer)
KAINT(AUTO)
KAINT
(integer)
LIKE
(channel-name)
LOCLADDR(' ')
LOCLADDR
(string)
MAXMSGL(4 194 304)
MAXMSGL
(integer)
MCANAME
(' ')
MCATYPE(PROCESS)
MCATYPE(THREAD)
(4)
MCAUSER(' ')
MCAUSER
(string)
MODENAME(' ')
MODENAME
(string)
(5)
MONCHL(QMGR)
MONCHL
(
OFF
LOW
MEDIUM
HIGH
)
MRDATA(' ')
MRDATA
(string)
MREXIT(' ')
MREXIT
(string)
MRRTY(10)
MRRTY
(integer)


MRTMR(1000)
MRTMR(integer)
MSGDATA(' ')
MSGDATA(string)
MSGEXIT(' ')
MSGEXIT(string)
NPMSPEED(FAST)
NPMSPEED(NORMAL)
PASSWORD(' ')
PASSWORD(string)
(5) (4)
PUTAUT(DEF)
PUTAUT(CTX)
PUTAUT(ONLYMCA)
(3)
PUTAUT(ALTMCA)
(3)
QSGDISP(QMGR)
(3)
QSGDISP(COPY)
(2)
QSGDISP(GROUP)
(2)
RCVDATA(' ')
RCVDATA(string)
RCVEXIT(' ')
RCVEXIT(string)
NOREPLACE
REPLACE
SCYDATA(' ')
SCYDATA(string)
SCYEXIT(' ')
SCYEXIT(string)
SENDDATA(' ')
SENDDATA(string)
SENDEXIT(' ')
SENDEXIT(string)
SEQWRAP(999 999 999)
SEQWRAP(integer)
SSLCAUTH(REQUIRED)
SSLCAUTH(OPTIONAL)
SSLCIPH(' ')
SSLCIPH(string)
SSLPEER(' ')
SSLPEER(string)



Notes:

1 This parameter must follow immediately after the channel name z/OS except on z/OS.

2 Valid only on IBM MQ for z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Not valid on z/OS.

5 Valid only if TRPTYPE is LU62.

6 Default for z/OS.

7 Default for Multiplatforms.

8 Valid only on Windows.

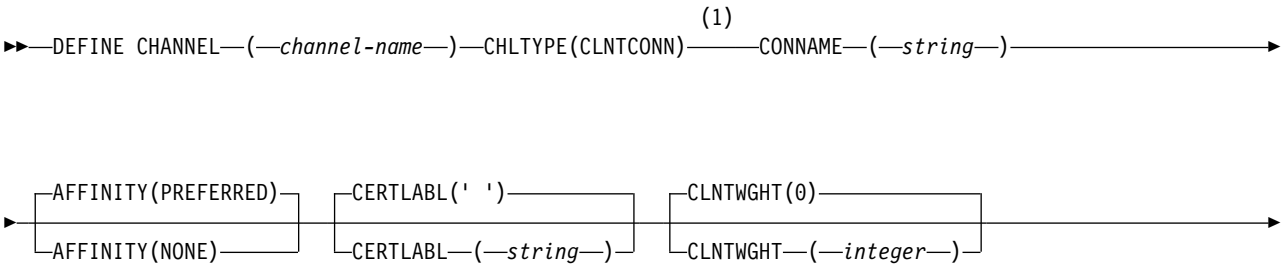
The parameters are described in "DEFINE CHANNEL" on page 575.

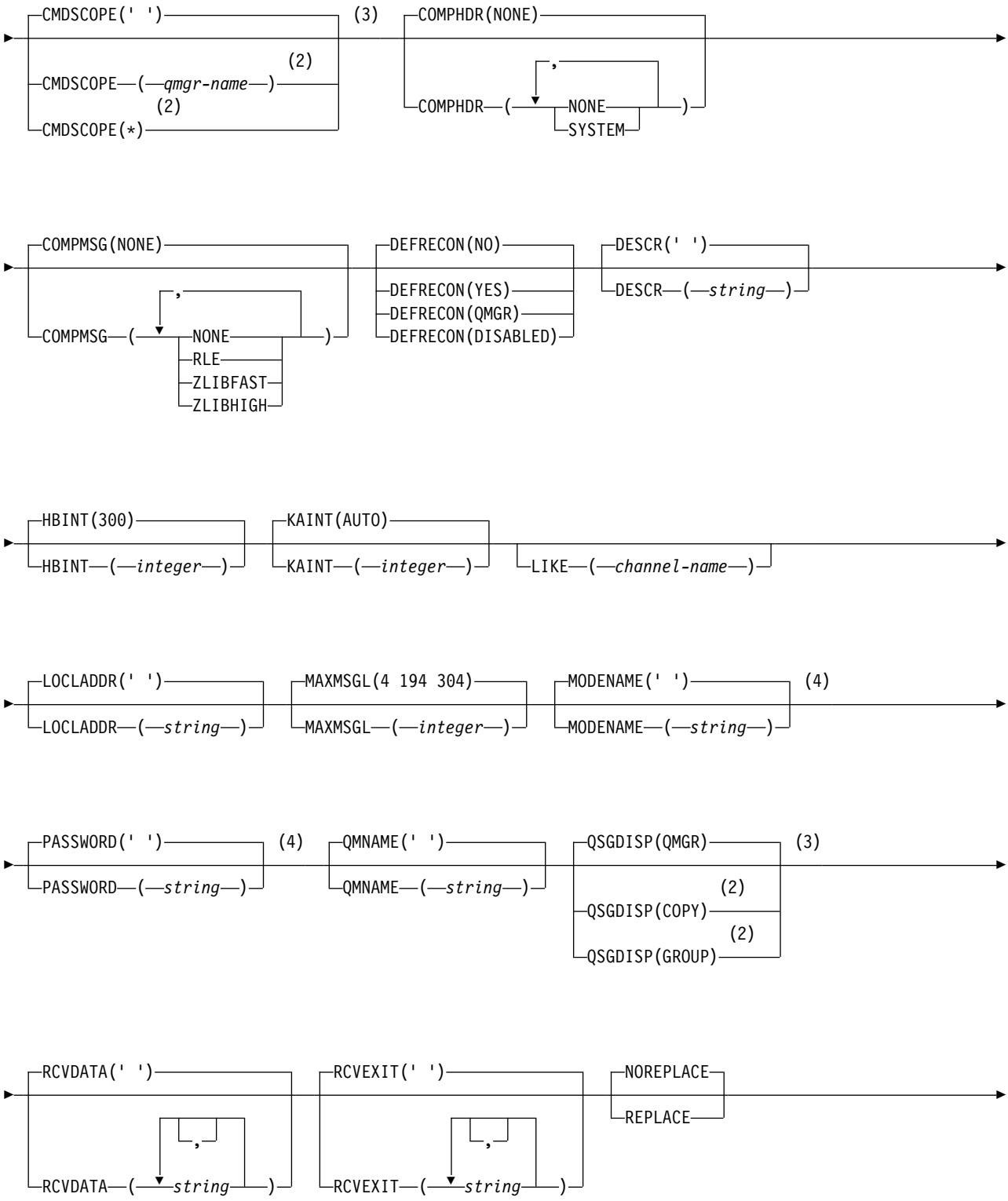
Client-connection channel:

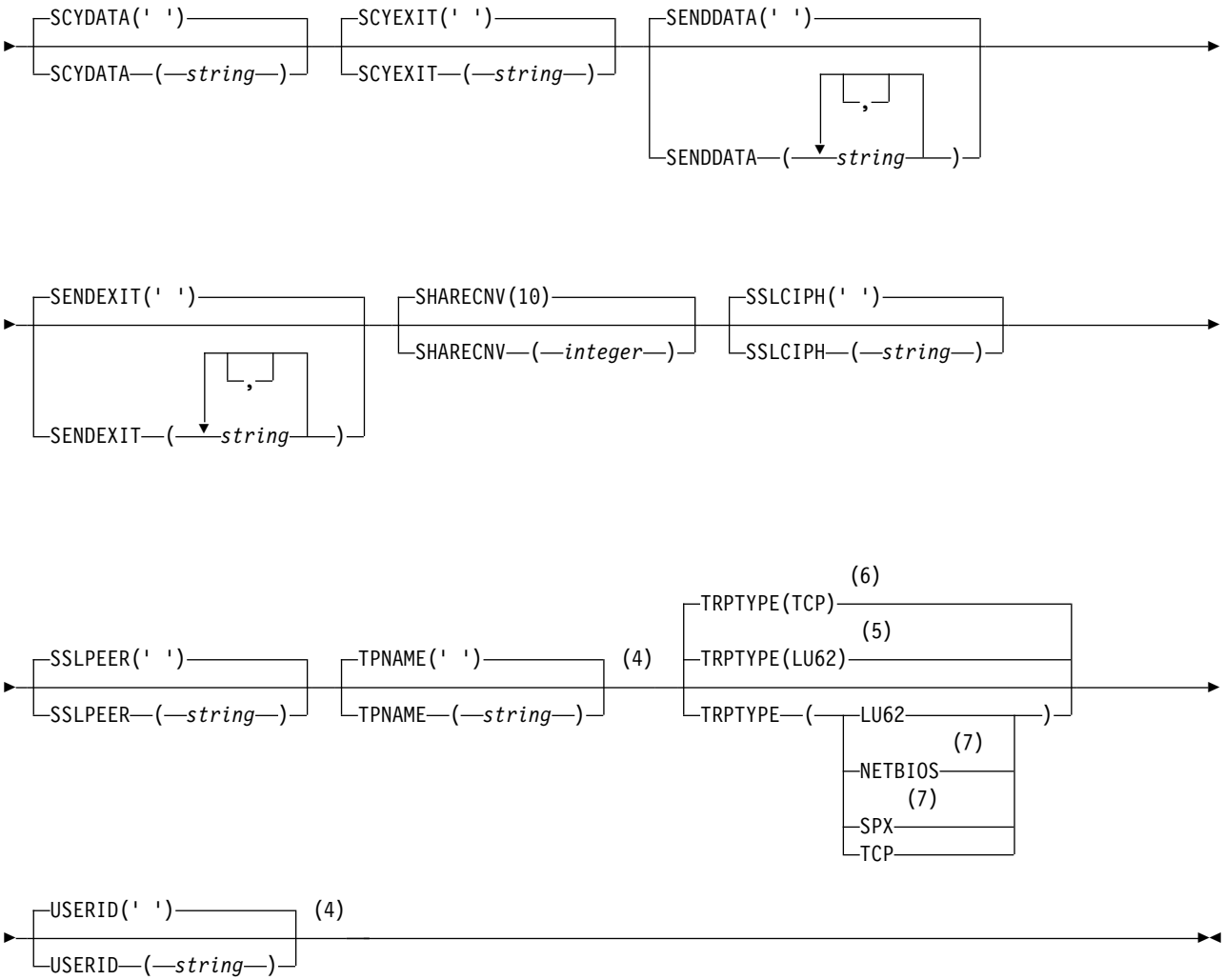
Syntax diagram for a client-connection channel when using the DEFINE CHANNEL command.

Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

DEFINE CHANNEL




CMDSCOPE(' ')
(3)
COMPHDR(NONE)
CMDSCOPE(*)
(2)
CMDSCOPE(qmgr-name)
(2)
COMPHDR(NONE)
SYSTEM
COMPMSG(NONE)
DEFRECON(NO)
DESCR(' ')
COMPMSG(NONE)
RLE
ZLIBFAST
ZLIBHIGH
DEFRECON(YES)
DEFRECON(QMGR)
DEFRECON(DISABLED)
DESCR(string)
HBINT(300)
KAINT(AUTO)
HBINT(integer)
KAINT(integer)
LIKE(channel-name)
LOCLADDR(' ')
MAXMSGL(4 194 304)
MODENAME(' ')
(4)
LOCLADDR(string)
MAXMSGL(integer)
MODENAME(string)
PASSWORD(' ')
(4)
QMNAME(' ')
QSGDISP(QMGR)
(3)
PASSWORD(string)
QMNAME(string)
(2)
QSGDISP(COPY)
(2)
QSGDISP(GROUP)
RCVDATA(' ')
RCVEXIT(' ')
NOREPLACE
REPLACE
RCVDATA(string)
RCVEXIT(string)



Notes:

1 This parameter must follow immediately after the channel name z/OS except on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Valid only if TRPTYPE is LU62.

5 Default for z/OS.

6 Default for Multiplatforms.

7 Valid only for clients to be run on DOS or Windows.

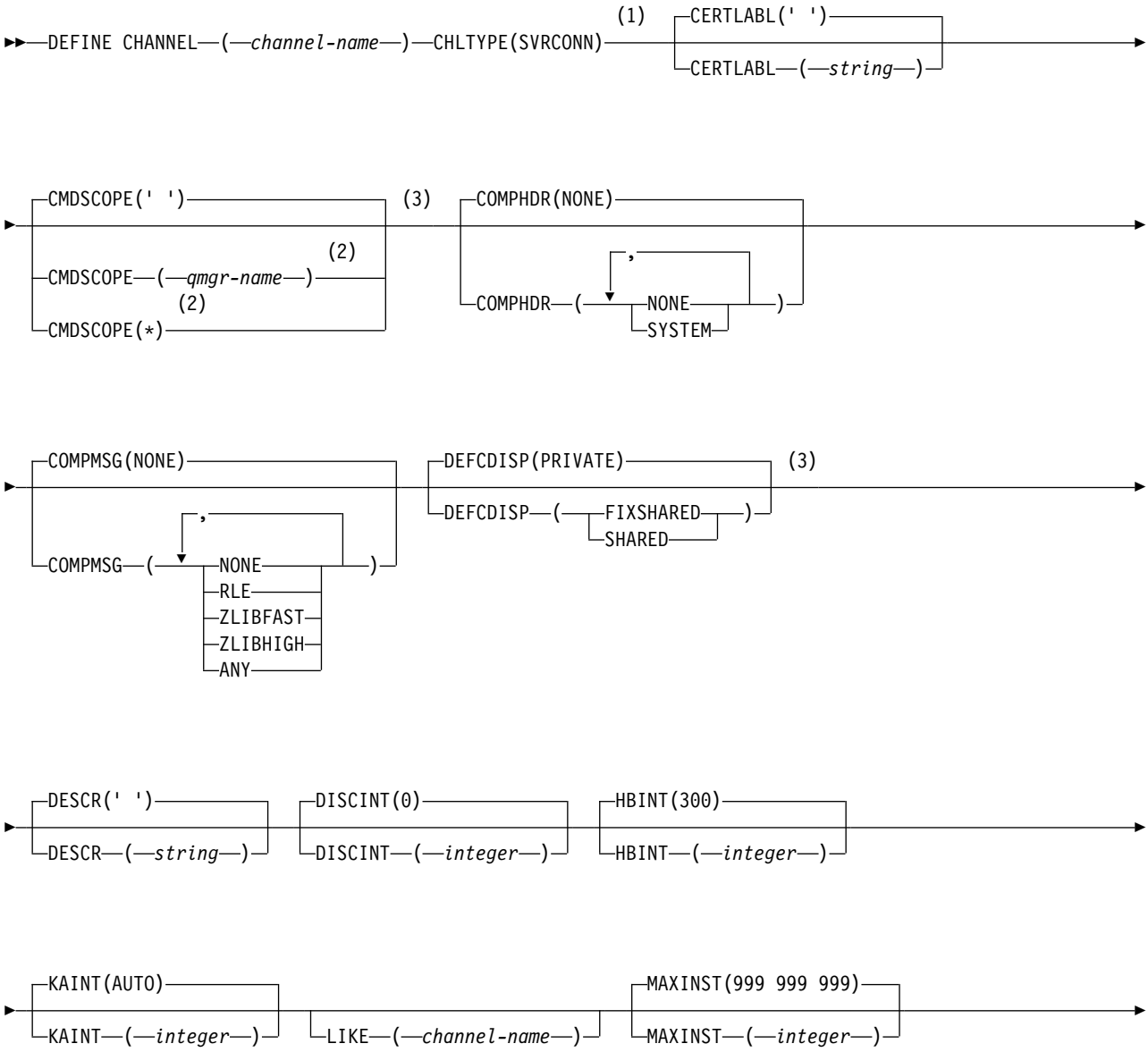
The parameters are described in “DEFINE CHANNEL” on page 575.

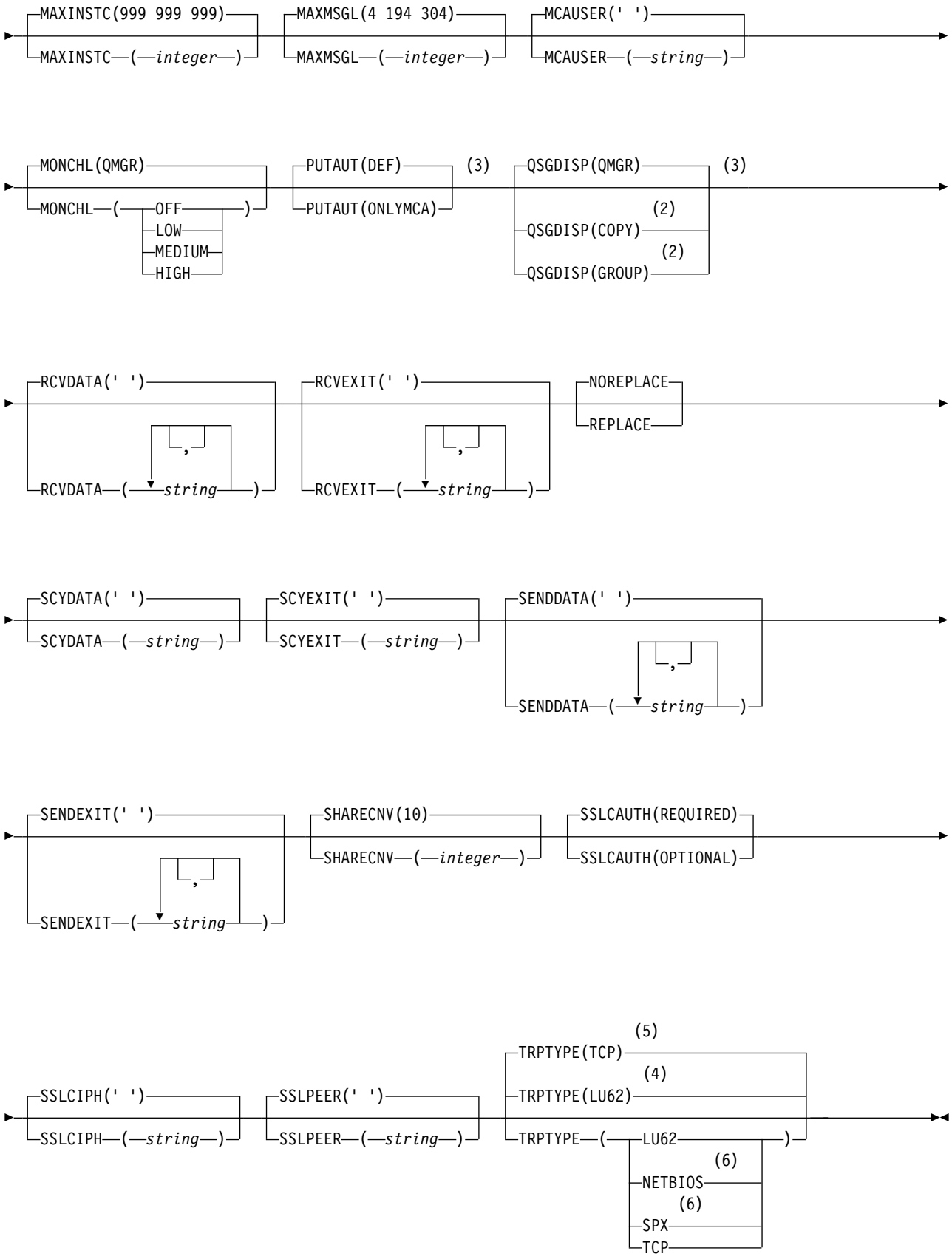
Server-connection channel:

Syntax diagram for a server-connection channel when using the DEFINE CHANNEL command.

Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

DEFINE CHANNEL




MAXINSTC(999 999 999)
MAXINSTC—(—integer—)
MAXMSGL(4 194 304)
MAXMSGL—(—integer—)
MCAUSER(' ')
MCAUSER—(—string—)
MONCHL(QMGR)
MONCHL—(—OFF—)
LOW
MEDIUM
HIGH
PUTAUT(DEF)
PUTAUT(ONLYMCA)
(3)
QSGDISP(QMGR)
(3)
(2)
QSGDISP(COPY)
(2)
QSGDISP(GROUP)
RCVDATA(' ')
RCVDATA—(—string—)
RCVEXIT(' ')
RCVEXIT—(—string—)
NOREPLACE
REPLACE
SCYDATA(' ')
SCYDATA—(—string—)
SCYEXIT(' ')
SCYEXIT—(—string—)
SENDDATA(' ')
SENDDATA—(—string—)
SENDEXIT(' ')
SENDEXIT—(—string—)
SHARECNV(10)
SHARECNV—(—integer—)
SSLCAUTH(REQUIRED)
SSLCAUTH(OPTIONAL)
SSLCIPH(' ')
SSLCIPH—(—string—)
SSLPEER(' ')
SSLPEER—(—string—)
(5)
TRPTYPE(TCP)
(4)
TRPTYPE(LU62)
TRPTYPE—(—LU62—)
(6)
NETBIOS
(6)
SPX
TCP

Notes:

1 This parameter must follow immediately after the channel name z/OS except on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Default for z/OS.

5 Default for Multiplatforms.

6 Valid only for clients to be run on Windows.

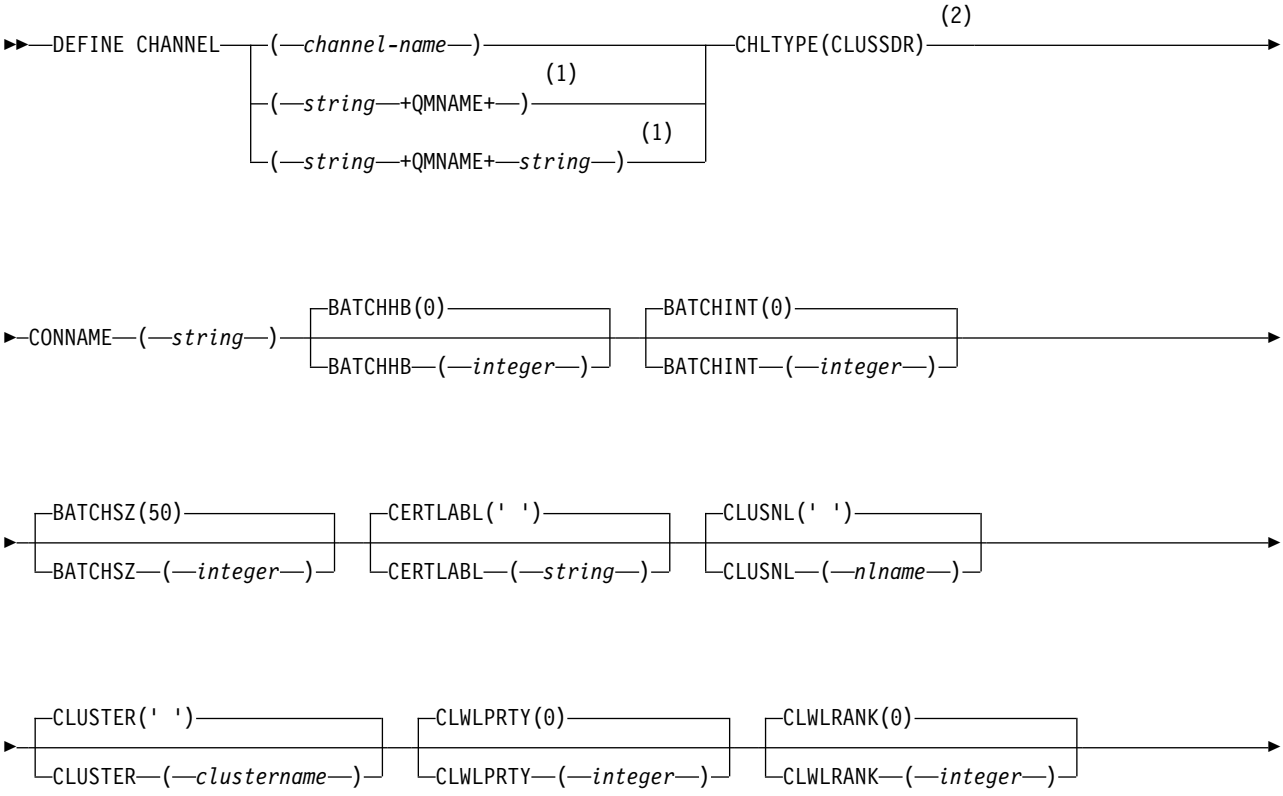
The parameters are described in "DEFINE CHANNEL" on page 575.

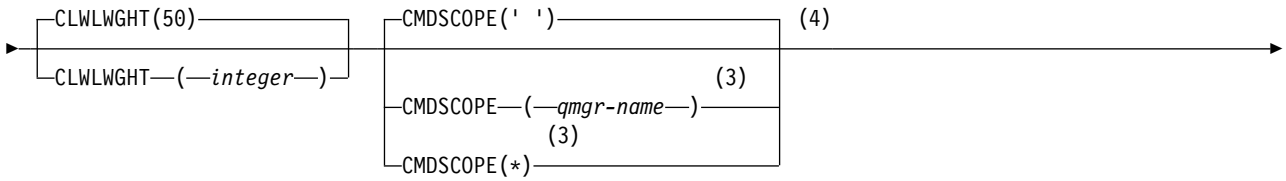
Cluster-sender channel:

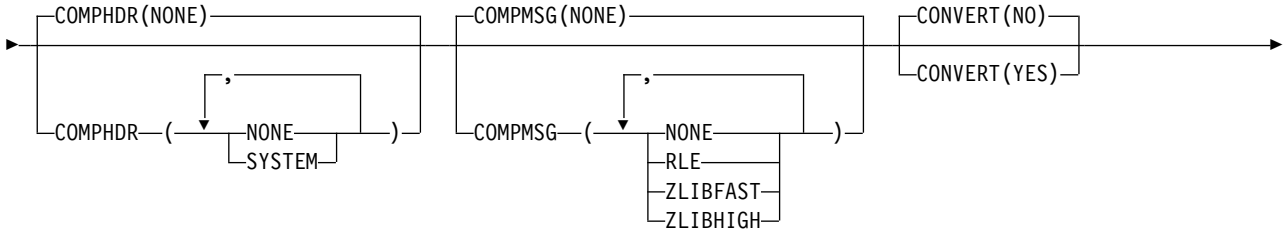
Syntax diagram for a cluster-sender channel when using the DEFINE CHANNEL command.

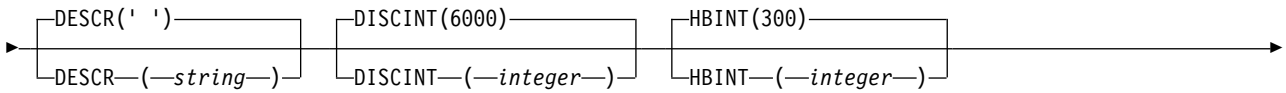
Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

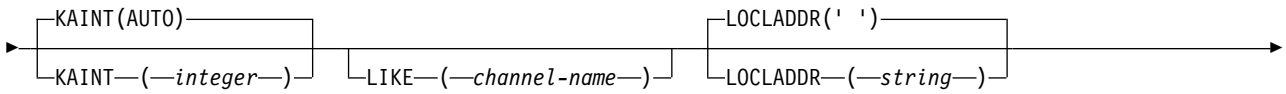
DEFINE CHANNEL

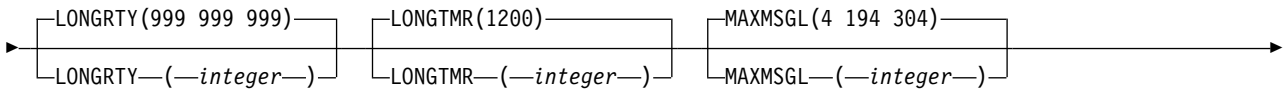


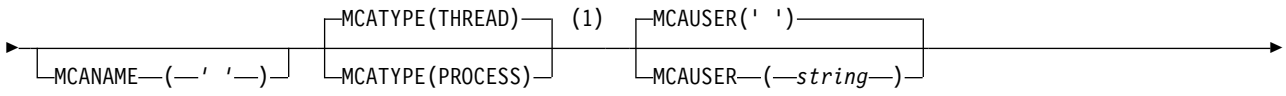

CLWLWGHT(50)
CLWLWGHT—(—integer—)
CMDSCOPE(' ')
CMDSCOPE—(—qmgr-name—)
CMDSCOPE(*)
(3)
(4)

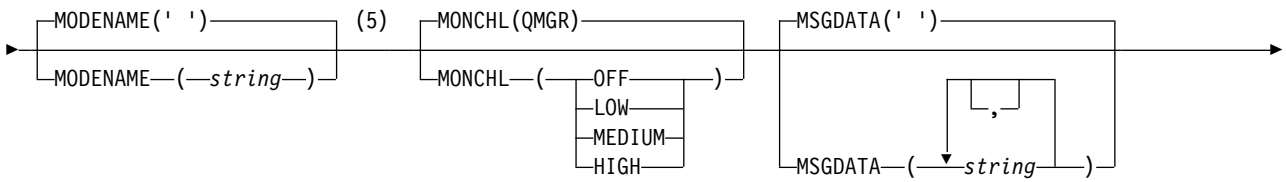

COMPHDR(NONE)
COMPHDR—(—NONE—)
SYSTEM
COMPMSG(NONE)
COMPMSG—(—NONE—)
RLE
ZLIBFAST
ZLIBHIGH
CONVERT(NO)
CONVERT(YES)

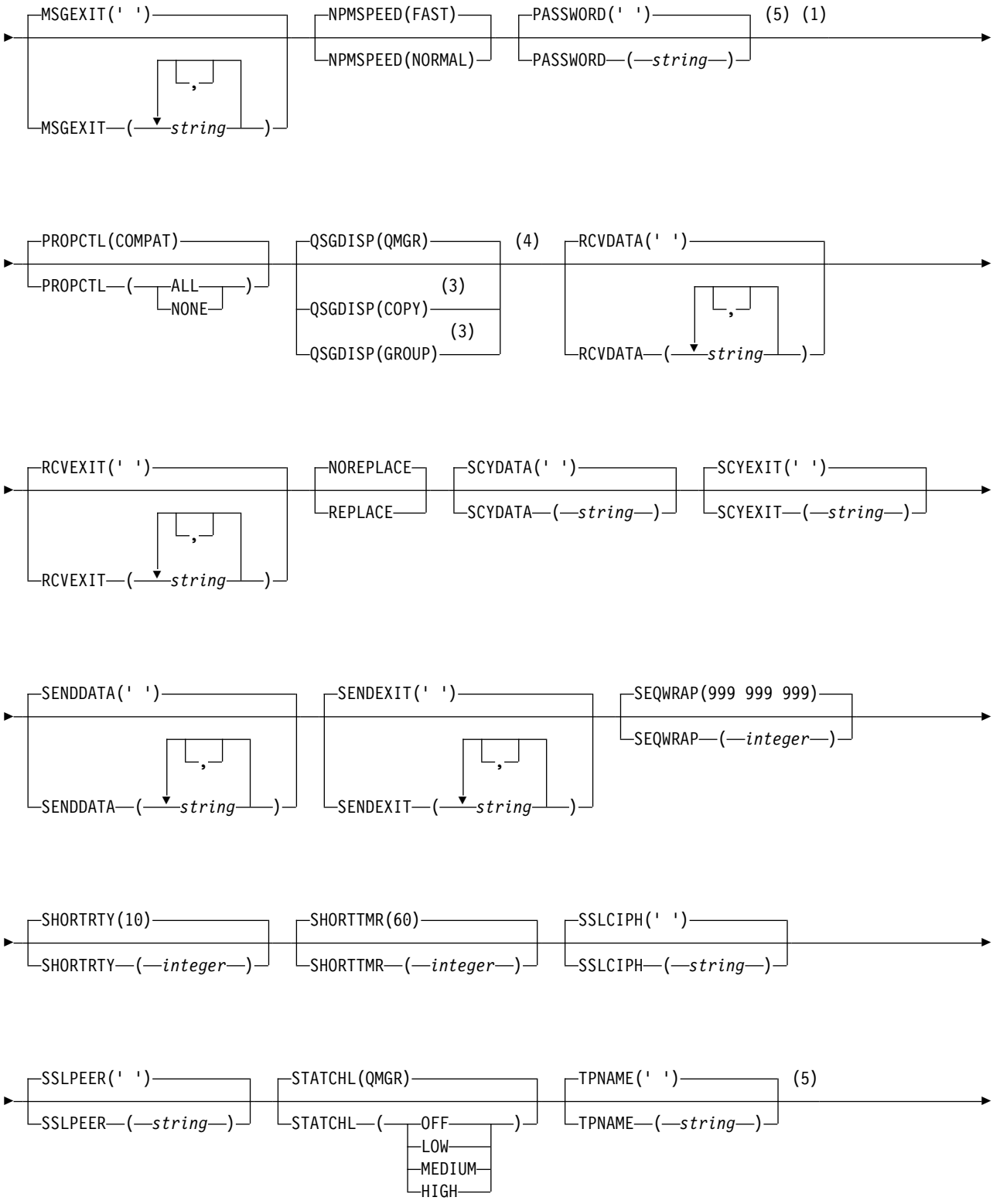

DESCR(' ')
DESCR—(—string—)
DISCINT(6000)
DISCINT—(—integer—)
HBINT(300)
HBINT—(—integer—)

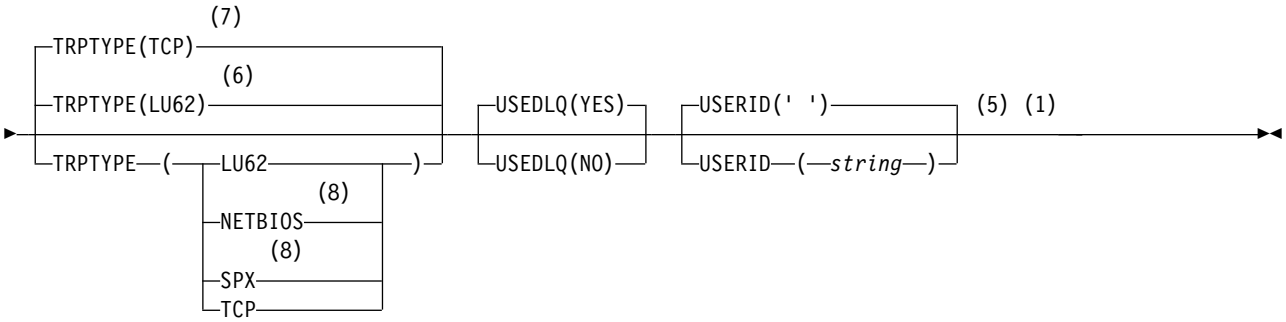

KAINT(AUTO)
KAINT—(—integer—)
LIKE—(—channel-name—)
LOCLADDR(' ')
LOCLADDR—(—string—)


LONGRTY(999 999 999)
LONGRTY—(—integer—)
LONGTMR(1200)
LONGTMR—(—integer—)
MAXMSGL(4 194 304)
MAXMSGL—(—integer—)


MCANAME—(—' '—)
MCATYPE(THREAD)
MCATYPE(PROCESS)
(1)
MCAUSER(' ')
MCAUSER—(—string—)


MODENAME(' ')
MODENAME—(—string—)
(5)
MONCHL(QMGR)
MONCHL—(—OFF—)
LOW
MEDIUM
HIGH
MSGDATA(' ')
MSGDATA—(—string—)


MSGEXIT(' ')
MSGEXIT
(
,
string
)
NPMSPEED(FAST)
NPMSPEED(NORMAL)
PASSWORD(' ')
PASSWORD
(
string
)
(5) (1)
PROPCTL(COMPAT)
PROPCTL
(
ALL
NONE
)
QSGDISP(QMGR)
QSGDISP(COPY)
QSGDISP(GROUP)
(3)
(3)
(4)
RCVDATA(' ')
RCVDATA
(
,
string
)
RCVEXIT(' ')
RCVEXIT
(
,
string
)
NOREPLACE
REPLACE
SCYDATA(' ')
SCYDATA
(
string
)
SCYEXIT(' ')
SCYEXIT
(
string
)
SENDDATA(' ')
SENDDATA
(
,
string
)
SENDEXIT(' ')
SENDEXIT
(
,
string
)
SEQWRAP(999 999 999)
SEQWRAP
(
integer
)
SHORTRTY(10)
SHORTRTY
(
integer
)
SHORTTMR(60)
SHORTTMR
(
integer
)
SSLCIPH(' ')
SSLCIPH
(
string
)
SSLPEER(' ')
SSLPEER
(
string
)
STATCHL(QMGR)
STATCHL
(
OFF
LOW
MEDIUM
HIGH
)
TPNAME(' ')
TPNAME
(
string
)
(5)



Notes:

1 Not valid on z/OS.

2 This parameter must follow immediately after the channel name z/OS except on z/OS.

3 Valid only on IBM MQ for z/OS when the queue manager is a member of a queue-sharing group.

4 Valid only on z/OS.

5 Valid only if TRPTYPE is LU62.

6 Default for z/OS.

7 Default for Multiplatforms.

8 Valid only on Windows.

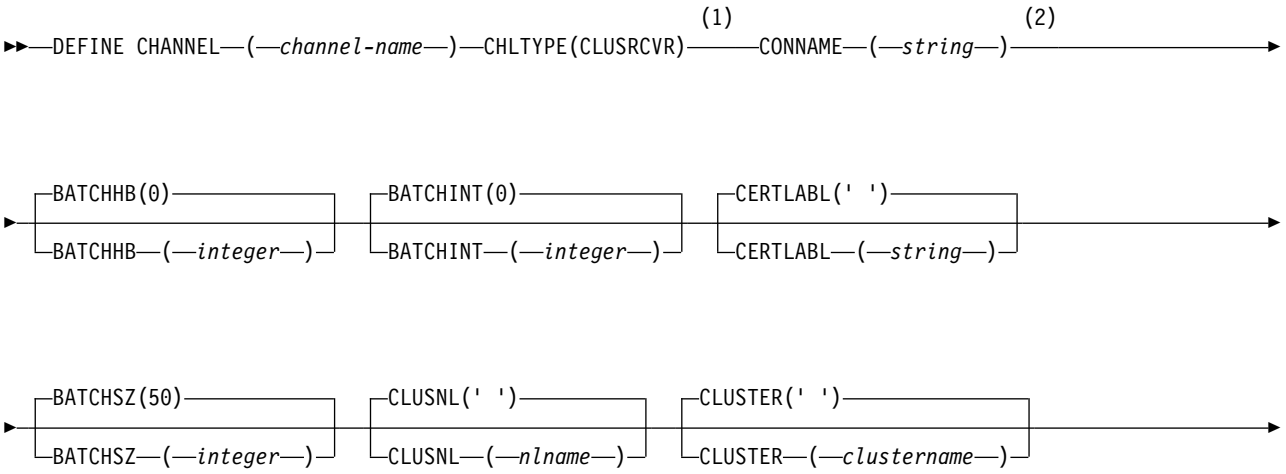
The parameters are described in "DEFINE CHANNEL" on page 575.

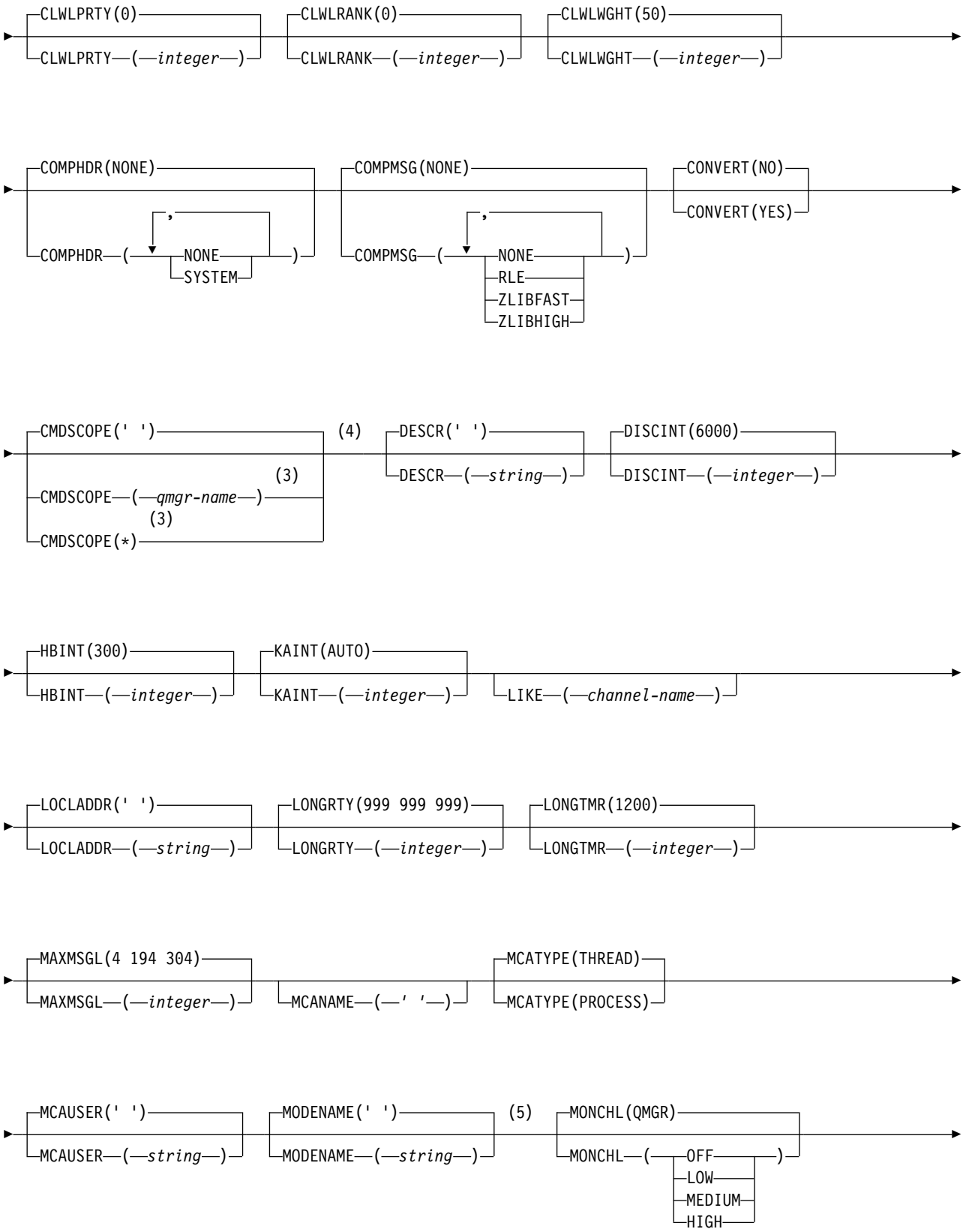
Cluster-receiver channel:

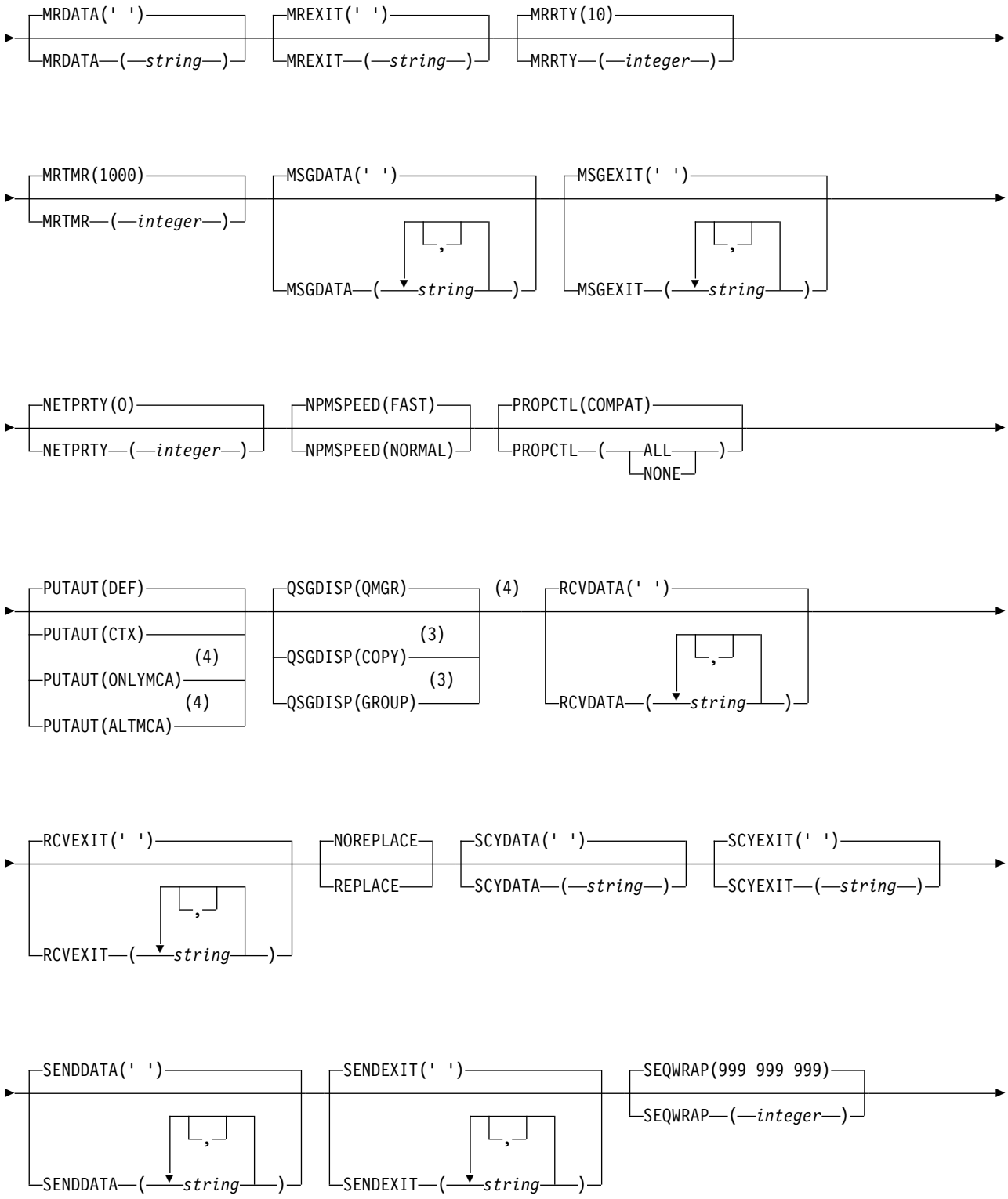
Syntax diagram for a cluster-receiver channel when using the DEFINE CHANNEL command.

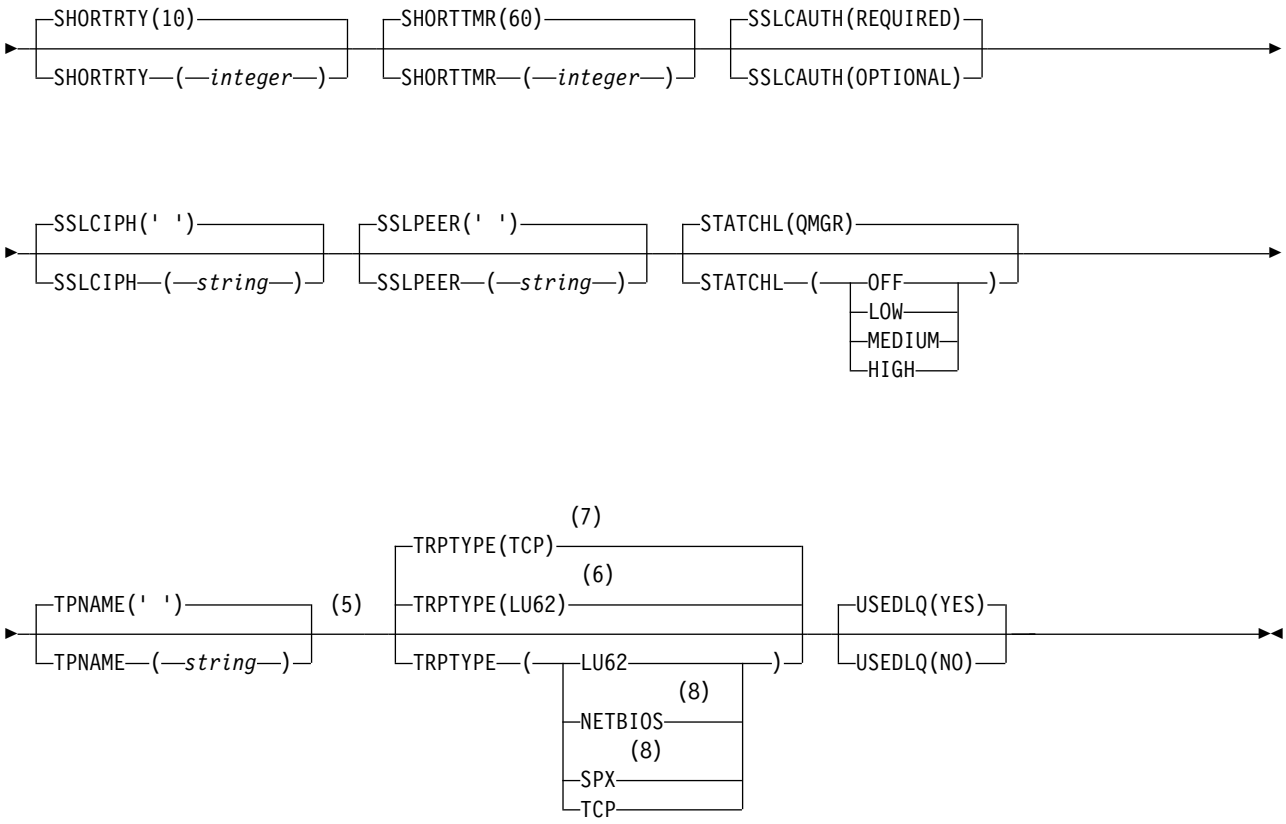
Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

DEFINE CHANNEL




CLWLPRTY(0)
CLWLPRTY—(—integer—)
CLWLRANK(0)
CLWLRANK—(—integer—)
CLWLWGHT(50)
CLWLWGHT—(—integer—)
COMPHDR(NONE)
COMPHDR—(—,—NONE—)
SYSTEM
COMPMSG(NONE)
COMPMSG—(—,—NONE—)
RLE
ZLIBFAST
ZLIBHIGH
CONVERT(NO)
CONVERT(YES)
CMDSCOPE(' ')
CMDSCOPE—(—qmgr-name—)
(3)
CMDSCOPE(*)
(4)
DESCR(' ')
DESCR—(—string—)
DISCINT(6000)
DISCINT—(—integer—)
HBINT(300)
HBINT—(—integer—)
KAINT(AUTO)
KAINT—(—integer—)
LIKE—(—channel-name—)
LOCLADDR(' ')
LOCLADDR—(—string—)
LONGRTY(999 999 999)
LONGRTY—(—integer—)
LONGTMR(1200)
LONGTMR—(—integer—)
MAXMSGL(4 194 304)
MAXMSGL—(—integer—)
MCANAME—(—' '—)
MCATYPE(THREAD)
MCATYPE(PROCESS)
MCAUSER(' ')
MCAUSER—(—string—)
MODENAME(' ')
MODENAME—(—string—)
(5)
MONCHL(QMGR)
MONCHL—(—OFF—)
LOW
MEDIUM
HIGH


MRDATA(' ')
MRDATA—(—string—)
MREXIT(' ')
MREXIT—(—string—)
MRRTY(10)
MRRTY—(—integer—)
MRTMR(1000)
MRTMR—(—integer—)
MSGDATA(' ')
,
MSGDATA—(—string—)
MSGEXIT(' ')
,
MSGEXIT—(—string—)
NETPRTY(0)
NETPRTY—(—integer—)
NPMSPEED(FAST)
NPMSPEED(NORMAL)
PROPCTL(COMPAT)
PROPCTL—(—ALL—)
NONE
PUTAUT(DEF)
PUTAUT(CTX)
PUTAUT(ONLYMCA)
(4)
PUTAUT(ALTMCA)
(4)
QSGDISP(QMGR)
(4)
QSGDISP(COPY)
(3)
QSGDISP(GROUP)
(3)
RCVDATA(' ')
,
RCVDATA—(—string—)
RCVEXIT(' ')
,
RCVEXIT—(—string—)
NOREPLACE
REPLACE
SCYDATA(' ')
SCYDATA—(—string—)
SCYEXIT(' ')
SCYEXIT—(—string—)
SENDDATA(' ')
,
SENDDATA—(—string—)
SENDEXIT(' ')
,
SENDEXIT—(—string—)
SEQWRAP(999 999 999)
SEQWRAP—(—integer—)



Notes:

1 This parameter must follow immediately after the channel name z/OS except on z/OS.

2 This parameter is optional if TRPTYPE is TCP.

3 Valid only on IBM MQ for z/OS when the queue manager is a member of a queue-sharing group.

4 Valid only on z/OS.

5 Valid only if TRPTYPE is LU62.

6 Default for z/OS.

7 Default for Multiplatforms.

8 Valid only on Windows.

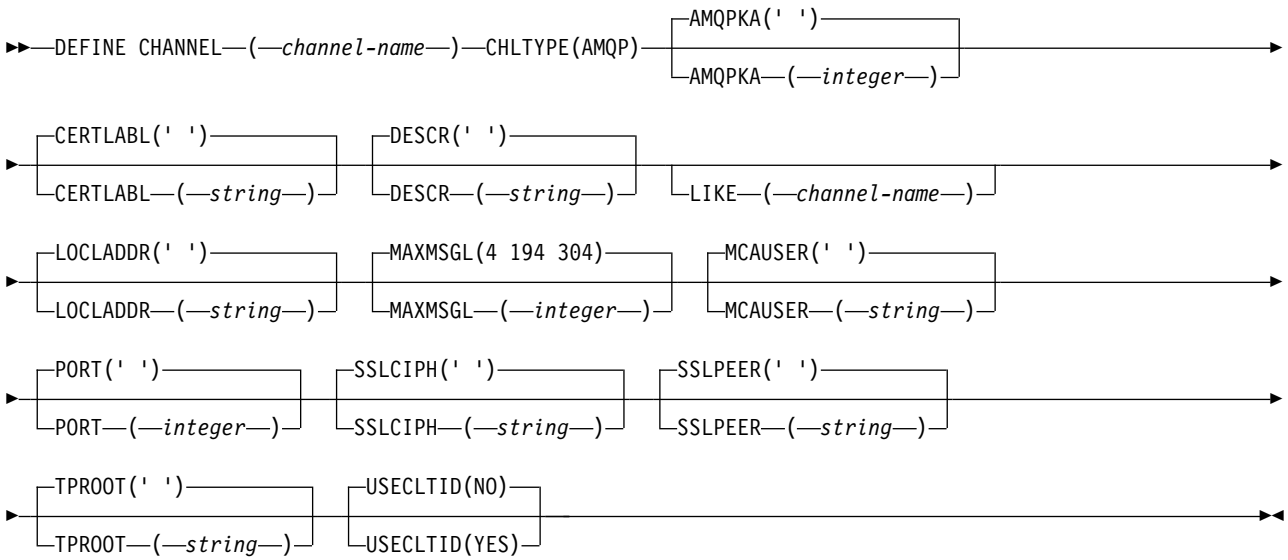
The parameters are described in “DEFINE CHANNEL” on page 575.

AMQP channel: ULW V 9.0.0

Syntax diagram for an AMQP channel when using the DEFINE CHANNEL command.

Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See “How to read railroad diagrams” on page 187.

DEFINE CHANNEL



The parameters are described in “DEFINE CHANNEL” on page 575.

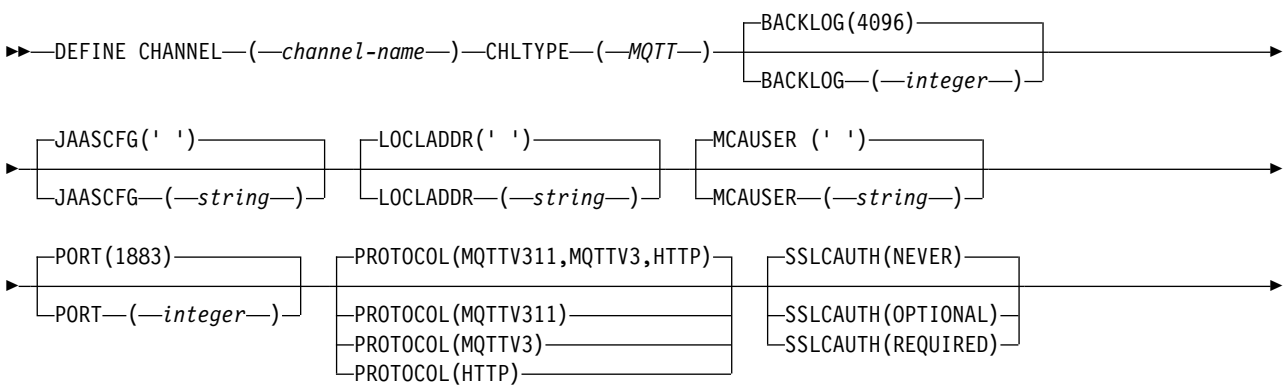
DEFINE CHANNEL (MQTT): AIX Linux Windows

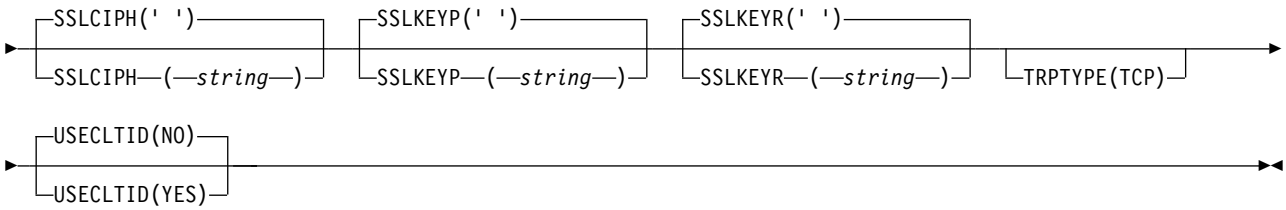
Syntax diagram for a telemetry channel when using the **DEFINE CHANNEL** command.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

DEFINE CHANNEL (MQTT)





Usage notes

The telemetry (MQXR) service must be running when you issue this command. For instructions on how to start the telemetry (MQXR) service, see Configuring a queue manager for telemetry on Linux or Configuring a queue manager for telemetry on Windows.

Parameter descriptions for DEFINE CHANNEL (MQTT)

(channel-name)

The name of the new channel definition.

The name must not be the same as any existing channel defined on this queue manager (unless REPLACE or ALTER is specified).

The maximum length of the string is 20 characters, and the string must contain only valid characters; see Rules for naming IBM MQ objects.

BACKLOG(*integer*)

The number of outstanding connection requests that the telemetry channel can support at any one time. When the backlog limit is reached, any further clients trying to connect will be refused connection until the current backlog is processed.

The value is in the range 0 - 999999999.

The default value is 4096.

CHLTYPE

Channel type. MQTT (telemetry) channel.

JAASCFG (*string*)

The name of a stanza in the JAAS configuration file.

See Authenticating an MQTT client Java app with JAAS

LOCLADDR (*ip-addr*)

LOCLADDR is the local communications address for the channel. Use this parameter if you want to force the client to use a particular IP address. LOCLADDR is also useful to force a channel to use an IPv4 or IPv6 address if a choice is available, or to use a particular network adapter on a system with multiple network adapters.

The maximum length of **LOCLADDR** is MQ_LOCAL_ADDRESS_LENGTH.

If you omit **LOCLADDR**, a local address is automatically allocated.

ip-addr
ip-addr is a single network address, specified in one of three forms:

IPv4 dotted decimal
For example 192.0.2.1

IPv6 hexadecimal notation
For example 2001:DB8:0:0:0:0:0:0

Alphanumeric host name form
For example `WWW.EXAMPLE.COM`

If an IP address is entered, only the address format is validated. The IP address itself is not validated.

MCAUSER(*string*)

Message channel agent user identifier.

The maximum length of the string is 64 characters on Windows and 12 characters on other platforms. On Windows, you can optionally qualify a user identifier with the domain name in the format `user@domain`.

If this parameter is nonblank, and **USECLNTID** is set to `NO`, then this user identifier is used by the telemetry service for authorization to access IBM MQ resources.

If this parameter is blank, and **USECLNTID** is set to `NO`, then the user name flowed in the MQTT CONNECT Packet is used. See MQTT client identity and authorization.

PORT(*integer*)

The port number on which the telemetry (MQXR) service accepts client connections. The default port number for a telemetry channel is 1883; and the default port number for a telemetry channel secured using SSL is 8883. Specifying a port value of 0 causes MQTT to dynamically allocate an available port number.

PROTOCOL

The following communication protocols are supported by the channel:

MQTTV311
The channel accepts connections from clients using the protocol defined by the MQTT Version 3.1.1 Oasis standard. The functionality provided by this protocol is almost identical to that provided by the pre-existing MQTTV3 protocol.

MQTTV3 The channel accepts connections from clients using the MQTT V3.1 Protocol Specification from mqtt.org.

HTTP The channel accepts HTTP requests for pages, or WebSockets connections to MQ Telemetry.

To accept connections from clients using different protocols, specify the acceptable values as a comma-delimited list. For example if you specify `MQTTV3,HTTP` the channel accepts connections from clients using either MQTTV3 or HTTP. If you specify no client protocols, the channel accepts connections from clients using any of the supported protocols.

If you are using IBM MQ Version 8.0.0, Fix Pack 3 or later, and your configuration includes an MQTT channel that was last modified in an earlier version of the product, you must explicitly change the protocol setting to prompt the channel to use the `MQTTV311` option. This is so even if the channel does not specify any client protocols, because the specific protocols to use with the channel are stored at the time the channel is configured, and previous versions of the product have no awareness of the `MQTTV311` option. To prompt a channel in this state to use the `MQTTV311` option, explicitly add the option then save your changes. The channel definition is now aware of the option. If you subsequently change the settings again, and specify no client protocols, the `MQTTV311` option is still included in the stored list of supported protocols.

SSLCAUTH

Defines whether IBM MQ requires a certificate from the TLS client. The initiating end of the channel acts as the TLS client, so this parameter applies to the end of the channel that receives the initiation flow, which acts as the TLS server.

NEVER IBM MQ never requests a certificate from the TLS client.

REQUIRED
IBM MQ requires and validates a certificate from the TLS client.

OPTIONAL
IBM MQ lets the TLS client decide whether to provide a certificate. If the client sends a certificate, the contents of this certificate are validated as normal.

SSLCIPH(*string*)

When **SSLCIPH** is used with a telemetry channel, it means TLS Cipher Suite. The TLS cipher suite is the one supported by the JVM that is running the telemetry (MQXR) service. If the **SSLCIPH** parameter is blank, no attempt is made to use TLS on the channel.

If you plan to use SHA-2 cipher suites, see System requirements for using SHA-2 cipher suites with MQTT channels.

SSLKEYP(*string*)

The passphrase for the TLS key repository.

SSLKEYR(*string*)

The full path name of the TLS key repository file, the store for digital certificates and their associated private keys. If you do not specify a key file, TLS is not used.

The maximum length of the string is 256 characters;

- AIX Linux On AIX and Linux, the name is of the form *pathname/keyfile*.
- Windows On Windows, the name is of the form *pathname\keyfile*.

where *keyfile* is specified without the suffix `.kdb`, and identifies a Java keystore file.

TRPTYPE (*string*)

The transmission protocol to be used:

TCP
TCP/IP.

USECLTID

Decide whether you want to use the MQTT client ID for the new connection as the IBM MQ user ID for that connection. If this property is specified, the user name supplied by the client is ignored.

If you set this parameter to `YES`, then **MCAUSER** must be blank.

If **USECLNTID** is set to `NO`, and **MCAUSER** is blank, then the user name flowed in the MQTT CONNECT Packet is used. See MQTT client identity and authorization.

Related reference:

"ALTER CHANNEL (MQTT)" on page 440

Syntax diagram for a telemetry channel when using the **ALTER CHANNEL** command.

Related information:

Telemetry channel configuration for MQTT client authentication using TLS

Telemetry channel configuration for channel authentication using TLS

CipherSpecs and CipherSuites

System requirements for using SHA-2 cipher suites with MQTT channels

DEFINE COMMINFO on Multiplatforms: Multi

Use the MQSC command DEFINE COMMINFO to define a new communication information object. These objects contain the definitions required for Multicast messaging.

Using MQSC commands

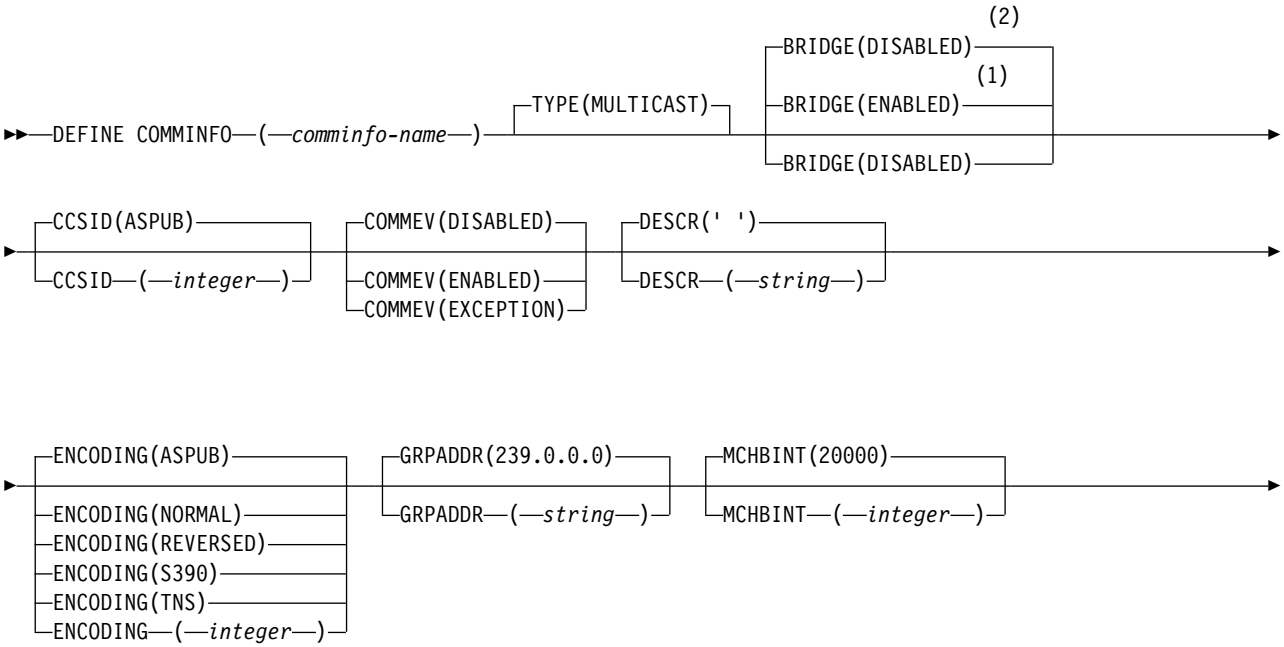
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

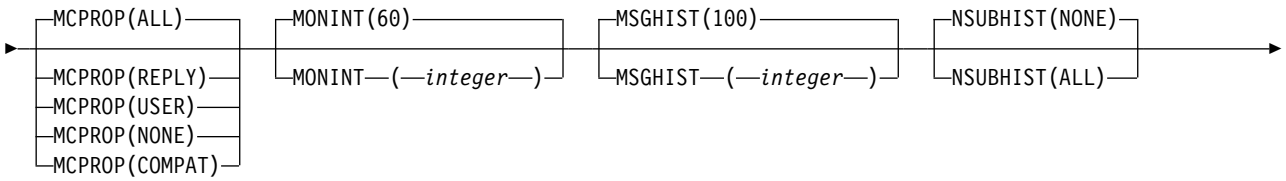
- Syntax diagram
- "Parameter descriptions for DEFINE COMMINFO" on page 639

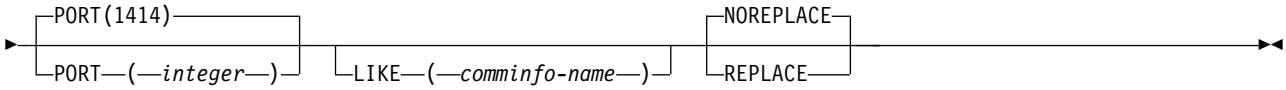
Synonym: DEF COMMINFO

Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

DEFINE COMMINFO







Notes:

1 Default for platforms other than IBM i.

2 Default for IBM i.

Parameter descriptions for DEFINE COMMINFO

(comminfo name)
Name of the communications information object. This is required.

The name must not be the same as any other communications information object name currently defined on this queue manager. See Rules for naming IBM MQ objects.

TYPE The type of the communications information object. The only type supported is `MULTICAST`.

BRIDGE
Controls whether publications from applications not using Multicast are bridged to applications using Multicast. Bridging does not apply to topics that are marked as **MCAST(ONLY)**. As these topics can only be Multicast traffic, it is not applicable to bridge to the queue's publish/subscribe domain.

DISABLED
Publications from applications not using Multicast are not bridged to applications that do use Multicast. This is the default for IBM i.

ENABLED
Publications from applications not using Multicast are bridged to applications that do use Multicast. This is the default for platforms other than IBM i.

CCSID(*integer*)
The coded character set identifier that messages are transmitted on. Specify a value in the range 1 through 65535.

The CCSID must specify a value that is defined for use on your platform, and use a character set that is appropriate to the platform. If you use this parameter to change the CCSID, applications that are running when the change is applied continue to use the original CCSID. Because of this, you must stop and restart all running applications before you continue. This includes the command server and channel programs. To do this, stop and restart the queue manager after making the change.

The default value is `ASPUB` which means that the coded character set is taken from the one that is supplied in the published message.

COMMEV

Controls whether event messages are generated for Multicast handles that are created using this COMMINFO object. Events will only be generated if they are enabled using the **MONINT** parameter.

DISABLED
Event messages are not generated for Multicast handles that are created using the COMMINFO object. This is the default value.

ENABLED
Event messages are generated for Multicast handles that are created using the COMMINFO object.

EXCEPTION
Event messages are written if the message reliability is below the reliability threshold The reliability threshold is set to 90 by default.

DESCR(*string* **)**

Plain-text comment. It provides descriptive information about the communication information object when an operator issues the DISPLAY COMMINFO command (see "DISPLAY COMMINFO on Multiplatforms" on page 820).

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

ENCODING

The encoding that the messages are transmitted in.

ASPUB
The encoding of the message is taken from the one that is supplied in the published message. This is the default value.

REVERSED

NORMAL

S390

TNS

encoding

GRPADDR

The group IP address or DNS name.

It is the administrator's responsibility to manage the group addresses. It is possible for all multicast clients to use the same group address for every topic; only the messages that match outstanding subscriptions on the client are delivered. Using the same group address can be inefficient because every client must examine and process every multicast packet in the network. It is more efficient to allocate different IP group addresses to different topics or sets of topics, but this requires careful management, especially if other non-MQ multicast applications are in use on the network. The default value is `239.0.0.0`.

MCHBINT

The heartbeat interval is measured in milliseconds, and specifies the frequency at which the transmitter notifies any receivers that there is no further data available. The value is in the range 0 to 999 999. The default value is `2000` milliseconds.

MCPROP
The multicast properties control how many of the MQMD properties and user properties flow with the message.

All
All user properties and all the fields of the MQMD are transported.

Reply
Only user properties, and MQMD fields that deal with replying to the messages, are transmitted. These properties are:

- MsgType
- MessageId
- CorrelId
- ReplyToQ
- ReplyToQmgr

User
Only the user properties are transmitted.

NONE
No user properties or MQMD fields are transmitted.

COMPAT
This value causes the transmission of the message to be done in a compatible mode to RMM. This allows some inter-operation with the current XMS applications and Broker RMM applications.

MONINT(*integer* **)**
How frequently, in seconds, that monitoring information is updated. If events messages are enabled, this parameter also controls how frequently event messages are generated about the status of the Multicast handles created using this COMMINFO object.

A value of 0 means that there is no monitoring.

The default value is 60.

MSGHIST
This value is the amount of message history in kilobytes that is kept by the system to handle retransmissions in the case of NACKs (negative acknowledgments).

The value is in the range 0 to 999 999 999. A value of `0` gives the least level of reliability. The default value is `100`.

NSUBHIST
The new subscriber history controls whether a subscriber joining a publication stream receives as much data as is currently available, or receives only publications made from the time of the subscription.

NONE
A value of `NONE` causes the transmitter to transmit only publication made from the time of the subscription. This is the default value.

ALL
A value of `ALL` causes the transmitter to retransmit as much history of the topic as is known. In some circumstances this can give a similar behavior to retained publications.

Note: Using the value of `ALL` might have a detrimental effect on performance if there is a large topic history because all the topic history is retransmitted.

PORT(*integer* **)**
The port number to transmit on. The default port number is `1414`.

LIKE(*authinfo-name*)

The name of a communication information object, with parameters that are used to model this definition.

If this field is not complete, and you do not complete the parameter fields related to the command, the values are taken from the default definition for an object of this type.

This default communication information object definition can be altered by the installation to the default values required.

REPLACE and NOREPLACE

Whether the existing definition is to be replaced with this one. This is optional. The default is NOREPLACE. Any object with a different disposition is not changed.

REPLACE

The definition replaces an existing definition of the same name. If a definition does not exist, one is created.

NOREPLACE

The definition does not replace an existing definition of the same name.

DEFINE LISTENER on Multiplatforms: Multi

Use the MQSC command DEFINE LISTENER to define a new IBM MQ listener definition, and set its parameters.

Using MQSC commands

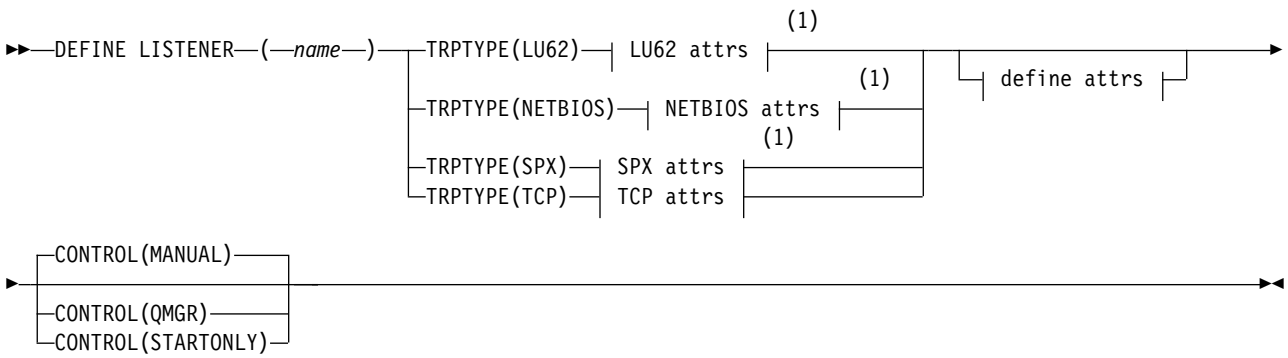
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Parameter descriptions for DEFINE LISTENER" on page 643

Synonym: DEF LSTR

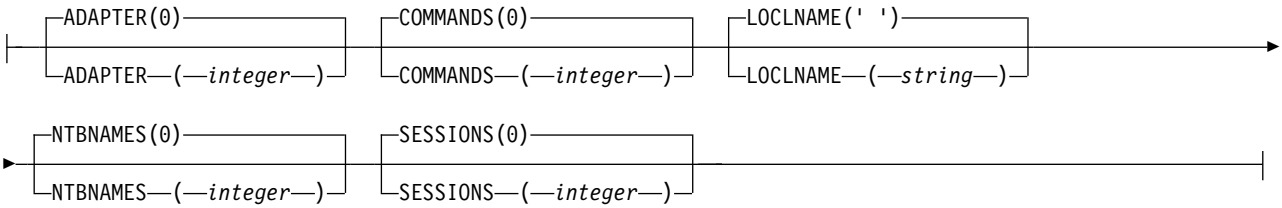
Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

DEFINE LISTENER

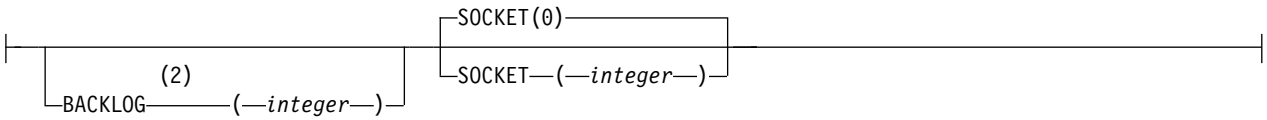


LU62 attrs:

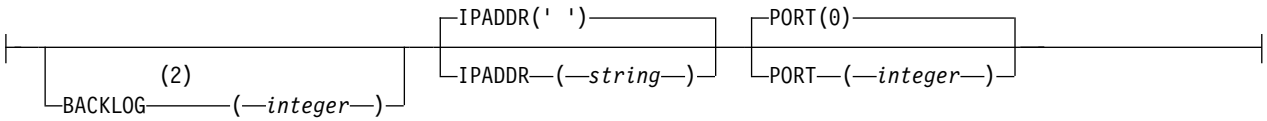
NETBIOS attrs:



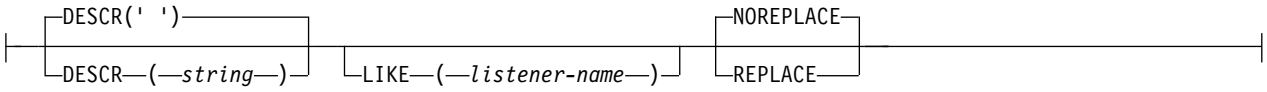
SPX attrs:



TCP attrs:



Define attrs:



Notes:

1 Valid only on Windows.

2 When the BACKLOG attribute is left unchanged or when it is explicitly set to zero, the attribute is stored as zero by default in the listener object created by the **DEFINE LISTENER** command. However, when the listener is started, the default backlog value takes effect. For information about the default value of the BACKLOG attribute, see Using the TCP listener backlog option.

Parameter descriptions for DEFINE LISTENER

listener-name)
Name of the IBM MQ listener definition (see Rules for naming IBM MQ objects). This is required.

The name must not be the same as any other listener definition currently defined on this queue manager (unless REPLACE is specified).

Windows **ADAPTER(*integer*)**
The adapter number on which NetBIOS listens. This parameter is valid only on Windows when TRPTYPE is NETBIOS.

BACKLOG(*integer*)
The number of concurrent connection requests that the listener supports.

Windows **COMMANDS(*integer*)**
The number of commands that the listener can use. This parameter is valid only on Windows when **TRPTYPE** is NETBIOS.

CONTROL(*string*)
Specifies how the listener is to be started and stopped.:

MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by use of the **START LISTENER** and **STOP LISTENER** commands.

QMGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY
The listener is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

DESCR(*string*)
Plain-text comment. It provides descriptive information about the listener when an operator issues the **DISPLAY LISTENER** command (see "DISPLAY LISTENER on Multiplatforms" on page 839).

It should contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

IPADDR(*string*)
IP address for the listener specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric host name form. If you do not specify a value for this parameter, the listener listens on all configured IPv4 and IPv6 stacks.

LIKE(*listener-name*)
The name of a listener, with parameters that are used to model this definition.

This parameter applies only to the **DEFINE LISTENER** command.

If this field is not filled in, and you do not complete the parameter fields related to the command, the values are taken from the default definition for listeners on this queue manager. This is equivalent to specifying:

```
LIKE(SYSTEM.DEFAULT.LISTENER)
```

A default listener is provided but it can be altered by the installation of the default values required. See Rules for naming IBM MQ objects.

Windows **LOCLNAME(*string*)**
The NetBIOS local name that the listener uses. This parameter is valid only on Windows when **TRPTYPE** is NETBIOS.

Windows **NTBNAMES(*integer*)**
The number of names that the listener can use. This parameter is valid only on Windows when **TRPTYPE** is NETBIOS.

PORT(*integer*)
The port number for TCP/IP. This is valid only when TRPTYPE is TCP. It must not exceed 65535.

Windows **SESSIONS(*integer*)**
The number of sessions that the listener can use. This parameter is valid only on Windows when **TRPTYPE** is NETBIOS.

SOCKET(*integer*)
The SPX socket on which to listen. This is valid only if **TRPTYPE** is SPX.

Windows **TPNAME(*string*)**
The LU 6.2 transaction program name (maximum length 64 characters). This parameter is valid only on Windows when **TRPTYPE** is LU62.

TRPTYPE(*string*)
The transmission protocol to be used:

Windows **LU62**
SNA LU 6.2. This is valid only on Windows.

Windows **NETBIOS**
NetBIOS. This is valid only on Windows.

Windows **SPX**
Sequenced packet exchange. This is valid only on Windows.

TCP
TCP/IP.

DEFINE LOG on z/OS: z/OS

Use the MQSC command DEFINE LOG to add a new active log data set in the ring of active logs.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

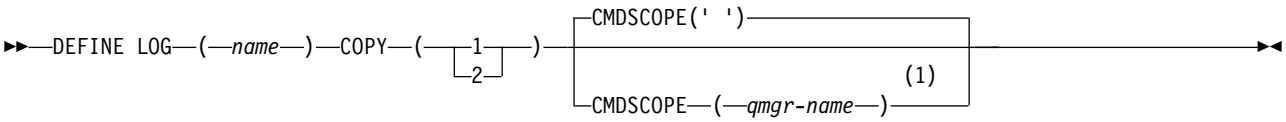
You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

The named data set is dynamically allocated to the running queue manager, added to either the COPY1 or COPY2 active log and the BSDS updated with the information so it is retained over a queue manager restart. The data set is added to the active log ring in a position such that it will be the next active log used when the current active log fills and an active log switch occurs.

- Syntax diagram
- "Usage note for DEFINE LOG" on page 646
- "Parameter descriptions for DEFINE LOG" on page 646

Synonym: DEF LOG

DEFINE LOG



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Usage note for DEFINE LOG

If a log data set has to be added because there is no more log space and the queue manager is waiting, you must issue the command from the z/OS console, and not through the command server.

Parameter descriptions for DEFINE LOG

(name)

The name of the new log data set. This is required and is the name of a VSAM linear data set which will have already been defined by Access Method Services (and, optionally, formatted by utility CSQJUFMT). This is allocated dynamically to the queue manager.

The maximum length of the string is 44 characters. The string must conform to z/OS data set naming conventions.

COPY

Specifies the number of an active log ring to which to add the new log data set. This is either 1 or 2 and is required.

CMDSCOPE

This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

DEFINE MAXSMSGS on z/OS: z/OS

Use the MQSC command DEFINE MAXSMSGS to define the maximum number of messages that a task can get or put within a single unit of recovery.

Using MQSC commands

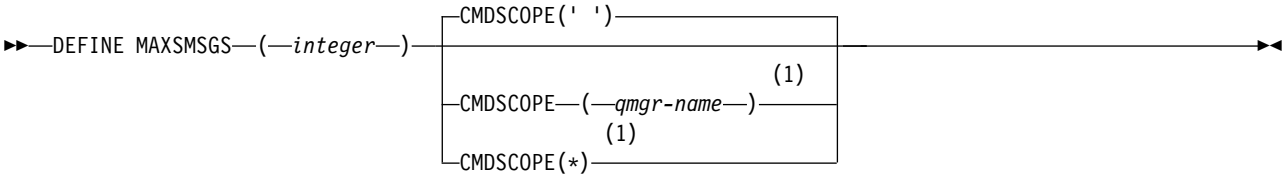
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes"
- "Parameter descriptions for DEFINE MAXSMSGS"

Synonym: DEF MAXSM

DEFINE MAXSMSGS



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Usage notes

1. This command is valid only on z/OS and is retained for compatibility with earlier releases, although it can no longer be issued from the CSQINP1 initialization input data set. You should use the MAXUMSGS parameter of the ALTER QMGR command instead.
2. You can issue the DEFINE MAXSMSGS command to change the number of messages allowed. Once a value is set, it is preserved during a queue manager restart.

Parameter descriptions for DEFINE MAXSMSGS

(integer)
The maximum number of messages that a task can get or put within a single unit of recovery. This value must be an integer in the range 1 through 999999999. The default value is 10000.

The number includes any trigger messages and report messages generated within the same unit of recovery.

CMDSCOPE
This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

' '
The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

DEFINE NAMELIST:

Use the MQSC command DEFINE NAMELIST to define a list of names. This is most commonly a list of cluster names or queue names.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

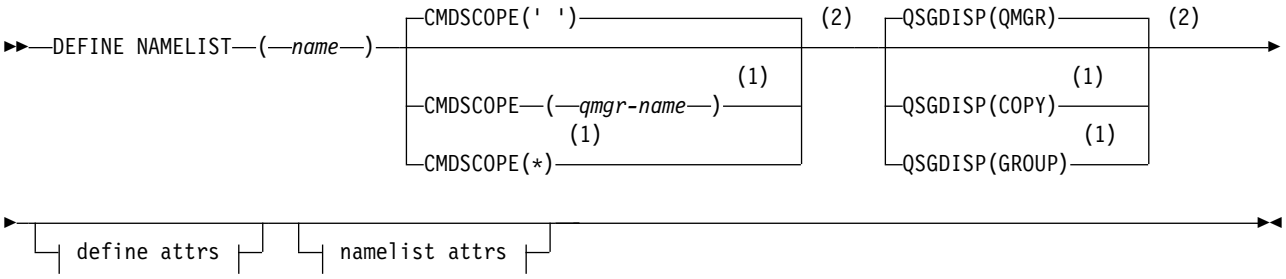
z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes" on page 649
- "Parameter descriptions for DEFINE NAMELIST" on page 649

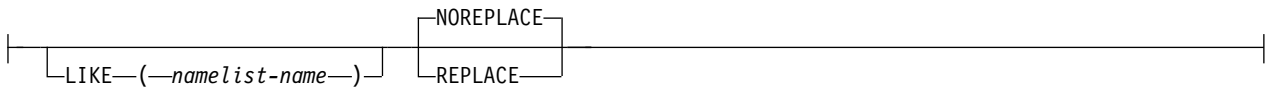
Synonym: DEF NL

Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

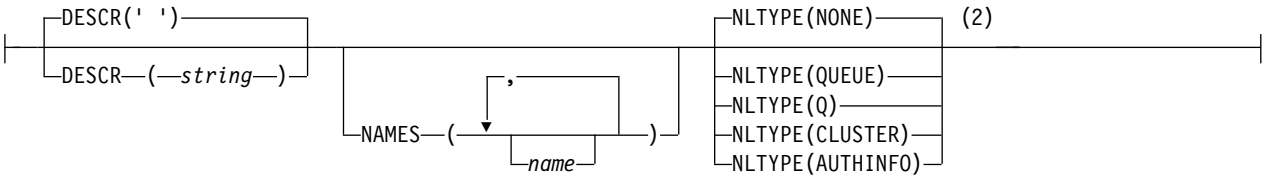
DEFINE NAMELIST



Define attrs:



Namelist attrs:



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes

Successful completion of the command does not mean that the action completed. To check for true completion, see the DEFINE NAMELIST step in Checking that async commands for distributed networks have finished.

Parameter descriptions for DEFINE NAMELIST

(name)

Name of the list.

The name must not be the same as any other namelist name currently defined on this queue manager (unless REPLACE or ALTER is specified). See Rules for naming IBM MQ objects.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command runs on the queue manager on which it was entered.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of specifying * is the same as entering the command on every queue manager in the queue-sharing group.

DESCR(***string*** **)**

Plain-text comment. It provides descriptive information about the namelist when an operator issues the DISPLAY NAMELIST command (see “DISPLAY NAMELIST” on page 848).

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

LIKE(***namelist-name*** **)**

The name of a namelist, with parameters that are used to model this definition.

If this field is not completed and you do not complete the parameter fields related to the command, the values are taken from the default definition for namelists on this queue manager.

Not completing this parameter is equivalent to specifying:

```
LIKE(SYSTEM.DEFAULT.NAMELIST)
```

A default namelist definition is provided, but it can be altered by the installation to the default values required. See Rules for naming IBM MQ objects.

z/OS On z/OS, the queue manager searches page set zero for an object with the name you specify and a disposition of QMGR or COPY. The disposition of the LIKE object is not copied to the object you are defining.

Note:

1. QSGDISP (GROUP) objects are not searched.
2. LIKE is ignored if QSGDISP(COPY) is specified.

NAMES(*name, ...*)

List of names.

The names can be of any type, but must conform to the rules for naming IBM MQ objects, with a maximum length of 48 characters.

An empty list is valid: specify NAMES(). The maximum number of names in the list is 256.

NLTYPE

Indicates the type of names in the namelist.

This parameter is valid only on z/OS.

NONE
The names are of no particular type.

QUEUE or Q
A namelist that holds a list of queue names.

CLUSTER
A namelist that is associated with clustering, containing a list of the cluster names.

AUTHINFO
This namelist is associated with TLS and contains a list of authentication information object names.

Namelists used for clustering must have NLTYPE(CLUSTER) or NLTYPE(NONE).

Namelists used for TLS must have NLTYPE(AUTHINFO).

z/OS **QSGDISP**

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

QSGDISP	DEFINE
COPY	The object is defined on the page set of the queue manager that executes the command using the QSGDISP(GROUP) object of the same name as the 'LIKE' object.
GROUP	The object definition resides in the shared repository but only if the queue manager is in a queue-sharing group. If the definition is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to attempt to make or refresh local copies on page set zero: `DEFINE NAMELIST(name)` `REPLACE QSGDISP(COPY)` The DEFINE for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.
PRIVATE	Not permitted.
QMGR	The object is defined on the page set of the queue manager that executes the command.

REPLACE and NOREPLACE

Whether the existing definition (and on z/OS, with the same disposition) is to be replaced with this one. Any object with a different disposition is not changed.

REPLACE

The definition replaces any existing definition of the same name. If a definition does not exist, one is created.

NOREPLACE

The definition does not replace any existing definition of the same name.

DEFINE PROCESS:

Use the MQSC command DEFINE PROCESS to define a new IBM MQ, process definition, and set its parameters.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

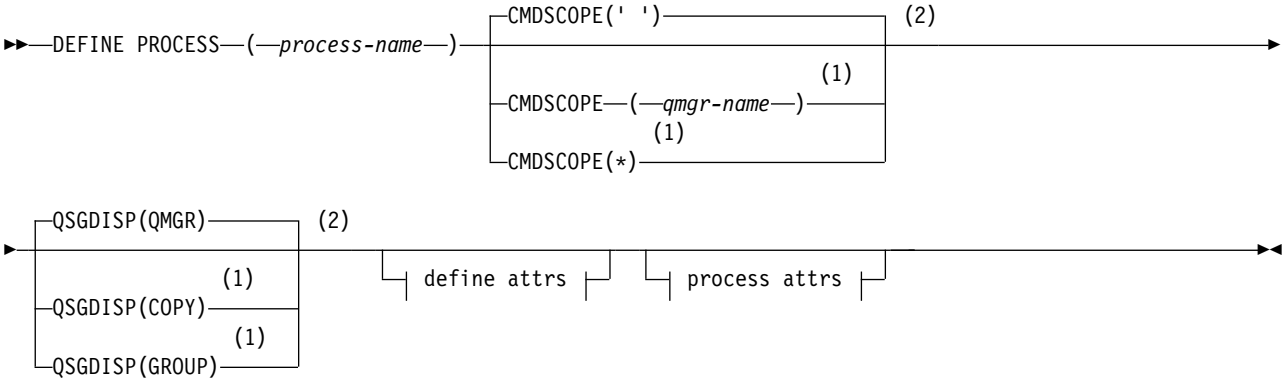
z/OS You can issue this command from sources `2CR`. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- “Parameter descriptions for DEFINE PROCESS” on page 652

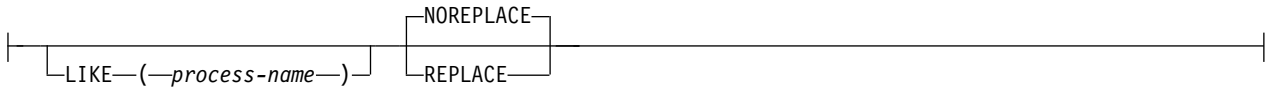
Synonym: DEF PRO

Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See “How to read railroad diagrams” on page 187.

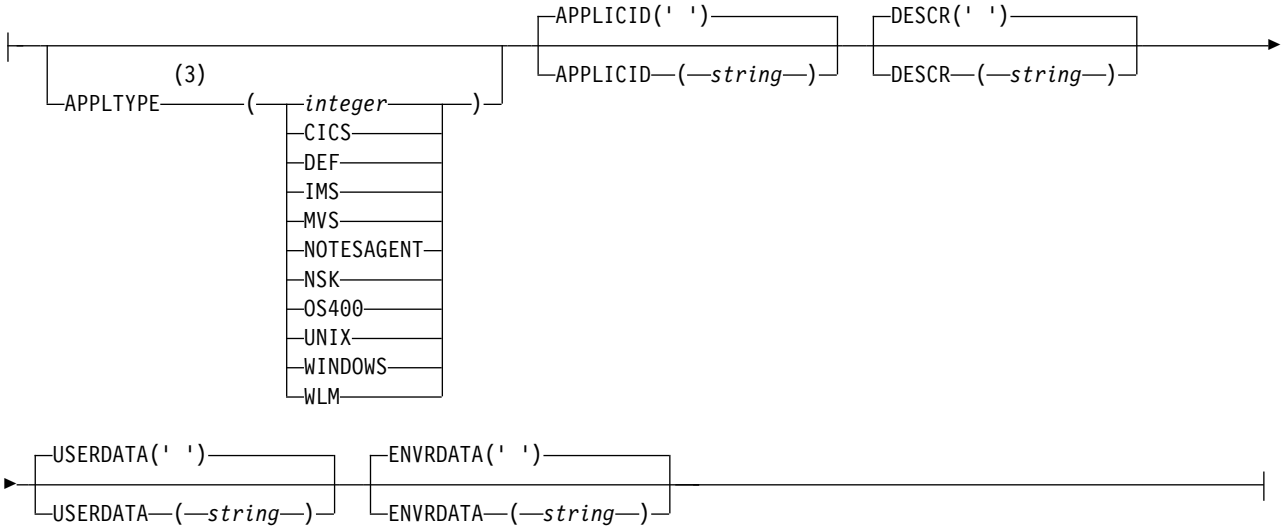
DEFINE PROCESS



Define attrs:



Process attrs:



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 The default depends on the platform, and can be changed by your installation.

Parameter descriptions for DEFINE PROCESS

(process-name)

Name of the IBM MQ process definition (see Rules for naming IBM MQ objects). *process-name* is required.

The name must not be the same as any other process definition currently defined on this queue manager (unless REPLACE is specified).

APPLICID(*string*)

The name of the application to be started. The name might typically be a fully qualified file name of an executable object. Qualifying the file name is particularly important if you have multiple IBM MQ installations, to ensure the correct version of the application is run. The maximum length is 256 characters.

For a CICS application the name is a CICS transaction ID.

z/OS For an IMS application, it is an IMS transaction ID.

z/OS On z/OS, for distributed queuing, it must be **CSQX START**.

APPLTYPE(*string*)

The type of application to be started. Valid application types are:

integer
A system-defined application type in the range zero through 65 535 or a user-defined application type in the range 65 536 through 999 999 999.

For certain values in the system range, a parameter from the following list can be specified instead of a numeric value:

CICS Represents a CICS transaction.

z/OS **IMS**
Represents an IMS transaction.

z/OS **MVS**
Represents a z/OS application (batch or TSO).

NOTESAGENT
Represents a Lotus Notes agent.

NSK Represents an HP Integrity NonStop Server application.

IBM i **OS400**
Represents an IBM i application.

UNIX Represents a UNIX application.

WINDOWS
Represents a Windows application.

z/OS **WLM**
Represents a z/OS workload manager application.

DEF Specifying DEF causes the default application type for the platform at which the command is interpreted to be stored in the process definition. This default cannot be changed by the installation. If the platform supports clients, the default is interpreted as the default application type of the server.

Only use application types (other than user-defined types) that are supported on the platform at which the command is run:

- z/OS On z/OS, CICS, IMS, MVS, UNIX, WINDOWS, WLM, and DEF are supported.
- IBM i On IBM i, OS400, CICS, and DEF are supported.
- UNIX On UNIX, UNIX, WINDOWS, CICS, and DEF are supported.
- Windows On Windows, WINDOWS, UNIX, CICS, and DEF are supported.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command runs on the queue manager on which it was entered.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

In a shared queue environment, you can provide a different queue manager name from the one you are using to enter the command. The command server must be enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect is the same as entering the command on every queue manager in the queue-sharing group.

DESCR(***string*** **)**
Plain-text comment. It provides descriptive information about the object when an operator issues the DISPLAY PROCESS command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: Use characters from the coded character set identifier (CCSID) for this queue manager. Other characters might be translated incorrectly if the information is sent to another queue manager.

ENVRDATA(***string*** **)**
A character string that contains environment information pertaining to the application to be started. The maximum length is 128 characters.

The meaning of ENVRDATA is determined by the trigger-monitor application. The trigger monitor provided by IBM MQ appends ENVRDATA to the parameter list passed to the started application. The parameter list consists of the MQTMC2 structure, followed by one blank, followed by ENVRDATA with trailing blanks removed.

Notes:

1. z/OS On z/OS, ENVRDATA is not used by the trigger-monitor applications provided by IBM MQ.
2. z/OS On z/OS, if APPLTYPE is WLM, the default values for the ServiceName and ServiceStep fields in the work information header (MQWIH) can be supplied in ENVRDATA. The format must be:

   ```
   SERVICENAME=servname,SERVICESTEP=stepname
   ```

   where:

   SERVICENAME=
   is the first 12 characters of ENVRDATA.

   servname
   is a 32-character service name. It can contain embedded blanks or any other data, and have trailing blanks. It is copied to the MQWIH as is.

   SERVICESTEP=
   is the next 13 characters of ENVRDATA.

   stepname
   is a 1 - 8 character service step name. It is copied as-is to the MQWIH, and padded to eight characters with blanks.

   If the format is incorrect, the fields in the MQWIH are set to blanks.

3. On UNIX, ENVRDATA can be set to the ampersand character to make the started application run in the background.

LIKE(*process-name*)

The name of an object of the same type, with parameters that are used to model this definition.

If this field is not provided, the values of fields you do not provide are taken from the default definition for this object.

Using `LIKE` is equivalent to specifying:

```
LIKE(SYSTEM.DEFAULT.PROCESS)
```

A default definition for each object type is provided. You can alter the provided defaults to the default values required. See Rules for naming IBM MQ objects.

z/OS On z/OS, the queue manager searches page set zero for an object with the name you specify and a disposition of QMGR or COPY. The disposition of the LIKE object is not copied to the object you are defining.

Note:

1. QSGDISP (GROUP) objects are not searched.
2. LIKE is ignored if QSGDISP(COPY) is specified.

z/OS **QSGDISP**

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

QSGDISP	DEFINE
COPY	The object is defined on the page set of the queue manager that executes the command. It uses the QSGDISP(GROUP) object of the same name as the 'LIKE' object.
GROUP	The object definition resides in the shared repository. `GROUP` is allowed only if the queue manager is in a queue-sharing group. If the definition is successful, the following command is generated. `DEFINE PROCESS(process-name)` `REPLACE QSGDISP(COPY)` The command is sent to all active queue managers in the queue-sharing group to attempt to make or refresh local copies on page set zero. The DEFINE for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.
PRIVATE	Not permitted.
QMGR	The object is defined on the page set of the queue manager that executes the command.

REPLACE and NOREPLACE

Whether the existing definition z/OS (and on z/OS, with the same disposition) is to be replaced with this one. REPLACE is optional. Any object with a different disposition is not changed.

REPLACE

The definition replaces any existing definition of the same name. If a definition does not exist, one is created.

NOREPLACE

The definition does not replace any existing definition of the same name.

USERDATA(*string*)

A character string that contains user information pertaining to the application defined in the APPLICID that is to be started. The maximum length is 128 characters.

The meaning of USERDATA is determined by the trigger-monitor application. The trigger monitor provided by IBM MQ simply passes USERDATA to the started application as part of the parameter list. The parameter list consists of the MQTMC2 structure (containing USERDATA), followed by one blank, followed by ENVRDATA with trailing blanks removed.

For IBM MQ message channel agents, the format of this field is a channel name of up to 20 characters. See Managing objects for triggering for information about what APPLICID to provide to message channel agents.

For Microsoft Windows, the character string must not contain double quotation marks if the process definition is going to be passed to **runmqtrm**.

DEFINE PSID on z/OS: z/OS

Use the MQSC command DEFINE PSID to define a page set and associated buffer pool.

Using MQSC commands

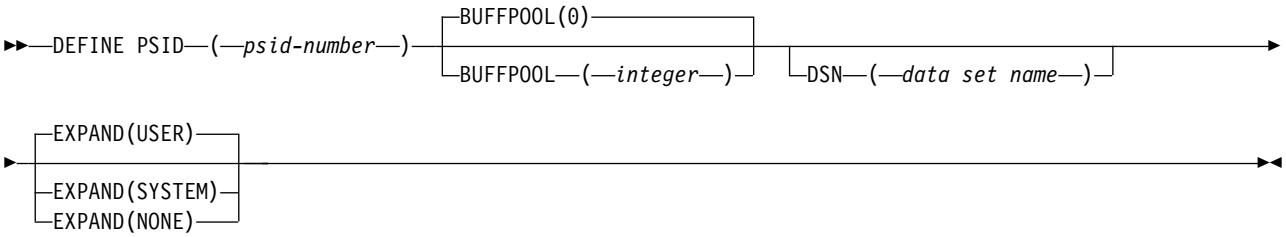
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 1CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for DEFINE PSID"
- "Parameter descriptions for DEFINE PSID" on page 657

Synonym: DEF PSID

DEFINE PSID



Usage notes for DEFINE PSID

The command can be used in two ways:

1. **At restart, from the CSQINP1 initialization input data set, to specify your standard page sets:**
   - You cannot specify the DSN keyword if issuing the command from CSQINP1.
   - If more than one DEFINE PSID command is issued for the same page set, only the last one is processed.
2. **While the queue manager is running, to dynamically add a page set:**
   - The command must specify the DSN keyword and can be issued from either of the following:
     - The z/OS console.
     - The command server and command queue by means of CSQUTIL, CSQINPX, or applications.

- The page set identifier (that is the PSID number) may have previously been used by a queue manager. It should therefore be freshly formatted by a FORMAT(RECOVER) statement in CSQUTIL, or formatted by with a FORMAT(REPLACE) in CSQUTIL.
- You cannot dynamically add page set zero.
- The BUFFPOOL parameter can specify a currently unused buffer pool. If the buffer pool was defined in CSQINP1 but not used by any PSID, then the number of buffers specified there is created if the required virtual storage is available. If this is not available, or if the buffer pool was not defined in CSQINP1, the queue manager attempts to allocate 1000 buffers. If this is not possible, 100 buffers are allocated.
- You should update your queue manager started task procedure JCL and your CSQINP1 initialization input data set to include the new page set.

One of the messages CSQP042I or CSQP041E is output when the command is complete.

You can use ALTER PSID to dynamically change the expansion method.

You can use the DISPLAY USAGE TYPE(PAGESET) command to display information about page sets (see "DISPLAY USAGE on z/OS" on page 960).

Parameter descriptions for DEFINE PSID

(psid-number)
Identifier of the page set. This is required.

A one-to-one relationship exists between page sets and the VSAM data sets used to store the pages. The identifier consists of a number in the range 00 through 99. It is used to generate a *ddname*, which references the VSAM LDS data set, in the range CSQP0000 through CSQP0099.

The identifier must not be the same as any other page set identifier currently defined on this queue manager.

BUFFPOOL(***integer*** **)**
The buffer pool number (in the range zero through 15). If OPMODE is set to OPMODE=(NEWFUNC, 800), this number is in the range zero through 99. This is optional. The default is zero.

If the buffer pool has not already been created by a DEFINE BUFFPOOL command, the buffer pool is created with 1000 buffers, and a LOCATION value of BELOW.

If the psid-number is zero, the buffer pool number must be in the range 0 to 15, otherwise the command fails, and the queue manager does not start.

DSN(***data set name*** **)**
The name of a cataloged VSAM LDS data set. This is optional. There is no default.

EXPAND
Controls how the queue manager should expand a page set when it becomes nearly full, and further pages are required in a page set.

USER The secondary extent size that was specified when the page set was defined is used. If no secondary extent size was specified, or it was specified as zero, no dynamic page set expansion can take place if page set data set is non-striped.

At restart, if a previously used page set has been replaced with a data set that is smaller, it is expanded until it reaches the size of the previously used data set. Only one extent is required to reach this size.

SYSTEM

A secondary extent size that is approximately 10 per cent of the current size of the page set is used. It can be rounded up depending on the characteristics of the DASD.

NONE
No further page set expansion is to take place.

DEFINE queues:

Use the MQSC **DEFINE** command to define a local, model, or remote queue, or a queue alias, reply-to queue alias, or a queue manager alias.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

This section contains the following commands:

- "DEFINE QALIAS" on page 682
- "DEFINE QLOCAL" on page 683
- "DEFINE QMODEL" on page 687
- "DEFINE QREMOTE" on page 690

Define a reply-to queue or queue manager alias with the "DEFINE QREMOTE" on page 690 command.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

Usage notes for DEFINE queues

- Successful completion of the command does not mean that the action completed. To check for true completion, see the DEFINE queues step in Checking that async commands for distributed networks have finished.
- For local queues
  1. z/OS You can define a local queue with QSGDISP(SHARED) even though another queue manager in the queue-sharing group already has a local version of the queue. However, when you try to access the locally defined queue, it fails with reason code MQRC_OBJECT_NOT_UNIQUE (2343). A local version of the queue with the same name can be of type QLOCAL, QREMOTE, or QALIAS and has the disposition, QSGDISP(QMGR).

     To resolve the conflict, you must delete one of the queues using the **DELETE** command. If the queue you want to delete contains messages, use the PURGE option or remove the messages first using the **MOVE** command.

     For example, to delete the QSGDISP(LOCAL) version, which contains messages, and copy those messages to the QSGDISP(SHARED) version, then issue the following commands:

     ```
     MOVE QLOCAL(QUEUE.1) QSGDISP(PRIVATE) TOQLOCAL(QUEUE.1) TYPE(ADD)
     DELETE QLOCAL(QUEUE.1) QSGDISP(QMGR)
     ```
- For alias queues:
  1. DEFINE QALIAS(*aliasqueue*) TARGET(*otherqname*) CLUSTER(*c*) advertises the queue *otherqname* by the name *aliasqueue*.
  2. DEFINE QALIAS(*aliasqueue*) TARGET(*otherqname*) allows a queue advertised by the name *otherqname* to be used on this queue manager by the name *aliasqueue*.
  3. TARGTYPE and TARGET are not cluster attributes, that is, they are not shared in a cluster environment.
- For remote queues:
  1. DEFINE QREMOTE(*rqueue*) RNAME(*otherq*) RQMNAME(*otherqm*) CLUSTER(*cl*) advertises this queue manager as a store and forward gateway to which messages for queue *rqueue* can be sent. It has no effect as a reply-to queue alias, except on the local queue manager.

DEFINE QREMOTE(*otherqm*) RNAME() RQMNAME(*anotherqm*) XMITQ(*xq*) CLUSTER advertises this queue manager as a store and forward gateway to which messages for *anotherqm* can be sent.

2. RQMNAME can itself be the name of a cluster queue manager within the cluster. You can map the advertised queue manager name to another name locally. The pattern is the same as with QALIAS definitions.
3. It is possible for the values of RQMNAME and QREMOTE to be the same if RQMNAME is itself a cluster queue manager. If this definition is also advertised using a CLUSTER attribute, do not choose the local queue manager in the cluster workload exit. If you do so, a cyclic definition results.
4. Remote queues do not have to be defined locally. The advantage of doing so is that applications can refer to the queue by a simple, locally defined name. If you do then the queue name is qualified by the name of the queue manager on which the queue resides. Using a local definition means that applications do not need to be aware of the real location of the queue.
5. A remote queue definition can also be used as a mechanism for holding a queue manager alias definition, or a reply-to queue alias definition. The name of the definition in these cases is:
   - The queue manager name being used as the alias for another queue manager name (queue manager alias), or
   - The queue name being used as the alias for the reply-to queue (reply-to queue alias).

Parameter descriptions for DEFINE QUEUE and ALTER QUEUE

Table 104 shows the parameters that are relevant for each type of queue. There is a description of each parameter after the table.

Table 104. DEFINE and ALTER QUEUE parameters

Parameter	Local queue	Model queue	Alias queue	Remote queue
ACCTQ	✓	✓		
BOQNAME	✓	✓		
BOTHRESH	✓	✓		
z/OS CFSTRUCT	✓	✓		
CLCHNAME	✓			
CLUSNL	✓		✓	✓
CLUSTER	✓		✓	✓
CLWLPRTY	✓		✓	✓
CLWLRANK	✓		✓	✓
CLWLUSEQ	✓			
z/OS CMDSCOPE	✓	✓	✓	✓
CUSTOM	✓	✓	✓	✓
DEFBIND	✓		✓	✓

Table 104. DEFINE and ALTER QUEUE parameters (continued)

Parameter	Local queue	Model queue	Alias queue	Remote queue
DEFPRESP	✓	✓	✓	✓
DEFPRTY	✓	✓	✓	✓
DEFPSIST	✓	✓	✓	✓
DEFREADA	✓	✓	✓	
DEFSOPT	✓	✓		
DEFTYPE		✓		
DESCR	✓	✓	✓	✓
DISTL	✓	✓		
FORCE	✓		✓	✓
GET	✓	✓	✓	
HARDENBO or NOHARDENBO	✓	✓		
V 9.0.2 IMGRCOVQ	✓	✓		
INDXTYPE	✓	✓		
INITQ	✓	✓		
LIKE	✓	✓	✓	✓
MAXDEPTH	✓	✓		
MAXMSGL	✓	✓		
MONQ	✓	✓		
MSGDLVSQ	✓	✓		
NOREPLACE	✓	✓	✓	✓
NPMCLASS	✓	✓		
PROCESS	✓	✓		
PROPCTL	✓	✓	✓	
PUT	✓	✓	✓	✓

Table 104. DEFINE and ALTER QUEUE parameters (continued)

Parameter	Local queue	Model queue	Alias queue	Remote queue
queue-name	✓	✓	✓	✓
QDEPTHHI	✓	✓		
QDEPTHLO	✓	✓		
QDPHIEV	✓	✓		
QDPLOEV	✓	✓		
QDPMAXEV	✓	✓		
z/OS QSGDISP	✓	✓	✓	✓
QSVCIEV	✓	✓		
QSVCINT	✓	✓		
REPLACE	✓	✓	✓	✓
RETINTVL	✓	✓		
RNAME				✓
RQMNAME				✓
SCOPE	✓		✓	✓
SHARE or NOSHARE	✓	✓		
STATQ	✓	✓		
z/OS STGCLASS	✓	✓		
TARGET			✓	
TARGQ			✓	
TARGTYPE			✓	
TRIGDATA	✓	✓		
TRIGDPTH	✓	✓		
TRIGGER or NOTRIGGER	✓	✓		
TRIGMPRI	✓	✓		

Table 104. DEFINE and ALTER QUEUE parameters (continued)

Parameter	Local queue	Model queue	Alias queue	Remote queue
TRIGTYPE	✓	✓		
USAGE	✓	✓		
XMITQ				✓

queue-name
Local name of the queue, except the remote queue where it is the local definition of the remote queue.

The name must not be the same as any other queue name of any queue type currently defined on this queue manager, unless you specify **REPLACE** or **ALTER**. See Rules for naming IBM MQ objects.

ACCTQ
Specifies whether accounting data collection is to be enabled for the queue. On z/OS, the data collected is class 3 accounting data (thread-level and queue-level accounting). In order for accounting data to be collected for this queue, accounting data for this connection must also be enabled. Turn on accounting data collection by setting either the **ACCTQ** queue manager attribute, or the options field in the MQCNO structure on the MQCONNX call.

QMGR The collection of accounting data is based on the setting of the **ACCTQ** parameter on the queue manager definition.

ON Accounting data collection is enabled for the queue unless the **ACCTQ** queue manager parameter has a value of NONE.

z/OS On z/OS systems, you must enable class 3 accounting using the **START TRACE** command.

OFF Accounting data collection is disabled for the queue.

BOQNAME (*queue-name*)
The excessive backout requeue name.

This parameter is supported only on local and model queues.

Use this parameter to set or change the back out queue name attribute of a local or model queue. Apart from allowing its value to be queried, the queue manager does nothing based on the value of this attribute. IBM MQ classes for JMS transfers a message that is backed out the maximum number of times to this queue. The maximum is specified by the **BOTHRESH** attribute.

BOTHRESH(*integer*)
The backout threshold.

This parameter is supported only on local and model queues.

Use this parameter to set or change the value of the back out threshold attribute of a local or model queue. Apart from allowing its value to be queried, the queue manager does nothing based on the value of this attribute. IBM MQ classes for JMS use the attribute to determine how many times back a message out. When the value is exceeded the message is transferred to the queue named by the **BOQNAME** attribute.

Specify a value in the range 0 - 999,999,999.

z/OS **CFSTRUCT(*structure-name*)**
Specifies the name of the coupling facility structure where you want messages stored when you use shared queues.

This parameter is supported only on z/OS for local and model queues.

The name:

- Cannot have more than 12 characters
- Must start with an uppercase letter (A - Z)
- Can include only the characters A - Z and 0 - 9

The name of the queue-sharing group to which the queue manager is connected is prefixed to the name you supply. The name of the queue-sharing group is always four characters, padded with @ symbols if necessary. For example, if you use a queue-sharing group named NY03 and you supply the name PRODUCT7, the resultant coupling facility structure name is NY03PRODUCT7. The administrative structure for the queue-sharing group (in this case NY03CSQ_ADMIN) cannot be used for storing messages.

For **ALTER QLOCAL**, **ALTER QMODEL**, **DEFINE QLOCAL** with **REPLACE**, and **DEFINE QMODEL** with **REPLACE** the following rules apply:

- On a local queue with **QSGDISP**(SHARED), **CFSTRUCT** cannot change.

  If you change either the **CFSTRUCT** or **QSGDISP** value you must delete and redefine the queue. To preserve any of the messages on the queue you must offload the messages before you delete the queue. Reload the messages after you redefine the queue, or move the messages to another queue.
- On a model queue with **DEFTYPE**(SHAREDYN), **CFSTRUCT** cannot be blank.
- On a local queue with a **QSGDISP** other than SHARED, or a model queue with a **DEFTYPE** other than SHAREDYN, the value of **CFSTRUCT** does not matter.

For **DEFINE QLOCAL** with **NOREPLACE** and **DEFINE QMODEL** with **NOREPLACE**, the coupling facility structure:

- On a local queue with **QSGDISP**(SHARED) or a model queue with a **DEFTYPE**(SHAREDYN), **CFSTRUCT** cannot be blank.
- On a local queue with a **QSGDISP** other than SHARED, or a model queue with a **DEFTYPE** other than SHAREDYN, the value of **CFSTRUCT** does not matter.

Note: Before you can use the queue, the structure must be defined in the coupling facility Resource Management (CFRM) policy data set.

CLCHNAME(*channel name*)

This parameter is supported only on transmission queues.

CLCHNAME is the generic name of the cluster-sender channels that use this queue as a transmission queue. The attribute specifies which cluster-sender channels send messages to a cluster-receiver channel from this cluster transmission queue.

You can also set the transmission queue attribute **CLCHNAME** attribute to a cluster-sender channel manually. Messages that are destined for the queue manager connected by the cluster-sender channel are stored in the transmission queue that identifies the cluster-sender channel. They are not stored in the default cluster transmission queue. If you set the **CLCHNAME** attribute to blanks, the channel switches to the default cluster transmission queue when the channel restarts. The default queue is either SYSTEM.CLUSTER.TRANSMIT.*ChannelName* or SYSTEM.CLUSTER.TRANSMIT.QUEUE, depending on the value of the queue manager **DEFCLXQ** attribute.

By specifying asterisks, “" * "”, in **CLCHNAME**, you can associate a transmission queue with a set of cluster-sender channels. The asterisks can be at the beginning, end, or any number of places in the middle of the channel name string. **CLCHNAME** is limited to a length of 48 characters, MQ_OBJECT_NAME_LENGTH. A channel name is limited to 20 characters: MQ_CHANNEL_NAME_LENGTH. If you specify an asterisk you must also set the SHARE attribute so that multiple channels can concurrently access the transmission queue.

z/OS If you specify a “" * "” in **CLCHNAME**, to obtain a channel profile name, you must specify the channel profile name within quotation marks. If you do not specify the generic channel name within quotation marks you receive message CSQ9030E.

The default queue manager configuration is for all cluster-sender channels to send messages from a single transmission queue, SYSTEM.CLUSTER.TRANSMIT.QUEUE. The default configuration can be changed by modified by changing the queue manager attribute, **DEFCLXQ**. The default value of the attribute is SCTQ. You can change the value to CHANNEL. If you set the **DEFCLXQ** attribute to CHANNEL, each cluster-sender channel defaults to using a specific cluster transmission queue, SYSTEM.CLUSTER.TRANSMIT.*ChannelName*.

z/OS On z/OS, if this parameter is set, the queue:

- Must be shareable, by specifying the queue attribute SHARE.
- Must be indexed on the correlation ID by specifying INDXTYPE(CORRELID).
- Must not be a dynamic or a shared queue.

ULW z/OS **CLUSNL(*namelist name*)**

The name of the namelist that specifies a list of clusters to which the queue belongs.

This parameter is supported only on alias, local, and remote queues.

Changes to this parameter do not affect instances of the queue that are already open.

Only one of the resultant values of **CLUSNL** or **CLUSTER** can be nonblank; you cannot specify a value for both.

On local queues, this parameter cannot be set for the following queues:

- Transmission queues
- SYSTEM.CHANNEL.*xx* queues
- SYSTEM.CLUSTER.*xx* queues
- SYSTEM.COMMAND.*xx* queues
- z/OS On z/OS only, SYSTEM.QSG.*xx* queues

This parameter is valid only on the following platforms:

- UNIX, Linux, and Windows
- z/OS

ULW z/OS **CLUSTER(*cluster name*)**

The name of the cluster to which the queue belongs.

This parameter is supported only on alias, local, and remote queues.

The maximum length is 48 characters conforming to the rules for naming IBM MQ objects. Changes to this parameter do not affect instances of the queue that are already open.

Only one of the resultant values of **CLUSNL** or **CLUSTER** can be nonblank; you cannot specify a value for both.

On local queues, this parameter cannot be set for the following queues:

- Transmission queues
- SYSTEM.CHANNEL.*xx* queues
- SYSTEM.CLUSTER.*xx* queues
- SYSTEM.COMMAND.*xx* queues
- z/OS On z/OS only, SYSTEM.QSG.*xx* queues

This parameter is valid only on the following platforms:

- UNIX, Linux, and Windows

- z/OS

CLWLPRTY(*integer*)
Specifies the priority of the queue for the purposes of cluster workload distribution. This parameter is valid only for local, remote, and alias queues. The value must be in the range zero through 9 where zero is the lowest priority and 9 is the highest. For more information about this attribute, see CLWLPRTY queue attribute.

CLWLRANK (*integer*)
Specifies the rank of the queue for the purposes of cluster workload distribution. This parameter is valid only for local, remote, and alias queues. The value must be in the range zero through 9 where zero is the lowest rank and 9 is the highest. For more information about this attribute, see CLWLRANK queue attribute.

CLWLUSEQ
Specifies the behavior of an MQPUT operation when the target queue has a local instance and at least one remote cluster instance. The parameter has no effect when the MQPUT originates from a cluster channel. This parameter is valid only for local queues.

QMGR The behavior is as specified by the **CLWLUSEQ** parameter of the queue manager definition.

ANY The queue manager is to treat the local queue as another instance of the cluster queue for the purposes of workload distribution.

LOCAL The local queue is the only target of the MQPUT operation.

z/OS z/OS **CMDSCOPE**
This parameter applies to z/OS only. It specifies where the command is run when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if **QSGDISP** is set to GROUP or SHARED.

'' The command runs on the queue manager on which it was entered.

QmgrName
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered. You can specify another name, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

CUSTOM(*string*)
The custom attribute for new features.

This attribute contains the values of attributes, as pairs of attribute name and value, separated by at least one space. The attribute name-value pairs have the form NAME(VALUE). Single quotation marks must be escaped with another single quotation mark.

CAPEXPRY(*integer*)
The maximum time, expressed in tenths of a second, until a message put using an object handle with this object in the resolution path, becomes eligible for expiry processing.

For more information on message expiry processing, see Enforcing lower expiration times.

integer
The value must be in the range one through to 999 999 999.

NOLIMIT
There is no limit on the expiry time of messages put using this object. This is the default value.

Specifying a value for **CAPEXPRY** that is not valid, does not cause the command to fail. Instead the default value is used.
Note that existing messages in the queue, prior to a change in **CAPEXPRY**, are not affected by the change (that is, their expiry time remains intact). Only new messages that are put into the queue after the change in **CAPEXPRY** have the new expiry time.

DEFBIND
Specifies the binding to be used when the application specifies MQOO_BIND_AS_Q_DEF on the MQOPEN call, and the queue is a cluster queue.

OPEN The queue handle is bound to a specific instance of the cluster queue when the queue is opened.

NOTFIXED
The queue handle is not bound to any instance of the cluster queue. The queue manager selects a specific queue instance when the message is put using MQPUT. It changes that selection later, if the need arises.

GROUP Allows an application to request that a group of messages is allocated to the same destination instance.

Multiple queues with the same name can be advertised in a queue manager cluster. An application can send all messages to a single instance, MQOO_BIND_ON_OPEN. It can allow a workload management algorithm to select the most suitable destination on a per message basis, MQOO_BIND_NOT_FIXED. It can allow an application to request that a group of messages be all allocated to the same destination instance. The workload balancing reselects a destination between groups of messages, without requiring an MQCLOSE and MQOPEN of the queue.

The MQPUT1 call always behaves as if NOTFIXED is specified.

This parameter is valid on all platforms.

DEFPRESP
Specifies the behavior to be used by applications when the put response type, within the MQPMO options, is set to MQPMO_RESPONSE_AS_Q_DEF.

SYNC Put operations to the queue specifying MQPMO_RESPONSE_AS_Q_DEF are issued as if MQPMO_SYNC_RESPONSE is specified instead.

ASYNC Put operations to the queue specifying MQPMO_RESPONSE_AS_Q_DEF are issued as if MQPMO_ASYNC_RESPONSE is specified instead; see MQPMO options (MQLONG).

DEFPRTY(*integer*)
The default priority of messages put on the queue. The value must be in the range 0 - 9. Zero is the lowest priority, through to the **MAXPRTY** queue manager parameter. The default value of **MAXPRTY** is 9.

DEFPSIST
Specifies the message persistence to be used when applications specify the MQPER_PERSISTENCE_AS_Q_DEF option.

NO Messages on this queue are lost across a restart of the queue manager.

YES Messages on this queue survive a restart of the queue manager.

z/OS On z/OS, N and Y are accepted as synonyms of NO and YES.

DEFREADA

Specifies the default read ahead behavior for non-persistent messages delivered to the client. Enabling read ahead can improve the performance of client applications consuming non-persistent messages.

NO Non-persistent messages are not read ahead unless the client application is configured to request read ahead.

YES Non-persistent messages are sent to the client before an application requests them. Non-persistent messages can be lost if the client ends abnormally or if the client does not delete all the messages it is sent.

DISABLED
Read ahead of non-persistent messages in not enabled for this queue. Messages are not sent ahead to the client regardless of whether read ahead is requested by the client application.

DEFSOPT

The default share option for applications opening this queue for input:

EXCL The open request is for exclusive input from the queue

SHARED The open request is for shared input from the queue

DEFTYPE

Queue definition type.

This parameter is supported only on model queues.

PERMDYN
A permanent dynamic queue is created when an application issues an MQOPEN MQI call with the name of this model queue specified in the object descriptor (MQOD).

z/OS On z/OS, the dynamic queue has a disposition of QMGR.

z/OS **SHAREDYN**
This option is available on z/OS only.

A permanent dynamic queue is created when an application issues an MQOPEN API call with the name of this model queue specified in the object descriptor (MQOD).

The dynamic queue has a disposition of SHARED.

TEMPDYN
A temporary dynamic queue is created when an application issues an MQOPEN API call with the name of this model queue specified in the object descriptor (MQOD).

z/OS On z/OS, the dynamic queue has a disposition of QMGR.

Do not specify this value for a model queue definition with a **DEFPSIST** parameter of YES.

If you specify this option, do not specify **INDXTYPE**(MSGTOKEN).

DESCR(*string*)

Plain-text comment. It provides descriptive information about the object when an operator issues the **DISPLAY QUEUE** command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: Use characters that are in the coded character set identifier (CCSID) of this queue manager. If you do not do so and if the information is sent to another queue manager, they might be translated incorrectly.

ULW **DISTL**

DISTL sets whether distribution lists are supported by the partner queue manager.

YES Distribution lists are supported by the partner queue manager.

NO Distribution lists are not supported by the partner queue manager.

Note: You do not normally change this parameter, because it is set by the MCA. However you can set this parameter when defining a transmission queue if the distribution list capability of the destination queue manager is known.

This parameter is valid only on UNIX, Linux, and Windows.

FORCE

This parameter applies only to the **ALTER** command on alias, local and remote queues.

Specify this parameter to force completion of the command in the following circumstances.

For an alias queue, if both of the following statements are true:

- The **TARGQ** parameter is specified
- An application has this alias queue open

For a local queue, if both of the following statements are true:

- The **NOSHARE** parameter is specified
- More than one application has the queue open for input

FORCE is also needed if both of the following statements are true:

- The **USAGE** parameter is changed
- Either one or more messages are on the queue, or one or more applications have the queue open

Do not change the **USAGE** parameter while there are messages on the queue; the format of messages changes when they are put on a transmission queue.

For a remote queue, if both of the following statements are true:

- The **XMITQ** parameter is changed
- One or more applications has this queue open as a remote queue

FORCE is also needed if both of the following statements are true:

- Any of the **RNAME**, **RQMNAME**, or **XMITQ** parameters are changed
- One or more applications has a queue open that resolved through this definition as a queue manager alias

Note: **FORCE** is not required if this definition is in use as a reply-to queue alias only.

If **FORCE** is not specified in the circumstances described, the command is unsuccessful.

GET Specifies whether applications are to be permitted to get messages from this queue:

ENABLED
Messages can be retrieved from the queue, by suitably authorized applications.

DISABLED
Applications cannot retrieve messages from the queue.

This parameter can also be changed using the MQSET API call.

HARDENBO &NOHARDENBO

Specifies whether hardening is used to ensure that the count of the number of times that a message is backed out is accurate.

This parameter is supported only on local and model queues.

HARDENBO
The count is hardened.

NOHARDENBO
The count is not hardened.

Note: z/OS This parameter affects only IBM MQ for z/OS. You can set this parameter on Multiplatforms, but it is ineffective.

V 9.0.2 Multi V 9.0.2 **IMGRCOVQ**
Specifies whether a local or permanent dynamic queue object is recoverable from a media image, if linear logging is being used. Possible values are:

YES These queue objects are recoverable.

NO The "rcdmqimg (record media image)" on page 283 and"rcrmqobj (re-create object)" on page 289 commands are not permitted for these objects, and automatic media images, if enabled, are not written for these objects.

QMGR
If you specify `QMGR`, and the **IMGRCOVQ** attribute for the queue manager specifies `YES`, these queue objects are recoverable.

If you specify `QMGR` and the **IMGRCOVQ** attribute for the queue manager specifies `NO`, the "rcdmqimg (record media image)" on page 283 and"rcrmqobj (re-create object)" on page 289 commands are not permitted for these objects, and automatic media images, if enabled, are not written for these objects.

`QMGR` is the default value.

This parameter is not valid on z/OS.

z/OS **INDXTYPE**
The type of index maintained by the queue manager to expedite MQGET operations on the queue. For shared queues, the type of index determines the type of MQGET operations that can be used.

This parameter is supported only on z/OS. On other platforms, all queues are automatically indexed.

This parameter is supported only on local and model queues.

Messages can be retrieved using a selection criterion only if an appropriate index type is maintained, as the following table shows:

Table 105. Index type required for different retrieval selection criteria

Retrieval selection criterion	**Index type required**	
	Shared queue	Other queue
None (sequential retrieval)	Any	Any
Message identifier	`MSGID` or `NONE`	Any
Correlation identifier	`CORRELID`	Any
Message and correlation identifiers	`MSGID` or `CORRELID`	Any
Group identifier	`GROUPID`	Any
Grouping	`GROUPID`	`GROUPID`
Message token	Not allowed	`MSGTOKEN`

where the value of **INDXTYPE** parameter has the following values:

NONE No index is maintained. Use NONE when messages are typically retrieved sequentially or use both the message identifier and the correlation identifier as a selection criterion on the MQGET call.

MSGID An index of message identifiers is maintained. Use MSGID when messages are typically retrieved using the message identifier as a selection criterion on the MQGET call with the correlation identifier set to NULL.

CORRELID
An index of correlation identifiers is maintained. Use CORRELID when messages are typically retrieved using the correlation identifier as a selection criterion on the MQGET call with the message identifier set to NULL.

GROUPID
An index of group identifiers is maintained. Use GROUPID when messages are retrieved using message grouping selection criteria.

Note:

1. You cannot set **INDXTYPE** to GROUPID if the queue is a transmission queue.
2. The queue must use a CF structure at CFLEVEL(3), to specify a shared queue with **INDXTYPE**(GROUPID).

z/OS **MSGTOKEN**
An index of message tokens is maintained. Use MSGTOKEN when the queue is a WLM-managed queue that you are using with the Workload Manager functions of z/OS.

Note: You cannot set **INDXTYPE** to MSGTOKEN if:

- The queue is a model queue with a definition type of SHAREDYN
- The queue is a temporary dynamic queue
- The queue is a transmission queue
- You specify **QSGDISP**(SHARED)

For queues that are not shared and do not use grouping or message tokens, the index type does not restrict the type of retrieval selection. However, the index is used to expedite **GET** operations on the queue, so choose the type that corresponds to the most common retrieval selection.

If you are altering or replacing an existing local queue, you can change the **INDXTYPE** parameter only in the cases indicated in the following table:

Table 106. Index type change permitted depending upon queue-sharing and presence of messages in the queue

Queue type		**NON-SHARED**			**SHARED**	
Queue state		Uncommitted activity	No uncommitted activity, messages present	No uncommitted activity, and empty	Open or messages present	Not open, and empty
Change **INDXTYPE** from:	To:	Change allowed?				
NONE	MSGID	No	Yes	Yes	No	Yes
NONE	CORRELID	No	Yes	Yes	No	Yes
NONE	MSGTOKEN	No	No	Yes	-	-
NONE	GROUPID	No	No	Yes	No	Yes
MSGID	NONE	No	Yes	Yes	No	Yes
MSGID	CORRELID	No	Yes	Yes	No	Yes
MSGID	MSGTOKEN	No	No	Yes	-	-

Table 106. Index type change permitted depending upon queue-sharing and presence of messages in the queue (continued)

Queue type		NON-SHARED			SHARED	
MSGID	GROUPID	No	No	Yes	No	Yes
CORRELID	NONE	No	Yes	Yes	No	Yes
CORRELID	MSGID	No	Yes	Yes	No	Yes
CORRELID	MSGTOKEN	No	No	Yes	-	-
CORRELID	GROUPID	No	No	Yes	No	Yes
MSGTOKEN	NONE	No	Yes	Yes	-	-
MSGTOKEN	MSGID	No	Yes	Yes	-	-
MSGTOKEN	CORRELID	No	Yes	Yes	-	-
MSGTOKEN	GROUPID	No	No	Yes	-	-
GROUPID	NONE	No	No	Yes	No	Yes
GROUPID	MSGID	No	No	Yes	No	Yes
GROUPID	CORRELID	No	No	Yes	No	Yes
GROUPID	MSGTOKEN	No	No	Yes	-	-

INITQ(*string*)
The local name of the initiation queue on this queue manager, to which trigger messages relating to this queue are written; see Rules for naming IBM MQ objects.

This parameter is supported only on local and model queues.

LIKE(*qtype-name*)
The name of a queue, with parameters that are used to model this definition.

If this field is not completed, the values of undefined parameter fields are taken from one of the following definitions. The choice depends on the queue type:

Table 107. Queue types and their corresponding definitions

Queue type	Definition
Alias queue	SYSTEM.DEFAULT.ALIAS.QUEUE
Local queue	SYSTEM.DEFAULT.LOCAL.QUEUE
Model queue	SYSTEM.DEFAULT.MODEL.QUEUE
Remote queue	SYSTEM.DEFAULT.REMOTE.QUEUE

For example, not completing this parameter is equivalent to defining the following value of **LIKE** for an alias queue:

```
LIKE(SYSTEM.DEFAULT.ALIAS.QUEUE)
```

If you require different default definitions for all queues, alter the default queue definitions instead of using the **LIKE** parameter.

z/OS On z/OS, the queue manager searches for an object with the name and queue type you specify with a disposition of QMGR, COPY, or SHARED. The disposition of the **LIKE** object is not copied to the object you are defining.

Note:

1. **QSGDISP**(GROUP) objects are not searched.
2. **LIKE** is ignored if **QSGDISP**(COPY) is specified.

ULW z/OS **MAXDEPTH(*integer*)**

The maximum number of messages allowed on the queue.

This parameter is supported only on local and model queues.

On the following platforms, specify a value in the range zero through 999999999:

- ULW UNIX, Linux, and Windows
- z/OS z/OS

On any other IBM MQ platform, specify a value in the range zero through 640000.

Other factors can still cause the queue to be treated as full, for example, if there is no further hard disk space available.

If this value is reduced, any messages that are already on the queue that exceed the new maximum remain intact.

MAXMSGL(*integer*)

The maximum length (in bytes) of messages on this queue.

This parameter is supported only on local and model queues.

ULW On UNIX, Linux, and Windows, specify a value in the range zero to the maximum message length for the queue manager. See the **MAXMSGL** parameter of the ALTER QMGR command, ALTER QMGR MAXMSGL.

z/OS On z/OS, specify a value in the range zero through 100 MB (104 857 600 bytes).

Message length includes the length of user data and the length of headers. For messages put on the transmission queue, there are additional transmission headers. Allow an additional 4000 bytes for all the message headers.

If this value is reduced, any messages that are already on the queue with length that exceeds the new maximum are not affected.

Applications can use this parameter to determine the size of buffer for retrieving messages from the queue. Therefore, the value can be reduced only if it is known that this reduction does not cause an application to operate incorrectly.

Note that by adding the digital signature and key to the message, Advanced Message Security increases the length of the message.

MONQ

Controls the collection of online monitoring data for queues.

This parameter is supported only on local and model queues.

QMGR Collect monitoring data according to the setting of the queue manager parameter **MONQ**.

OFF Online monitoring data collection is turned off for this queue.

LOW If the value of the **MONQ** parameter of the queue manager is not NONE, online monitoring data collection is turned on for this queue.

MEDIUM If the value of the **MONQ** parameter of the queue manager is not NONE, online monitoring data collection is turned on for this queue.

HIGH If the value of the **MONQ** parameter of the queue manager is not NONE, online monitoring data collection is turned on for this queue.

There is no distinction between the values LOW, MEDIUM, and HIGH. These values all turn data collection on, but do not affect the rate of collection.

When this parameter is used in an **ALTER** queue command, the change is effective only when the queue is next opened.

MSGDLVSQ
Message delivery sequence.

This parameter is supported only on local and model queues.

PRIORITY
Messages are delivered (in response to MQGET API calls) in first-in-first-out (FIFO) order within priority.

FIFO Messages are delivered (in response to MQGET API calls) in FIFO order. Priority is ignored for messages on this queue.

The message delivery sequence parameter can be changed from PRIORITY to FIFO while there are messages on the queue. The order of the messages already on the queue is not changed. Messages added to the queue later take the default priority of the queue, and so might be processed before some of the existing messages.

If the message delivery sequence is changed from FIFO to PRIORITY, the messages put on the queue while the queue was set to FIFO take the default priority.

Note: z/OS If **INDXTYPE**(GROUPID) is specified with **MSGDLVSQ**(PRIORITY), the priority in which groups are retrieved is based on the priority of the first message within each group. The priorities 0 and 1 are used by the queue manager to optimize the retrieval of messages in logical order. The first message in each group must not use these priorities. If it does, the message is stored as if it was priority two.

Multi **NPMCLASS**
The level of reliability to be assigned to non-persistent messages that are put to the queue:

NORMAL Non-persistent messages are lost after a failure, or queue manager shutdown. These messages are discarded on a queue manager restart.

HIGH The queue manager attempts to retain non-persistent messages on this queue over a queue manager restart or switch over.

z/OS You cannot set this parameter on z/OS.

PROCESS(*string*)
The local name of the IBM MQ process.

This parameter is supported only on local and model queues.

This parameter is the name of a process instance that identifies the application started by the queue manager when a trigger event occurs; see Rules for naming IBM MQ objects.

The process definition is not checked when the local queue is defined, but it must be available for a trigger event to occur.

If the queue is a transmission queue, the process definition contains the name of the channel to be started. This parameter is optional for transmission queues on the following platforms:

- IBM i IBM i
- ULW UNIX, Linux, and Windows
- z/OS z/OS

If you do not specify it, the channel name is taken from the value specified for the **TRIGDATA** parameter.

PROPCTL

Property control attribute. The attribute is optional. It is applicable to local, alias, and model queues.

PROPCTL options are as follows. The options do not affect message properties in the MQMD or MQMD extension.

ALL

Set ALL so that an application can read all the properties of the message either in MQRFH2 headers, or as properties of the message handle.

The ALL option enables applications that cannot be changed to access all the message properties from MQRFH2 headers. Applications that can be changed, can access all the properties of the message as properties of the message handle.

In some cases, the format of data in MQRFH2 headers in the received message might be different to the format in the message when it was sent.

COMPAT

Set COMPAT so that unmodified applications that expect JMS-related properties to be in an MQRFH2 header in the message data continue to work as before. Applications that can be changed, can access all the properties of the message as properties of the message handle.

If the message contains a property with a prefix of mcd., jms., usr., or mqext., all message properties are delivered to the application. If no message handle is supplied, properties are returned in an MQRFH2 header. If a message handle is supplied, all properties are returned in the message handle.

If the message does not contain a property with one of those prefixes, and the application does not provide a message handle, no message properties are returned to the application. If a message handle is supplied, all properties are returned in the message handle.

In some cases, the format of data in MQRFH2 headers in the received message might be different to the format in the message when it was sent.

FORCE

Force all applications to read message properties from MQRFH2 headers.

Properties are always returned in the message data in an MQRFH2 header regardless of whether the application specifies a message handle.

A valid message handle supplied in the MsgHandle field of the MQGMO structure on the MQGET call is ignored. Properties of the message are not accessible using the message handle.

In some cases, the format of data in MQRFH2 headers in the received message might be different to the format in the message when it was sent.

NONE

If a message handle is supplied, all the properties are returned in the message handle.

All message properties are removed from the message body before it is delivered to the application.

PUT Specifies whether messages can be put on the queue.

ENABLED
Messages can be added to the queue (by suitably authorized applications).

DISABLED
Messages cannot be added to the queue.

This parameter can also be changed using the MQSET API call.

QDEPTHHI(*integer*)
The threshold against which the queue depth is compared to generate a Queue Depth High event.

This parameter is supported only on local and model queues.

z/OS For more information about the effect that shared queues on z/OS have on this event; see Shared queues and queue depth events on z/OS.

This event indicates that an application put a message on a queue resulting in the number of messages on the queue becoming greater than or equal to the queue depth high threshold. See the **QDPHIEV** parameter.

The value is expressed as a percentage of the maximum queue depth (**MAXDEPTH** parameter), and must be in the range zero through 100 and no less than **QDEPTHLO**.

QDEPTHLO(*integer*)
The threshold against which the queue depth is compared to generate a Queue Depth Low event.

This parameter is supported only on local and model queues.

z/OS For more information about the effect that shared queues on z/OS have on this event; see Shared queues and queue depth events on z/OS.

This event indicates that an application retrieved a message from a queue resulting in the number of messages on the queue becoming less than or equal to the queue depth low threshold. See the **QDPLOEV** parameter.

The value is expressed as a percentage of the maximum queue depth (**MAXDEPTH** parameter), and must be in the range zero through 100 and no greater than **QDEPTHHI**.

QDPHIEV
Controls whether Queue Depth High events are generated.

This parameter is supported only on local and model queues.

A Queue Depth High event indicates that an application put a message on a queue resulting in the number of messages on the queue becoming greater than or equal to the queue depth high threshold. See the **QDEPTHHI** parameter.

ENABLED
Queue Depth High events are generated.

DISABLED
Queue Depth High events are not generated.

Note: The value of this parameter can change implicitly.

z/OS On z/OS, shared queues affect the event.

For more information about this event, see Queue Depth High.

QDPLOEV
Controls whether Queue Depth Low events are generated.

This parameter is supported only on local and model queues.

A Queue Depth Low event indicates that an application retrieved a message from a queue resulting in the number of messages on the queue becoming less than or equal to the queue depth low threshold. See the **QDEPTHLO** parameter.

ENABLED
Queue Depth Low events are generated.

DISABLED
Queue Depth Low events are not generated.

Note: The value of this parameter can change implicitly.

z/OS On z/OS, shared queues affect the event.

For more information about this event, see Queue Depth Low.

QDPMAXEV
Controls whether Queue Full events are generated.

This parameter is supported only on local and model queues.

A Queue Full event indicates that a put to a queue was rejected because the queue is full. The queue depth reached its maximum value.

ENABLED
Queue Full events are generated.

DISABLED
Queue Full events are not generated.

Note: The value of this parameter can change implicitly.

z/OS On z/OS, shared queues affect the event.

For more information about this event, see Queue Full.

z/OS **QSGDISP**
This parameter applies to z/OS only.

Specifies the disposition of the object within the group.

Table 108. QSGDISP parameters.

Definitions of the QSGDISP parameters when defining a queue.

QSGDISP	DEFINE
COPY	The object is defined on the page set of the queue manager that executes the command using the QSGDISP(GROUP) object of the same name as the LIKE object. For local queues, messages are stored on the page sets of each queue manager and are available only through that queue manager.
GROUP	The object definition resides in the shared repository but only if there is a shared queue manager environment. If the definition is successful, the following command is generated. The command is sent to all active queue managers to attempt to make or refresh local copies on page set zero: `DEFINE QUEUE(q-name)` `REPLACE QSGDISP(COPY)` The **DEFINE** command for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.
PRIVATE	Not permitted.
QMGR	The object is defined on the page set of the queue manager that executes the command. For local queues, messages are stored on the page sets of each queue manager and are available only through that queue manager.

Table 108. QSGDISP parameters (continued).

Definitions of the QSGDISP parameters when defining a queue.

QSGDISP	DEFINE
SHARED	This option applies only to local queues. The object is defined in the shared repository. Messages are stored in the coupling facility and are available to any queue manager in the queue-sharing group. You can specify SHARED only if: • CFSTRUCT is nonblank • INDXTYPE is not MSGTOKEN • The queue is not: – SYSTEM.CHANNEL.INITQ – SYSTEM.COMMAND.INPUT If the queue is clustered, a command is generated. The command is sent to all active queue managers in the queue-sharing group to notify them of this clustered, shared queue.

QSVCIEV

Controls whether Service Interval High or Service Interval OK events are generated.

This parameter is supported only on local and model queues and is ineffective if it is specified on a shared queue.

A Service Interval High event is generated when a check indicates that no messages were retrieved from the queue for at least the time indicated by the **QSVCINT** parameter.

A Service Interval OK event is generated when a check indicates that messages were retrieved from the queue within the time indicated by the **QSVCINT** parameter.

Note: The value of this parameter can change implicitly. For more information, see the description of the Service Interval High and Service Interval OK events in Queue Service Interval High and Queue Service Interval OK.

HIGH Service Interval High events are generated

OK Service Interval OK events are generated

NONE No service interval events are generated

QSVCINT(*integer*)

The service interval used for comparison to generate Service Interval High and Service Interval OK events.

This parameter is supported only on local and model queues and is ineffective if it is specified on a shared queue.

See the **QSVCIEV** parameter.

The value is in units of milliseconds, and must be in the range zero through 999999999.

REPLACE & NOREPLACE

This option controls whether any existing definition is to be replaced with this one.

Note: z/OS On IBM MQ for z/OS, an existing definition is replaced only if it is of the same disposition. Any object with a different disposition is not changed.

REPLACE

If the object does exist, the effect is like issuing the **ALTER** command without the **FORCE** parameter and with all the other parameters specified. In particular, note that any messages that are on the existing queue are retained.

There is a difference between the **ALTER** command without the **FORCE** parameter, and the **DEFINE** command with the **REPLACE** parameter. The difference is that **ALTER** does not change unspecified parameters, but **DEFINE** with **REPLACE** sets all the parameters. If you use **REPLACE**, unspecified parameters are taken either from the object named on the **LIKE** parameter, or from the default definition, and the parameters of the object being replaced, if one exists, are ignored.

The command fails if both of the following statements are true:

- The command sets parameters that would require the use of the **FORCE** parameter if you were using the **ALTER** command
- The object is open

The **ALTER** command with the **FORCE** parameter succeeds in this situation.

If **SCOPE(CELL)** is specified on UNIX, Linux, or Windows, and there is already a queue with the same name in the cell directory, the command fails, even if **REPLACE** is specified.

NOREPLACE
The definition must not replace any existing definition of the object.

RETINTVL(*integer*)
The number of hours from when the queue was defined, after which the queue is no longer needed. The value must be in the range 0 - 999,999,999.

This parameter is supported only on local and model queues.

The **CRDATE** and **CRTIME** can be displayed using the **DISPLAY QUEUE** command.

This information is available for use by an operator or a housekeeping application to delete queues that are no longer required.

Note: The queue manager does not delete queues based on this value, nor does it prevent queues from being deleted if their retention interval is not expired. It is the responsibility of the user to take any required action.

RNAME(*string*)
Name of remote queue. This parameter is the local name of the queue as defined on the queue manager specified by **RQMNAME**.

This parameter is supported only on remote queues.

- If this definition is used for a local definition of a remote queue, **RNAME** must not be blank when the open occurs.
- If this definition is used for a queue manager alias definition, **RNAME** must be blank when the open occurs.

  In a queue manager cluster, this definition applies only to the queue manager that made it. To advertise the alias to the whole cluster, add the **CLUSTER** attribute to the remote queue definition.
- If this definition is used for a reply-to queue alias, this name is the name of the queue that is to be the reply-to queue.

The name is not checked to ensure that it contains only those characters normally allowed for queue names; see Rules for naming IBM MQ objects.

RQMNAME(*string*)
The name of the remote queue manager on which the queue **RNAME** is defined.

This parameter is supported only on remote queues.

- If an application opens the local definition of a remote queue, **RQMNAME** must not be blank or the name of the local queue manager. When the open occurs, if **XMITQ** is blank there must be a local queue of this name, which is to be used as the transmission queue.
- If this definition is used for a queue manager alias, **RQMNAME** is the name of the queue manager that is being aliased. It can be the name of the local queue manager. Otherwise, if **XMITQ** is blank, when the open occurs there must be a local queue of this name, which is to be used as the transmission queue.
- If **RQMNAME** is used for a reply-to queue alias, **RQMNAME** is the name of the queue manager that is to be the reply-to queue manager.

The name is not checked to ensure that it contains only those characters normally allowed for IBM MQ object names; see Rules for naming IBM MQ objects.

ULW **SCOPE**

Specifies the scope of the queue definition.

This parameter is supported only on alias, local, and remote queues.

QMGR The queue definition has queue manager scope. This means that the definition of the queue does not extend beyond the queue manager that owns it. You can open a queue for output that is owned by another queue manager in either of two ways:

1. Specify the name of the owning queue manager.
2. Open a local definition of the queue on the other queue manager.

CELL The queue definition has cell scope. Cell scope means that the queue is known to all the queue managers in the cell. A queue with cell scope can be opened for output merely by specifying the name of the queue. The name of the queue manager that owns the queue need not be specified.

If there is already a queue with the same name in the cell directory, the command fails. The **REPLACE** option does not affect this situation.

This value is valid only if a name service supporting a cell directory is configured.

Restriction: The DCE name service is no longer supported.

This parameter is valid only on UNIX, Linux, and Windows.

SHARE and NOSHARE

Specifies whether multiple applications can get messages from this queue.

This parameter is supported only on local and model queues.

SHARE More than one application instance can get messages from the queue.

NOSHARE
Only a single application instance can get messages from the queue.

Multi **STATQ**

Specifies whether statistics data collection is enabled:

QMGR Statistics data collection is based on the setting of the **STATQ** parameter of the queue manager.

ON If the value of the **STATQ** parameter of the queue manager is not NONE, statistics data collection for the queue is enabled.

OFF Statistics data collection for the queue is disabled.

If this parameter is used in an **ALTER** queue command, the change is effective only for connections to the queue manager made after the change to the parameter.

This parameter is valid only on Multiplatforms.

z/OS z/OS **STGCLASS(*string*)**
The name of the storage class.

This parameter is supported only on local and model queues.

Note: You can change this parameter only if the queue is empty and closed.

This parameter is an installation-defined name. The first character of the name must be uppercase A through Z, and subsequent characters either uppercase A through Z or numeric 0 through 9.

This parameter is valid only on z/OS; see Storage classes.

If you specify **QSGDISP**(SHARED) or **DEFTYPE**(SHAREDYN), this parameter is ignored.

TARGET(*string*)
The name of the queue or topic object being aliased; See Rules for naming IBM MQ objects. The object can be a queue or a topic as defined by **TARGTYPE**. The maximum length is 48 characters.

This parameter is supported only on alias queues.

This object needs to be defined only when an application process opens the alias queue.

This parameter is a synonym of the parameter **TARGQ**; **TARGQ** is retained for compatibility. If you specify **TARGET**, you cannot also specify **TARGQ**.

TARGTYPE(*string*)
The type of object to which the alias resolves.

QUEUE The alias resolves to a queue.

TOPIC The alias resolves to a topic.

TRIGDATA(*string*)
The data that is inserted in the trigger message. The maximum length of the string is 64 bytes.

This parameter is supported only on local and model queues.

For a transmission queue, you can use this parameter to specify the name of the channel to be started.

This parameter can also be changed using the MQSET API call.

TRIGDPTH(*integer*)
The number of messages that have to be on the queue before a trigger message is written, if **TRIGTYPE** is DEPTH. The value must be in the range 1 - 999,999,999.

This parameter is supported only on local and model queues.

This parameter can also be changed using the MQSET API call.

TRIGGER & NOTRIGGER
Specifies whether trigger messages are written to the initiation queue, named by the **INITQ** parameter, to trigger the application, named by the **PROCESS** parameter:

TRIGGER
Triggering is active, and trigger messages are written to the initiation queue.

NOTRIGGER
Triggering is not active, and trigger messages are not written to the initiation queue.

This parameter is supported only on local and model queues.

This parameter can also be changed using the MQSET API call.

TRIGMPRI(*integer*)
The message priority number that triggers this queue. The value must be in the range zero through to the **MAXPRTY** queue manager parameter; see "DISPLAY QMGR" on page 860 for details.

This parameter can also be changed using the MQSET API call.

TRIGTYPE

Specifies whether and under what conditions a trigger message is written to the initiation queue. The initiation queue is (named by the **INITQ** parameter.

This parameter is supported only on local and model queues.

FIRST Whenever the first message of priority equal to or greater than the priority specified by the **TRIGMPRI** parameter of the queue arrives on the queue.

EVERY Every time a message arrives on the queue with priority equal to or greater than the priority specified by the **TRIGMPRI** parameter of the queue.

DEPTH When the number of messages with priority equal to or greater than the priority specified by **TRIGMPRI** is equal to the number indicated by the **TRIGDPTH** parameter.

NONE No trigger messages are written.

This parameter can also be changed using the MQSET API call.

USAGE

Queue usage.

This parameter is supported only on local and model queues.

NORMAL The queue is not a transmission queue.

XMITQ The queue is a transmission queue, which is used to hold messages that are destined for a remote queue manager. When an application puts a message to a remote queue, the message is stored on the appropriate transmission queue. It stays there, awaiting transmission to the remote queue manager.

If you specify this option, do not specify values for **CLUSTER** and **CLUSNL**.

z/OS Additionally, on z/OS, do not specify **INDXTYPE**(MSGTOKEN) or **INDXTYPE**(GROUPID).

XMITQ(*string*)

The name of the transmission queue to be used for forwarding messages to the remote queue. **XMITQ** is used with either remote queue or queue manager alias definitions.

This parameter is supported only on remote queues.

If **XMITQ** is blank, a queue with the same name as **RQMNAME** is used as the transmission queue.

This parameter is ignored if the definition is being used as a queue manager alias and **RQMNAME** is the name of the local queue manager.

It is also ignored if the definition is used as a reply-to queue alias definition.

Related information:
Copying a local queue definition

DEFINE QALIAS:

Use **DEFINE QALIAS** to define a new alias queue, and set its parameters.

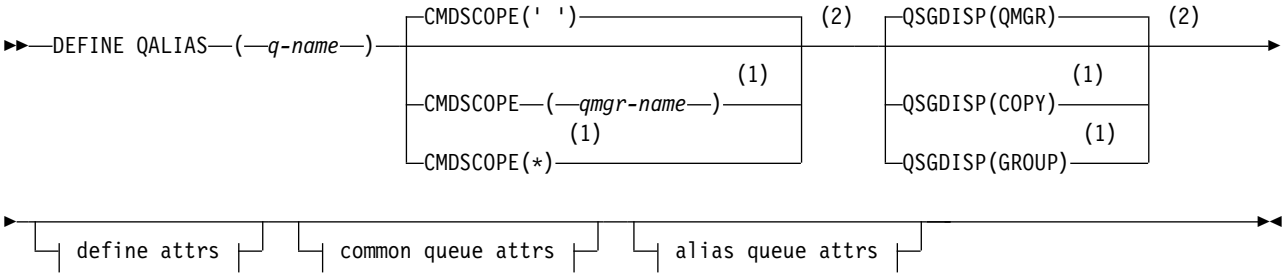
Note: An alias queue provides a level of indirection to another queue or a topic object. If the alias refers to a queue, it must be another local or remote queue, defined at this queue manager, or a clustered alias queue defined on another queue manager. It cannot be another alias queue on this queue manager. If the alias refers to a topic, it must be a topic object defined at this queue manager.

- Syntax diagram
- "Usage notes for DEFINE queues" on page 658
- "Parameter descriptions for DEFINE QUEUE and ALTER QUEUE" on page 659

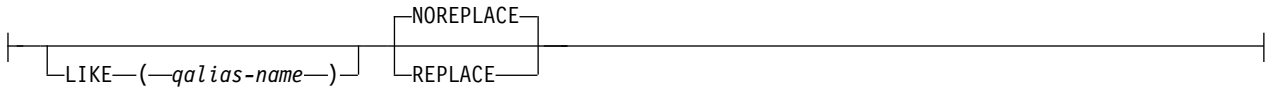
Synonym: DEF QA

Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

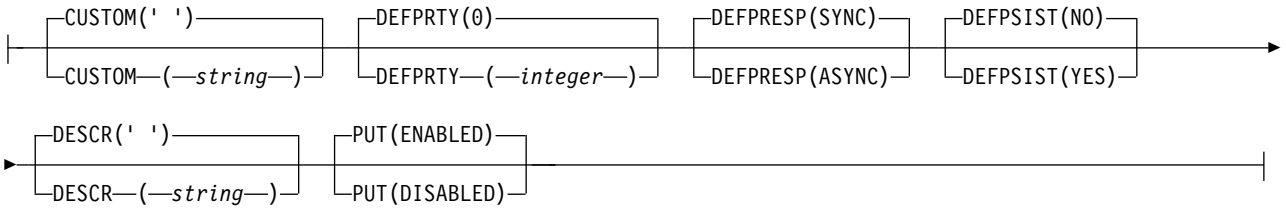
DEFINE QALIAS



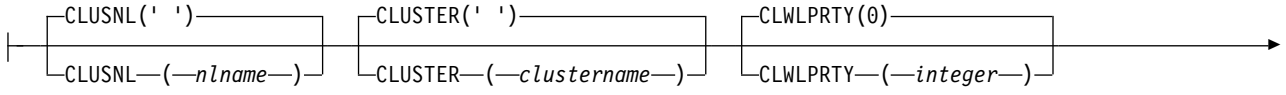
Define attrs:

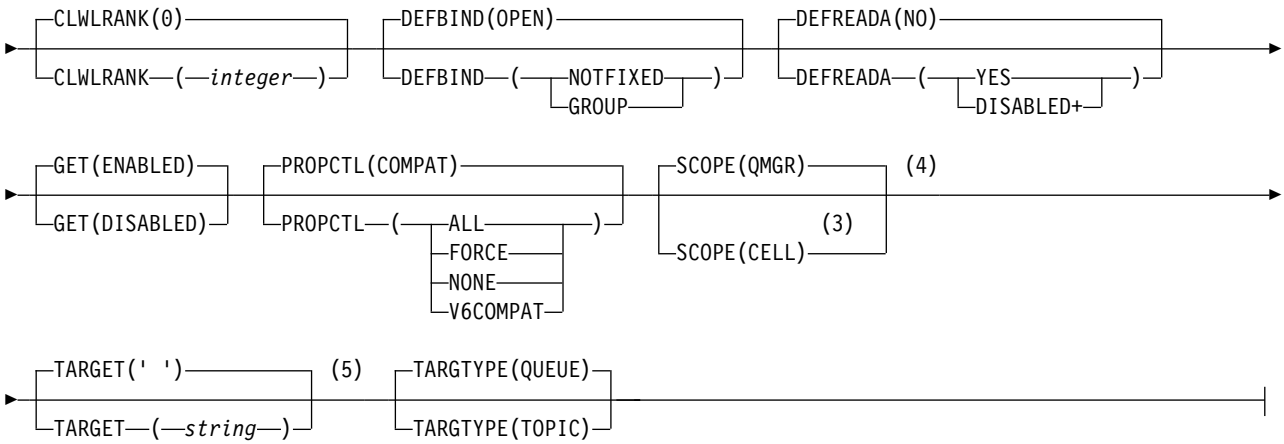


Common queue attrs:



Alias queue attrs:





Notes:

1 Valid only on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on UNIX, Linux, and Windows.

4 Not valid on z/OS.

5 The TARGQ parameter is available for compatibility with previous releases. It is a synonym of TARGET; you cannot specify both parameters.

Related information:

Working with alias queues

DEFINE QLOCAL:

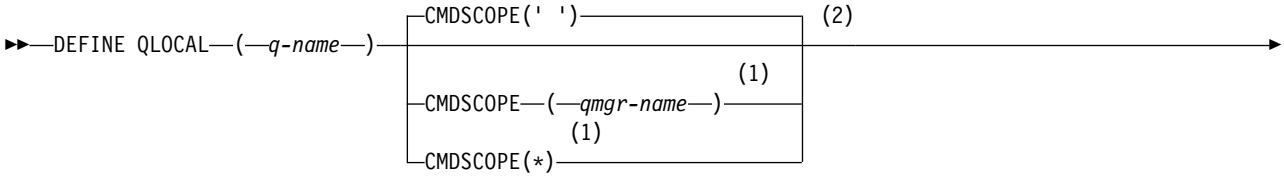
Use **DEFINE QLOCAL** to define a new local queue, and set its parameters.

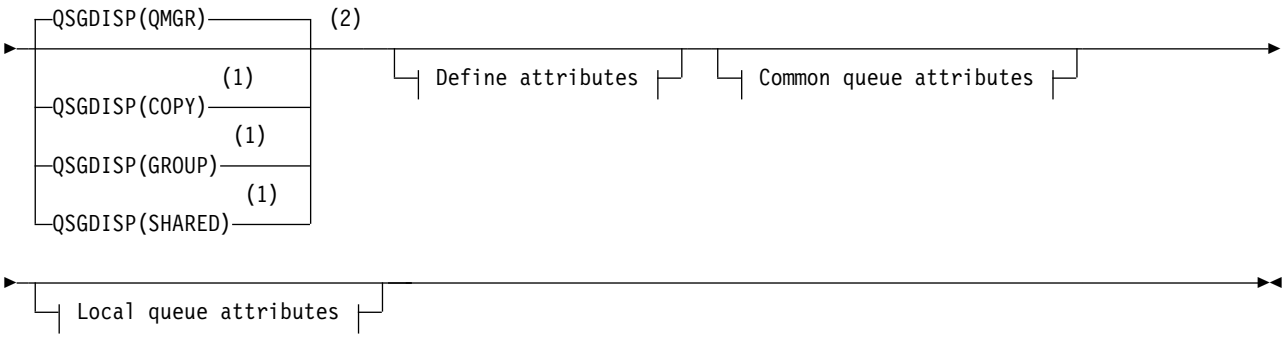
- Syntax diagram
- "Usage notes for DEFINE queues" on page 658
- "Parameter descriptions for DEFINE QUEUE and ALTER QUEUE" on page 659

Synonym: **DEF QL**

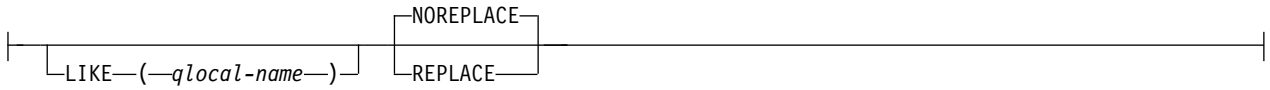
Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

DEFINE QLOCAL

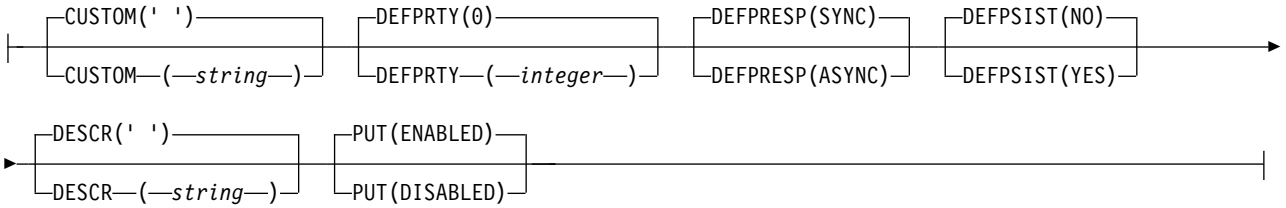




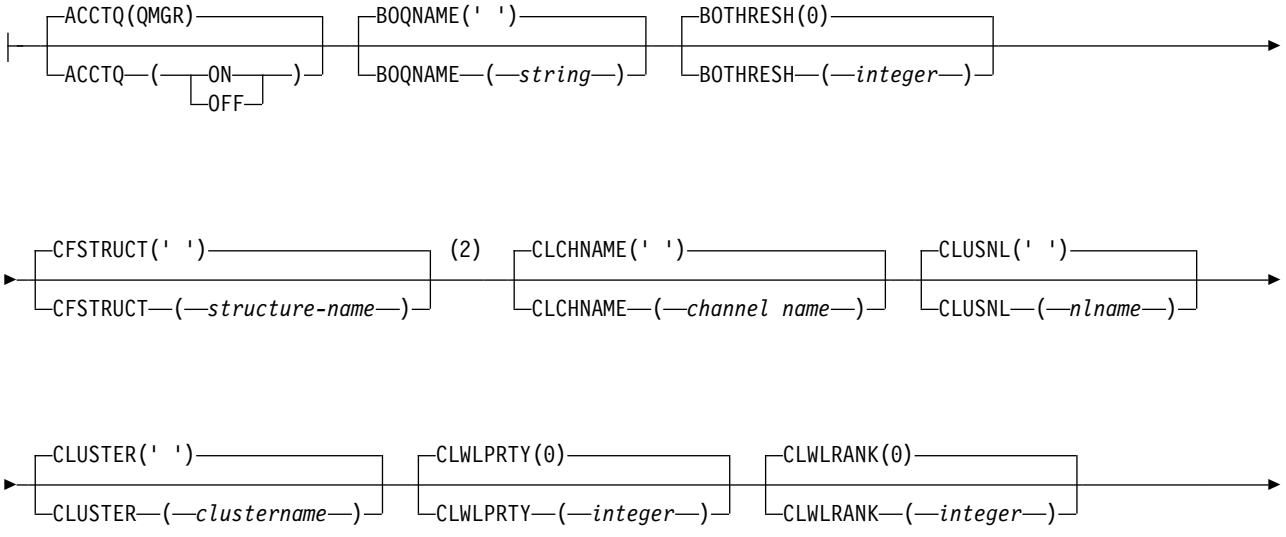
Define attributes:

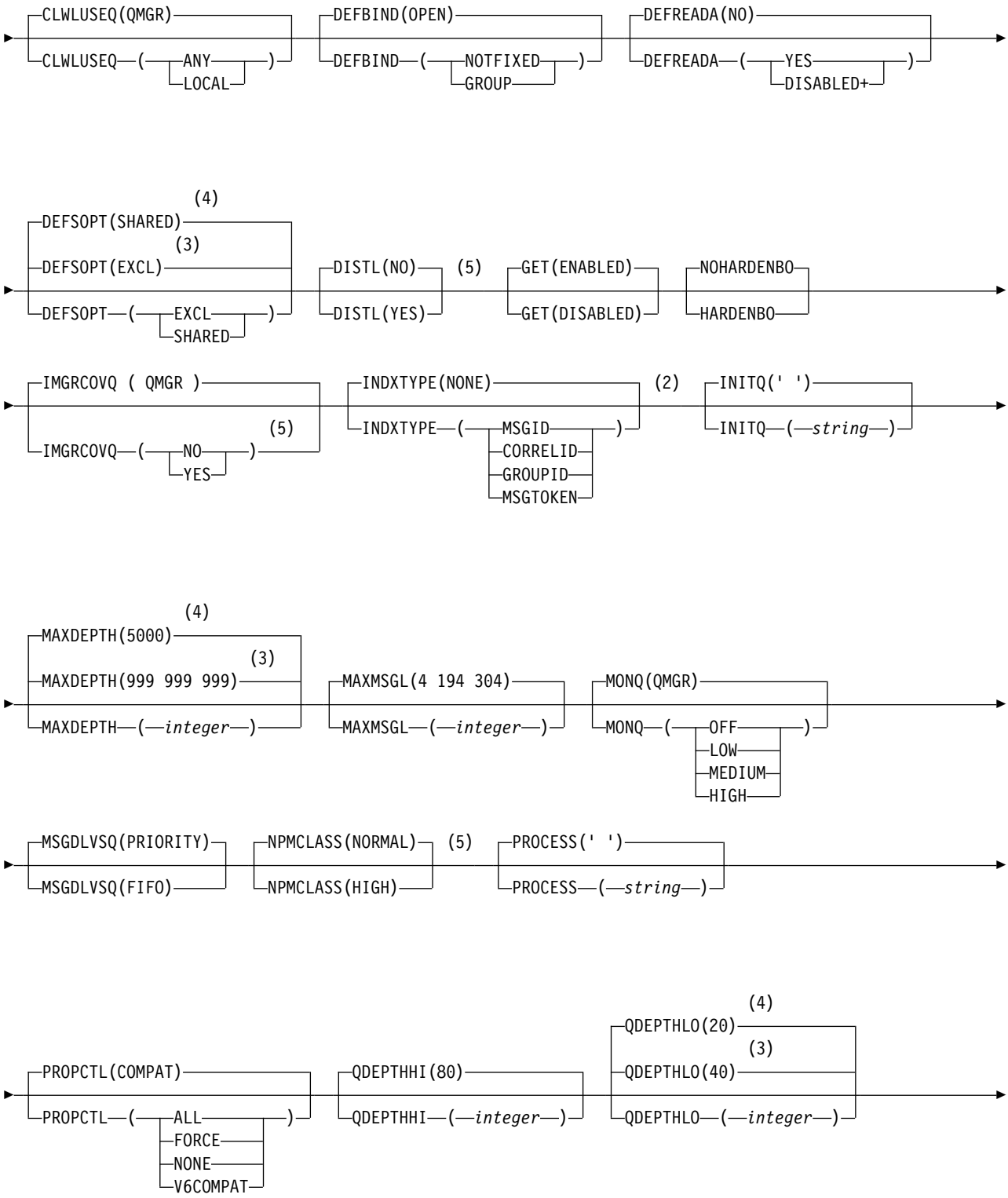


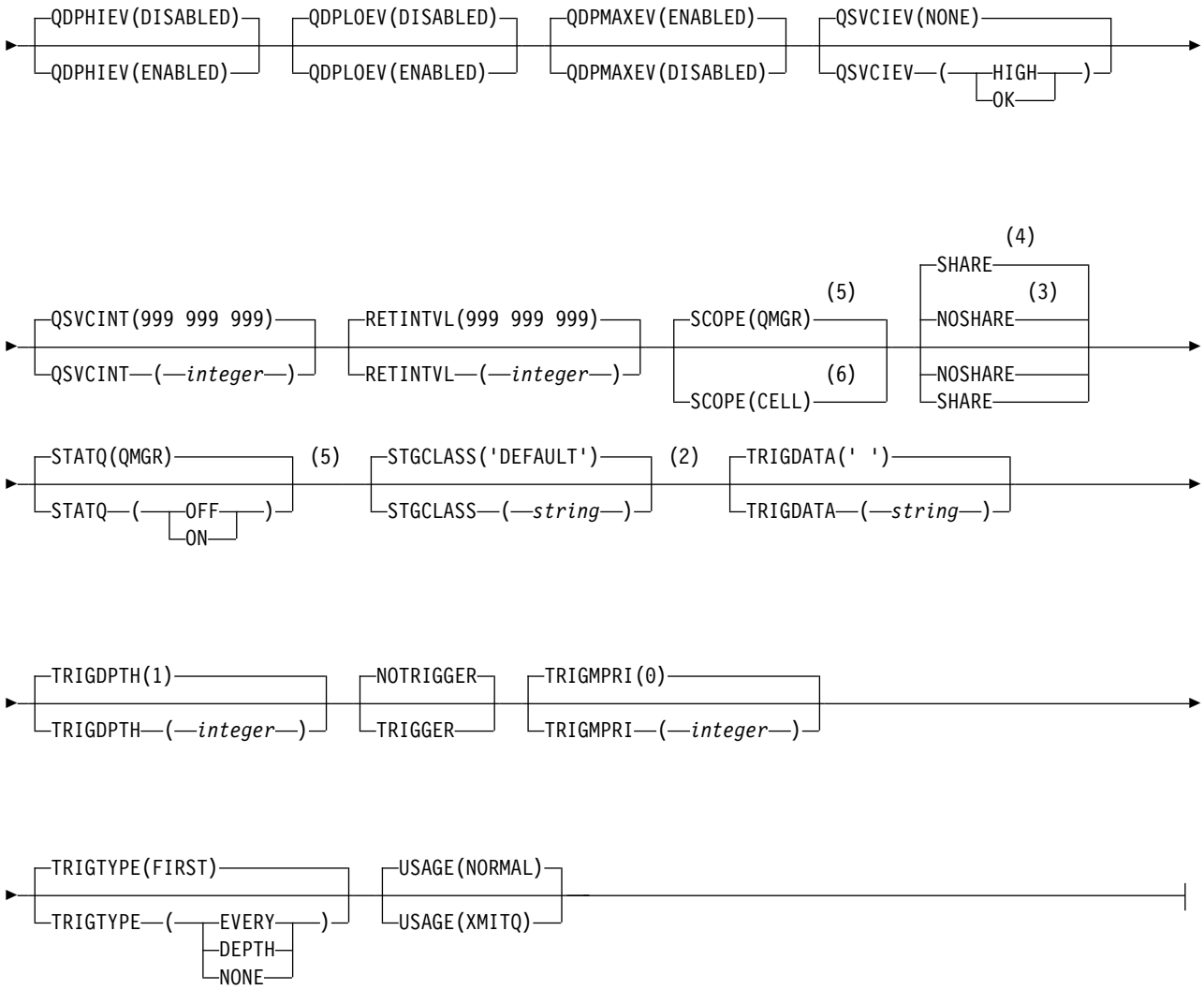
Common queue attributes:



Local queue attributes:




CLWLUSEQ(QMGR)
CLWLUSEQ—(—ANY—)
LOCAL
DEFBIND(OPEN)
DEFBIND—(—NOTFIXED—)
GROUP
DEFREADA(NO)
DEFREADA—(—YES—)
DISABLED+
(4)
DEFSOPT(SHARED)
(3)
DEFSOPT(EXCL)
DEFSOPT—(—EXCL—)
SHARED
DISTL(NO)
DISTL(YES)
(5)
GET(ENABLED)
GET(DISABLED)
NOHARDENBO
HARDENBO
IMGRCOVQ (QMGR)
(5)
IMGRCOVQ—(—NO—)
YES
INDXTYPE(NONE)
INDXTYPE—(—MSGID—)
CORRELID
GROUPID
MSGTOKEN
(2)
INITQ(' ')
INITQ—(—string—)
(4)
MAXDEPTH(5000)
(3)
MAXDEPTH(999 999 999)
MAXDEPTH—(—integer—)
MAXMSGL(4 194 304)
MAXMSGL—(—integer—)
MONQ(QMGR)
MONQ—(—OFF—)
LOW
MEDIUM
HIGH
MSGDLVSQ(PRIORITY)
MSGDLVSQ(FIFO)
NPMCLASS(NORMAL)
NPMCLASS(HIGH)
(5)
PROCESS(' ')
PROCESS—(—string—)
PROPCTL(COMPAT)
PROPCTL—(—ALL—)
FORCE
NONE
V6COMPAT
QDEPTHHI(80)
QDEPTHHI—(—integer—)
(4)
QDEPTHLO(20)
(3)
QDEPTHLO(40)
QDEPTHLO—(—integer—)



Notes:

1 Valid only on z/OS and when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Default for z/OS.

4 Default for Multiplatforms.

5 Not valid on z/OS.

6 Valid only on UNIX, Linux, and Windows.

Related information:
Defining a local queue
Changing local queue attributes

DEFINE QMODEL:

Use **DEFINE QMODEL** to define a new model queue, and set its parameters.

A model queue is not a real queue, but a collection of attributes that you can use when creating dynamic queues with the MQOPEN API call.

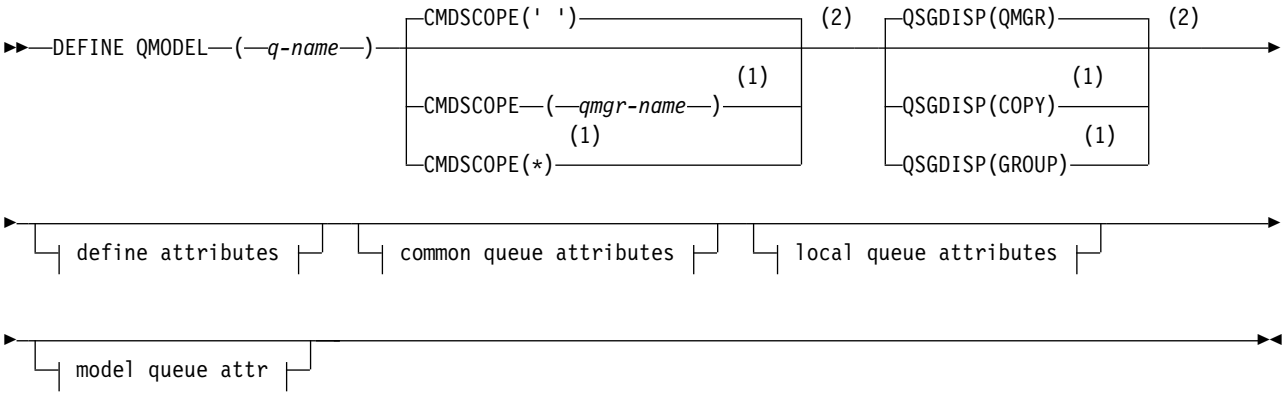
When it has been defined, a model queue (like any other queue) has a complete set of applicable attributes, even if some of these are defaults.

- Syntax diagram
- "Usage notes for DEFINE queues" on page 658
- "Parameter descriptions for DEFINE QUEUE and ALTER QUEUE" on page 659

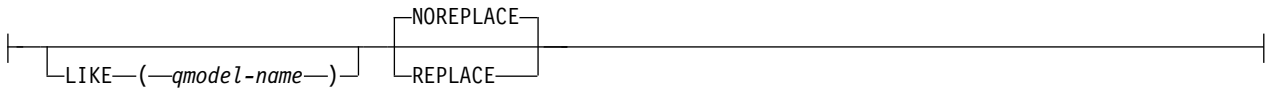
Synonym: **DEF QM**

Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

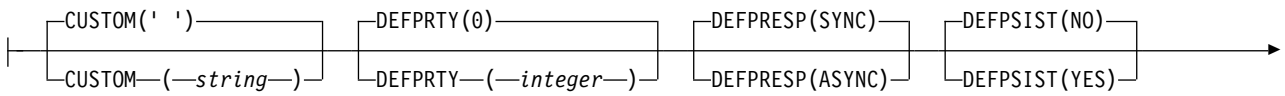
DEFINE QMODEL



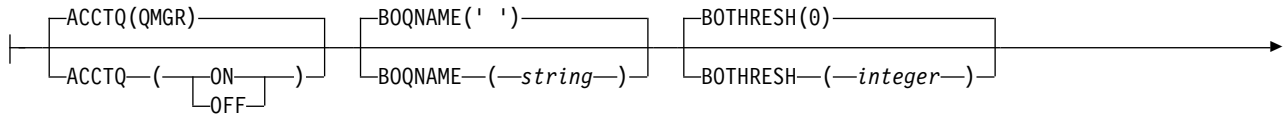
Define attributes:

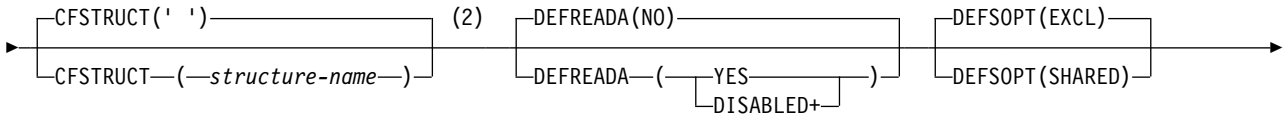


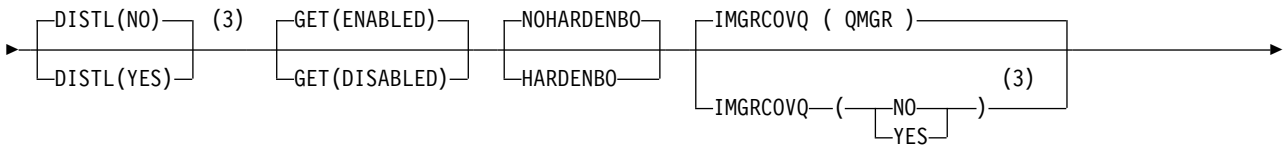
Common queue attributes:

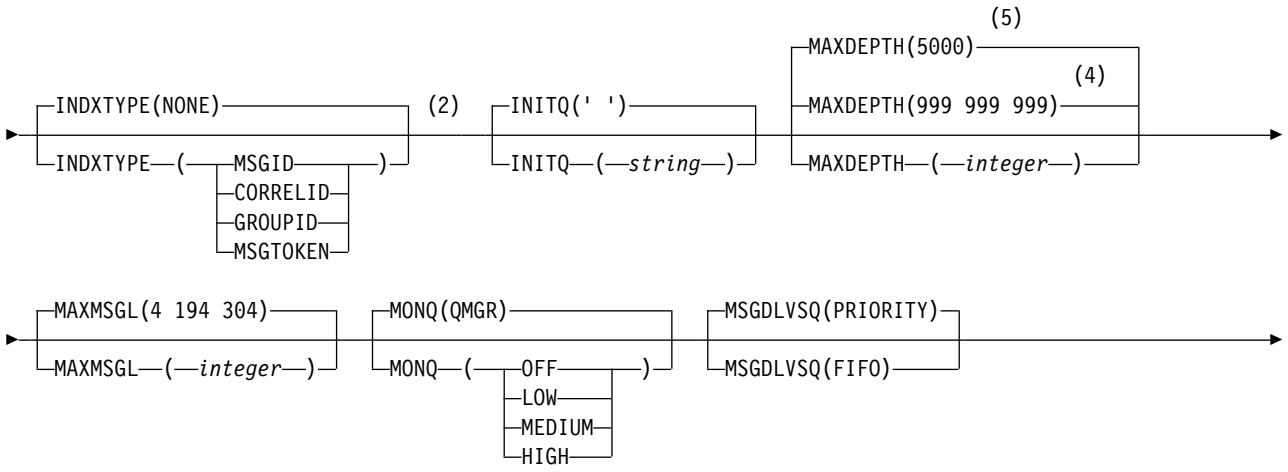


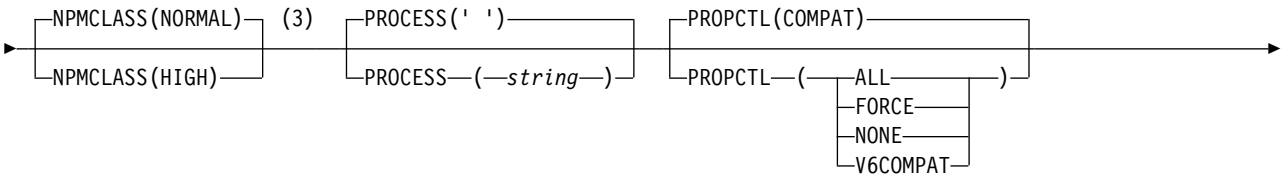
Local queue attributes:

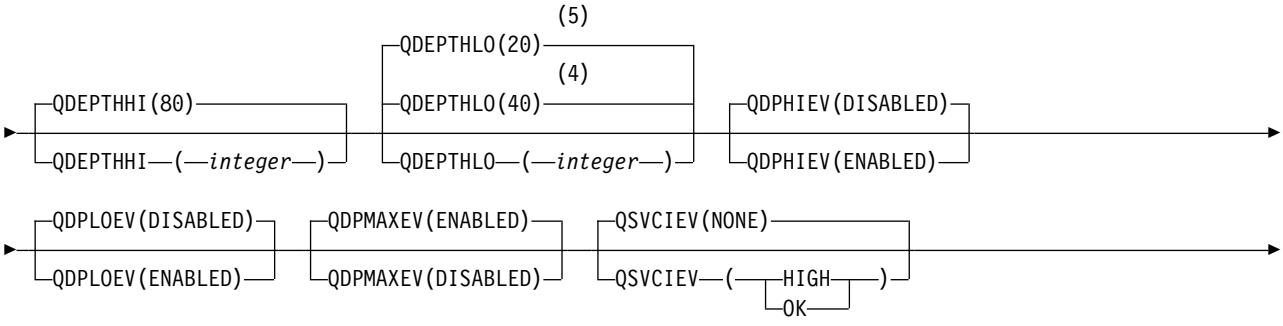


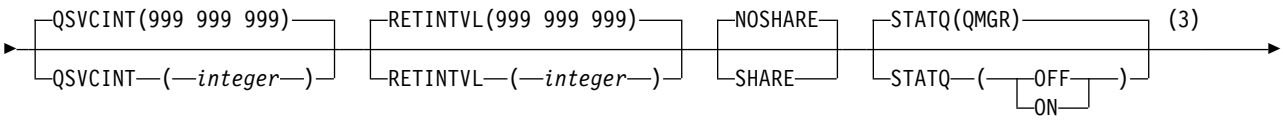


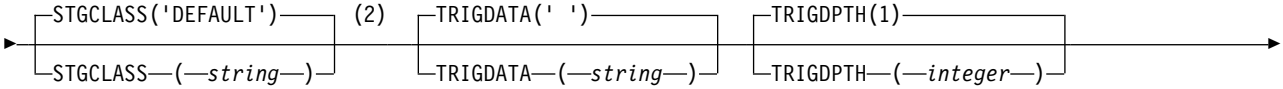


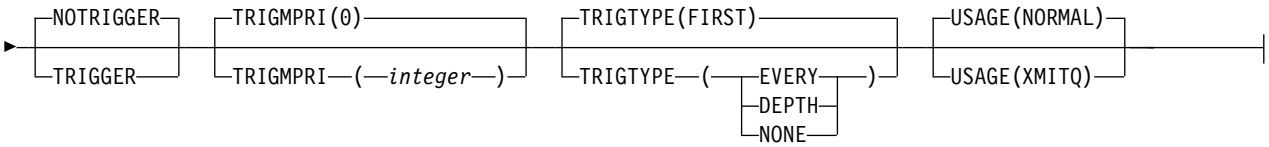












Model queue attr:

Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Used only on z/OS.

3 Not valid on z/OS.

4 Default for z/OS.

5 Default for Multiplatforms.

Related information:

Working with model queues

DEFINE QREMOTE:

Use DEFINE QREMOTE to define a new local definition of a remote queue, a queue manager alias, or a reply-to queue alias, and to set its parameters.

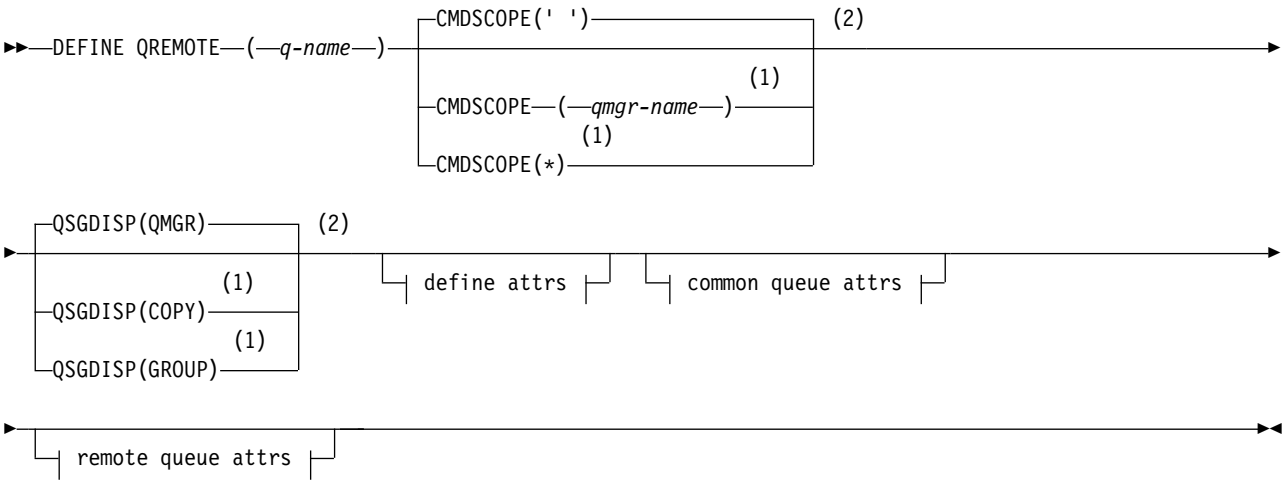
A remote queue is one that is owned by another queue manager that application processes connected to this queue manager need to access.

- Syntax diagram
- "Usage notes for DEFINE queues" on page 658
- "Parameter descriptions for DEFINE QUEUE and ALTER QUEUE" on page 659

Synonym: DEF QR

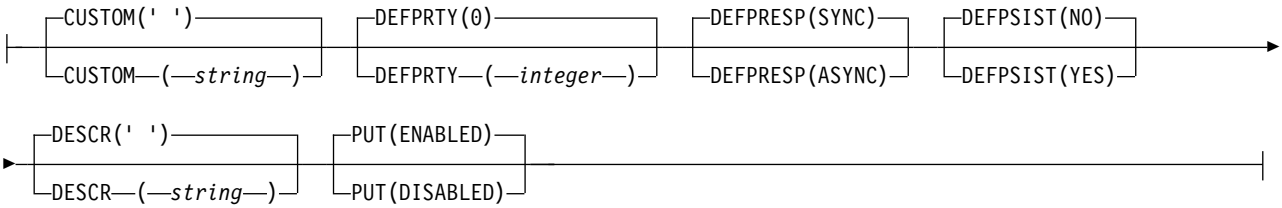
Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

DEFINE QREMOTE

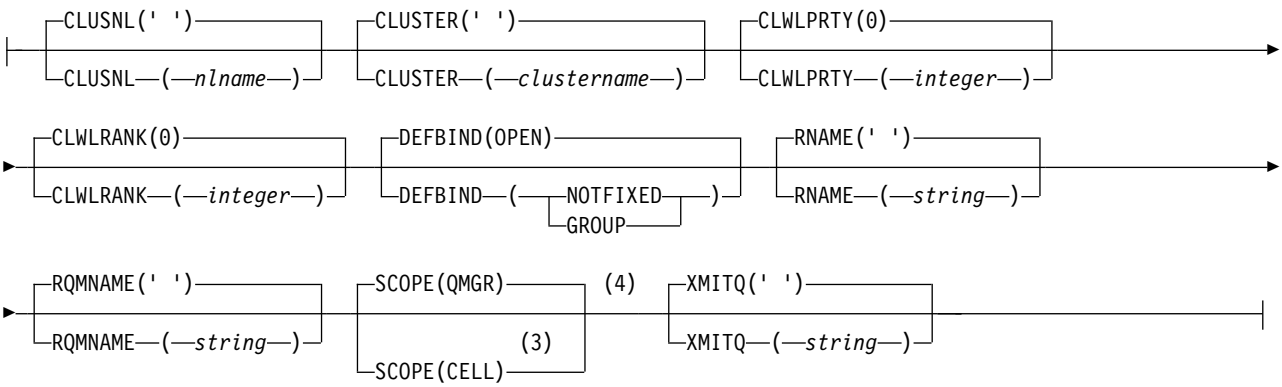


Define attrs:

Common queue attrs:



Remote queue attrs:



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Valid only on UNIX, Linux, and Windows.

4 Not valid on z/OS.

DEFINE SERVICE on Multiplatforms: Multi

Use the MQSC command **DEFINE SERVICE** to define a new IBM MQ service definition, and set its parameters.

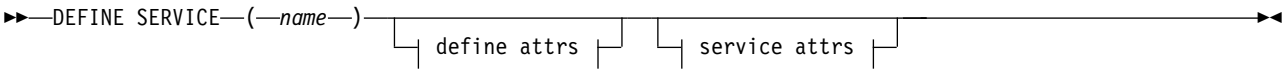
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

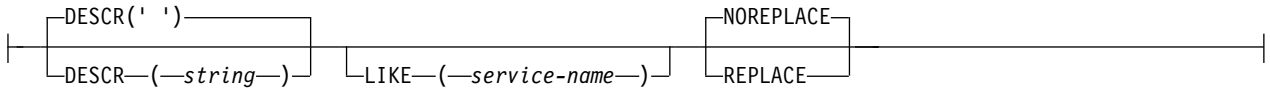
- Syntax diagram
- "Usage notes" on page 692
- "Parameter descriptions for DEFINE SERVICE" on page 692

Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

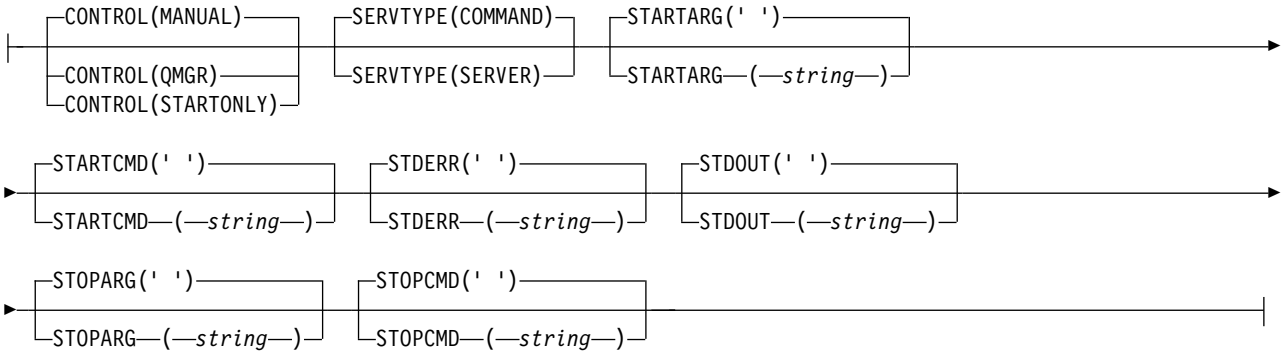
DEFINE SERVICE



Define attrs:



Service attrs:



Usage notes

A service is used to define the user programs that are to be started and stopped when the queue manager is started and stopped. You can also start and stop these programs by issuing the **START SERVICE** and **STOP SERVICE** commands.

Attention: This command allows a user to run an arbitrary command with mqm authority. If granted rights to use this command, a malicious or careless user could define a service which damages your systems or data, for example, by deleting essential files.

For more information about services, see Services.

Parameter descriptions for DEFINE SERVICE

The parameter descriptions apply to the **ALTER SERVICE** and **DEFINE SERVICE** commands, with the following exceptions:

- The **LIKE** parameter applies only to the **DEFINE SERVICE** command.
- The **NOREPLACE** and **REPLACE** parameter applies only to the **DEFINE SERVICE** command.

(service-name)
Name of the IBM MQ service definition (see Rules for naming IBM MQ objects).

The name must not be the same as any other service definition currently defined on this queue manager (unless **REPLACE** is specified).

CONTROL(*string*)
Specifies how the service is to be started and stopped:

MANUAL
The service is not to be started automatically or stopped automatically. It is to be controlled by use of the **START SERVICE** and **STOP SERVICE** commands.

QMGR
The service being defined is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY
The service is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

DESCR(*string*)
Plain-text comment. It provides descriptive information about the service when an operator issues the **DISPLAY SERVICE** command (see "DISPLAY SERVICE on Multiplatforms" on page 912).

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

LIKE(*service-name*)
The name of a service the parameters of which are used to model this definition.

This parameter applies only to the **DEFINE SERVICE** command.

If this field is not completed, and you do not complete the parameter fields related to the command, the values are taken from the default definition for services on this queue manager. Not completing this parameter is equivalent to specifying:

```
LIKE(SYSTEM.DEFAULT.SERVICE)
```

A default service is provided but it can be altered by the installation of the default values required. See Rules for naming IBM MQ objects.

REPLACE and NOREPLACE
Whether the existing definition is to be replaced with this one.

This parameter applies only to the **DEFINE SERVICE** command.

REPLACE
The definition must replace any existing definition of the same name. If a definition does not exist, one is created.

NOREPLACE
The definition should not replace any existing definition of the same name.

SERVTYPE
Specifies the mode in which the service is to run:

COMMAND
A command service object. Multiple instances of a command service object can be executed concurrently. You cannot monitor the status of command service objects.

SERVER
A server service object. Only one instance of a server service object can be executed at a time. The status of server service objects can be monitored using the **DISPLAY SVSTATUS** command.

STARTARG(*string*)
Specifies the arguments to be passed to the user program at queue manager startup.

STARTCMD(*string*)
Specifies the name of the program which is to run. You must specify a fully qualified path name to the executable program.

STDERR(*string*)
Specifies the path to a file to which the standard error (stderr) of the service program is redirected. If the file does not exist when the service program is started, the file is created. If this value is blank then any data written to stderr by the service program is discarded.

STDOUT(*string*)
Specifies the path to a file to which the standard output (stdout) of the service program is redirected. If the file does not exist when the service program is started, the file is created. If this value is blank then any data written to stdout by the service program is discarded.

STOPARG(*string*)
Specifies the arguments to be passed to the stop program when instructed to stop the service.

STOPCMD(*string*)
Specifies the name of the executable program to run when the service is requested to stop. You must specify a fully qualified path name to the executable program.

Replaceable inserts can be used for any of the **STARTCMD**, **STARTARG**, **STOPCMD**, **STOPARG**, **STDOUT** or **STDERR** strings, for more information, see Replaceable inserts on service definitions.

Related reference:

"ALTER SERVICE on Multiplatforms" on page 526
Use the MQSC command **ALTER SERVICE** to alter the parameters of an existing IBM MQ service definition.

"DISPLAY SVSTATUS on Multiplatforms" on page 931
Use the MQSC command **DISPLAY SVSTATUS** to display status information for one or more services. Only services with a **SERVTYPE** of SERVER are displayed.

"START SERVICE on Multiplatforms" on page 1039
Use the MQSC command **START SERVICE** to start a service. The identified service definition is started within the queue manager and inherits the environment and security variables of the queue manager.

"STOP SERVICE on Multiplatforms" on page 1057
Use the MQSC command **STOP SERVICE** to stop a service.

Related information:

Working with services

Defining a service object

Examples of using service objects

DEFINE STGCLASS on z/OS: z/OS

Use the MQSC command DEFINE STGCLASS to define a storage class to page set mapping.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

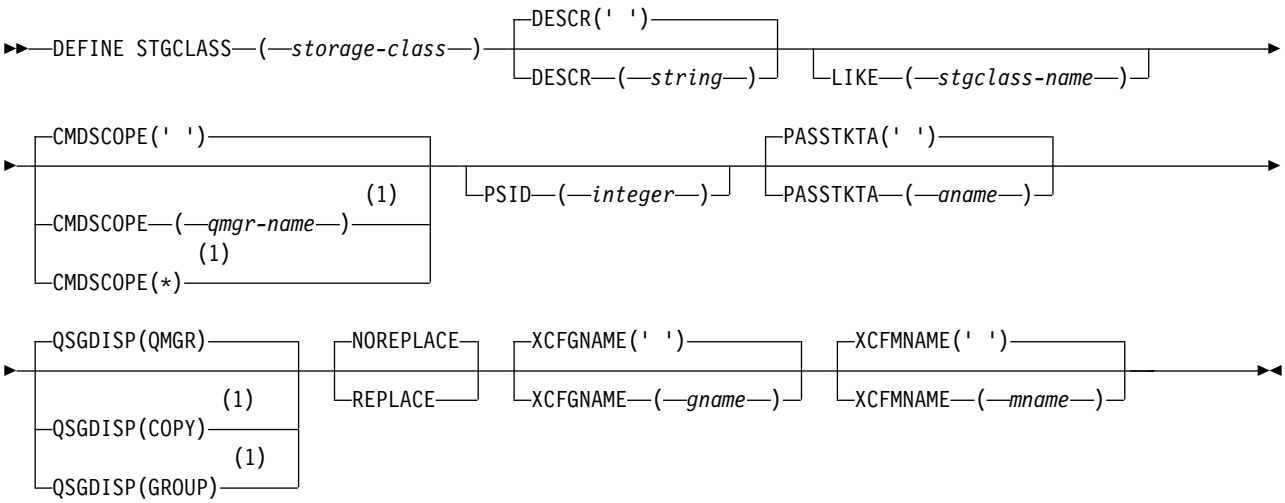
You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for DEFINE STGCLASS" on page 695
- "Parameter descriptions for DEFINE STGCLASS" on page 695

Synonym: DEF STC

Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See “How to read railroad diagrams” on page 187.

DEFINE STGCLASS



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

Usage notes for DEFINE STGCLASS

1. The resultant values of XCFGNAME and XCFMNAME must either both be blank or both be nonblank.
2. You can change a storage class only if it is not being used by any queues. To determine whether any queues are using the storage class, you can use the following command:

   ```
   DISPLAY QUEUE(*) STGCLASS(ABC) PSID(n)
   ```

   where 'ABC' is the name of the storage class, and *n* is the identifier of the page set that the storage class is associated with.

   This command gives a list of all queues that reference the storage class, and have an active association to page set *n*, and therefore identifies the queues that are actually preventing the change to the storage class. If you do not specify the PSID, you just get a list of queues that are potentially stopping the change.

   See the DISPLAY QUEUE PSID command for more information about active association of a queue to a page set.

Parameter descriptions for DEFINE STGCLASS

(storage-class)

Name of the storage class.

This name is one to 8 characters. The first character is in the range A through Z; subsequent characters are A through Z or 0 through 9.

Note: Exceptionally, certain all numeric storage class names are allowed, but are reserved for the use of IBM service personnel.

The storage class must not be the same as any other storage class currently defined on this queue manager.

CMDSCOPE

This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command runs on the queue manager on which it was entered.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

DESCR(*description*)

Plain-text comment. It provides descriptive information about the object when an operator issues the DISPLAY STGCLASS command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager

LIKE(*stgclass-name*)

The name of an object of the same type, with parameters that are used to model this definition.

If this field is not completed, and you do not complete the parameter fields related to the command, the values are taken from the default definition for this object.

Not completing this parameter is equivalent to specifying:

```
LIKE(SYSTEMST)
```

This default storage class definition can be altered by your installation to the default values required.

The queue manager searches for an object with the name you specify and a disposition of QMGR or COPY. The disposition of the LIKE object is not copied to the object you are defining.

Note:

1. QSGDISP (GROUP) objects are not searched.
2. LIKE is ignored if QSGDISP(COPY) is specified.

PASSTKTA(*application name*)

The application name that is passed to RACF when authenticating the PassTicket specified in the MQIIH header.

PSID(*integer*)

The page set identifier that this storage class is to be associated with.

Note: No check is made that the page set has been defined; an error is raised only when you try to put a message to a queue that specifies this storage class (MQRC_PAGESET_ERROR).

The string consists of two numeric characters, in the range 00 through 99. See “DEFINE PSID on z/OS” on page 656.

QSGDISP

Specifies the disposition of the object in the group.

QSGDISP	DEFINE
COPY	The object is defined on the page set of the queue manager that executes the command using the QSGDISP(GROUP) object of the same name as the 'LIKE' object.
GROUP	The object definition resides in the shared repository but only if the queue manager is in a queue-sharing group. If the definition is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to attempt to make or refresh local copies on page set zero: `DEFINE STGCLASS(storage-class)` `REPLACE QSGDISP(COPY)` The DEFINE for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.
PRIVATE	Not permitted.
QMGR	The object is defined on the page set of the queue manager that executes the command.

REPLACE and NOREPLACE

Whether the existing definition, and with the same disposition, is to be replaced with this one. Any object with a different disposition is not changed.

REPLACE

The definition replaces any existing definition of the same name. If a definition does not exist, one is created.

If you use the REPLACE option, all queues that use this storage class must be temporarily altered to use another storage class while the command is issued.

NOREPLACE

The definition does not replace any existing definition of the same name.

XCFGNAME(*group name* **)**

If you are using the IMS bridge, this name is the name of the XCF group to which the IMS system belongs. (This name is the group name specified in the IMS parameter list.)

This name is 1 - 8 characters. The first character is in the range A through Z; subsequent characters are A through Z or 0 - 9.

XCFMNAME(*member name* **)**

If you are using the IMS bridge, this name is the XCF member name of the IMS system within the XCF group specified in XCFGNAME. (This name is the member name specified in the IMS parameter list.)

This name is 1 - 16 characters. The first character is in the range A through Z; subsequent characters are A through Z or 0 - 9.

DEFINE SUB:

Use **DEFINE SUB** to allow an existing application to participate in a publish/subscribe application by allowing the administrative creation of a durable subscription.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

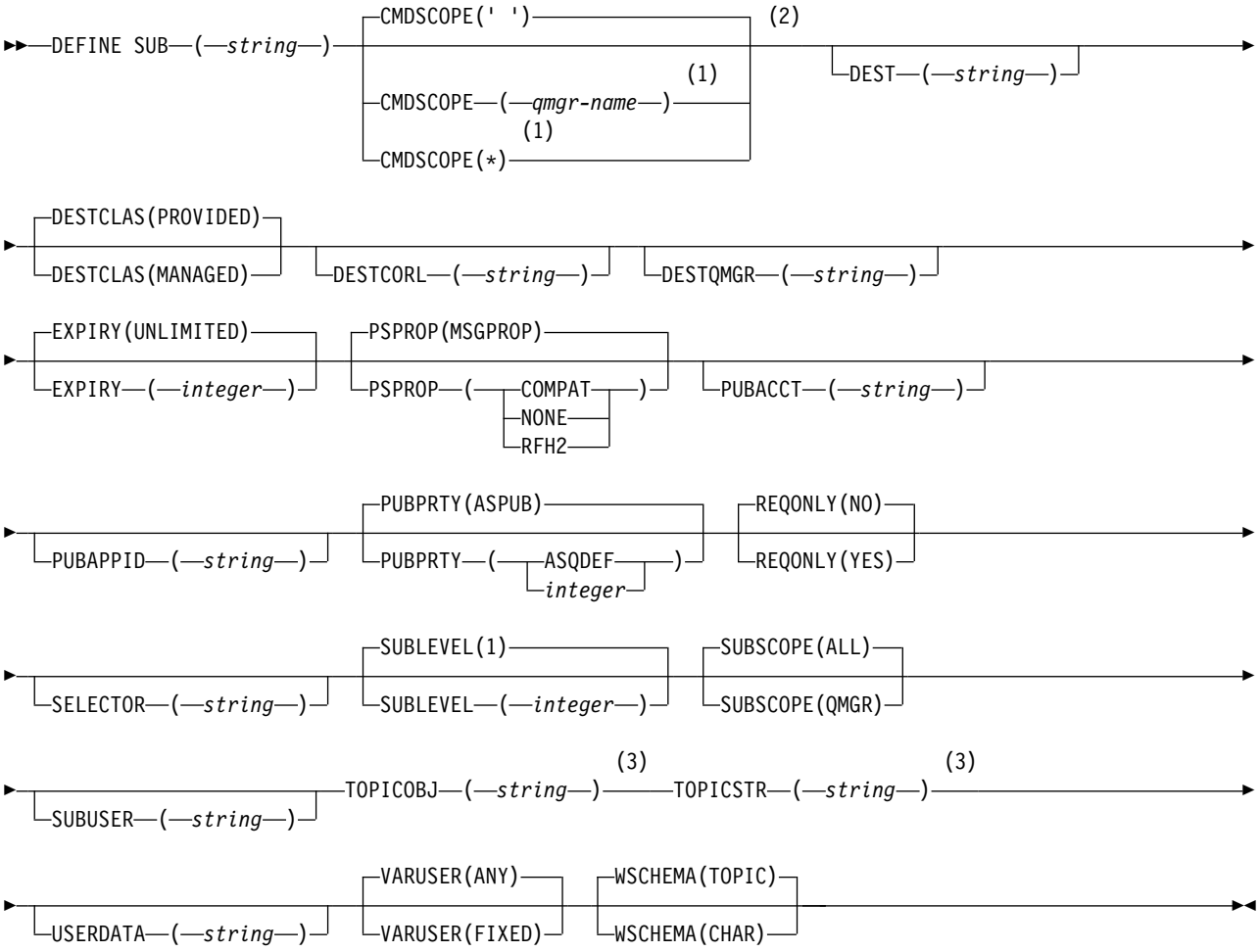
z/OS You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for DEFINE SUB" on page 699
- "Parameter descriptions for DEFINE SUB" on page 699

Synonym: **DEF SUB**

Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

DEFINE SUB



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 At least one of **TOPICSTR** and **TOPICOBJ** must be present on **DEFINE**.

Usage notes for DEFINE SUB

- You must provide the following information when you define a subscription:
  - The **SUBNAME**
  - A destination for messages
  - The topic to which the subscription applies
- You can provide the topic name in the following ways:

  TOPICSTR
  The topic is fully specified as the **TOPICSTR** attribute.

  TOPICOBJ
  The topic is obtained from the **TOPICSTR** attribute of the named topic object. The named topic object is retained as the **TOPICOBJ** attribute of the new subscription. This method is provided to help you enter long topic strings through an object definition.

  TOPICSTR and TOPICOBJ
  The topic is obtained by the concatenation of the **TOPICSTR** attribute of the named topic object and the value of **TOPICSTR** (see the MQSUB API specification for concatenation rules). The named topic object is retained as the **TOPICOBJ** attribute of the new subscription.
- If you specify **TOPICOBJ**, the parameter must name an IBM MQ topic object. The existence of the named topic object is checked at the time the command processes.
- You can explicitly specify the destination for messages through the use of the **DEST** and **DESTQMGR** keywords.

  You must provide the **DEST** keyword for the default option of **DESTCLAS(PROVIDED)**; if you specify **DESTCLAS(MANAGED)**, a managed destination is created on the local queue manager, so you cannot specify either the **DEST** or **DESTQMGR** attribute. For more information, see Managed queues and publish/subscribe.
- z/OS On z/OS only, at the time the **DEF SUB** command processes, no check is performed that the named **DEST** or **DESTQMGR** exists.

  These names are used at publishing time as the `ObjectName` and `ObjectQMgrName` for an MQOPEN call. These names are resolved according to the IBM MQ name resolution rules.
- When a subscription is defined administratively using MQSC or PCF commands, the selector is not validated for invalid syntax. The **DEFINE SUB** command has no equivalent to the MQRC_SELECTION_NOT_AVAILABLE reason code that can be returned by the MQSUB API call.
- **TOPICOBJ**, **TOPICSTR**, **WSCHEMA**, **SELECTOR**, **SUBSCOPE**, and **DESTCLAS** cannot be changed with **DEFINE REPLACE**.
- When a publication has been retained, it is no longer available to subscribers at higher levels because it is republished at PubLevel 1.
- Successful completion of the command does not mean that the action completed. To check for true completion, see the DEFINE SUB step in Checking that async commands for distributed networks have finished.

Parameter descriptions for DEFINE SUB

(string)
A mandatory parameter. Specifies the unique name for this subscription, see **SUBNAME** property.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

' '
The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

*
The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of setting this value is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

DEST(*string*)

The destination for messages published to this subscription; this parameter is the name of a queue.

DESTCLAS

System managed destination.

PROVIDED
The destination is a queue.

MANAGED
The destination is managed.

DESTCORL(*string*)

The **CorrelId** used for messages published to this subscription.

A blank value (default) results in a system generated correlation identifier being used.

If set to ' 00000000000000000000000000000000000000000000000000000000 ' (48 zeros) the **CorrelId** set by the publishing application will be maintained in the copy of the message delivered to the subscription, unless messages are propagated across a publish/subscribe hierarchy.

DESTQMGR(*string*)

The destination queue manager for messages published to this subscription. You must define the channels to the remote queue manager, for example, the XMITQ, and a sender channel. If you do not, messages do not arrive at the destination.

EXPIRY

The time to expiry of the subscription object from the creation date and time.

(*integer*)
The time to expiry, in tenths of a second, from the creation date and time.

UNLIMITED
There is no expiry time. This is the default option supplied with the product.

LIKE(*subscription-name*)

The name of a subscription, the parameters of which are used as a model for this definition.

This parameter applies only to the **DEFINE SUB** command.

If this field is not supplied, and you do not complete the parameter fields related to the command, the values are taken from the default definition for subscriptions on this queue manager. Not completing this parameter is equivalent to specifying:

LIKE (SYSTEM.DEFAULT.SUB)

PSPROP
The manner in which publish subscribe related message properties are added to messages sent to this subscription.

NONE Do not add publish subscribe properties to the message.

COMPAT Publish/subscribe properties are added within an MQRFH version 1 header unless the message was published in PCF format.

MSGPROP
Publish/subscribe properties are added as message properties.

RFH2 Publish/subscribe properties are added within an MQRFH version 2 header.

PUBACCT(*string*)
Accounting token passed by the subscriber, for propagation into messages published to this subscription in the AccountingToken field of the MQMD.

PUBAPPID(*string*)
Identity data passed by the subscriber, for propagation into messages published to this subscription in the ApplIdentityData field of the MQMD.

PUBPRTY
The priority of the message sent to this subscription.

ASPUB Priority of the message sent to this subscription is taken from the priority supplied in the published message.

ASQDEF Priority of the message sent to this subscription is taken from the default priority of the queue defined as a destination.

(integer)
An integer providing an explicit priority for messages published to this subscription.

REPLACE and NOREPLACE
This parameter controls whether any existing definition is to be replaced with this one.

REPLACE
The definition replaces any existing definition of the same name. If a definition does not exist, one is created.

You cannot change **TOPICOBJ**, **TOPICSTR**, **WSCHEMA**, **SELECTOR**, **SUBSCOPE**, or **DESTCLAS** with **DEFINE REPLACE**.

NOREPLACE
The definition does not replace any existing definition of the same name.

REQONLY
Indicates whether the subscriber polls for updates using the MQSUBRQ API call, or whether all publications are delivered to this subscription.

NO All publications on the topic are delivered to this subscription.

YES Publications are only delivered to this subscription in response to an MQSUBRQ API call.

This parameter is equivalent to the subscribe option MQSO_PUBLICATIONS_ON_REQUEST.

SELECTOR(*string*)
A selector that is applied to messages published to the topic.

SUBLEVEL(*integer*)
The level within the subscription hierarchy at which this subscription is made. The range is zero through 9.

SUBSCOPE
Determines whether this subscription is forwarded to other queue managers, so that the subscriber receives messages published at those other queue managers.

ALL The subscription is forwarded to all queue managers directly connected through a publish/subscribe collective or hierarchy.

QMGR The subscription forwards messages published on the topic only within this queue manager.

Note: Individual subscribers can only restrict **SUBSCOPE**. If the parameter is set to ALL at topic level, then an individual subscriber can restrict it to QMGR for this subscription. However, if the parameter is set to QMGR at topic level, then setting an individual subscriber to ALL has no effect.

SUBNAME
The application's unique subscription name that is associated with the handle. This parameter is relevant only for handles of subscriptions to topics. It is not returned for other handles. Not all subscriptions will have a subscription name.

SUBUSER(*string*)
Specifies the user ID that is used for security checks that are performed to ensure that publications can be put to the destination queue associated with the subscription. This ID is either the user ID associated with the creator of the subscription or, if subscription takeover is permitted, the user ID that last took over the subscription. The length of this parameter must not exceed 12 characters.

TOPICOBJ(*string*)
The name of a topic object used by this subscription.

TOPICSTR(*string*)
Specifies a fully qualified topic name, or a topic set using wildcard characters for the subscription.

USERDATA(*string*)
Specifies the user data associated with the subscription. The string is a variable length value that can be retrieved by the application on an MQSUB API call and passed in a message sent to this subscription as a message property. The **USERDATA** is stored in the RFH2 header in the mqps folder with the key Sud.

V 9.0.2 V 9.0.0.2 An IBM MQ classes for JMS application can retrieve the subscription user data from the message by using the constant JMS_IBM_SUBSCRIPTION_USER_DATA. For more information, see Retrieval of user subscription data.

VARUSER
Specifies whether a user other than the subscription creator can connect to and take over ownership of the subscription.

ANY Any user can connect to and takeover ownership of the subscription.

FIXED Takeover by another USERID is not permitted.

WSCHEMA
The schema to be used when interpreting any wildcard characters in the topic string.

CHAR Wildcard characters represent portions of strings.

TOPIC Wildcard characters represent portions of the topic hierarchy.

Related information:
Defining an administrative subscription
Changing local subscription attributes
Copying a local subscription definition

DEFINE TOPIC:

Use **DEFINE TOPIC** to define a new IBM MQ administrative topic in a topic tree, and set its parameters.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

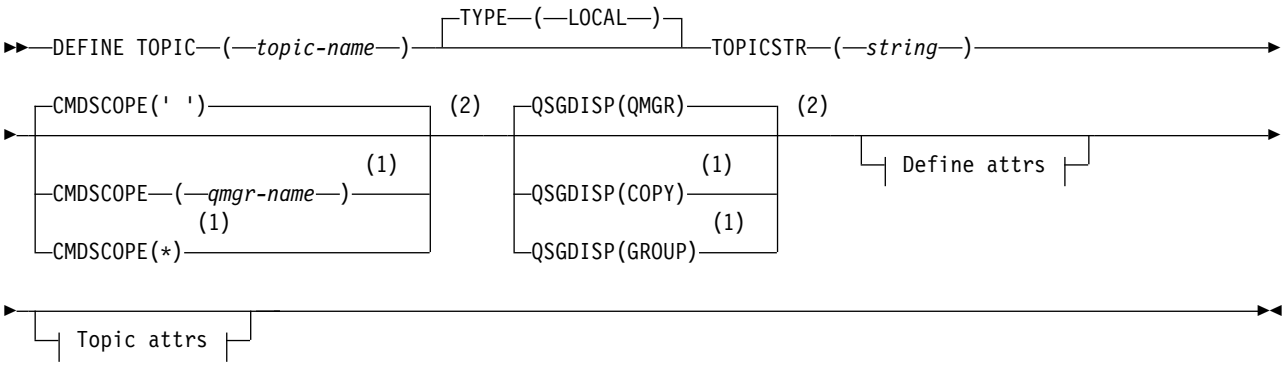
z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for DEFINE TOPIC" on page 704
- "Parameter descriptions for DEFINE TOPIC" on page 705

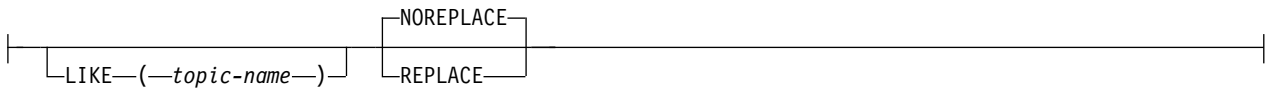
Synonym: DEF TOPIC

Values shown above the main line in the railroad diagram are the defaults supplied with IBM MQ, but your installation might have changed them. See "How to read railroad diagrams" on page 187.

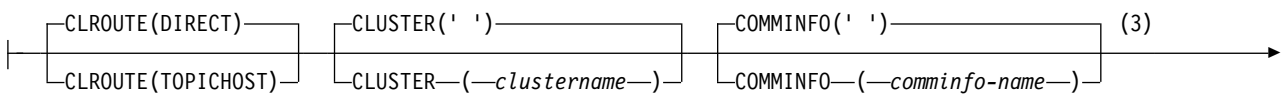
DEFINE TOPIC

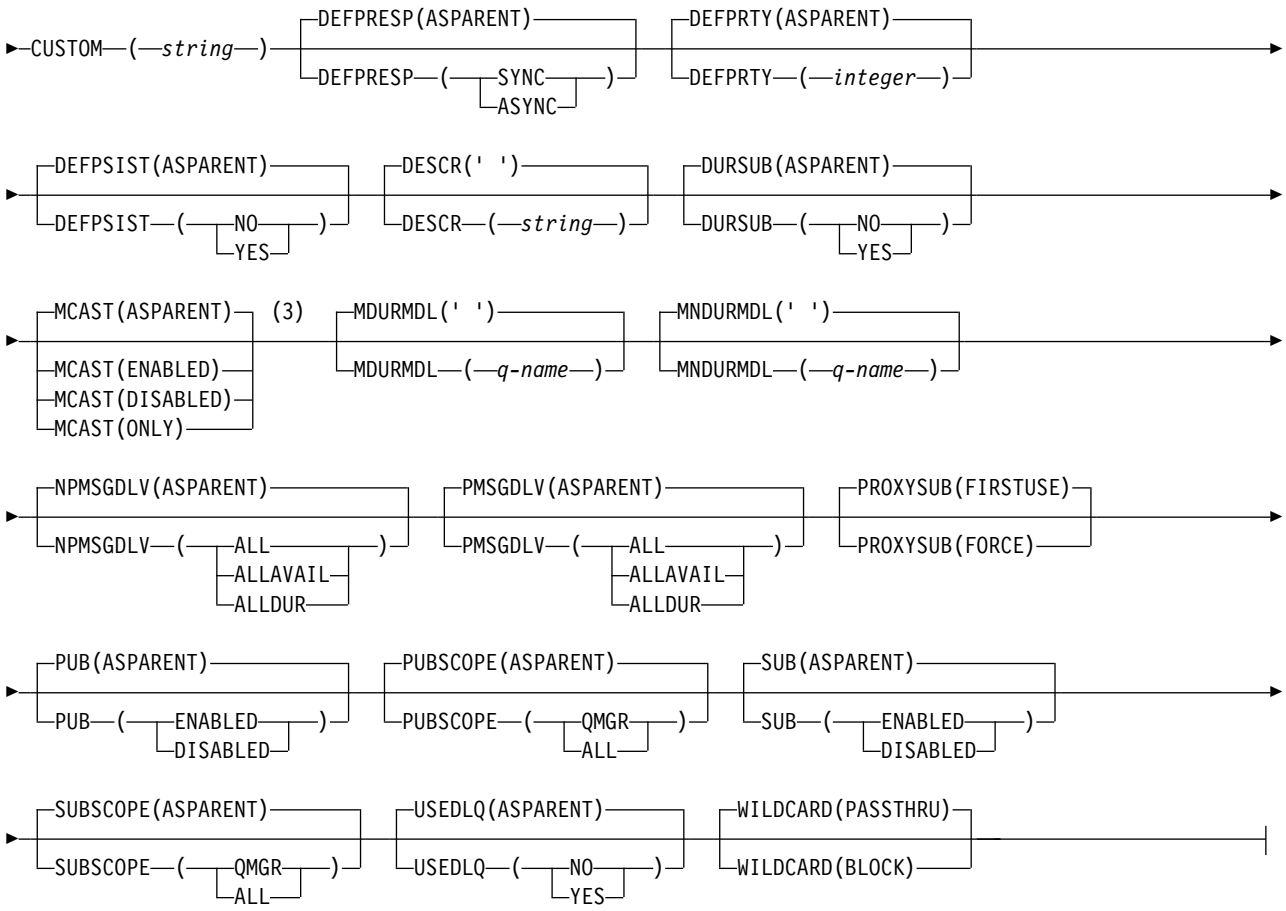


Define attrs:



Topic attrs:





Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Not valid on z/OS.

Usage notes for DEFINE TOPIC

- When an attribute has the value ASPARENT, the value is taken from the setting of the first parent administrative node that is found in the topic tree. Administered nodes are based on either locally defined topic objects or remotely defined cluster topics when participating in a publish/subscribe cluster. If the first parent topic object also has the value ASPARENT, the next object is looked for. If every object that is found, when looking up the tree, uses ASPARENT, the values are taken from the SYSTEM.BASE.TOPIC, if it exists. If SYSTEM.BASE.TOPIC does not exist, the values are the same as the values supplied with IBM MQ in the definition of the SYSTEM.BASE.TOPIC.
- The ASPARENT attribute is applied at each queue manager in the cluster collective by inspecting the set of local definitions and cluster definitions that is visible in the queue manager at the time.
- When a publication is sent to multiple subscribers, the attributes used from the topic object are used consistently for all subscribers that receive the publication. For example, inhibiting publication on a topic is applied for the next application MQPUT to the topic. A publication that is in progress to multiple subscribers completes to all subscribers. This publication does not take note of a change that has happened, part of the way through, to any attribute on the topic.
- Successful completion of the command does not mean that the action completed. To check for true completion, see the DEFINE TOPIC step in Checking that async commands for distributed networks have finished.

Parameter descriptions for DEFINE TOPIC

(topic-name)

Name of the IBM MQ topic definition (see Rules for naming IBM MQ objects). The maximum length is 48 characters.

The name must not be the same as any other topic definition currently defined on this queue manager (unless REPLACE is specified).

CLROUTE

The routing behavior to use for topics in the cluster defined by the **CLUSTER** parameter.

DIRECT

When you configure a direct routed clustered topic on a queue manager, all queue managers in the cluster become aware of all other queue managers in the cluster. When performing publish and subscribe operations, each queue manager can connect direct to any other queue manager in the cluster.

TOPICHOST

When you use topic host routing, all queue managers in the cluster become aware of the cluster queue managers that host the routed topic definition (that is, the queue managers on which you have defined the topic object). When performing publish and subscribe operations, queue managers in the cluster connect only to these topic host queue managers, and not directly to each other. The topic host queue managers are responsible for routing publications from queue managers on which publications are published to queue managers with matching subscriptions.

After a topic object has been clustered (through setting the **CLUSTER** property) you cannot change the value of the **CLROUTE** property. The object must be un-clustered (**CLUSTER** set to `' '`) before you can change the value. Un-clustering a topic converts the topic definition to a local topic, which results in a period during which publications are not delivered to subscriptions on remote queue managers; this should be considered when performing this change. See The effect of defining a non-cluster topic with the same name as a cluster topic from another queue manager. If you try to change the value of the **CLROUTE** property while it is clustered, the system generates an `MQRCCF_CLROUTE_NOT_ALTERABLE` exception.

See also Routing for publish/subscribe clusters: Notes on behavior and Designing publish/subscribe clusters.

CLUSTER

The name of the cluster to which this topic belongs. Setting this parameter to a cluster that this queue manager is a member of makes all queue managers in the cluster aware of this topic. Any publication to this topic or a topic string below it put to any queue manager in the cluster is propagated to subscriptions on any other queue manager in the cluster. For more details, see Distributed publish/subscribe networks.

`' '` If no topic object above this topic in the topic tree has set this parameter to a cluster name, then this topic does not belong to a cluster. Publications and subscriptions for this topic are not propagated to publish/subscribe cluster-connected queue managers. If a topic node higher in the topic tree has a cluster name set, publications and subscriptions to this topic are also propagated throughout the cluster.

string The topic belongs to this cluster. It is not recommended that this is set to a different cluster from a topic object above this topic object in the topic tree. Other queue managers in the cluster will honor this object's definition unless a local definition of the same name exists on those queue managers.

To prevent all subscriptions and publications being propagated throughout a cluster, leave this parameter blank on the system topics SYSTEM.BASE.TOPIC and SYSTEM.DEFAULT.TOPIC, except in special circumstances, for example to support migration.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command runs on the queue manager on which it was entered.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

COMMINFO(*comminfo-name*)
The name of the Multicast communication information object associated with this topic object.

CUSTOM(*string*)
The custom attribute for new features.

This attribute contains the values of attributes, as pairs of attribute name and value, separated by at least one space. The attribute name-value pairs have the form `NAME(VALUE)`. Single quotation marks must be escaped with another single quotation mark.

`CAPEXPRY`(*integer*)
The maximum time, expressed in tenths of a second, until a message published to a topic which inherits properties from this object, remains in the system until it becomes eligible for expiry processing.

For more information on message expiry processing, see Enforcing lower expiration times.

integer
The value must be in the range one through to 999 999 999.

`NOLIMIT`
There is no limit on the expiry time of messages put to this topic.

`ASPARENT`
The maximum message expiry time is based on the setting of the closest parent administrative topic object in the topic tree. This is the default value.

Specifying a value for `CAPEXPRY` that is not valid, does not cause the command to fail. Instead, the default value is used.

DEFPRESP
Specifies the put response to be used when applications specify the MQPMO_RESPONSE_AS_DEF option.

ASPARENT
The default put response is based on the setting of the closest parent administrative topic object in the topic tree.

SYNC Put operations to the queue that specify MQPMO_RESPONSE_AS_Q_DEF are issued as if MQPMO_SYNC_RESPONSE had been specified instead. Fields in the MQMD and MQPMO are returned by the queue manager to the application.

ASYNC
Put operations to the queue that specify MQPMO_RESPONSE_AS_Q_DEF are always

issued as if MQPMO_ASYNC_RESPONSE had been specified instead. Some fields in the MQMD and MQPMO are not returned by the queue manager to the application; but an improvement in performance might be seen for messages put in a transaction and any non-persistent messages

DEFPRTY(*integer*)
The default priority of messages published to the topic.

(*integer*)
The value must be in the range zero (the lowest priority), through to the MAXPRTY queue manager parameter (MAXPRTY is 9).

ASPARENT
The default priority is based on the setting of the closest parent administrative topic object in the topic tree.

DEFPSIST
Specifies the message persistence to be used when applications specify the MQPER_PERSISTENCE_AS_TOPIC_DEF option.

ASPARENT
The default persistence is based on the setting of the closest parent administrative topic object in the topic tree.

NO Messages on this queue are lost during a restart of the queue manager.

YES Messages on this queue survive a restart of the queue manager.

On z/OS, N and Y are accepted as synonyms of NO and YES.

DESCR(*string*)
Plain-text comment. It provides descriptive information about the object when an operator issues the DISPLAY TOPIC command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

DURSUB
Specifies whether applications are permitted to make durable subscriptions on this topic.

ASPARENT
Whether durable subscriptions can be made on this topic is based on the setting of the closest parent administrative topic object in the topic tree.

NO Durable subscriptions cannot be made on this topic.

YES Durable subscriptions can be made on this topic.

LIKE(*topic-name*)
The name of a topic. The topic parameters are used to model this definition.

If this field is not completed, and you do not complete the parameter fields related to the command, the values are taken from the default definition for topics on this queue manager.

Not completing this field is equivalent to specifying:

```
LIKE(SYSTEM.DEFAULT.TOPIC)
```

A default topic definition is provided, but it can be altered by the installation to the default values required. See Rules for naming IBM MQ objects.

z/OS On z/OS, the queue manager searches page set zero for an object with the name you specify and a disposition of QMGR or COPY. The disposition of the LIKE object is not copied to the object you are defining.

Note:

1. QSGDISP (GROUP) objects are not searched.
2. LIKE is ignored if QSGDISP(COPY) is specified.

MCAST

Specifies whether multicast is allowable in the topic tree. The values are:

ASPARENT
The multicast attribute of the topic is inherited from the parent.

DISABLED
No multicast traffic is allowed at this node.

ENABLED
Multicast traffic is allowed at this node.

ONLY Only subscriptions from a multicast capable client are allowed.

MDURMDL(*string*)

The name of the model queue to be used for durable subscriptions that request that the queue manager manages the destination of its publications (see Rules for naming IBM MQ objects). The maximum length is 48 characters.

If **MDURMDL** is blank, it operates in the same way as ASPARENT values on other attributes. The name of the model queue to be used is based on the closest parent administrative topic object in the topic tree with a value set for **MDURMDL**.

If you use **MDURMDL** to specify a model queue for a clustered topic, you must ensure that the queue is defined on every queue manager in the cluster where a durable subscription using this topic can be made.

The dynamic queue created from this model has a prefix of SYSTEM.MANAGED.DURABLE

MNDURMDL(*string*)

The name of the model queue to be used for non-durable subscriptions that request that the queue manager manages the destination of its publications (see Rules for naming IBM MQ objects). The maximum length is 48 characters.

If **MNDURMDL** is blank, it operates in the same way as ASPARENT values on other attributes. The name of the model queue to be used is based on the closest parent administrative topic object in the topic tree with a value set for **MNDURMDL**.

If you use **MNDURMDL** to specify a model queue for a clustered topic, you must ensure that the queue is defined on every queue manager in the cluster where a non-durable subscription using this topic can be made.

The dynamic queue created from this model has a prefix of SYSTEM.MANAGED.NDURABLE.

NPMSGDLV

The delivery mechanism for non-persistent messages published to this topic:

ASPARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

ALL Non-persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT call fails.

ALLAVAIL
Non-persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

ALLDUR
Non-persistent messages must be delivered to all durable subscribers. Failure to deliver a non-persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no subscribers receive the message and the MQPUT calls fails.

PMSGDLV
The delivery mechanism for persistent messages published to this topic:

ASPARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

ALL Persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT call fails.

ALLAVAIL
Persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

ALLDUR
Persistent messages must be delivered to all durable subscribers. Failure to deliver a persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no subscribers receive the message and the MQPUT calls fails.

PROXYSUB
Controls when a proxy subscription is sent for this topic, or topic strings below this topic, to neighboring queue managers when in a publish/subscribe cluster or hierarchy. For more details, see Subscription performance in publish/subscribe networks.

FIRSTUSE
For each unique topic string at or below this topic object, a proxy subscription is asynchronously sent to all neighboring queue managers in the following scenarios:

- When a local subscription is created.
- When a proxy subscription is received that must be propagated to further directly connected queue managers.

FORCE
A wildcard proxy subscription that matches all topic strings at and below this point in the topic tree is sent to neighboring queue managers even if no local subscriptions exist.

Note: The proxy subscription is sent when this value is set on DEFINE or ALTER. When set on a clustered topic, all queue managers in the cluster issue the wildcard proxy subscription to all other queue managers in the cluster.

PUB Controls whether messages can be published to this topic.

ASPARENT
Whether messages can be published to the topic is based on the setting of the closest parent administrative topic object in the topic tree.

ENABLED
Messages can be published to the topic (by suitably authorized applications).

DISABLED
Messages cannot be published to the topic.

See also Special handling for the **PUB** parameter.

PUBSCOPE
Determines whether this queue manager propagates publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster.

Note: You can restrict the behavior on a publication-by-publication basis, using MQPMO_SCOPE_QMGR on the Put Message options.

ASPARENT
Determines whether this queue manager propagates publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster. This is based on the setting of the first parent administrative node found in the topic tree that relates to this topic.

QMGR
Publications for this topic are not propagated to connected queue managers.

ALL Publications for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

z/OS **QSGDISP**
This parameter applies to z/OS only.

Specifies the disposition of the object within the group.

QSGDISP	DEFINE
COPY	The object is defined on the page set of the queue manager that executes the command using the QSGDISP(GROUP) object of the same name as the 'LIKE' object.
GROUP	The object definition resides in the shared repository but only if the queue manager is in a queue-sharing group. If the definition is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to attempt to make or refresh local copies on page set zero: `DEFINE TOPIC(name)` `REPLACE QSGDISP(COPY)` The DEFINE for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.
PRIVATE	Not permitted.
QMGR	The object is defined on the page set of the queue manager that executes the command.

REPLACE and NOREPLACE
Determines whether the existing definition (and on z/OS, with the same disposition) is to be replaced with this one. Any object with a different disposition is not changed.

REPLACE
If the object does exist, the effect is like issuing the **ALTER** command without the **FORCE** option and with *all* the other parameters specified.

(The difference between the **ALTER** command without the **FORCE** option, and the **DEFINE** command with the **REPLACE** option, is that **ALTER** does not change unspecified parameters, but **DEFINE** with **REPLACE** sets *all* the parameters. When you use **REPLACE**, unspecified parameters are taken either from the object named on the **LIKE** option, or from the default definition, and the parameters of the object being replaced, if one exists, are ignored.)

The command fails if both of the following statements are true:

- The command sets parameters that would require the use of the **FORCE** option if you were using the **ALTER** command.
- The object is open.

The ALTER command with the FORCE option succeeds in this situation.

NOREPLACE
The definition must not replace any existing definition of the object.

SUB Controls whether applications are to be permitted to subscribe to this topic.

ASPARENT
Whether applications can subscribe to the topic is based on the setting of the closest parent administrative topic object in the topic tree.

ENABLED
Subscriptions can be made to the topic (by suitably authorized applications).

DISABLED
Applications cannot subscribe to the topic.

SUBSCOPE
Determines whether this queue manager subscribes to publications in this queue manager or in the network of connected queue managers. If subscribing to all queue managers, the queue manager propagates subscriptions to them as part of a hierarchy or as part of a publish/subscribe cluster.

Note: You can restrict the behavior on a subscription-by-subscription basis, using **MQPMO_SCOPE_QMGR** on the Subscription Descriptor or **SUBSCOPE(QMGR)** on **DEFINE SUB**. Individual subscribers can override the **SUBSCOPE** setting of ALL by specifying the **MQSO_SCOPE_QMGR** subscription option when creating a subscription.

ASPARENT
Whether this queue manager subscribes to publications in the same way as the setting of the first parent administrative node found in the topic tree relating to this topic.

QMGR Only publications that are published on this queue manager reach the subscriber.

ALL A publication made on this queue manager or on another queue manager reaches the subscriber. Subscriptions for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

TOPICSTR(*string*)
The topic string represented by this topic object definition. This parameter is required and cannot contain the empty string.

The topic string must not be the same as any other topic string already represented by a topic object definition.

The maximum length of the string is 10,240 characters.

TYPE(*topic-type*)
If this parameter is used it must follow immediately after the *topic-name* parameter on all platforms z/OS except z/OS.

LOCAL A local topic object.

USEDLQ
Determines whether the dead-letter queue is used when publication messages cannot be delivered to their correct subscriber queue.

ASPARENT
Determines whether to use the dead-letter queue using the setting of the closest

administrative topic object in the topic tree. This value is the default supplied with IBM MQ, but your installation might have changed it.

NO Publication messages that cannot be delivered to their correct subscriber queue are treated as a failure to put the message. The MQPUT of an application to a topic fails in accordance with the settings of **NPMSGDLV** and **PMSGDLV**.

YES When the **DEADQ** queue manager attribute provides the name of a dead-letter queue, then it is used. If the queue manager does not provide the name of a dead-letter queue, then the behavior is as for NO.

WILDCARD

The behavior of wildcard subscriptions with respect to this topic.

PASSTHRU

Subscriptions made to a wildcarded topic less specific than the topic string at this topic object receive publications made to this topic and to topic strings more specific than this topic.

BLOCK Subscriptions made to a wildcarded topic less specific than the topic string at this topic object do not receive publications made to this topic or to topic strings more specific than this topic.

The value of this attribute is used when subscriptions are defined. If you alter this attribute, the set of topics covered by existing subscriptions is not affected by the modification. This scenario applies also if the topology is changed when topic objects are created or deleted; the set of topics matching subscriptions created following the modification of the **WILDCARD** attribute is created using the modified topology. If you want to force the matching set of topics to be re-evaluated for existing subscriptions, you must restart the queue manager.

Related information:

Defining an administrative topic

DELETE AUTHINFO:

Use MQSC command DELETE AUTHINFO to delete an authentication information object.

Using MQSC commands

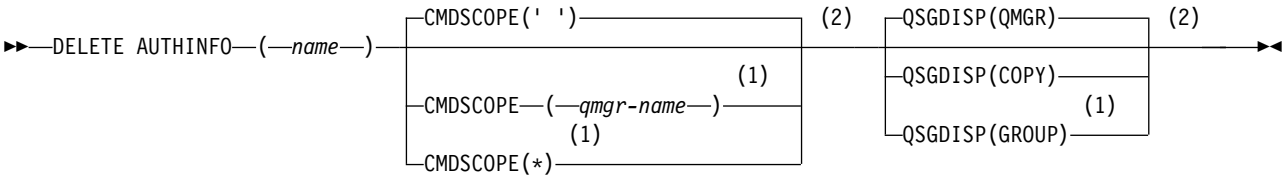
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for DELETE AUTHINFO" on page 713

Synonym: None

DELETE AUTHINFO



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group. You can use queue-sharing groups only on IBM MQ for z/OS.

2 Valid only on z/OS.

Parameter descriptions for DELETE AUTHINFO

(name)

Name of the authentication information object. This is required.

The name must be that of an existing authentication information object.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

z/OS **QSGDISP**

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

COPY The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.

GROUP

The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE AUTHINFO(name) QSGDISP(COPY)
```

The deletion of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

QMGR

The object definition resides on the page set of the queue manager that executes the

command. The object was defined using a command that had the parameters QSGDISP(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

This is the default value.

DELETE AUTHREC on Multiplatforms: Multi

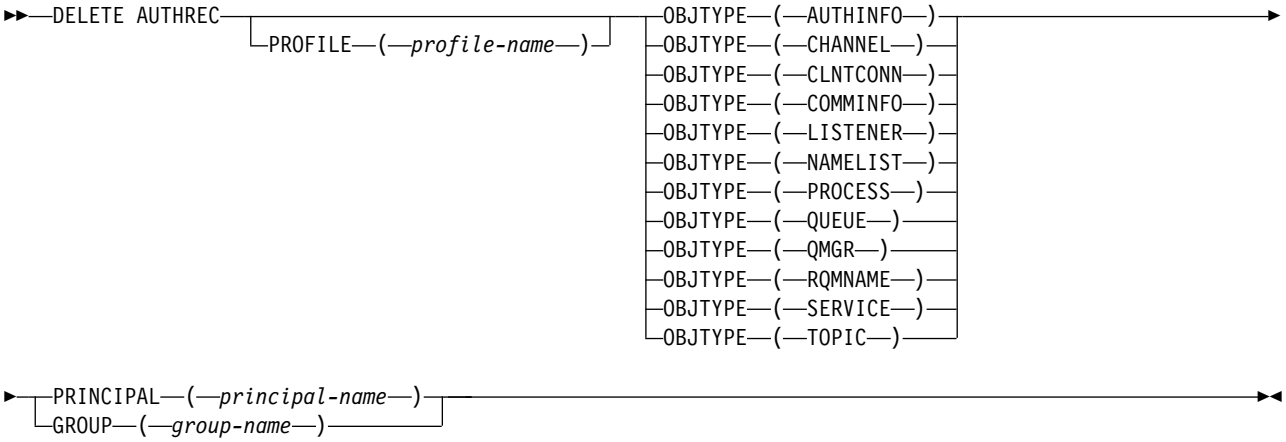
Use the MQSC command DELETE AUTHREC to delete authority records associated with a profile name.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Parameter descriptions"

DELETE AUTHREC



Parameter descriptions

PROFILE(*profile-name*)
The name of the object or generic profile for which to remove the authority record. This parameter is required unless the **OBJTYPE** parameter is QMGR, in which case it can be omitted.

OBJTYPE
The type of object referred to by the profile. Specify one of the following values:

AUTHINFO
Authentication information record

CHANNEL
Channel

CLNTCONN
Client connection channel

COMMINFO
Communication information object

LISTENER
Listener

NAMELIST
Namelist

PROCESS
Process

QUEUE
Queue

QMGR
Queue manager

RQMNAME
Remote queue manager

SERVICE
Service

TOPIC
Topic

PRINCIPAL(*principal-name*)
A principal name. This is the name of a user for whom to remove authority records for the specified profile. On IBM MQ for Windows, the name of the principal can optionally include a domain name, specified in this format: `user@domain`.

You must specify either PRINCIPAL or GROUP.

GROUP(*group-name*)
A group name. This is the name of the user group for which to remove authority records for the specified profile. You can specify one name only and it must be the name of an existing user group.

Windows For IBM MQ for Windows only, the group name can optionally include a domain name, specified in the following formats:

```
GroupName@domain
domain\GroupName
```

You must specify either PRINCIPAL or GROUP.

DELETE BUFFPOOL on z/OS: z/OS

Use the MQSC command DELETE BUFFPOOL to delete a buffer pool that is used for holding messages in main storage.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources `2CR`. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage note for DELETE BUFFPOOL" on page 716
- "Parameter descriptions for DELETE BUFFPOOL" on page 716

Synonym: DEL BP

DELETE BUFFPOOL

Usage note for DELETE BUFFPOOL

- Ensure there are no current page set definitions using the named buffer pool, otherwise the command will fail.
- DELETE BUFFPOOL cannot be issued from CSQINPT.

Parameter descriptions for DELETE BUFFPOOL

(integer)

CD This is the number of the buffer pool to be deleted. If IBM MQ Version 8.0 new functions are enabled with OPMODE, the value is an integer in the range zero through 99. Otherwise, the value is an integer in the range zero through 15.

DELETE CFSTRUCT on z/OS: z/OS

Use the MQSC command DELETE CFSTRUCT to delete a CF application structure definition.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for DELETE CFSTRUCT"
- "Keyword and parameter descriptions for DELETE CFSTRUCT"

Synonym: None

DELETE CFSTRUCT

Usage notes for DELETE CFSTRUCT

1. This command is valid only z/OS when the queue manager is a member of a queue-sharing group.
2. The command fails if there are any queues in existence that reference this CF structure name that are not both empty and closed.
3. The command cannot specify the CF administration structure (CSQ_ADMIN).
4. The command deletes the Db2 CF structure record only. It does **not** delete the CF structure definition from the CFRM policy data set.
5. CF structures at CFLEVEL(1) are automatically deleted when the last queue on that structure is deleted.

Keyword and parameter descriptions for DELETE CFSTRUCT

(structure-name)

The name of the CF structure definition to be deleted. The name must be defined within the queue-sharing group.

DELETE CHANNEL:

Use the MQSC command DELETE CHANNEL to delete a channel definition.

Using MQSC commands

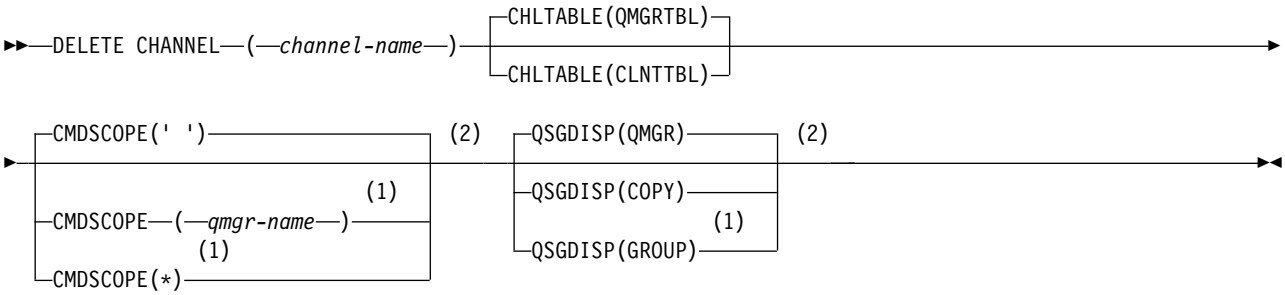
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes"
- "Parameter descriptions"

Synonym: DELETE CHL

DELETE CHANNEL



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes

- Successful completion of the command does not mean that the action completed. To check for true completion, see the DELETE CHANNEL step in Checking that async commands for distributed networks have finished.
- z/OS On z/OS systems, the command fails if the channel initiator and command server have not been started, or the channel status is RUNNING, except client-connection channels, which can be deleted without the channel initiator or command server running.
- z/OS On z/OS systems, you can only delete cluster-sender channels that have been created manually.

Parameter descriptions

(channel-name)
The name of the channel definition to be deleted. This is required. The name must be that of an existing channel.

CHLTABLE

Specifies the channel definition table that contains the channel to be deleted. This is optional.

QMGRTBL

The channel table is that associated with the target queue manager. This table does not contain any channels of type CLNTCONN. This is the default.

CLNTTBL

The channel table for CLNTCONN channels. On z/OS, this is associated with the target queue manager, but separate from the main channel table. On all other platforms, this channel table is normally associated with a queue manager, but can be a system-wide, queue manager independent channel table if you set up a number of environment variables. For more information about setting up environment variables, see Using IBM MQ environment variables.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

z/OS **QSGDISP**

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

COPY The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.

GROUP

The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE CHANNEL(channel-name) QSGDISP(COPY)
```

The deletion of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

QMGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

This is the default value.

DELETE CHANNEL (MQTT): AIX Linux Windows

Use the MQSC command DELETE CHANNEL to delete an MQ Telemetry channel definition.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

The DELETE CHANNEL (MQTT) command is only valid for MQ Telemetry channels.

Synonym: DELETE CHL

DELETE CHANNEL

Parameter descriptions

(channel-name)
The name of the channel definition to be deleted. This is required. The name must be that of an existing channel.

CHLTYPE
This parameter is required. There is only one possible value: MQTT.

DELETE COMMINFO on Multiplatforms: Multi

Use the MQSC command DELETE COMMINFO to delete a communication information object.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Parameter descriptions for DELETE COMMINFO" on page 720

Synonym: DEL COMMINFO

DELETE COMMINFO

Parameter descriptions for DELETE COMMINFO

(comminfo name)
The name of the communications information object to be deleted. This is required.

DELETE LISTENER on Multiplatforms: Multi

Use the MQSC command DELETE LISTENER to delete a listener definition.

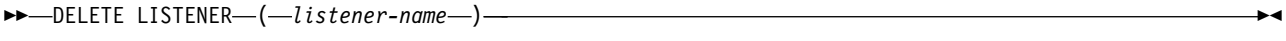
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Usage notes for DELETE LISTENER"
- "Keyword and parameter descriptions for DELETE LISTENER"

Synonym: DELETE LSTR

DELETE LISTENER



Usage notes for DELETE LISTENER

1. The command fails if an application has the specified listener object open, or if the listener is currently running.

Keyword and parameter descriptions for DELETE LISTENER

(listener-name)

The name of the listener definition to be deleted. This is required. The name must be that of an existing listener defined on the local queue manager.

DELETE NAMELIST:

Use the MQSC command DELETE NAMELIST to delete a namelist definition.

Using MQSC commands

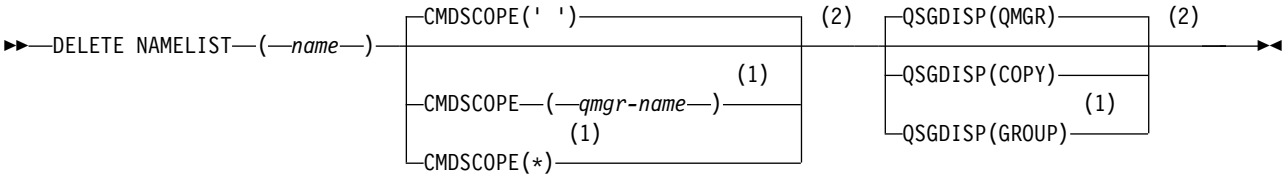
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes"
- "Parameter descriptions for DELETE NAMELIST"

Synonym: DELETE NL

DELETE NAMELIST



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes

Successful completion of the command does not mean that the action completed. To check for true completion, see the DELETE NAMELIST step in Checking that async commands for distributed networks have finished.

Parameter descriptions for DELETE NAMELIST

You must specify which namelist definition you want to delete.

(name)
The name of the namelist definition to be deleted. The name must be defined to the local queue manager.

If an application has this namelist open, the command fails.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

*
The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

z/OS **QSGDISP**

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

COPY The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.

GROUP
The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE NAMELIST(name) QSGDISP(COPY)
```

The deletion of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

QMGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

This is the default value.

DELETE POLICY on Multiplatforms: Multi

Use the MQSC command DELETE POLICY to delete a security policy.

- Syntax diagram
- "Parameter descriptions for DELETE POLICY"

DELETE POLICY

Parameter descriptions for DELETE POLICY

(policy-name)
Specifies the policy name to be deleted.

The name of the policy, or policies, to delete are the same as the name of the queue, or queues, that the policies control.

DELETE PROCESS:

Use the MQSC command DELETE PROCESS to delete a process definition.

Using MQSC commands

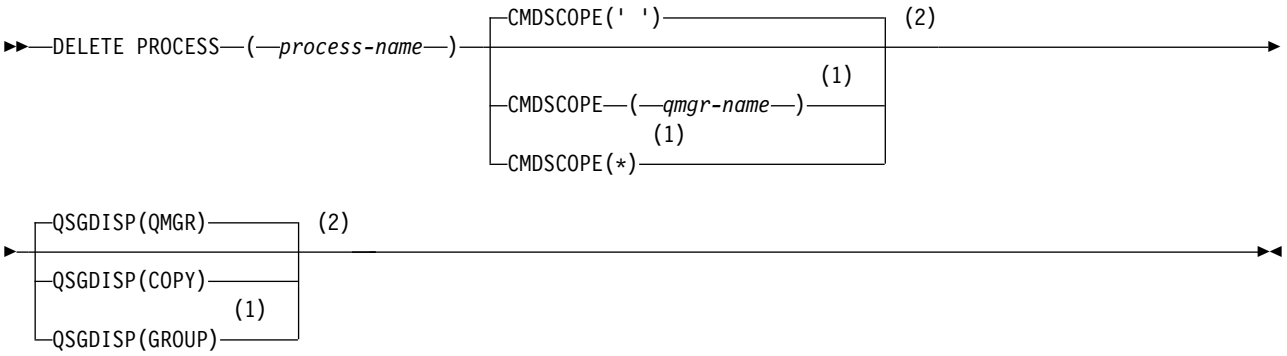
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for DELETE PROCESS" on page 724

Synonym: DELETE PRO

DELETE PROCESS



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Parameter descriptions for DELETE PROCESS

You must specify which process definition you want to delete.

(process-name)
The name of the process definition to be deleted. The name must be defined to the local queue manager.

If an application has this process open, the command fails.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

z/OS **QSGDISP**
This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

COPY The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.

GROUP
The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE PROCESS(process-name) QSGDISP(COPY)
```

The deletion of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

QMGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

This is the default value.

DELETE PSID on z/OS: z/OS

Use the MQSC command DELETE PSID to delete a page set. This command closes the page set and de-allocates it from the queue manager.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for DELETE PSID"
- "Parameter descriptions for DELETE PSID"

Synonym: DEL PSID

DELETE PSID

Usage notes for DELETE PSID

1. The identified page set must have no storage class (STGCLASS) referencing it.
2. If the page set still has buffers in the buffer pool when you issue this command, the command fails and an error message is issued. You cannot delete the page set until 3 checkpoints have been completed since the page set was emptied.
3. If the page set is not to be used again by the queue manager, update the queue manager started task procedure JCL, and remove the corresponding DEFINE PSID command from the CSQINP1 initialization data set. If the page set had a dedicated buffer pool, remove its definitions also from CSQINP1.
4. If you want to reuse the data set again as a page set, format it before doing so.

Parameter descriptions for DELETE PSID

(psid-number)
Identifier of the page set. This is required. You cannot delete page set 0.

DELETE queues:

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

This section contains the following commands:

- "DELETE QALIAS" on page 728
- "DELETE QLOCAL" on page 729
- "DELETE QMODEL" on page 729
- "DELETE QREMOTE" on page 730

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

Usage notes for DELETE queues

- Successful completion of the command does not mean that the action completed. To check for true completion, see the DELETE queues step in Checking that async commands for distributed networks have finished.

Parameter descriptions for DELETE queues

(q-name)

The name of the queue must be defined to the local queue manager for all the queue types.

For an alias queue this is the local name of the alias queue to be deleted.

For a model queue this is the local name of the model queue to be deleted.

For a remote queue this is the local name of the remote queue to be deleted.

For a local queue this is the name of the local queue to be deleted. You must specify which queue you want to delete.

Note: A queue cannot be deleted if it contains uncommitted messages.

If an application has this queue open, or has open a queue that eventually resolves to this queue, the command fails. The command also fails if this queue is a transmission queue, and any queue that is, or resolves to, a remote queue that references this transmission queue, is open.

If this queue has a SCOPE attribute of CELL, the entry for the queue is also deleted from the cell directory.

AUTHREC

This parameter does not apply to z/OS.

Specifies whether the associated authority record is also deleted:

YES The authority record associated with the object is deleted. This is the default.

NO The authority record associated with the object is not deleted.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP or SHARED.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

PURGE and NOPURGE
Specifies whether any existing committed messages on the queue named by the DELETE command are to be purged for the delete command to work. The default is NOPURGE.

PURGE
The deletion is to go ahead even if there are committed messages on the named queue, and these messages are also to be purged.

NOPURGE
The deletion is not to go ahead if there are any committed messages on the named queue.

z/OS **QSGDISP**
Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). If the object definition is shared, you do not need to delete it on every queue manager that is part of a queue-sharing group. (Queue-sharing groups are available only on IBM MQ for z/OS.)

COPY The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.

GROUP
The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). Any object residing on the page set of the queue manager that executes the command, or any object defined using a command that had the parameters QSGDISP(SHARED), is not affected by this command.

If the deletion is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to make, or delete, local copies on page set zero:

```
DELETE queue(q-name) QSGDISP(COPY)
```

or, for a local queue only:

```
DELETE QLOCAL(q-name) NOPURGE QSGDISP(COPY)
```

The deletion of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

Note: You always get the NOPURGE option even if you specify PURGE. To delete messages on local copies of the queues, you must explicitly issue the command:

```
DELETE QLOCAL(q-name) QSGDISP(COPY) PURGE
```

for each copy.

QMGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

This is the default value.

SHARED
This option applies only to local queues.

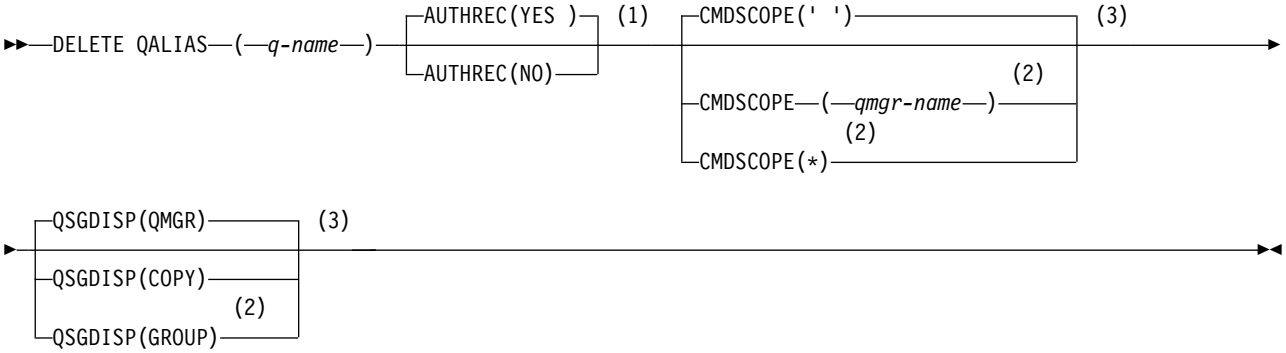
The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(SHARED). Any object residing on the page set of the queue manager that executes the command, or any object defined using a command that had the parameters QSGDISP(GROUP), is not affected by this command.

DELETE QALIAS:

Use DELETE QALIAS to delete an alias queue definition.

Synonym: DELETE QA

DELETE QALIAS



Notes:

1 Not valid on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

The parameters are described in "DELETE queues" on page 726.

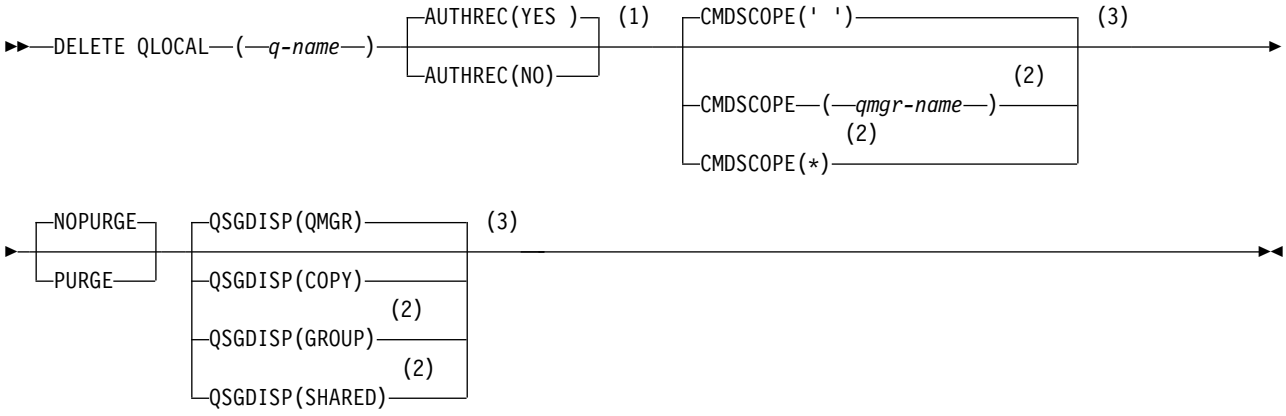
Related information:
Working with alias queues

DELETE QLOCAL:

Use DELETE QLOCAL to delete a local queue definition. You can specify that the queue must not be deleted if it contains messages, or that it can be deleted even if it contains messages.

Synonym: DELETE QL

DELETE QLOCAL



Notes:

1 Not valid on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

The parameters are described in "DELETE queues" on page 726.

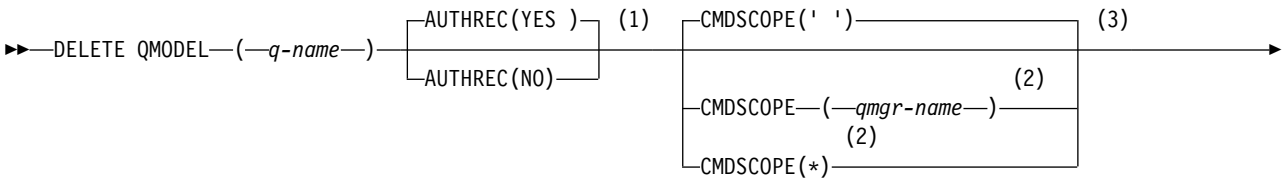
Related information:
Deleting a local queue

DELETE QMODEL:

Use **DELETE QMODEL** to delete a model queue definition.

Synonym: DELETE QM

DELETE QMODEL



Notes:

1 Not valid on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

The parameters are described in "DELETE queues" on page 726.

Related information:

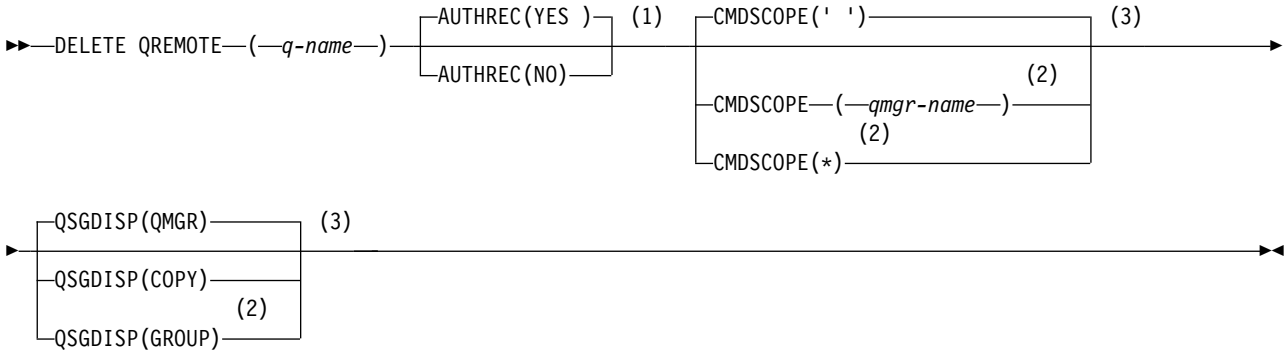
Working with model queues

DELETE QREMOTE:

Use DELETE QREMOTE to delete a local definition of a remote queue. It does not affect the definition of that queue on the remote system.

Synonym: DELETE QR

DELETE QREMOTE



Notes:

1 Not valid on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

The parameters are described in "DELETE queues" on page 726.

DELETE SERVICE on Multiplatforms: Multi

Use the MQSC command DELETE SERVICE to delete a service definition.

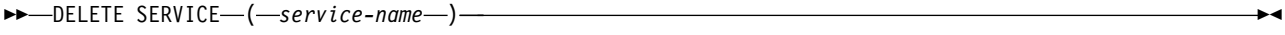
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Usage notes for DELETE SERVICE"
- "Keyword and parameter descriptions for DELETE SERVICE"

Synonym:

DELETE SERVICE



Usage notes for DELETE SERVICE

1. The command fails if an application has the specified service object open, or if the service is currently running.

Keyword and parameter descriptions for DELETE SERVICE

(service-name)

The name of the service definition to be deleted. This is required. The name must be that of an existing service defined on the local queue manager.

DELETE SUB:

Use the MQSC command **DELETE SUB** to remove a durable subscription from the system. For a managed destination, any unprocessed messages left on the destination are removed.

Using MQSC commands

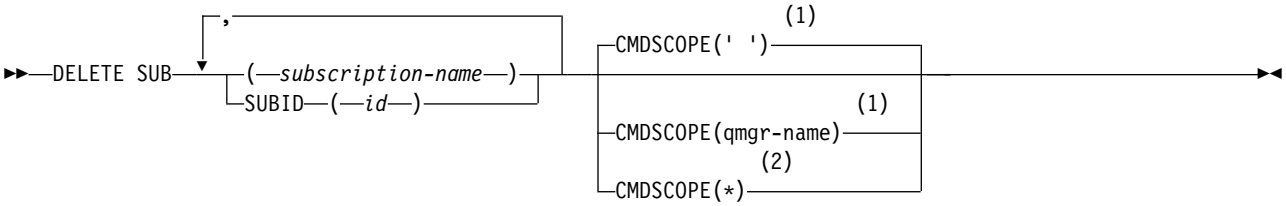
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- Usage notes for DELETE SUB
- "Parameter descriptions for DELETE SUB" on page 732

Synonym: **DEL SUB**

DELETE SUB



Notes:

1 Valid only on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

Usage notes for DELETE SUB

- You can specify either the name, the identifier, or both, of the subscription you want to delete.

  Examples of valid forms:

  ```
  DELETE SUB(xyz)
  DELETE SUB SUBID(123)
  DELETE SUB(xyz) SUBID(123)
  ```

- Successful completion of the command does not mean that the action completed. To check for true completion, see the DELETE SUB step in Checking that async commands for distributed networks have finished.

Parameter descriptions for DELETE SUB

subscription-name
The local name of the subscription definition to be deleted.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if **QSGDISP** is set to GROUP.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use **CMDSCOPE** as a filter keyword.

SUBID(*string* **)**
The internal, unique key identifying a subscription.

Related information:
Deleting a subscription

DELETE STGCLASS on z/OS: z/OS

Use the MQSC command DELETE STGCLASS to delete a storage class definition.

Using MQSC commands

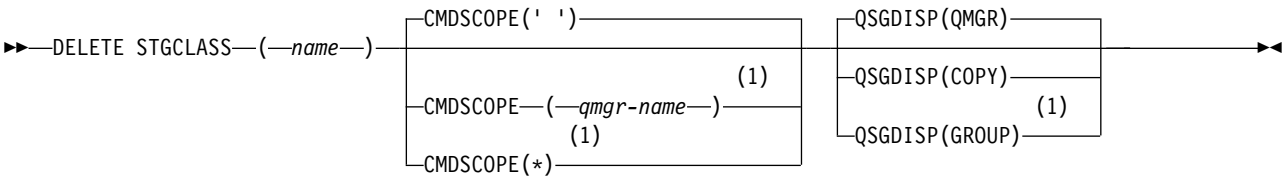
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for DELETE STGCLASS"

Synonym: DELETE STC

DELETE STGCLASS



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

Parameter descriptions for DELETE STGCLASS

You must specify which storage class definition you want to delete.

All queues that use this storage class must be altered to use another storage class.

(name)
The name of the storage class definition to be deleted. The name must be defined to the local queue manager.

The command fails unless all queues referencing the storage class are empty and closed.

CMDSCOPE
This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

QSGDISP

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

COPY The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.

GROUP

The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE STGCLASS(name) QSGDISP(COPY)
```

The deletion of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

QMGR

The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

This is the default value.

DELETE TOPIC:

Use **DELETE TOPIC** to delete an IBM MQ administrative topic node.

Using MQSC commands

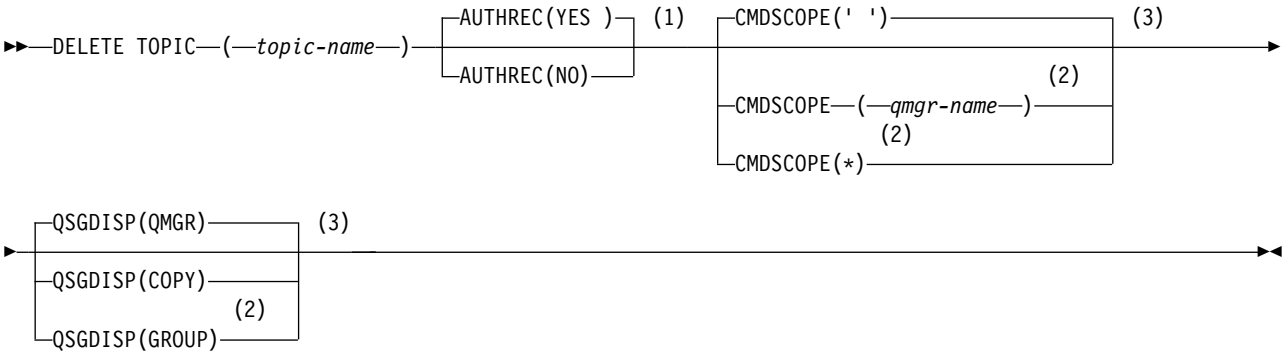
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for DELETE TOPIC" on page 735
- "Parameter descriptions for DELETE TOPIC" on page 735

Synonym: None

DELETE TOPIC



Notes:

1 Not valid on z/OS

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

Usage notes for DELETE TOPIC

- Successful completion of the command does not mean that the action completed. To check for true completion, see the DELETE TOPIC step in Checking that async commands for distributed networks have finished.

Parameter descriptions for DELETE TOPIC

(topic-name)

The name of the administrative topic object to be deleted. This parameter is required.

The name must be that of an existing administrative topic object.

AUTHREC

This parameter does not apply to z/OS

Specifies whether the associated authority record is also deleted:

YES The authority record associated with the object is deleted. This is the default.

NO The authority record associated with the object is not deleted.

z/OS **CMDSCOPE**

This parameter applies to only z/OS and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue

manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

z/OS **QSGDISP**

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

COPY The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.

GROUP

The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to make, or delete, local copies on page set zero:

```
DELETE TOPIC(topic-name) QSGDISP(COPY)
```

The deletion of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

QMGR

The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

This is the default value.

Related information:

Deleting an administrative topic definition

DISPLAY ARCHIVE on z/OS: z/OS

Use the MQSC command DISPLAY ARCHIVE to display archive system parameters and information.

Using MQSC commands

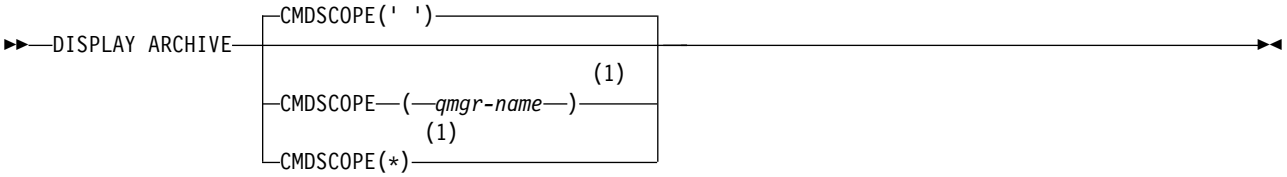
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 12CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for DISPLAY ARCHIVE" on page 737
- "Parameter descriptions for DISPLAY ARCHIVE" on page 737

Synonym: DIS ARC

DISPLAY ARCHIVE



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

Usage notes for DISPLAY ARCHIVE

1. DISPLAY ARCHIVE returns a report that shows the initial values for the archiving parameters, and the current values as changed by the SET ARCHIVE command.
   - Units in which primary and secondary space allocations are made (ALCUNIT).
   - Prefix for first archive log data set name (ARCPFX1).
   - Prefix for second archive log data set name (ARCPFX2).
   - The retention period of the archive log data set in days (ARCRETN).
   - List of route codes for messages to the operator about archive log data sets (ARCWRTC).
   - Whether to send message to operator and wait for reply before trying to mount an archive log data set (ARCWTOR).
   - Block size of archive log data set (BLKSIZE).
   - Whether archive log data sets are cataloged in the ICF (CATALOG).
   - Whether archive log data sets should be compacted (COMPACT).
   - Primary space allocation for DASD data sets (PRIQTY).
   - Whether archive log data sets are protected by ESM profiles when the data sets are created (PROTECT).
   - Maximum time, in seconds, allowed for quiesce when ARCHIVE LOG with MODE(QUIESCE) specified (QUIESCE).
   - Secondary space allocation for DASD data sets. See the ALCUNIT parameter for the units to be used (SECQTY).
   - Whether the archive data set name should include a time stamp (TSTAMP).
   - Device type or unit name on which the first copy of archive log data sets is stored (UNIT).
   - Device type or unit name on which the second copy of archive log data sets is stored (UNIT2).

   It also reports the status of tape units used for archiving.

   For more details of these parameters, see "SET ARCHIVE on z/OS" on page 1003.
2. This command is issued internally by IBM MQ at the end of queue manager startup.

Parameter descriptions for DISPLAY ARCHIVE

CMDSCOPE

This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE cannot be used for commands issued from the first initialization input data set CSQINP1.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

DISPLAY AUTHINFO:

Use the MQSC command DISPLAY AUTHINFO to display the attributes of an authentication information object.

Using MQSC commands

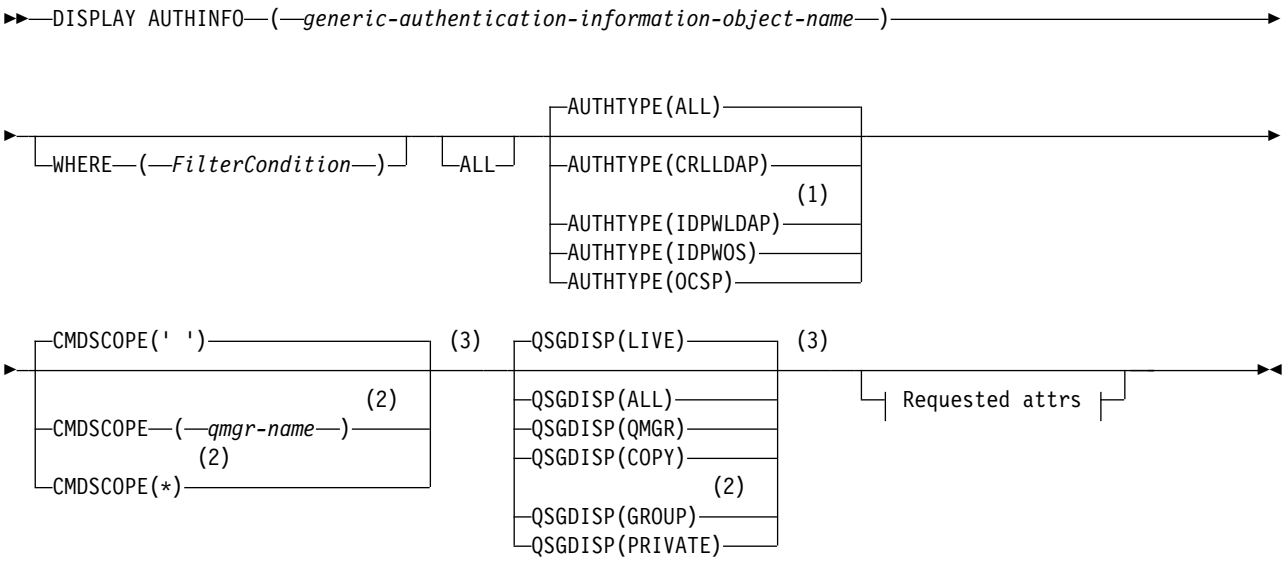
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

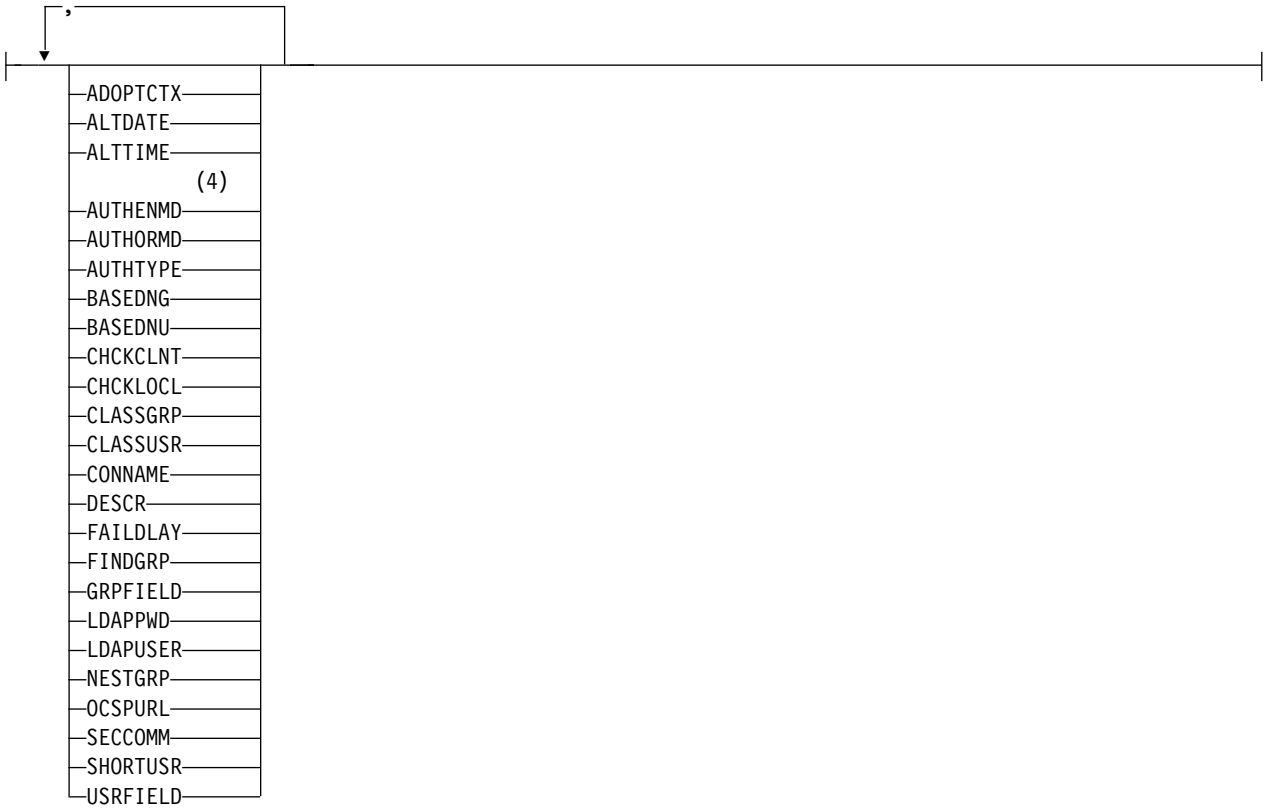
- Syntax diagram
- "Parameter descriptions for DISPLAY AUTHINFO" on page 739
- "Requested parameters" on page 741

Synonym: DIS AUTHINFO

DISPLAY AUTHINFO



Requested attrs:



Notes:

1 Not valid on IBM MQ for z/OS.

2 Valid only when the queue manager is a member of a queue-sharing group. You can use queue-sharing groups only on IBM MQ for z/OS.

3 Valid only on z/OS.

4 Not valid on z/OS and AUTHENMD PAM value valid only on UNIX.

Parameter descriptions for DISPLAY AUTHINFO

(generic-authentication-information-object-name)
The name of the authentication information object to be displayed (see Rules for naming IBM MQ objects). A trailing asterisk (*) matches all authentication information objects with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all authentication information objects.

WHERE
Specify a filter condition to display only those authentication information objects that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

`filter-keyword`
Almost any parameter that can be used to display attributes for this DISPLAY command. However, you cannot use the CMDSCOPE or QSGDISP parameters as filter keywords.

`operator`
This is used to determine whether an authentication information object satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

`filter-value`
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use any of the operators except LK and NL.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. The characters must be valid for the attribute you are testing. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. You cannot use a generic filter-value with numeric values. Only a single trailing wildcard character (asterisk) is permitted.

  You can only use operators LK or NL for generic values on the DISPLAY AUTHINFO command.

ALL Specify this to display all the parameters. If this parameter is specified, any parameters that are requested specifically have no effect; all parameters are still displayed.

This is the default if you do not specify a generic name and do not request any specific parameters.

z/OS On z/OS this is also the default if you specify a filter condition using the WHERE parameter, but on other platforms only requested attributes are displayed.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

AUTHTYPE
Specifies the authentication information type of the objects for which information is to be displayed. Values are:

ALL
This is the default value and displays information for objects defined with AUTHTYPE(CRLLDAP) and with AUTHTYPE(OCSP).

CRLLDAP
Displays information only for objects defined with AUTHTYPE(CRLLDAP).

IDPWLDAP
Displays information only for objects defined with AUTHTYPE(IDPWLDAP).

IDPWOS
Displays information only for objects defined with AUTHTYPE(IDPWOS).

OCSP
Displays information only for objects defined with AUTHTYPE(OCSP).

z/OS **QSGDISP**

Specifies the disposition of the objects for which information is to be displayed. Values are:

LIVE This is the default value and displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

ALL Displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with QSGDISP(GROUP).

If QSGDISP(LIVE) is specified or defaulted, or if QSGDISP(ALL) is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

COPY Displays information only for objects defined with QSGDISP(COPY).

GROUP
Displays information only for objects defined with QSGDISP(GROUP). This is allowed only if there is a shared queue manager environment.

PRIVATE
Displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY). Note that QSGDISP(PRIVATE) displays the same information as QSGDISP(LIVE).

QMGR
Displays information only for objects defined with QSGDISP(QMGR).

QSGDISP displays one of the following values:

QMGR
The object was defined with QSGDISP(QMGR).

GROUP
The object was defined with QSGDISP(GROUP).

COPY The object was defined with QSGDISP(COPY).

You cannot use QSGDISP as a filter keyword.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

The default, if no parameters are specified (and the ALL parameter is not specified) is that the object names and their AUTHTYPEs, and, on z/OS, their QSGDISPs, are displayed.

ADOPTCTX
Displays the presented credentials as the context for this application.

ALTDATE
The date on which the definition was last altered, in the form `yyyy-mm-dd`

ALTTIME
The time at which the definition was last altered, in the form `hh.mm.ss`

AUTHENMD
Authentication method. Possible values are:

OS Displays the traditional UNIX password verification method permissions.

PAM
Displays the Pluggable Authentication Method permissions.

You can set the PAM value only on UNIX and Linux platforms.

AUTHORMD
Displays the authorization method. Possible values are:

OS Use operating system groups to determine permissions associated with a user.

SEARCHGRP
A group entry in the LDAP repository contains an attribute listing the Distinguished Name of all users belonging to that group.

SEARCHUSR
A user entry in the LDAP repository contains an attribute listing the Distinguished Name of all the groups to which the specified user belongs.

V 9.0.5 **SRCHGRPSN**
A group entry in the LDAP repository contains an attribute listing the short user name of all users belonging to that group.

AUTHTYPE
The type of the authentication information

BASEDNG
Displays the Base DN for groups.

BASEDNU
Displays the base distinguished name to search for users within the LDAP server.

CHCKLOCL or CHCKCLNT
These attributes are valid only for an **AUTHTYPE** of *IDPWOS* or *IDPWLDAP*. The possible values are:

NONE Displays all locally bound applications that have no user ID and password authentication.

OPTIONAL
Displays the user IDs and passwords provided by an application. Note that it is not mandatory to provide these attributes. This option might be useful during migration, for example.

REQUIRED
Displays all applications providing a valid user ID and password.

REQDADM
Displays privileged users supplying a valid user ID and password, Non-privileged users are treated as with the `OPTIONAL` setting. See also the following note. z/OS (This setting is not allowed on z/OS systems.)

CLASSGRP
Displays the LDAP object class for group records.

CLASSUSR
Displays the LDAP object class for user records within the LDAP repository.

CONNAME
The host name, IPv4 dotted decimal address, or IPv6 hexadecimal notation of the host on which the LDAP server is running. Applies only to objects with AUTHTYPE(CRLLDAP) or AUTHTYPE(IDPWLDAP).

DESCR
Description of the authentication information object.

FAILDLAY
Delay in seconds before an authentication failure is returned to an application.

FINDGRP
Displays the name of the attribute within an LDAP entry to determine group membership.

GRPFIELD
Displays the LDAP attribute that represents a simple name for the group.

LDAPPWD
Password associated with the Distinguished Name of the user on the LDAP server. If nonblank, this is displayed as asterisks z/OS on all platforms except z/OS. Applies only to objects with AUTHTYPE(CRLLDAP) or AUTHTYPE(IDPWLDAP).

LDAPUSER
Distinguished Name of the user on the LDAP server. Applies only to objects with AUTHTYPE(CRLLDAP) or AUTHTYPE(IDPWLDAP).

NESTGRP
Displays whether a group is a member of another group..

OCSPURL
The URL of the OCSP responder used to check for certificate revocation. Applies only to objects with AUTHTYPE(OCSP).

SECCOMM
Displays the method used to connect the LDAP server.

SHORTUSR
Displays the user record being used as a short name.

USRFIELD
Displays the user record being used in the LDAP user record, only if the user ID does not contain a qualifier.

See “Usage notes for DEFINE AUTHINFO” on page 557 for more information about individual parameters.

DISPLAY AUTHREC on Multiplatforms: Multi

Use the MQSC command DISPLAY AUTHREC to display the authority records associated with a profile name.

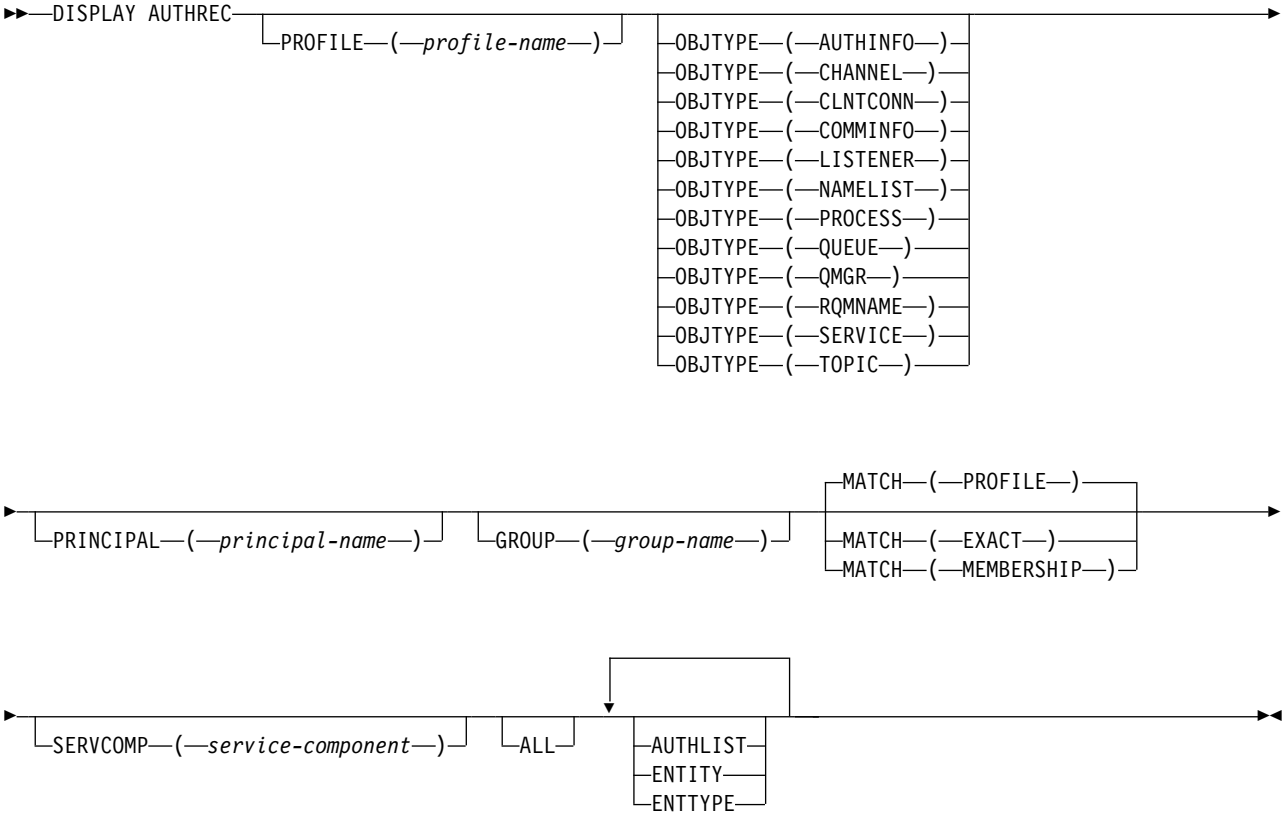
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Parameter descriptions"
- "Requested parameters" on page 746

Synonym: DIS AUTHREC

DISPLAY AUTHREC



Parameter descriptions

PROFILE(*profile-name*)
The name of the object or generic profile for which to display the authority records. If you omit this parameter, all authority records that satisfy the values of the other parameters are displayed.

OBJTYPE
The type of object referred to by the profile. Specify one of the following values:

AUTHINFO
Authentication information record

CHANNEL
Channel

CLNTCONN
Client connection channel

COMMINFO
Communication information object

LISTENER
Listener

NAMELIST
Namelist

PROCESS
Process

QUEUE
Queue

QMGR
Queue manager

RQMNAME
Remote queue manager

SERVICE
Service

TOPIC
Topic

If you omit this parameter, authority records for all object types are displayed.

PRINCIPAL(*principal-name*)
A principal name. This is the name of a user for whom to retrieve authorizations to the specified object. On IBM MQ for Windows, the name of the principal can optionally include a domain name, specified in this format: `user@domain`.

This parameter cannot be specified with GROUP.

GROUP(*group-name*)
A group name. This is the name of the user group on which to make the inquiry. You can specify one name only and it must be the name of an existing user group.

Windows For IBM MQ for Windows only, the group name can optionally include a domain name, specified in the following formats:

```
GroupName@domain
domain\GroupName
```

This parameter cannot be specified with PRINCIPAL.

MATCH
Specify this parameter to control the set of authority records that is displayed. Specify one of the following values:

PROFILE
Return only those authority records which match the specified profile, principal, and group names. This means that a profile of ABCD results in the profiles ABCD, ABC*, and AB* being returned (if ABC* and AB* have been defined as profiles). If the profile name is a generic profile, only authority records which exactly match the specified profile name are returned. If a principal is specified, no profiles are returned for any group in which the principal is a member; only the profiles defined for the specified principal or group.

This is the default value.

MEMBERSHIP
Return only those authority records which match the specified profile, and the entity field of which matches the specified principal and the profiles pertaining to any groups in which the principal is a member that contribute to the cumulative authority for the specified entity.

If this option is specified, the PROFILE and OBJTYPE parameters must also be specified. In addition, either the PRINCIPAL or GROUP parameter must also be supplied. If OBJTYPE(QMGR) is specified, the profile name is optional.

EXACT
Return only those authority records which exactly match the specified profile name and EntityName. No matching generic profiles are returned unless the profile name is, itself, a generic profile. If a principal is specified, no profiles are returned for any group in which the principal is a member; only the profile defined for the specified principal or group.

SERVCOMP(*service-component*)
The name of the authorization service for which information is to be displayed.

If you specify this parameter, it specifies the name of the authorization service to which the authorizations apply. If you omit this parameter, the inquiry is made to the registered authorization services in turn in accordance with the rules for chaining authorization services.

ALL
Specify this parameter to display all of the authorization information available for the entity and the specified profile.

Requested parameters

You can request the following information about the authorizations:

AUTHLIST
Specify this parameter to display the list of authorizations.

ENTITY
Specify this parameter to display the entity name.

ENTTYPE
Specify this parameter to display the entity type.

DISPLAY AUTHSERV: 

Use the MQSC command DISPLAY AUTHSERV to display information about the level of function supported by the installed authorization services.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- “Parameter descriptions” on page 747
- “Requested parameters” on page 747

Synonym: DIS AUTHSERV

DISPLAY AUTHSERV

Parameter descriptions

ALL Specify this parameter to display all the information for each authorization service.

Requested parameters

You can request the following information for the authorization service:

IFVER
Specify this parameter to display the current interface version of the authorization service.

UIDSUPP
Specify this parameter to display whether the authorization service supports user IDs.

DISPLAY CFSTATUS on z/OS: z/OS

Use the MQSC command DISPLAY CFSTATUS to display the status of one or more CF application structures. This command is valid only on IBM MQ for z/OS when the queue manager is a member of a queue-sharing group.

Using MQSC commands

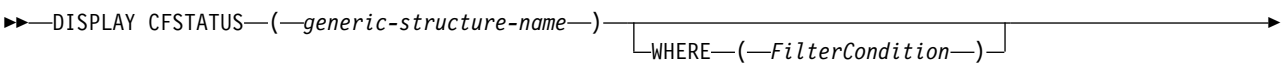
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

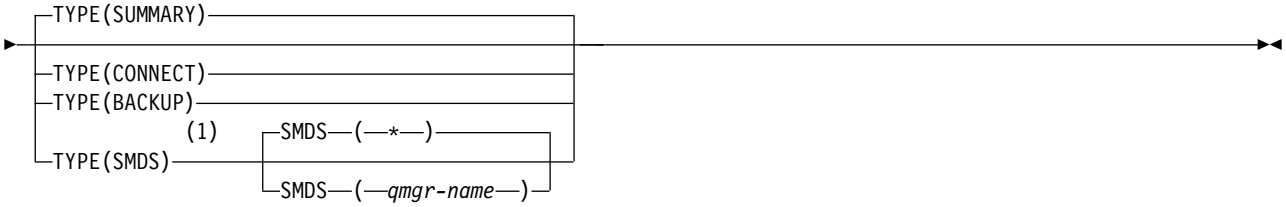
You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Keyword and parameter descriptions for DISPLAY CFSTATUS" on page 748
- "Summary status" on page 750
- "Connection status" on page 751
- "Backup status" on page 752
- "SMDS status" on page 753

Synonym: DIS CFSTATUS

DISPLAY CFSTATUS





Notes:

1 This option is only supported when the CFSTRUCT is defined with OFFLOAD(SMDS).

Keyword and parameter descriptions for DISPLAY CFSTATUS

The name of the application structure for the status information to be displayed must be specified. This can be a specific application structure name or a generic name. By using a generic name, it is possible to display either:

- status information for all application structure definitions
- status information for one or more application structures that match the specified name

The type of status information to be returned can also be specified. This can be:

- summary status information for the application structure in the queue-sharing group
- connection status information for each queue manager in the queue-sharing group for each matching application structure name
- backup status information for each backup taken for each matching application structure defined in the queue-sharing group

(generic-structure-name)

The 12-character name of the CF application structure to be displayed. A trailing asterisk (*) matches all structure names with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all structure names.

The CF structure name must be defined within the queue-sharing group.

The CFSTATUS generic name can be the administration CF structure name (CSQ_ADMIN) or any generic form of this name. Data for this structure, however, is only displayed when TYPE is set to SUMMARY.

WHERE

Specify a filter condition to display status information for those CF application structures that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

`filter-keyword`

Almost any parameter that is returned by this DISPLAY command. However, you cannot use the TYPE parameter as a filter keyword.

`operator`

This is used to determine whether a CF application structure satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which contain the specified item.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not contain the specified item.

CTG Contains an item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use this to display objects the attributes of which match the generic string.

EXG Does not contain any item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not match the generic string.

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE, GE, only. However, if the value is one from a possible set of values returnable on a parameter (for example, the value ACTIVE on the STATUS parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string in the QMNAME parameter) with an asterisk at the end, for example ABC*. The characters must be valid for the attribute you are testing. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.
- An item in a list of values. The value can be explicit or, if it is a character value, it can be explicit or generic. If it is explicit, use CT or EX as the operator. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed. If it is generic, use CTG or EXG as the operator. If ABC* is specified with the operator CTG, all items where one of the attribute values begins with ABC are listed.

TYPE Specifies the type of status information required to be displayed. Values are:

SUMMARY
Display summary status information for each application structure. This is the default.

CONNECT
Display connection status information for each application structure for each active queue manager.

BACKUP
Display backup status information for each application structure.

SMDS
Display shared message data set information.

SMDS

qmgr-name
Specifies the queue manager for which the shared message data set status is to be displayed.

***** Displays the status for all shared message data sets associated with the specified CFSTRUCT except those which have both STATUS(NOTFOUND) and ACCESS(ENABLED).

Summary status

For summary status, the following information is returned for each structure that satisfies the selection criteria:

- The name of the application structure matching the generic name.
- The type of information returned.

CFTYPE
The CF structure type. This is one of the following:

ADMIN
This is the CF administration structure.

APPL
This is a CF application structure.

STATUS
The status of the CF application structure. This is one of the following:

ACTIVE
The structure is active.

FAILED
The structure has failed.

NOTFOUND
The structure is not allocated in the CF, but has been defined to Db2. Check and resolve any messages in the job log about this structure.

INBACKUP
The structure is in the process of being backed-up.

INRECOVER
The structure is in the process of being recovered.

UNKNOWN
The status of the CF structure is not known because, for example, Db2 might be unavailable.

SIZEMAX *(size)*
The size in kilobytes of the application structure.

SIZEUSED *(integer)*
The percentage of the size of the application structure that is in use. Therefore SIZEUSED(25) would indicate that a quarter of the space allocated to this application structure is in use.

ENTSMAX *(integer)*
The number of CF list entries defined for this application structure.

Note: The number does not include any entries that are in storage class memory (SCM), and which might have been allocated to the structure.

ENTSUSED *(integer)*
The number of CF list entries for this application structure that are in use.

Note: The number does not include any entries that are in storage class memory (SCM), and which might have been allocated to the structure.

FAILTIME *(time)*
The time that this application structure failed. The format of this field is `hh.mm.ss`. This parameter is only applicable when the CF structure is in FAILED or INRECOVER state. If the structure is not in a failed state, this is displayed as FAILTIME().

FAILDATE *(date)*
The date that this application-structure failed. The format of this field is yyyy-mm-dd. This parameter is only applicable when the CF structure is in FAILED or INRECOVER state. If the structure is not in a failed state, then this is displayed as FAILDATE().

OFFLDUSE
This indicates whether offloaded large message data potentially exists in shared message data sets, Db2 or both.

When the offload method is switched, the previous offload method needs to remains available for retrieving and deleting old messages, so the OFFLDUSE status is changed to indicate BOTH. When a queue manager disconnects normally from a structure that has OFFLDUSE(BOTH) it checks whether there still are any messages which were stored using the old offload method. If not, it changes the OFFLDUSE status to match the current offload method and issues message CSQE245I to indicate that the switch is complete.

This parameter is one of the following:

NONE
No offloaded large messages are present.

SMDS
Offloaded large messages can exist in shared message data sets.

Db2
Offloaded large messages can exist in Db2.

BOTH
Offloaded large messages can exist both in shared message data sets and in Db2.

Connection status

For connection status, the following information is returned for each connection to each structure that satisfies the selection criteria:

- The name of the application structure matching the generic name.
- The type of information returned.

QMNAME *(qmgrname)*
The queue manager name.

SYSNAME *(systemname)*
The name of the z/OS image of the queue manager that last connected to the application structure. These can be different across queue managers depending on the customer configuration setup.

STATUS
A status indicating whether this queue manager is connected to this application structure. This is one of the following:

ACTIVE
The structure is connected to this queue manager.

FAILED
The queue manager connection to this structure has failed.

NONE
The structure has never been connected to this queue manager.

UNKNOWN
The status of the CF structure is not known.

FAILTIME *(time)*
The time that this queue manager lost connectivity to this application structure. The format of this

field is hh.mm.ss. This parameter is only applicable when the CF structure is in FAILED state. If the structure is not in a failed state, this is displayed as FAILTIME().

FAILDATE *(date)*
The date that this queue manager lost connectivity to this application structure. The format of this field is yyyy-mm-dd. This parameter is only applicable when the CF structure is in FAILED state. If the structure is not in a failed state, this is displayed as FAILDATE().

Backup status

For backup status, the following information is returned for each structure that satisfies the selection criteria:

- The name of the application structure matching the generic name.
- The type of information returned.

STATUS
The status of the CF application structure. This is one of the following:

ACTIVE
The structure is active.

FAILED
The structure has failed.

NONE
The structure is defined as RECOVER(YES), but has never been backed up.

INBACKUP
The structure is in the process of being backed-up.

INRECOVER
The structure is in the process of being recovered.

UNKNOWN
The status of the CF structure is not known.

QMNAME *(qmgrname)*
The name of the queue manager that took the last successful backup for this application structure.

BKUPTIME *(time)*
The end time of the last successful backup taken for this application structure. The format of this field is hh.mm.ss.

BKUPDATE *(date)*
The date of the last successful backup taken for this application structure. The format of this field is yyyy-mm-dd.

BKUPSIZE *(size)*
The size in megabytes of the last successful backup taken for this application structure.

BKUPSRBA *(hexadecimal)*
This is the backup data set start RBA for the start of the last successful backup taken for this application structure.

BKUPERBA *(hexadecimal)*
This is the backup data set end RBA for the end of the last successful backup taken for this application structure.

LOGS *(qmgrname-list)*
This is the list of queue managers, the logs of which are required to perform a recovery.

FAILTIME ***(time)***
The time that this CF structure failed. The format of this field is `hh.mm.ss`. This parameter is only applicable when the CF structure is in FAILED state. If the structure is not in a failed state, this is displayed as FAILTIME().

FAILDATE ***(date)***
The date that this CF structure failed. The format of this field is `yyyy-mm-dd`. This parameter is only applicable when the CF structure is in FAILED state. If the structure is not in a failed state, this is displayed as FAILDATE().

SMDS status

The DISPLAY CFSTATUS command with TYPE(SMDS) displays status information relating to one or more shared message data sets associated with a specific application structure.

The following data is returned for each selected data set:

SMDS
The queue manager name which owns the shared message data set for which properties are being displayed

STATUS
The current status of the shared message data set. This is one of the following:

NOTFOUND
The data set has never been used, or the attempt to open it for the first time failed. Check and resolve any messages in the job log about this structure.

NEW
The data set is being opened and initialized for the first time, ready to be made active.

ACTIVE
The data set is available for normal use.

FAILED
The data set is in an unusable state and probably requires recovery.

INRECOVER
Data set recovery (using RECOVER CFSTRUCT) is in progress.

RECOVERED
The data set has been recovered or otherwise repaired, and is ready for use again, but requires some restart processing the next time it is opened. This restart processing ensures that obsolete references to any deleted messages have been removed from the coupling facility structure before the data set is made available again. The restart processing also rebuilds the data set space map.

EMPTY
The data set contains no messages. The data set is put into this state if it is closed normally by the owning queue manager at a time when it does not contain any messages. It can also be put into EMPTY state when the previous data set contents are to be discarded because the application structure has been emptied (using **RECOVER CFSTRUCT** with TYPE PURGE or, for a nonrecoverable structure only, by deleting the previous instance of the structure). The next time the data set is opened by its owning queue manager, the space map is reset to empty, and the status is changed to ACTIVE. As the previous data set contents are no longer required, a data set in this state can be replaced with a newly allocated data set, for example to change the space allocation or move it to another volume.

ACCESS
The current availability state of the shared message data set. This parameter is one of the following:

ENABLED
The data set can be used, and no error has been detected since the time that it was enabled. If the

data set has STATUS(RECOVERED) it can only be opened by the owning queue manager for restart purposes, but if it has STATUS(ACTIVE) all queue managers can open it.

SUSPENDED
The data set is unavailable because of an error.

This occurs specifically when the STATUS is set to FAILED either because of an error accessing the data set, or using the ALTER SMDS command.

The queue manager can try to enable access again automatically if the error might no longer be present, for example when recovery completes, or if the status is manually set to RECOVERED. Otherwise, it can be enabled again by a command in order to retry the action which originally failed.

DISABLED
The shared message data set cannot be used because it has been explicitly disabled using a command. It can only be enabled again by using another command to enable it. For more information, see "RESET SMDS on z/OS" on page 992.

RCVDATE
The recovery start date.

If recovery is currently enabled for the data set, this indicates the date when it was activated, in the form yyyy-mm-dd. If recovery is not enabled, this is displayed as RCVDATE().

RCVTIME
The recovery start time.

If recovery is currently enabled for the data set, this indicates the time when it was activated, in the form hh.mm.ss. If recovery is not enabled, this is displayed as RCVTIME().

FAILDATE
The failure date.

If the data set was put into a failed state, and has not yet been restored to the active state, this indicates the date when the failure was indicated, in the form yyyy-mm-dd. If the data set is in the active state, this is displayed as FAILDATE().

FAILTIME
The failure time.

If the data set was put into a failed state and has not yet been restored to the active state, this indicates the time when the failure was indicated, in the form hh.mm.ss. If the data set is in the active state, this is displayed as FAILTIME().

DISPLAY CFSTRUCT on z/OS: z/OS

Use the MQSC command DISPLAY CFSTRUCT to display the attributes of one or more CF application structures. This command is valid only on z/OS when the queue manager is a member of a queue-sharing group.

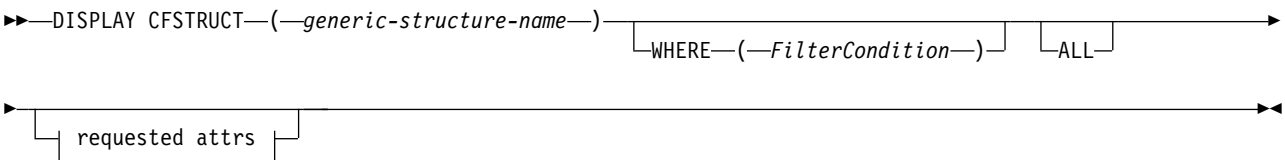
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

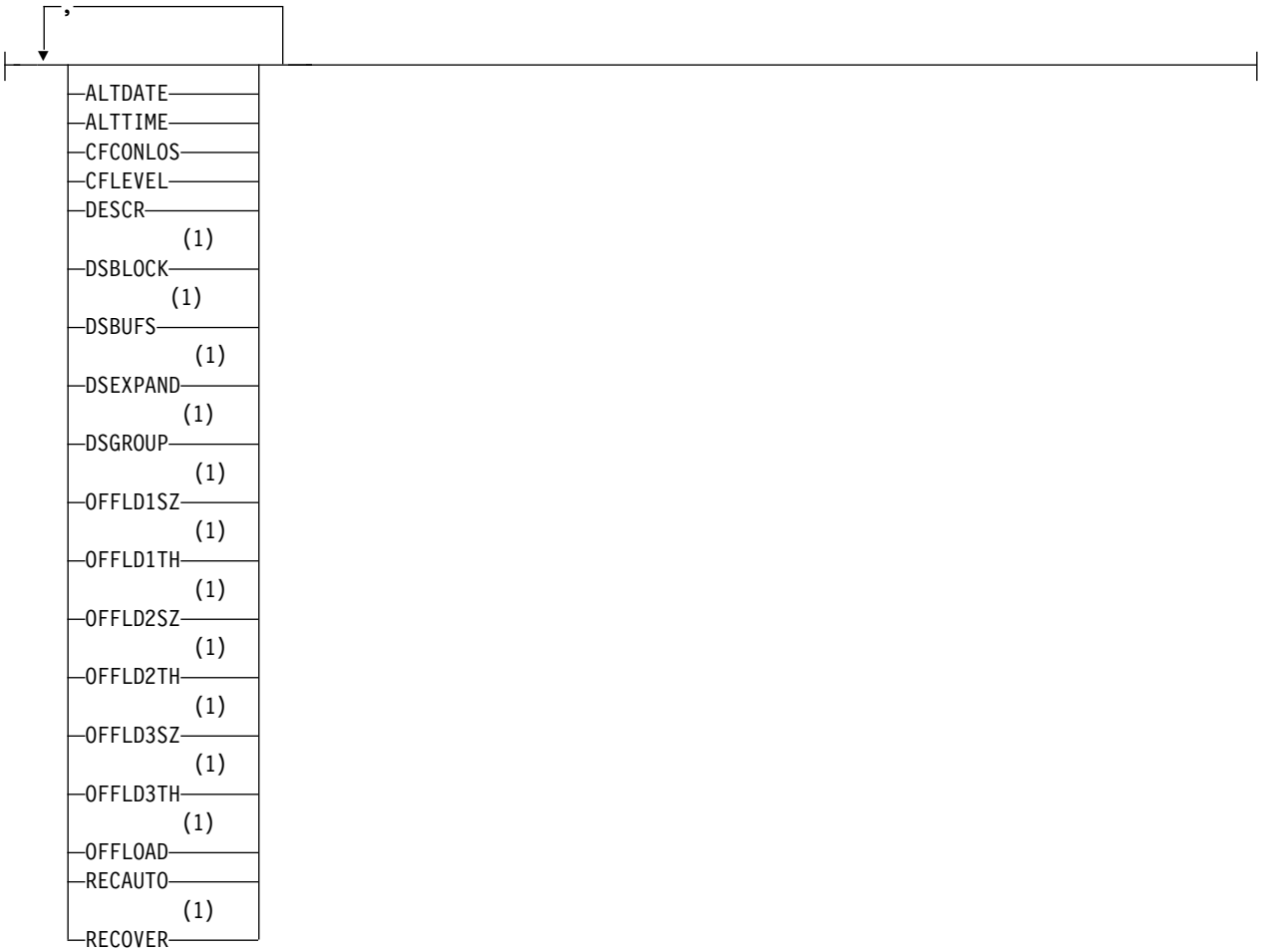
You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

Synonym: DIS CFSTRUCT

DISPLAY CFSTRUCT



Requested attrs:



Notes:

1 z/OS For more information about this parameter, see Planning your coupling facility and offload storage environment.

Usage notes for DISPLAY CFSTRUCT

1. The command cannot specify the CF administration structure (CSQ_ADMIN).

Keyword and parameter descriptions for DISPLAY CFSTRUCT

The name of the application structure to be displayed must be specified. This can be a specific application structure name or a generic name. By using a generic name, it is possible to display either:

- all application structure definitions
- one or more application structures that match the specified name

(*generic-structure-name*)

The 12-character name of the CF application structure to be displayed. A trailing asterisk (*) matches all structure names with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all structure names.

The CF structure name must be defined within the queue-sharing group.

WHERE

Specify a filter condition to display only those CF application structures that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword

Any parameter that can be used to display attributes for this DISPLAY command.

operator

This is used to determine whether a CF application structure satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value

The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use any of the operators except LK and NL. However, if the value is one from a possible set of values returnable on a parameter (for example, the value YES on the RECOVER parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. The characters must be valid for the attribute you are testing. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

  You can only use operators LK or NL for generic values on the DISPLAY CFSTRUCT command.

ALL Specify this to display all attributes. If this keyword is specified, any attributes that are requested specifically have no effect; all attributes are still displayed.

This is the default behavior if you do not specify a generic name and do not request any specific attributes.

Requested parameters

Specify one or more attributes that define the data to be displayed. The attributes can be specified in any order. Do not specify the same attribute more than once.

The default, if no parameters are specified (and the ALL parameter is not specified) is that the structure names are displayed.

ALTDATE

The date on which the definition was last altered, in the form `yyyy-mm-dd`.

ALTTIME

The time at which the definition was last altered, in the form `hh.mm.ss`.

CFCONLOS
The action to be taken when the queue manager loses connectivity to the CF application structure.

CFLEVEL
Indicates the functional capability level for this CF application structure.

DESCR
Descriptive comment.

DSBLOCK
The logical block size, which is the unit in which shared message data set space is allocated to individual queues.

DSBUFS
The number of buffers allocated in each queue manager for accessing shared message data sets.

DSEXPAND
Whether the queue manager expands a shared message data set.

DSGROUP
The generic data set name to be used for the group of shared message data sets.

OFFLD1SZ
Offload rule 1: The message size value specifying an integer followed by K, giving the number of kilobytes.

OFFLD1TH
Offload rule 1: The coupling facility structure percentage usage threshold value as an integer.

OFFLD2SZ
Offload rule 2: The message size value specifying an integer followed by K, giving the number of kilobytes.

OFFLD2TH
Offload rule 2: The coupling facility structure percentage usage threshold value as an integer.

OFFLD3SZ
Offload rule 3: The message size value specifying an integer followed by K, giving the number of kilobytes.

OFFLD3TH
Offload rule 3: The coupling facility structure percentage usage threshold value as an integer.

OFFLOAD
If the CFLEVEL is less than 4, the only value you can display is NONE.

If the CFLEVEL is 4, the only value can display is Db2.

If the CFLEVEL is 5, the values displayed are Db2, SMDS, or BOTH. These values depict whether offloaded message data is stored in a group of shared message data sets, or in Db2, or both.

In addition, the offload rules parameter values for OFFLD1SZ, OFFLD1TH, OFFLD2SZ, OFFLD2TH, OFFLD3SZ, and OFFLD3TH are displayed.

RECAUTO
Indicates whether automatic recovery action is taken when a queue manager detects that the structure is failed, or when a queue manager loses connectivity to the structure and no systems in the SysPlex have connectivity to the Coupling Facility that the structure is allocated in. Values are:

YES
The structure and associated shared message data sets which also need recovery are automatically recovered.

NO The structure is not automatically recovered.

RECOVER
Indicates whether CF recovery for the application structure is supported. Values are:

NO CF application structure recovery is not supported.

YES
CF application structure recovery is supported.

DISPLAY CHANNEL:

Use the MQSC command DISPLAY CHANNEL to display a channel definition.

Using MQSC commands

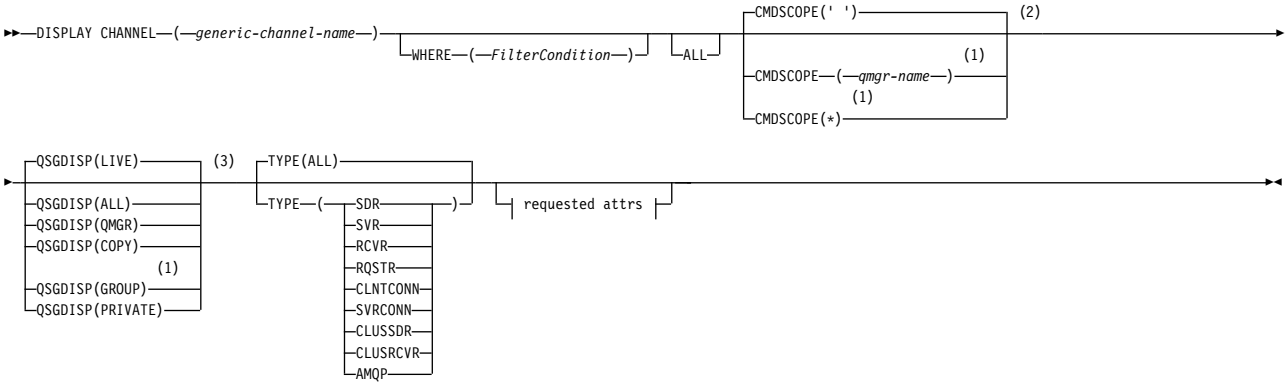
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

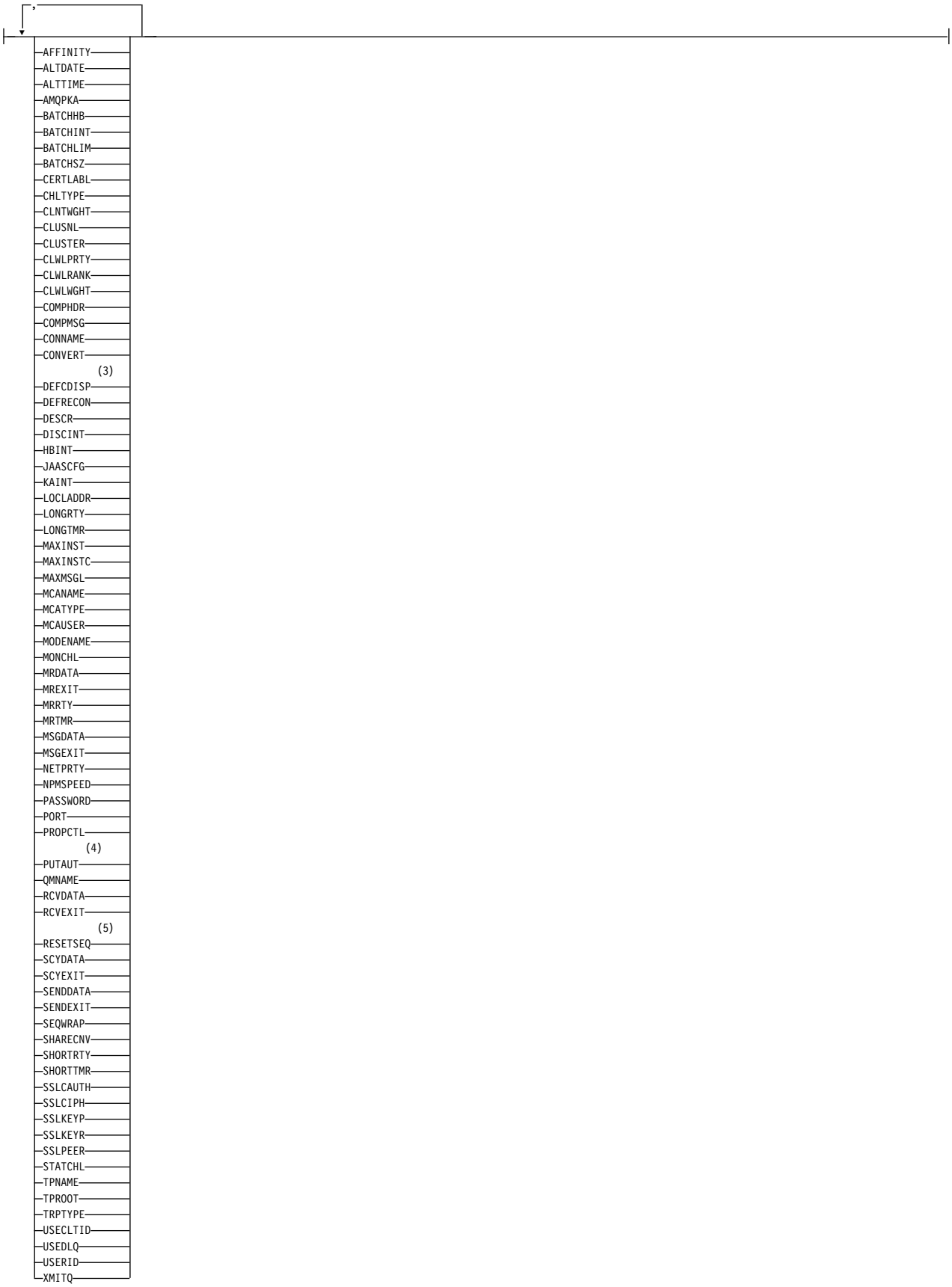
- Syntax diagram
- "Usage notes" on page 761
- "Parameter descriptions for DISPLAY CHANNEL" on page 761
- "Requested parameters" on page 764

Synonym: DIS CHL

DISPLAY CHANNEL



Requested attrs:


,
AFFINITY
ALTDATE
ALTTIME
AMQPKA
BATCHHB
BATCHINT
BATCHLIM
BATCHSZ
CERTLABL
CHLTYPE
CLNTWGHT
CLUSNL
CLUSTER
CLWLPRTY
CLWLRANK
CLWLWGHT
COMPHDR
COMPMSG
CONNAME
CONVERT
(3)
DEFCDISP
DEFRECON
DESCR
DISCINT
HBINT
JAASCFG
KAINT
LOCLADDR
LONGRTY
LONGTMR
MAXINST
MAXINSTC
MAXMSGL
MCANAME
MCATYPE
MCAUSER
MODENAME
MONCHL
MRDATA
MREXIT
MRRTY
MRTMR
MSGDATA
MSGEXIT
NETPRTY
NPMSPEED
PASSWORD
PORT
PROPCTL
(4)
PUTAUT
QMNAME
RCVDATA
RCVEXIT
(5)
RESETSEQ
SCYDATA
SCYEXIT
SENDDATA
SENDEXIT
SEQWRAP
SHARECNV
SHORTRTY
SHORTTMR
SSLCAUTH
SSLCIPH
SSLKEYP
SSLKEYR
SSLPEER
STATCHL
TPNAME
TPROOT
TRPTYPE
USECLTID
USEDLQ
USERID
XMITQ

Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Not valid for z/OS client-connection channels.

3 Valid only on z/OS.

4 Valid only for RCVR, RQSTR, CLUSRCVR and (for z/OS only) SVRCONN channel types.

5 Not valid on z/OS.

Usage notes

You can only display cluster-sender channels if they were created manually. See Cluster channels.

The values shown describe the current definition of the channel. If the channel has been altered since it was started, any currently running instance of the channel object might not have the same values as the current definition.

Parameter descriptions for DISPLAY CHANNEL

You must specify the name of the channel definition you want to display. It can be a specific channel name or a generic channel name. By using a generic channel name, you can display either:

- All channel definitions
- One or more channel definitions that match the specified name

(generic-channel-name)
The name of the channel definition to be displayed (see Rules for naming IBM MQ objects). A trailing asterisk (*) matches all channel definitions with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all channel definitions.

WHERE
Specify a filter condition to display only those channels that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

`filter-keyword`
Almost any parameter that can be used to display attributes for this DISPLAY command. However, you cannot use the CMDSCOPE, QSGDISP, or MCANAME parameters as filter keywords. You cannot use TYPE (or CHLTYPE) if it is also used to select channels. Channels of a type for which the filter keyword is not a valid attribute are not displayed.

`operator`
This is used to determine whether a channel satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which contain the specified item.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not contain the specified item.

CTG Contains an item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use this to display objects the attributes of which match the generic string.

EXG Does not contain any item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not match the generic string.

filter-value

The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE, or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value SDR on the TYPE parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. The characters must be valid for the attribute you are testing. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.
- An item in a list of values. The value can be explicit or, if it is a character value, it can be explicit or generic. If it is explicit, use CT or EX as the operator. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed. If it is generic, use CTG or EXG as the operator. If ABC* is specified with the operator CTG, all items where one of the attribute values begins with ABC are listed.

ALL Specify ALL to display the results of querying all the parameters. If ALL is specified, any request for a specific parameter is ignored. The result of querying with ALL is to return the results for all of the possible parameters.

This is the default, if you do not specify a generic name and do not request any specific parameters.

z/OS On z/OS this is also the default if you specify a filter condition using the WHERE parameter, but on other platforms, only requested attributes are displayed.

If no parameters are specified (and the ALL parameter is not specified or defaulted), the default is that the channel names only are displayed.

z/OS On z/OS, the CHLTYPE and QSGDISP values are also displayed.

z/OS **CMDSCOPE**

This parameter specifies how the command is executed when the queue manager is a member of a queue-sharing group.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name

The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

z/OS **QSGDISP**

Specifies the disposition of the objects for which information is to be displayed. Values are:

LIVE This is the default value and displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

ALL Displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with QSGDISP(GROUP).

If QSGDISP(ALL) is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

Note: In the QSGDISP(LIVE) case, this occurs only where a shared and a non-shared queue have the same name; such a situation should not occur in a well-managed system.

In a shared queue manager environment, use

```
DISPLAY CHANNEL(name) CMDSCOPE(*) QSGDISP(ALL)
```

to list ALL objects matching

```
name
```

in the queue-sharing group without duplicating those in the shared repository.

COPY Display information only for objects defined with QSGDISP(COPY).

GROUP
Display information only for objects defined with QSGDISP(GROUP). This is allowed only if there is a shared queue manager environment.

PRIVATE
Display information only for objects defined with QSGDISP(QMGR) or QSGDISP(COPY). Note that QSGDISP(PRIVATE) displays the same information as QSGDISP(LIVE).

QMGR
Display information only for objects defined with QSGDISP(QMGR).

QSGDISP displays one of the following values:

QMGR
The object was defined with QSGDISP(QMGR).

GROUP
The object was defined with QSGDISP(GROUP).

COPY The object was defined with QSGDISP(COPY).

You cannot use QSGDISP as a filter keyword.

TYPE This is optional. It can be used to restrict the display to channels of one type.

The value is one of the following:

ALL Channels of all types are displayed (this is the default).

SDR Sender channels only are displayed.

SVR Server channels only are displayed.

RCVR Receiver channels only are displayed.

RQSTR
Requester channels only are displayed.

CLNTCONN
Client-connection channels only are displayed.

SVRCONN
Server-connection channels only are displayed.

CLUSSDR
Cluster-sender channels only are displayed.).

CLUSRCVR
Cluster-receiver channels only are displayed.).

V 9.0.0 **AMQP**
AMQP channels only are displayed.

CHLTYPE(*type*) can be used as a synonym for this parameter. ,

Requested parameters

Specify one or more DISPLAY CHANNEL parameters that define the data to be displayed. You can specify the parameters in any order, but do not specify the same parameter more than once.

Some parameters are relevant only for channels of a particular type or types. Attributes that are not relevant for a particular type of channel cause no output, nor is an error raised. The following table shows the parameters that are relevant for each type of channel. There is a description of each parameter after the table. Parameters are optional unless the description states that they are required.

Table 109. Parameters that result in data being returned from the DISPLAY CHANNEL command

Parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR- CONN	CLUS- SDR	CLUS-RCVR	AMQP V 9.0.0
AFFINITY					✓				
ALTDATE	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
ALTTIME	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
AMQPKA									✓ 9.0.0
AUTOSTART		✓	✓	✓		✓			
BATCHHB	✓	✓					✓	✓	
BATCHINT	✓	✓					✓	✓	
BATCHLIM	✓	✓					✓	✓	
BATCHSZ	✓	✓	✓	✓			✓	✓	
CERTLABL	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
channel-name	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0

Table 109. Parameters that result in data being returned from the DISPLAY CHANNEL command (continued)

Parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR- CONN	CLUS- SDR	CLUS-RCVR	AMQP V 9.0.0
CHLTYPE	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
CLNTWGHT					✓				
CLUSNL							✓	✓	
CLUSTER							✓	✓	
CLWLPRTY							✓	✓	
CLWLRANK							✓	✓	
CLWLWGHT							✓	✓	
COMPHDR	✓	✓	✓	✓	✓	✓	✓	✓	
COMPMSG	✓	✓	✓	✓	✓	✓	✓	✓	
CONNAME	✓	✓		✓	✓		✓	✓	
CONVERT	✓	✓					✓	✓	
DEFCDISP	✓	✓	✓	✓		✓			
DEFRECON					✓				
DESCR	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
DISCINT	✓	✓				✓	✓	✓	
HBINT	✓	✓	✓	✓	✓	✓	✓	✓	
KAINT	✓	✓	✓	✓	✓	✓	✓	✓	
LOCLADDR	✓	✓		✓	✓		✓	✓	✓ 9.0.0
LONGRTY	✓	✓					✓	✓	
LONGTMR	✓	✓					✓	✓	
MAXINST						✓			✓ 9.0.0
MAXINSTC						✓			
MAXMSGL	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
MCANAME	✓	✓		✓			✓	✓	

Table 109. Parameters that result in data being returned from the DISPLAY CHANNEL command (continued)

Parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR- CONN	CLUS- SDR	CLUS-RCVR	AMQP V 9.0.0
MCATYPE	✓	✓		✓			✓	✓	
MCAUSER	✓	✓	✓	✓		✓	✓	✓	✓ 9.0.0
MODENAME	✓	✓		✓	✓		✓	✓	
MONCHL	✓	✓	✓	✓		✓	✓	✓	
MRDATA			✓	✓				✓	
MREXIT			✓	✓				✓	
MRRTY			✓	✓				✓	
MRTMR			✓	✓				✓	
MSGDATA	✓	✓	✓	✓			✓	✓	
MSGEXIT	✓	✓	✓	✓			✓	✓	
NETPRTY								✓	
NPMSPEED	✓	✓	✓	✓			✓	✓	
PASSWORD	✓	✓		✓	✓		✓		
PORT									✓ 9.0.0
PROPCTL	✓	✓					✓	✓	
PUTAUT			✓	✓		✓ [1]		✓	
QMNAME					✓				
RESETSEQ	✓	✓	✓	✓			✓	✓	
RCVDATA	✓	✓	✓	✓	✓	✓	✓	✓	
RCVEXIT	✓	✓	✓	✓	✓	✓	✓	✓	
SCYDATA	✓	✓	✓	✓	✓	✓	✓	✓	
SCYEXIT	✓	✓	✓	✓	✓	✓	✓	✓	
SENDDATA	✓	✓	✓	✓	✓	✓	✓	✓	
SENDEXIT	✓	✓	✓	✓	✓	✓	✓	✓	

Table 109. Parameters that result in data being returned from the DISPLAY CHANNEL command (continued)

Parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR- CONN	CLUS- SDR	CLUS-RCVR	AMQP V 9.0.0
SEQWRAP	✓	✓	✓	✓			✓	✓	
SHARECNV						✓			
SHORTRTY	✓	✓					✓	✓	
SHORTTMR	✓	✓					✓	✓	
SSLCAUTH		✓	✓	✓		✓		✓	✓ V 9.0.0
SSLCIPH	✓	✓	✓	✓	✓	✓	✓	✓	✓ V 9.0.0
SSLPEER	✓	✓	✓	✓	✓	✓	✓	✓	✓ V 9.0.0
STATCHL	✓	✓	✓	✓			✓	✓	
TPNAME	✓	✓		✓	✓	✓	✓	✓	
TPROOT									✓ V 9.0.0
TRPTYPE	✓	✓	✓	✓	✓	✓	✓	✓	
USECLTID									✓ V 9.0.0
USEDLQ	✓	✓	✓	✓			✓	✓	
USERID	✓	✓		✓	✓		✓		
XMITQ	✓	✓							

Note:

1. PUTAUT is valid for a channel type of SVRCONN on z/OS only.

AFFINITY
The channel affinity attribute.

PREFERRED
Subsequent connections in a process attempt to use the same channel definition as the first connection.

NONE
All connections in a process select an applicable definition based on the weighting with any applicable CLNTWGHT(0) definitions selected first in alphabetical order.

ALTDATE
The date on which the definition was last altered, in the form `yyyy-mm-dd`.

ALTTIME
The time at which the definition was last altered, in the form `hh.mm.ss`.

V 9.0.0 **AMQPKA**
The keep alive time for an AMQP channel in seconds.

AUTOSTART
Whether an LU 6.2 responder process should be started for the channel.

BATCHHB
The batch heartbeating value being used.

BATCHINT
Minimum batch duration.

BATCHLIM
Batch data limit.

The limit of the amount of data that can be sent through a channel.

BATCHSZ
Batch size.

CERTLABL
Certificate label.

CHLTYPE
Channel type.

The channel type is always displayed if you specify a generic channel name and do not request any other parameters. On z/OS, the channel type is always displayed.

Multi On Multiplatforms, TYPE can be used as a synonym for this parameter.

CLNTWGHT
The client channel weighting.

The special value 0 indicates that no random load balancing is performed and applicable definitions are selected in alphabetical order. If random load balancing is performed the value is in the range 1 - 99 where 1 is the lowest weighting and 99 is the highest.

CLUSTER
The name of the cluster to which the channel belongs.

CLUSNL
The name of the namelist that specifies the list of clusters to which the channel belongs.

CLWLPRTY
The priority of the channel for the purposes of cluster workload distribution.

CLWLRANK
The rank of the channel for the purposes of cluster workload distribution.

CLWLWGHT
The weighting of the channel for the purposes of cluster workload distribution.

COMPHDR
The list of header data compression techniques supported by the channel. For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference.

COMPMSG
The list of message data compression techniques supported by the channel. For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference.

CONNAME
Connection name.

CONVERT
Whether sender should convert application message data.

DEFCDISP
Specifies the default channel disposition of the channels for which information is to be returned. If this keyword is not present, channels of all default channel dispositions are eligible.

ALL Channels of all default channel dispositions are displayed.

This is the default setting.

PRIVATE
Only channels where the default channel disposition is PRIVATE are displayed.

SHARED
Only channels where the default channel disposition is FIXSHARED or SHARED are displayed.

Note: This does not apply to client-connection channel types on z/OS.

DESCR
Default client reconnection option.

DESCR
Description.

DISCINT
Disconnection interval.

HBINT
Heartbeat interval.

KAINT
KeepAlive timing for the channel.

LOCLADDR
Local communications address for the channel.

LONGRTY
Long retry count.

LONGTMR
Long retry timer.

MAXINST(*integer*)
The maximum number of instances of a server-connection channel that are permitted to run simultaneously.

MAXINSTC(*integer*)
The maximum number of instances of a server-connection channel, started from a single client, that are permitted to run simultaneously.

Note: In this context, connections originating from the same remote network address are regarded as coming from the same client.

MAXMSGL
Maximum message length for channel.

MCANAME
Message channel agent name.

You cannot use MCANAME as a filter keyword.

MCATYPE
Whether message channel agent runs as a separate process or a separate thread.

MCAUSER
Message channel agent user identifier.

MODENAME
LU 6.2 mode name.

MONCHL
Online monitoring data collection.

MRDATA
Channel message-retry exit user data.

MREXIT
Channel message-retry exit name.

MRRTY
Channel message-retry count.

MRTMR
Channel message-retry time.

MSGDATA
Channel message exit user data.

MSGEXIT
Channel message exit names.

NETPRTY
The priority for the network connection.

NPMSPEED
Nonpersistent message speed.

PASSWORD

Password for initiating LU 6.2 session. If nonblank, this is displayed as asterisks z/OS on all platforms except z/OS.

V 9.0.0 **PORT**
The port number used to connect an AMQP channel.

PROPCTL
Message property control.

Specifies what happens to properties of messages when the message is about to be sent to a V6 or prior queue manager (a queue manager that does not understand the concept of a property descriptor).

This parameter is applicable to Sender, Server, Cluster Sender, and Cluster Receiver channels.

This parameter is optional.

Permitted values are:

COMPAT
This is the default value.

Message properties	Result
The message contains a property with a prefix of **mcd.**, **jms.**, **usr.** or **mqext.**	All optional message properties (where the **Support** value is MQPD_SUPPORT_OPTIONAL), except those in the message descriptor or extension, are placed in one or more MQRFH2 headers in the message data before the message it sent to the remote queue manager.
The message does not contain a property with a prefix of **mcd.**, **jms.**, **usr.** or **mqext.**	All message properties, except those in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.
The message contains a property where the **Support** field of the property descriptor is not set to MQPD_SUPPORT_OPTIONAL	The message is rejected with reason MQRC_UNSUPPORTED_PROPERTY and treated in accordance with its report options.
The message contains one or more properties where the **Support** field of the property descriptor is set to MQPD_SUPPORT_OPTIONAL but other fields of the property descriptor are set to non-default values	The properties with non-default values are removed from the message before the message is sent to the remote queue manager.
The MQRFH2 folder that would contain the message property needs to be assigned with the *content='properties'* attribute	The properties are removed to prevent MQRFH2 headers with unsupported syntax flowing to a V6 or prior queue manager.

NONE
All properties of the message, except those in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.

If the message contains a property where the **Support** field of the property descriptor is not set to MQPD_SUPPORT_OPTIONAL then the message is rejected with reason MQRC_UNSUPPORTED_PROPERTY and treated in accordance with its report options.

ALL All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except those in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

PUTAUT
Put authority.

QMNAME
Queue manager name.

RESETSEQ
Pending reset sequence number.

This is the sequence number from an outstanding request and it indicates a user RESET CHANNEL command request is outstanding.

A value of zero indicates that there is no outstanding RESET CHANNEL. The value can be in the range 1 - 999999999.

This parameter is not applicable on z/OS.

RCVDATA
Channel receive exit user data.

RCVEXIT
Channel receive exit names.

SCYDATA
Channel security exit user data.

SCYEXIT
Channel security exit names.

SENDDATA
Channel send exit user data.

SENDEXIT
Channel send exit names.

SEQWRAP
Sequence number wrap value.

SHARECNV
Sharing conversations value.

SHORTRTY
Specifies the maximum number of times that the channel is to try allocating a session to its partner.

SHORTTMR
Short retry timer.

SSLCAUTH
Whether TLS client authentication is required.

SSLCIPH
Cipher specification for the TLS connection.

SSLPEER
Filter for the Distinguished Name from the certificate of the peer queue manager or client at the other end of the channel.

STATCHL
Statistics data collection.

TPNAME
LU 6.2 transaction program name.

V 9.0.0 **TPROOT**
The topic root for an AMQP channel.

TRPTYPE
Transport type.

V 9.0.0 **USECLTID**
Specifies that the client ID should be used for authorization checks for an AMQP channel, instead of the MCAUSER attribute value.

USEDLQ
Determines whether the dead-letter queue is used when messages cannot be delivered by channels.

USERID
User identifier for initiating LU 6.2 session.

XMITQ
Transmission queue name.

For more details of these parameters, see “DEFINE CHANNEL” on page 575.

DISPLAY CHANNEL (MQTT): AIX Linux Windows

Use the MQSC command DISPLAY CHANNEL (MQTT) to display an MQ Telemetry channel definition.

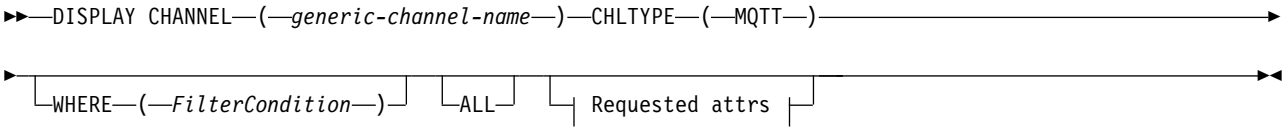
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Parameter descriptions for DISPLAY CHANNEL (MQTT)"
- "Requested parameters" on page 775

Synonym: DIS CHL

DISPLAY CHANNEL (MQTT)



Requested attrs:

DISPLAY CHANNEL (MQTT) command is only valid for MQ Telemetry channels.

Parameter descriptions for DISPLAY CHANNEL (MQTT)

You must specify the name of the channel definition you want to display. This can be a specific channel name or a generic channel name. By using a generic channel name, you can display either:

- All channel definitions
- One or more channel definitions that match the specified name

(generic-channel-name)

The name of the channel definition to be displayed (see Rules for naming IBM MQ objects). A trailing asterisk (*) matches all channel definitions with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all channel definitions.

CHLTYPE(*type* **)**

The value is always `MQTT`.

TYPE can be used as a synonym for this parameter.

WHERE

Specify a filter condition to display only those channels that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

`filter-keyword`
Almost any parameter that can be used to display attributes for this DISPLAY command. However, you cannot use the CMDSCOPE, QSGDISP, or MCANAME parameters as filter keywords. You cannot use TYPE (or CHLTYPE) if it is also used to select channels. Channels of a type for which the filter keyword is not a valid attribute are not displayed.

`operator`
This is used to determine whether a channel satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which contain the specified item.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not contain the specified item.

CTG Contains an item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use this to display objects the attributes of which match the generic string.

EXG Does not contain any item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not match the generic string.

`filter-value`
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE, or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value SDR on the TYPE parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. The characters must be valid for the attribute you are testing. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

- An item in a list of values. The value can be explicit or, if it is a character value, it can be explicit or generic. If it is explicit, use CT or EX as the operator. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed. If it is generic, use CTG or EXG as the operator. If ABC* is specified with the operator CTG, all items where one of the attribute values begins with ABC are listed.

ALL

Specify ALL to display the results of querying all the parameters. If ALL is specified, any request for a specific parameter is ignored. The result of querying with ALL is to return the results for all of the possible parameters.

This is the default, if you do not specify a generic name and do not request any specific parameters.

If no parameters are specified (and the ALL parameter is not specified or defaulted), the default is that the channel names only are displayed.

Requested parameters

Specify one or more DISPLAY CHANNEL parameters that define the data to be displayed. You can specify the parameters in any order, but do not specify the same parameter more than once.

Some parameters are relevant only for channels of a particular type or types. Attributes that are not relevant for a particular type of channel cause no output, nor is an error raised. The following table shows the parameters that are relevant for each type of channel. There is a description of each parameter after the table. Parameters are optional unless the description states that they are required.

BACKLOG

The number of outstanding connection requests that the telemetry channel can support at any one time. When the backlog limit is reached, any further clients trying to connect will be refused connection until the current backlog is processed. The value is in the range 0 - 999999999. The default value is 4096.

CHLTYPE

Channel type.

There is only one valid value for this parameter: MQTT.

JAASCFG

The name of a stanza in the JAAS configuration file.

LOCLADDR

The local communications address for the channel.

MCAUSER

The message channel agent user identifier.

PORT

The port number on which the telemetry (MQXR) service accepts client connections.

PROTOCOL

The communication protocol supported by the channel.

SSLCAUTH

Defines whether IBM MQ requires a certificate from the TLS client.

SSLCIPH

When **SSLCIPH** is used with a telemetry channel, it means TLS Cipher Suite.

SSLKEYP

The store for digital certificates and their associated private keys. If you do not specify a key file, TLS is not used.

SSLKEYR

The name of the TLS key repository. For full details, see the SSLKEYR parameter of the ALTER QMGR command.

TRPTYPE

The transmission protocol to be used. For a Telemetry channel, this is always TCP (that is, the TCP/IP protocol).

USECLTID

Indicates whether you want to use the MQTT client ID for the connection as the IBM MQ user ID for that connection.

For more details of these parameters, see "DEFINE CHANNEL (MQTT)" on page 634.

DISPLAY CHINIT on z/OS: z/OS

Use the MQSC command DISPLAY CHINIT to display information about the channel initiator. The command server must be running.

Using MQSC commands

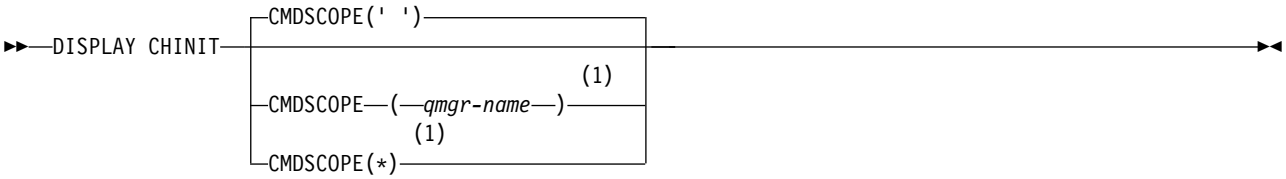
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for DISPLAY CHINIT"
- "Parameter descriptions for DISPLAY CHINIT" on page 777

Synonym: DIS CHI or DIS DQM

DISPLAY CHINIT



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

Usage notes for DISPLAY CHINIT

1. The response to this command is a series of messages showing the current status of the channel initiator. This includes the following:
   - Whether the channel initiator is running or not

- Which listeners are started, and information about them.
- How many dispatchers are started, and how many were requested
- How many adapter subtasks are started, and how many were requested
- How many TLS subtasks are started, and how many were requested
- The TCP system name
- How many channel connections are current, and whether they are active, stopped, or retrying
- The maximum number of current connections

Parameter descriptions for DISPLAY CHINIT

CMDSCOPE

This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

''
The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

*
The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

DISPLAY CHLAUTH:

Use the MQSC command DISPLAY CHLAUTH to display the attributes of a channel authentication record.

Using MQSC commands

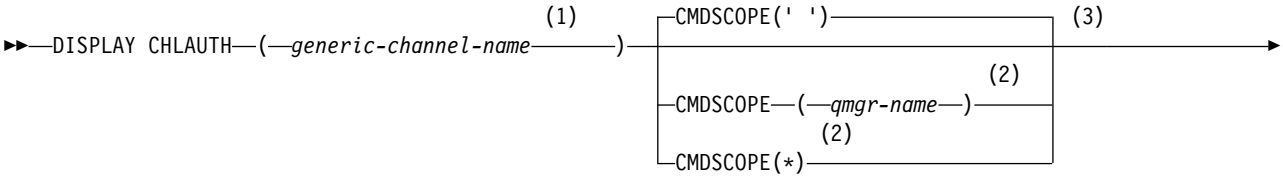
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

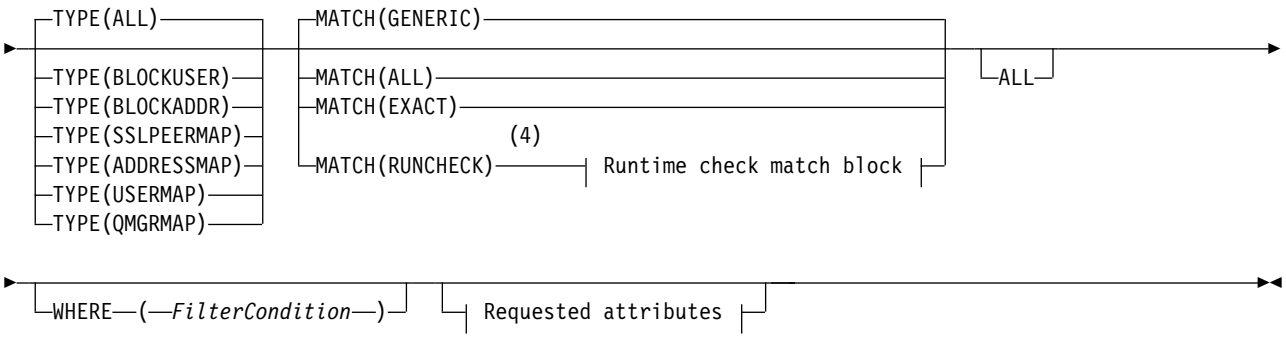
z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- Parameters

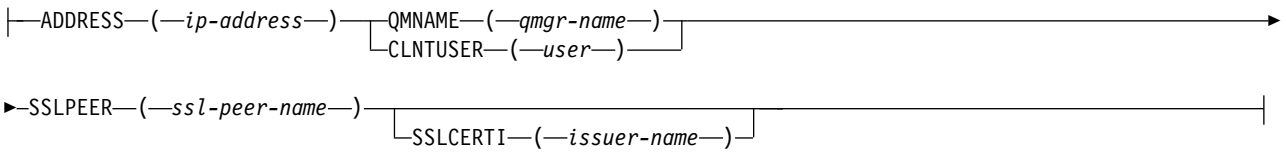
Synonym: DIS CHLAUTH

DISPLAY CHLAUTH





Runtime check match block:



Requested attributes:

Notes:

1 Must be * with TYPE(BLOCKADDR) and cannot be generic with MATCH(RUNCHECK)

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Must be combined with TYPE(ALL)

Parameters

generic-channel-name
The name of the channel or set of channels to display. You can use the asterisk (*) as a wildcard to specify a set of channels. When an asterisk is used on z/OS, single quotes must be used around the whole value. When **MATCH** is RUNCHECK this parameter must not be generic.

ADDRESS
The IP address to be matched.

This parameter is valid only when **MATCH** is RUNCHECK, must not be generic and must not be a host name.

ALL
Specify this parameter to display all attributes. If this keyword is specified, any attributes that are requested specifically have no effect; all attributes are still displayed.

This is the default behavior if you do not specify a generic name and do not request any specific attributes.

CLNTUSER
The client asserted user ID to be mapped to a new user ID, allowed through unchanged, or blocked.

This can be the user ID flowed from the client indicating the user ID the client side process is running under, or the user ID presented by the client on an MQCONNX call using MQCSP.

This parameter is valid only with TYPE(USERMAP) and when **Match** is MQMATCH_RUNCHECK.

The maximum length of the string is MQ_CLIENT_USER_ID_LENGTH.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command is run when the queue manager is a member of a queue-sharing group.

' ' The command is run on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is run on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command is run on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect is the same as entering the command on every queue manager in the queue-sharing group.

MATCH
Indicates the type of matching to be applied.

RUNCHECK
Returns the record that is matched by a specific inbound channel at run time if it connects to this queue manager. The specific inbound channel is described by providing values that are not generic:

- Channel name.
- **ADDRESS** attribute containing an IP address, that is then reverse looked up as part of running the command to discover the host name, if the queue manager is configured with **REVDNS(ENABLED)**.
- **SSLCERTI** attribute, only if the inbound channel uses TLS.
- **SSLPEER** attribute, only if the inbound channel uses TLS.
- **QMNAME** or **CLNTUSER** attribute, depending on whether the inbound channel is a client or queue manager channel.

If the record discovered has **WARN** set to YES, a second record might also be displayed to show the actual record the channel will use at run time. This parameter must be combined with **TYPE(ALL)**.

EXACT
Return only those records which exactly match the channel profile name supplied. If there are no asterisks in the channel profile name, this option returns the same output as MATCH(GENERIC).

GENERIC
Any asterisks in the channel profile name are treated as wildcards. If there are no asterisks in the channel profile name, this returns the same output as MATCH(EXACT). For example, a profile of ABC* could result in records for ABC, ABC*, and ABCD being returned.

ALL Return all possible records that match the channel profile name supplied. If the channel name is generic in this case, all records that match the channel name are returned even if more specific matches exist. For example, a profile of SYSTEM.*.SVRCONN could result in records for SYSTEM.*, SYSTEM.DEF.*, SYSTEM.DEF.SVRCONN, and SYSTEM.ADMIN.SVRCONN being returned.

QMNAME
The name of the remote partner queue manager to be matched

This parameter is valid only when **MATCH** is RUNCHECK and must not be generic.

SSLCERTI

The Certificate issuer Distinguished Name of the certificate to be matched.

The **SSLCERTI** field, if not blank, is matched in addition to the **SSLPEER** value.

This parameter is valid only when **MATCH** is RUNCHECK and must not be generic.

SSLPEER

The Subject Distinguished Name of the certificate to be matched.

The **SSLPEER** value is specified in the standard form used to specify a Distinguished Name.

This parameter is valid only when **MATCH** is RUNCHECK and must not be generic.

TYPE
The type of Channel Authentication Record for which to display details. Possible values are:
- ALL
- BLOCKUSER
- BLOCKADDR
- SSLPEERMAP
- ADDRESSMAP
- USERMAP
- QMGRMAP

WHERE
Specify a filter condition to display only those channel authentication records that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Any parameter that can be used to display attributes for this DISPLAY command.

operator
This is used to determine whether a channel authentication record satisfies the filter value on the given filter keyword. The operators are as follows:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which contain the specified item.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not contain the specified item.

CTG Contains an item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use this to display objects the attributes of which match the generic string.

EXG Does not contain any item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not match the generic string.

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, the value can be either explicit or generic:

- An explicit value, that is a valid value for the attribute being tested.

  You can use any of the operators except LK and NL. However, if the value is one from a possible set of values returnable on a parameter (for example, the value ALL on the MATCH parameter), you can only use EQ or NE.
- A generic value. This is a character string with an asterisk at the end, for example ABC*. The characters must be valid for the attribute you are testing. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

  You can only use operators LK or NL for generic values.
- An item in a list of values. The value can be explicit or, if it is a character value, it can be explicit or generic. If it is explicit, use CT or EX as the operator. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed. If it is generic, use CTG or EXG as the operator. If ABC* is specified with the operator CTG, all items where one of the attribute values begins with ABC are listed.

Note: z/OS On z/OS there is a 256 character limit for the filter-value of the MQSC **WHERE** clause. This limit is not in place for other platforms.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

TYPE
The type of channel authentication record

SSLPEER
The Distinguished Name of the certificate.

ADDRESS
The IP address

CHCKCLNT
Whether a user ID and password are to be supplied by connections which match this rule.

CLNTUSER
The client asserted user ID

QMNAME
The name of the remote partner queue manager

MCAUSER
The user identifier to be used when the inbound connection matches the TLS DN, IP address, client asserted user ID or remote queue manager name supplied.

ADDRLIST
A list of IP address patterns which are banned from connecting into this queue manager on any channel.

USERLIST
A list of user IDs which are banned from use of this channel or set of channels.

ALTDATE
The date on which the channel authentication record was last altered, in the format *yyyy-mm-dd*.

ALTTIME
The time on which the channel authentication record was last altered, in the form *hh.mm.ss*.

DESCR
Descriptive information about the channel authentication record.

SSLCERTI
The Certificate issuer Distinguished Name of the certificate to be matched.

CUSTOM
Reserved for future use.

Related information:
Channel authentication records

Generic IP addresses for channel authentication records:

In the various commands that create and display channel authentication records, you can specify certain parameters as either a single IP address or a pattern to match a set of IP addresses.

When you create a channel authentication record, using the MQSC command **SET CHLAUTH** or the PCF command **Set Channel Authentication Record**, you can specify a generic IP address in various contexts. You can also specify a generic IP address in the filter condition when you display a channel authentication record using the commands **DISPLAY CHLAUTH** or **Inquire Channel Authentication Records**.

You can specify the address in any of the following ways:

- a single IPv4 address, such as 192.0.2.0
- a pattern based on an IPv4 address, including an asterisk (*) as a wildcard. The wildcard represents one or more parts of the address, depending on context. For example, the following values are all valid:
  - 192.0.2.*
  - 192.0.*
  - 192.0.*.2
  - 192.*.2
  - *
- a pattern based on an IPv4 address, including a hyphen (-) to indicate a range, for example 192.0.2.1-8

- a pattern based on an IPv4 address, including both an asterisk and a hyphen, for example 192.0.*.1-8
- a single IPv6 address, such as 2001:DB8:0:0:0:0:0:0
- a pattern based on an IPv6 address including an asterisk (*) as a wildcard. The wildcard represents one or more parts of the address, depending on context. For example, the following values are all valid:
  - 2001:DB8:0:0:0:0:*
  - 2001:DB8:0:0:0:*
  - 2001:DB8:0:0:0:*:0:1
  - 2001:*:1
  - *
- a pattern based on an IPv6 address, including a hyphen (-) to indicate a range, for example 2001:DB8:0:0:0:0:0:0-8
- a pattern based on an IPv6 address, including both an asterisk and a hyphen, for example 2001:DB8:0:0:*:0:0-8

If your system supports both IPv4 and IPv6, you can use either address format. IBM MQ recognizes IPv4 mapped addresses in IPv6.

Certain patterns are invalid:

- A pattern cannot have fewer than the required number of parts, unless the pattern ends with a single trailing asterisk. For example 192.0.2 is invalid, but 192.0.2.* is valid.
- A trailing asterisk must be separated from the rest of the address by the appropriate part separator (a dot (.) for IPv4, a colon (:) for IPv6). For example, 192.0* is not valid because the asterisk is not in a part of its own.
- A pattern may contain additional asterisks provided that no asterisk is adjacent to the trailing asterisk. For example, 192.*.2.* is valid, but 192.0.*.* is not valid.
- An IPv6 address pattern cannot contain a double colon and a trailing asterisk, because the resulting address would be ambiguous. For example, 2001::* could expand to 2001:0000:*, 2001:0000:0000:* and so on

Related information:

Mapping an IP address to an MCAUSER user ID

DISPLAY CHSTATUS:

Use the MQSC command DISPLAY CHSTATUS to display the status of one or more channels.

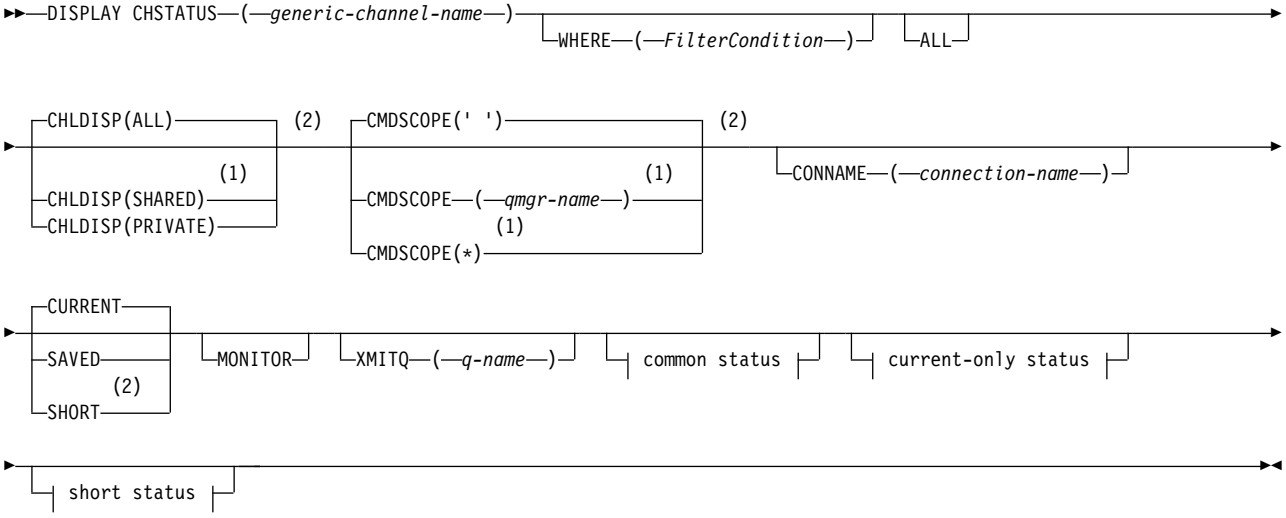
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

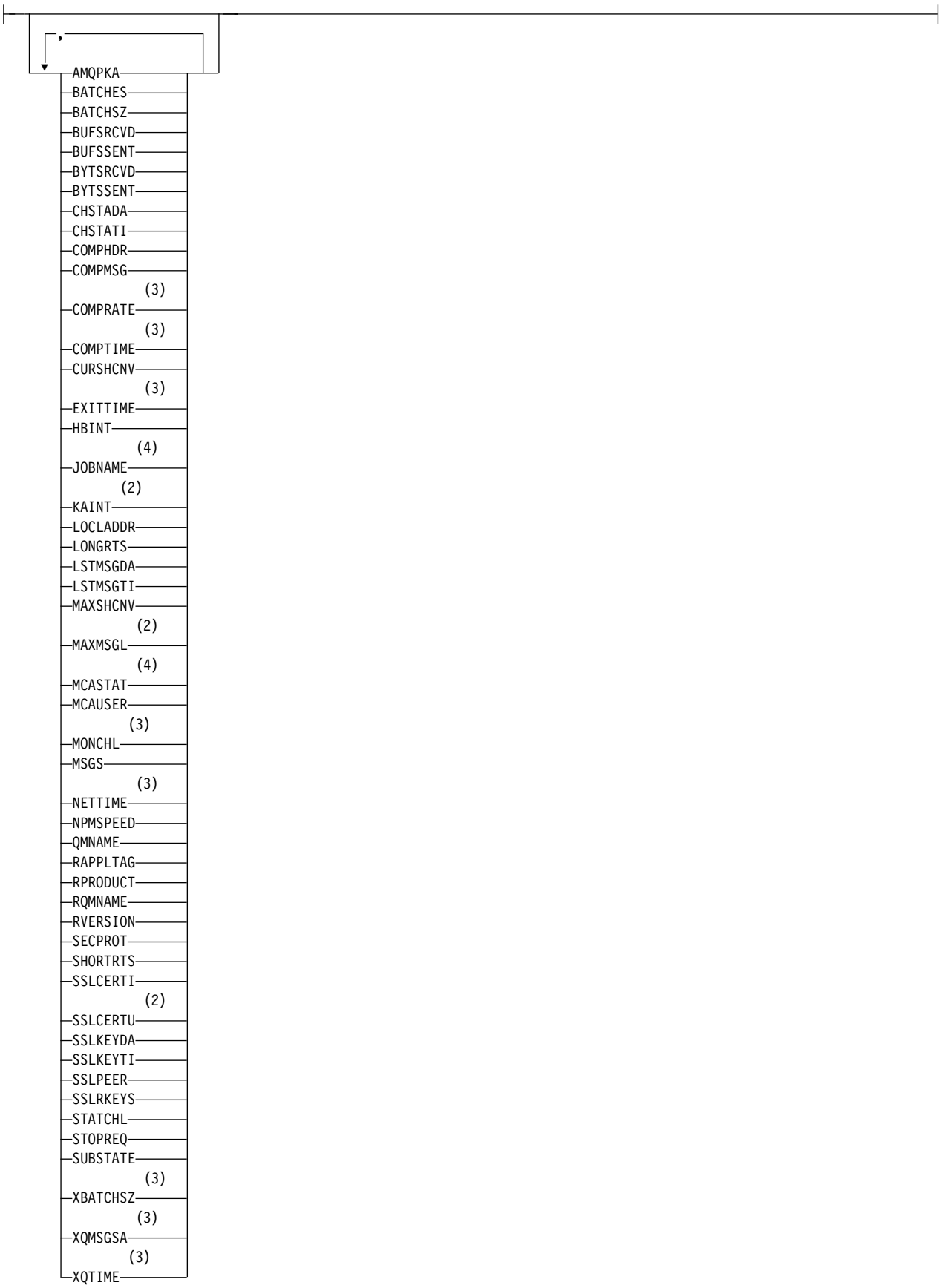
Synonym: DIS CHS

DISPLAY CHSTATUS

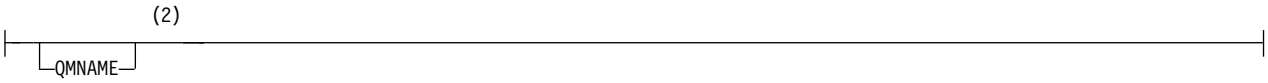


Common status:

Current-only status:


,
AMQPKA
BATCHES
BATCHSZ
BUFSRCVD
BUFSSENT
BYTSRCVD
BYTSSENT
CHSTADA
CHSTATI
COMPHDR
COMPMSG
(3)
COMPRATE
(3)
COMPTIME
CURSHCNV
(3)
EXITTIME
HBINT
(4)
JOBNAME
(2)
KAINT
LOCLADDR
LONGRTS
LSTMSGDA
LSTMSGTI
MAXSHCNV
(2)
MAXMSGL
(4)
MCASTAT
MCAUSER
(3)
MONCHL
MSGS
(3)
NETTIME
NPMSPEED
QMNAME
RAPPLTAG
RPRODUCT
RQMNAME
RVERSION
SECPROT
SHORTRTS
SSLCERTI
(2)
SSLCERTU
SSLKEYDA
SSLKEYTI
SSLPEER
SSLRKEYS
STATCHL
STOPREQ
SUBSTATE
(3)
XBATCHSZ
(3)
XQMSGSA
(3)
XQTIME

Short status:



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Also displayed by selection of the MONITOR parameter.

4 Ignored if specified on z/OS.

z/OS

Usage notes for DISPLAY CHSTATUS on z/OS

1. The command fails if the channel initiator has not been started.
2. The command server must be running.
3. If you want to see the overall status of the channel (that is, the status of the queue-sharing group) use the command **DISPLAY CHSTATUS SHORT**, which obtains the status information of the channel from Db2.
4. If any numeric parameter exceeds 999,999,999, it is displayed as 999999999.
5. The status information that is returned for various combinations of CHLDISP, CMDSCOPE, and status type are summarized in Table 110, Table 111, and Table 112 on page 787.

Table 110. CHLDISP and CMDSCOPE for DISPLAY CHSTATUS CURRENT

CHLDISP	CMDSCOPE( ) or CMDSCOPE (local-qmgr)	CMDSCOPE (qmgr-name)	CMDSCOPE(*)
PRIVATE	Common and current-only status for current private channels on the local queue manager	Common and current-only status for current private channels on the named queue manager	Common and current-only status for current private channels on all queue managers
SHARED	Common and current-only status for current shared channels on the local queue manager	Common and current-only status for current shared channels on the named queue manager	Common and current-only status for current shared channels on all queue managers
ALL	Common and current-only status for current private and shared channels on the local queue manager	Common and current-only status for current private and shared channels on the named queue manager	Common and current-only status for current private and shared channels on all active queue managers

Table 111. CHLDISP and CMDSCOPE for DISPLAY CHSTATUS SHORT

CHLDISP	CMDSCOPE( ) or CMDSCOPE (local-qmgr)	CMDSCOPE (qmgr-name)	CMDSCOPE(*)
PRIVATE	STATUS and short status for current private channels on the local queue manager	STATUS and short status for current private channels on the named queue manager	STATUS and short status for current private channels on all active queue managers
SHARED	STATUS and short status for current shared channels on all active queue managers in the queue-sharing group	Not permitted	Not permitted

Table 111. CHLDISP and CMDSCOPE for DISPLAY CHSTATUS SHORT (continued)

CHLDISP	CMDSCOPE( ) or CMDSCOPE (local-qmgr)	CMDSCOPE (qmgr-name)	CMDSCOPE(*)
ALL	STATUS and short status for current private channels on the local queue manager and current shared channels in the queue-sharing group (5a)	STATUS and short status for current private channels on the named queue manager	STATUS and short status for current private, and shared, channels on all active queue managers in the queue-sharing group (5a)

Note:

a. In this case you get two separate sets of responses to the command on the queue manager where it was entered; one for PRIVATE and one for SHARED.

Table 112. CHLDISP and CMDSCOPE for DISPLAY CHSTATUS SAVED

CHLDISP	CMDSCOPE( ) or CMDSCOPE (local-qmgr)	CMDSCOPE (qmgr-name)	CMDSCOPE(*)
PRIVATE	Common status for saved private channels on the local queue manager	Common status for saved private channels on the named queue manager	Common status for saved private channels on all active queue managers
SHARED	Common status for saved shared channels on all active queue managers in the queue-sharing group	Not permitted	Not permitted
ALL	Common status for saved private channels on the local queue manager and saved shared channels in the queue-sharing group	Common status for saved private channels on the named queue manager	Common status for saved private, and shared, channels on all active queue managers in the queue-sharing group

Parameter descriptions for DISPLAY CHSTATUS on all platforms

You must specify the name of the channel for which you want to display status information. This can be a specific channel name or a generic channel name. By using a generic channel name, you can display either the status information for all channels, or status information for one or more channels that match the specified name.

You can also specify whether you want the current status data (of current channels only), or the saved status data of all channels.

Status for all channels that meet the selection criteria is displayed, whether the channels were defined manually or automatically.

The classes of data available for channel status are **saved** and **current**, and (on z/OS only) **short**.

The status fields available for saved data are a subset of the fields available for current data and are called **common** status fields. Note that although the common data *fields* are the same, the data *values* might be different for saved and current status. The rest of the fields available for current data are called **current-only** status fields.

- **Saved** data consists of the common status fields noted in the syntax diagram.
  - For a sending channel data is updated before requesting confirmation that a batch of messages has been received and when confirmation has been received

  - For a receiving channel data is reset just before confirming that a batch of messages has been received
  - For a server connection channel no data is saved.
  - Therefore, a channel that has never been current cannot have any saved status.

    Note: Status is not saved until a persistent message is transmitted across a channel, or a nonpersistent message is transmitted with a NPMSPEED of NORMAL. Because status is saved at the end of each batch, a channel does not have any saved status until at least one batch has been transmitted.
- **Current** data consists of the common status fields and current-only status fields as noted in the syntax diagram. The data fields are continually updated as messages are sent/received.
- z/OS **Short** data consists of the STATUS current data item and the short status field as noted in the syntax diagram.

This method of operation has the following consequences:

- An inactive channel might not have any saved status - if it has never been current or has not yet reached a point where saved status is reset.
- The "common" data fields might have different values for saved and current status.
- A current channel always has current status and might have saved status.

Channels can either be current or inactive:

Current channels
: These are channels that have been started, or on which a client has connected, and that have not finished or disconnected normally. They might not yet have reached the point of transferring messages, or data, or even of establishing contact with the partner. Current channels have **current** status and might also have **saved** status.

  The term **Active** is used to describe the set of current channels that are not stopped.

Inactive channels
: These are channels that either:
  - Have not been started
  - On which a client has not connected
  - Have finished
  - Have disconnected normally

  (Note that if a channel is stopped, it is not yet considered to have finished normally - and is, therefore, still current.) Inactive channels have either **saved** status or no status at all.

There can be more than one instance of the same named receiver, requester, cluster-receiver, or server-connection channel current at the same time (the requester is acting as a receiver). This occurs if several senders, at different queue managers, each initiate a session with this receiver, using the same channel name. For channels of other types, there can only be one instance current at any time.

For all channel types, however, there can be more than one set of saved status information available for a channel name. At most one of these sets relates to a current instance of the channel, the rest relate to previously current instances. Multiple instances arise if different transmission queue names or connection names have been used with the same channel. This can happen in the following cases:

- At a sender or server:
  - If the same channel has been connected to by different requesters (servers only)
  - If the transmission queue name has been changed in the definition
  - If the connection name has been changed in the definition
- At a receiver or requester:

– If the same channel has been connected to by different senders or servers
– If the connection name has been changed in the definition (for requester channels initiating connection)

The number of sets that are displayed for a channel can be limited by using the XMITQ, CONNAME, and CURRENT parameters on the command.

(*generic-channel-name*)
The name of the channel definition for which status information is to be displayed. A trailing asterisk (*) matches all channel definitions with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all channel definitions. A value is required for all channel types.

WHERE
Specify a filter condition to display status information for those channels that satisfy the selection criterion of the filter condition.

The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

`filter-keyword`
The parameter to be used to display attributes for this DISPLAY command.

Multi You cannot use the following parameters as filter keywords on Multiplatforms: COMPRATE, COMPTIME, CURRENT, EXITTIME, JOBNAME, NETTIME, SAVED, SHORT, XBATCHSZ, or XQTIME.

z/OS You cannot use the following parameters as filter keywords on z/OS: CHLDISP, CMDSCOPE, MCASTAT, or MONITOR.

You cannot use CONNAME or XMITQ as filter keywords if you also use them to select channel status.
Status information for channels of a type for which the filter keyword is not valid is not displayed.

`operator`
This is used to determine whether a channel satisfies the filter value on the filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which contain the specified item.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not contain the specified item.

`filter-value`
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:
- An explicit value, that is a valid value for the attribute being tested.

You can use operators LT, GT, EQ, NE, LE, or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value SDR on the CHLTYPE parameter), you can only use EQ or NE.

- A generic value. This is a character string with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.
- An item in a list of values. Use CT or EX as the operator. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed.

ALL Specify this to display all the status information for each relevant instance.

If SAVED is specified, this causes only common status information to be displayed, not current-only status information.

If this parameter is specified, any parameters requesting specific status information that are also specified have no effect; all the information is displayed.

z/OS **CHLDISP**

This parameter applies to z/OS only and specifies the disposition of the channels for which information is to be displayed, as used in the START and STOP CHANNEL commands, and **not** that set by QSGDISP for the channel definition. Values are:

ALL This is the default value and displays requested status information for private channels.

If there is a shared queue manager environment and the command is being executed on the queue manager where it was issued, or if CURRENT is specified, this option also displays the requested status information for shared channels.

PRIVATE
Display requested status information for private channels.

SHARED
Display requested status information for shared channels. This is allowed only if there is a shared queue manager environment, and either:

- CMDSCOPE is blank or the local queue manager
- CURRENT is specified

CHLDISP displays the following values:

PRIVATE
The status is for a private channel.

SHARED
The status is for a shared channel.

FIXSHARED
The status is for a shared channel, tied to a specific queue manager.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which it was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

Note: See Table 1, Table 2, and Table 3 for the permitted combinations of CHLDISP and CMDSCOPE.

CONNAME(*connection-name* **)**
The connection name for which status information is to be displayed, for the specified channel or channels.

This parameter can be used to limit the number of sets of status information that is displayed. If it is not specified, the display is not limited in this way.

The value returned for CONNAME might not be the same as in the channel definition, and might differ between the current channel status and the saved channel status. (Using CONNAME for limiting the number of sets of status is therefore not recommended.)

For example, when using TCP, if CONNAME in the channel definition:

- Is blank or is in "host name" format, the channel status value has the resolved IP address.
- Includes the port number, the current channel status value includes the port number (except on z/OS), but the saved channel status value does not.

For SAVED or SHORT status, this value could also be the queue manager name, or queue-sharing group name, of the remote system.

CURRENT
This is the default, and indicates that current status information as held by the channel initiator for current channels only is to be displayed.

Both common and current-only status information can be requested for current channels.

Short status information is not displayed if this parameter is specified.

SAVED
Specify this to display saved status information for both current and inactive channels.

Only common status information can be displayed. Short and current-only status information is not displayed for current channels if this parameter is specified.

z/OS **SHORT**
This indicates that short status information and the STATUS item for current channels only is to be displayed.

Other common status and current-only status information is not displayed for current channels if this parameter is specified.

MONITOR
Specify this to return the set of online monitoring parameters. These are COMPRATE, COMPTIME, EXITTIME, MONCHL, NETTIME, XBATCHSZ, XQMSGSA, and XQTIME. If you specify this parameter, any of the monitoring parameters that you request specifically have no effect; all monitoring parameters are still displayed.

XMITQ(*q-name* **)**
The name of the transmission queue for which status information is to be displayed, for the specified channel or channels.

This parameter can be used to limit the number of sets of status information that is displayed. If it is not specified, the display is not limited in this way.

The following information is always returned, for each set of status information:

- The channel name
- The transmission queue name (for sender and server channels)
- The connection name
- The remote queue manager, or queue-sharing group, name (only for current status, and for all channel types except server-connection channels)
- The remote partner application name (for server-connection channels)
- The type of status information returned (CURRENT, or SAVED, or on z/OS only, SHORT)
- STATUS (except SAVED on z/OS)
- On z/OS, CHLDISP
- STOPREQ (only for current status)
- SUBSTATE

If no parameters requesting specific status information are specified (and the ALL parameter is not specified), no further information is returned.

If status information is requested that is not relevant for the particular channel type, this is not an error.

Common status

The following information applies to sets of current status data and also to sets of saved status data. Some of this information does not apply to server-connection channels.

CHLTYPE
The channel type. This is one of the following:

SDR A sender channel

SVR A server channel

RCVR A receiver channel

RQSTR
A requester channel

CLUSSDR
A cluster-sender channel

CLUSRCVR
A cluster-receiver channel

SVRCONN
A server-connection channel

AMQP
An AMQP channel

CURLUWID
The logical unit of work identifier associated with the current batch, for a sending or a receiving channel.

For a sending channel, when the channel is in doubt it is the LUWID of the in-doubt batch.

For a saved channel instance, this parameter has meaningful information only if the channel instance is in doubt. However, the parameter value is still returned when requested, even if the channel instance is not in doubt.

It is updated with the LUWID of the next batch when this is known.

This parameter does not apply to server-connection channels.

CURMSGS
For a sending channel, this is the number of messages that have been sent in the current batch. It is incremented as each message is sent, and when the channel becomes in doubt it is the number of messages that are in doubt.

For a saved channel instance, this parameter has meaningful information only if the channel instance is in doubt. However, the parameter value is still returned when requested, even if the channel instance is not in doubt.

For a receiving channel, it is the number of messages that have been received in the current batch. It is incremented as each message is received.

The value is reset to zero, for both sending and receiving channels, when the batch is committed.

This parameter does not apply to server-connection channels.

CURSEQNO
For a sending channel, this is the message sequence number of the last message sent. It is updated as each message is sent, and when the channel becomes in doubt it is the message sequence number of the last message in the in-doubt batch.

For a saved channel instance, this parameter has meaningful information only if the channel instance is in doubt. However, the parameter value is still returned when requested, even if the channel instance is not in doubt.

For a receiving channel, it is the message sequence number of the last message that was received. It is updated as each message is received.

This parameter does not apply to server-connection channels.

INDOUBT
Whether the channel is currently in doubt.

This is only YES while the sending Message Channel Agent is waiting for an acknowledgment that a batch of messages that it has sent has been successfully received. It is NO at all other times, including the period during which messages are being sent, but before an acknowledgment has been requested.

For a receiving channel, the value is always NO.

This parameter does not apply to server-connection channels.

LSTLUWID
The logical unit of work identifier associated with the last committed batch of messages transferred.

This parameter does not apply to server-connection channels.

LSTSEQNO
Message sequence number of the last message in the last committed batch. This number is not incremented by nonpersistent messages using channels with a NPMSPEED of FAST.

This parameter does not apply to server-connection channels.

STATUS
Current status of the channel. This is one of the following:

BINDING
Channel is performing channel negotiation and is not yet ready to transfer messages.

INITIALIZING
The channel initiator is attempting to start a channel.

On z/OS, this is displayed as INITIALIZI.

PAUSED
The channel is waiting for the message-retry interval to complete before retrying an MQPUT operation.

REQUESTING
A local requester channel is requesting services from a remote MCA.

RETRYING
A previous attempt to establish a connection has failed. The MCA will reattempt connection after the specified time interval.

RUNNING
The channel is either transferring messages at this moment, or is waiting for messages to arrive on the transmission queue so that they can be transferred.

STARTING
A request has been made to start the channel but the channel has not yet begun processing. A channel is in this state if it is waiting to become active.

STOPPED
This state can be caused by one of the following:

- Channel manually stopped

  A user has entered a stop channel command against this channel.

- Retry limit reached

  The MCA has reached the limit of retry attempts at establishing a connection. No further attempt will be made to establish a connection automatically.

A channel in this state can be restarted only by issuing the START CHANNEL command, or starting the MCA program in an operating-system dependent manner.

STOPPING
Channel is stopping or a close request has been received.

SWITCHING
The channel is switching transmission queues.

On z/OS, STATUS is not displayed if saved data is requested.

Multi On Multiplatforms, the value of the STATUS field returned in the saved data is the status of the channel at the time the saved status was written. Normally, the saved status value is RUNNING. To see the current status of the channel, the user can use the DISPLAY CHSTATUS CURRENT command.

Note: For an inactive channel, CURMSGS, CURSEQNO, and CURLUWID have meaningful information only if the channel is INDOUBT. However they are still displayed and returned if requested.

Current-only status

The following information applies only to current channel instances. The information applies to all channel types, except where stated.

AMQPKA
The keep alive time for an AMQP channel in seconds. If the AMQP client has not sent any frames within the keep alive interval, then the connection is closed with a `amqp:resource-limit-exceeded` AMQP error condition.

This parameter is valid only for channels with a channel type (CHLTYPE) of `AMQP`

BATCHES
Number of completed batches during this session (since the channel was started).

BATCHSZ
The batch size being used for this session.

This parameter does not apply to server-connection channels, and no values are returned; if specified on the command, this is ignored.

BUFSRCVD
Number of transmission buffers received. This includes transmissions to receive control information only.

BUFSSENT
Number of transmission buffers sent. This includes transmissions to send control information only.

BYTSRCVD
Number of bytes received during this session (since the channel was started). This includes control information received by the message channel agent.

BYTSSENT
Number of bytes sent during this session (since the channel was started). This includes control information sent by the message channel agent.

CHSTADA
Date when this channel was started (in the form yyyy-mm-dd).

CHSTATI
Time when this channel was started (in the form hh.mm.ss).

COMPHDR
The technique used to compress the header data sent by the channel. Two values are displayed:

- The default header data compression value negotiated for this channel.
- The header data compression value used for the last message sent. The header data compression value can be altered in a sending channels message exit. If no message has been sent, the second value is blank.

COMPMSG
The technique used to compress the message data sent by the channel. Two values are displayed:

- The default message data compression value negotiated for this channel.
- The message data compression value used for the last message sent. The message data compression value can be altered in a sending channels message exit. If no message has been sent, the second value is blank.

COMPRATE
The compression rate achieved displayed to the nearest percentage. Two values are displayed:

- The first value based on recent activity over a short period.
- The second value based on activity over a longer period.

These values are reset every time the channel is started and are displayed only when the STATUS of the channel is RUNNING. If monitoring data is not being collected, or if no messages have been sent by the channel, the values are shown as blank.

A value is only displayed for this parameter if MONCHL is set for this channel. See “Setting monitor values” on page 802.

COMPTIME
The amount of time for each message, displayed in microseconds, spent on compression or decompression. Two values are displayed:

- The first value based on recent activity over a short period.
- The second value based on activity over a longer period.

Note: On z/OS, COMPTIME is the amount of time for each message, provided that the message does not have to be processed in segments.

This segmenting of the message on z/OS occurs when the message is:

- 32 KB or larger, or
- 16 KB or larger, and the channel has TLS encryption.

If the message is split into segments, COMPTIME is the time spent compressing each segment. This means that a message that is split into 8 segments actually spends (COMPTIME * 8) microseconds during compression or decompression.

A value is only displayed for this parameter if MONCHL is set for this channel. See "Setting monitor values" on page 802.

CURSHCNV

The CURSHCNV value is blank for all channel types other than server-connection channels. For each instance of a server-connection channel, the CURSHCNV output gives a count of the number of conversations currently running over that channel instance.

A value of zero indicates that the channel is running as it did in versions of the product earlier than IBM WebSphere MQ Version 7.0, regarding:

- Administrator stop-quiesce
- Heartbeating
- Read ahead
- Sharing conversations
- Client Asynchronous consumption

EXITTIME

Amount of time, displayed in microseconds, spent processing user exits per message. Two values are displayed:

- The first value based on recent activity over a short period.
- The second value based on activity over a longer period.

These values depend on the configuration and behavior of your system, as well as the levels of activity within it, and serve as an indicator that your system is performing normally. A significant variation in these values may indicate a problem with your system. They are reset every time the channel is started and are displayed only when the STATUS of the channel is RUNNING.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONCHL is set for this channel. See "Setting monitor values" on page 802.

HBINT

The heartbeat interval being used for this session.

JOBNAME

A name that identifies the MQ process that is currently providing and hosting the channel.

Multi On Multiplatforms, this name is the concatenation of the process identifier and the thread identifier of the MCA program, displayed in hexadecimal.

z/OS This information is not available on z/OS. The parameter is ignored if specified.

z/OS You cannot use JOBNAME as a filter keyword on z/OS.

z/OS **KAINT**

The keepalive interval being used for this session. This is valid only on z/OS.

LOCLADDR
Local communications address for the channel. The value returned depends on the TRPTYPE of the channel (currently only TCP/IP is supported).

LONGRTS
Number of long retry wait start attempts left. This applies only to sender or server channels.

LSTMSGDA
Date when the last message was sent or MQI call was handled, see LSTMSGTI.

LSTMSGTI
Time when the last message was sent or MQI call was handled.

For a sender or server, this is the time the last message (the last part of it if it was split) was sent. For a requester or receiver, it is the time the last message was put to its target queue. For a server-connection channel, it is the time when the last MQI call completed.

In the case of a server-connection channel instance on which conversations are being shared, this is the time when the last MQI call completed on any of the conversations running on the channel instance.

z/OS **MAXMSGL**
The maximum message length being used for this session (valid only on z/OS).

MAXSHCNV
The MAXSHCNV value is blank for all channel types other than server-connection channels. For each instance of a server-connection channel, the MAXSHCNV output gives the negotiated maximum of the number of conversations that can run over that channel instance.

A value of zero indicates that the channel is running as it did in versions of IBM MQ earlier than Version 7.0, regarding:

- Administrator stop-quiesce
- Heartbeating
- Read ahead
- Sharing conversations
- Client asynchronous consumption

Multi **MCASTAT**
Whether the Message Channel Agent is currently running. This is either "running" or "not running". Note that it is possible for a channel to be in stopped state, but for the program still to be running.

z/OS This information is not available on z/OS. The parameter is ignored if specified.

z/OS You cannot use MCASTAT as a filter keyword on z/OS.

MCAUSER
The user ID used by the MCA. This can be the user ID set in the channel definition, the default user ID for message channels, a user ID transferred from a client if this is a server-connection channel, or a user ID specified by a security exit.

This parameter applies only to server-connection, receiver, requester, and cluster-receiver channels.

On server connection channels that share conversations, the MCAUSER field contains a user ID if all the conversations have the same MCA user ID value. If the MCA user ID in use varies across these conversations, the MCAUSER field contains a value of *.

Multi The maximum length on Multiplatforms is 64 characters.

z/OS The maximum length on z/OS is 12 characters.

MONCHL
Current level of monitoring data collection for the channel.

This parameter is also displayed when you specify the MONITOR parameter.

MSGS
Number of messages sent or received (or, for server-connection channels, the number of MQI calls handled) during this session (since the channel was started).

In the case of a server-connection channel instance on which conversations are being shared, this is the total number of MQI calls handled on all of the conversations running on the channel instance.

NETTIME
Amount of time, displayed in microseconds, to send a request to the remote end of the channel and receive a response. This time only measures the network time for such an operation. Two values are displayed:

- The first value based on recent activity over a short period.
- The second value based on activity over a longer period.

These values depend on the configuration and behavior of your system, as well as the levels of activity within it, and serve as an indicator that your system is performing normally. A significant variation in these values may indicate a problem with your system. They are reset every time the channel is started and are displayed only when the STATUS of the channel is RUNNING.

This parameter applies only to sender, server, and cluster-sender channels.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONCHL is set for this channel. See “Setting monitor values” on page 802.

NPMSPEED
The nonpersistent message handling technique being used for this session.

PORT The port number used to connect an AMQP channel. The default port for AMQP 1.0 connections is 5672.

RAPPLTAG
The remote partner application name. This is the name of the client application at the remote end of the channel. This parameter applies only to server-connection channels.

RPRODUCT
The remote partner product identifier. This is the product identifier of the IBM MQ code running at the remote end of the channel. The possible values are shown in Table 113.

Table 113. Product Identifier values

Product Identifier	Description
MQMM	Queue manager on a distributed platform
z/OS MQMV	Queue manager on z/OS
MQCC	IBM MQ C client
NSS Client MQNC	IBM MQ client for HP Integrity NonStop Server
MQNM	IBM MQ .NET fully managed client
MQJB	IBM MQ Classes for JAVA
V 9.0.1 MQJF	Managed File Transfer Agent
MQJM	IBM MQ Classes for JMS (normal mode)
MQJN	IBM MQ Classes for JMS (migration mode)

Table 113. Product Identifier values (continued)

Product Identifier	Description
MQJU	Common Java interface to the MQI
MQXC	XMS client C/C++ (normal mode)
MQXD	XMS client C/C++ (migration mode)
MQXN	XMS client .NET (normal mode)
MQXM	XMS client .NET (migration mode)
MQXU	IBM MQ .NET XMS client (unmanaged/XA)
MQNU	IBM MQ .NET unmanaged client

RQMNAME
The queue manager name, or queue-sharing group name, of the remote system. This parameter does not apply to server-connection channels.

RVERSION
The remote partner version. This is the version of the IBM MQ code running at the remote end of the channel.

The remote version is displayed as **VVRRMMFF**, where

VV Version

RR Release

MM Maintenance level

FF Fix level

SECPROT
Security protocol currently in use.

This parameter does not apply to client-connection channels.

The possible values are:

NONE
No security protocol

SSLV3 SSL version 3.0

TLSV1
TLS version 1.0

TLSV12
TLS version 1.2

z/OS SECPROT is not available on z/OS.

SHORTRTS
Number of short retry wait start attempts left. This applies only to sender or server channels.

SSLCERTI
The full Distinguished Name of the issuer of the remote certificate. The issuer is the Certificate Authority that issued the certificate.

The maximum length is 256 characters, so longer Distinguished Names are truncated.

z/OS **SSLCERTU**
The local user ID associated with the remote certificate. This is valid on z/OS only.

SSLKEYDA
Date on which the previous successful TLS secret key reset was issued.

SSLKEYTI
Time at which the previous successful TLS secret key reset was issued.

SSLPEER
Distinguished Name of the peer queue manager or client at the other end of the channel.

The maximum length is 256 characters, so longer Distinguished Names are truncated.

SSLRKEYS
Number of successful TLS key resets. The count of TLS secret key resets is reset when the channel instance ends.

STOPREQ
Whether a user stop request is outstanding. This is either YES or NO.

STATCHL
Current level of statistics data collection for the channel.

SUBSTATE
Action being performed by the channel when this command is issued. The following substates are listed in precedence order, starting with the substate of the highest precedence:

ENDBATCH
Channel is performing end-of-batch processing.

SEND A request has been made to the underlying communication subsystem to send some data.

RECEIVE
A request has been made to the underlying communication subsystem to receive some data.

z/OS **SERIALIZE**
Channel is serializing its access to the queue manager. Valid on z/OS only.

RESYNCH
Channel is resynchronizing with the partner.

HEARTBEAT
Channel is heartbeating with the partner.

SCYEXIT
Channel is running the security exit.

RCVEXIT
Channel is running one of the receive exits.

SENDEXIT
Channel is running one of the send exits.

MSGEXIT
Channel is running one of the message exits.

MREXIT
Channel is running the message retry exit.

CHADEXIT
Channel is running through the channel auto-definition exit.

NETCONNECT
A request has been made to the underlying communication subsystem to connect a partner machine.

SSLHANDSHK
Channel is processing a TLS handshake.

NAMESERVER
A request has been made to the name server.

MQPUT
A request has been made to the queue manager to put a message on the destination queue.

MQGET
A request has been made to the queue manager to get a message from the transmission queue (if this is a message channel) or from an application queue (if this is an MQI channel).

MQICALL
A MQ API call, other than MQPUT and MQGET, is being executed.

COMPRESS
Channel is compressing or extracting data.

Not all substates are valid for all channel types or channel states. There are occasions when no substate is valid, at which times a blank value is returned.

For channels running on multiple threads, this parameter displays the substate of the highest precedence.

TPROOT
The topic root for an AMQP channel. The default value for TPROOT is SYSTEM.BASE.TOPIC. With this value, the topic string an AMQP client uses to publish or subscribe has no prefix, and the client can exchange messages with other MQ pub/sub applications. To have AMQP clients publish and subscribe under a topic prefix, first create an MQ topic object with a topic string set to the prefix you want, then set TPROOT to the name of the MQ topic object you created.

This parameter is valid only for channels with a channel type (CHLTYPE) of `AMQP`

XBATCHSZ
Size of the batches transmitted over the channel. Two values are displayed:

- The first value based on recent activity over a short period.
- The second value based on activity over a longer period.

These values depend on the configuration and behavior of your system, as well as the levels of activity within it, and serve as an indicator that your system is performing normally. A significant variation in these values might indicate a problem with your system. They are reset every time the channel is started and are displayed only when the STATUS of the channel is RUNNING.

This parameter does not apply to server-connection channels.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONCHL is set for this channel. See “Setting monitor values” on page 802.

USECLTID
Specifies that the client ID should be used for authorization checks for an AMQP channel, instead of the MCAUSER attribute value.

XQMSGSA
Number of messages queued on the transmission queue available to the channel for MQGETs.

This parameter has a maximum displayable value of 999. If the number of messages available exceeds 999, a value of 999 is displayed.

z/OS On z/OS, if the transmission queue is not indexed by *CorrelId*, this value is shown as blank.

This parameter applies to cluster-sender channels only.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONCHL is set for this channel. See "Setting monitor values."

XQTIME
The time, in microseconds, that messages remained on the transmission queue before being retrieved. The time is measured from when the message is put onto the transmission queue until it is retrieved to be sent on the channel and, therefore, includes any interval caused by a delay in the putting application.

Two values are displayed:

- The first value based on recent activity over a short period.
- The second value based on activity over a longer period.

These values depend on the configuration and behavior of your system, as well as the levels of activity within it, and serve as an indicator that your system is performing normally. A significant variation in these values might indicate a problem with your system. They are reset every time the channel is started and are displayed only when the STATUS of the channel is RUNNING.

This parameter applies only to sender, server, and cluster-sender channels.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONCHL is set for this channel. See "Setting monitor values."

z/OS

Short status

The following information applies only to current channel instances.

QMNAME
The name of the queue manager that owns the channel instance.

Setting monitor values

For auto-defined cluster sender channels, these are controlled with the queue manager MONACLS parameter. See "ALTER QMGR" on page 458 for more information. You cannot display or alter auto-defined cluster sender channels. However you can get their status, or issue DISPLAY CLUSQMGR, as described here: Working with auto-defined cluster-sender channels.

For other channels, including manually-defined cluster sender channels, these are controlled with the channel MONCHL parameter. See "ALTER CHANNEL" on page 389 for more information.

DISPLAY CHSTATUS (AMQP): ULW V 9.0.0

Use the MQSC command DISPLAY CHSTATUS (AMQP) to display the status of one or more AMQP channels.

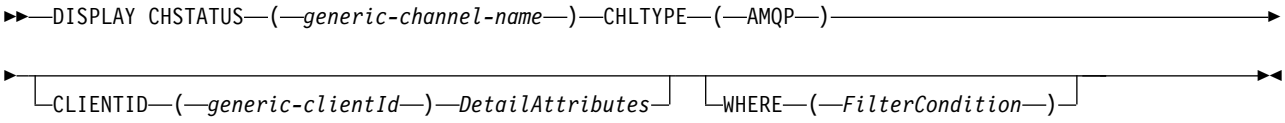
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Parameter descriptions for DISPLAY CHSTATUS" on page 804
- "Summary attributes" on page 805

Synonym: DIS CHS

DISPLAY CHSTATUS (AMQP)



SummaryAttributes:

DetailAttributes:

Note:

- The default behavior is for **RUNMQSC** to return a summary of the connections to the channel. If **CLIENTID** is specified then **RUNMQSC** returns details of each client connected to the channel.

Parameter descriptions for DISPLAY CHSTATUS

You must specify the name of the channel for which you want to display status information. This parameter can be a specific channel name or a generic channel name. By using a generic channel name, you can display either the status information for all channels, or status information for one or more channels that match the specified name.

(*generic-channel-name* **)**

The name of the channel definition for which status information is to be displayed. A trailing asterisk (*) matches all channel definitions with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all channel definitions. A value is required for all channel types.

WHERE

Specify a filter condition to display status information for those channels that satisfy the selection criterion of the filter condition.

The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
The parameter to be used to display attributes for this DISPLAY command.

Status information for channels of a type for which the filter keyword is not valid is not displayed.

operator
This is used to determine whether a channel satisfies the filter value on the filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use this operator to display objects the attributes of which contain the specified item.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use this operator to display objects the attributes of which do not contain the specified item.

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this value can be:

- An explicit value, that is a valid value for the attribute that is being tested.

  You can use operators LT, GT, EQ, NE, LE, or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value SDR on the CHLTYPE parameter), you can use EQ or NE only.
- A generic value. This value is a character string with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

- An item in a list of values. Use CT or EX as the operator. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed.

ALL

Specify this parameter to display all the status information for each relevant instance.

If this parameter is specified, any parameters that request specific status information which are also specified have no effect; all the information is displayed.

Summary attributes

When no CLIENTID parameter is added to the MQSC command DISPLAY CHSTATUS (AMQP), a summary of AMQP channel information is displayed. The number of connections is displayed as the CONNECTIONS attribute. The following attributes display a summary for each channel.

ALL

Specify this parameter to display all the status information for each relevant instance. This attribute is the default value if no attributes are requested.

This parameter is valid for AMQP channels.

If this parameter is specified, any specified parameters that are requesting specific status information have no effect; and all the information is displayed.

CONNS

The number of current connections to this channel.

STATUS

The status of this channel.

Client details mode

STATUS

The status of the client.

CONNAME

The name of the remote connection (IP address)

AMQPKA

The client's keep alive interval.

MCAUSER

The user ID that the client is using to access IBM MQ resources.

CLNTUSER

The user ID that the client provided when it connected.

MSGSNT

Number of messages sent by the client since it connected last.

MSGRCVD

Number of messages received by the client since it connected last.

LSTMSGDA

Date last message was received or sent.

LSTMSGTI

Time last message was received or sent.

CHSTADA

Date channel started.

CHSTATI

Time channel was started.

PROTOCOL

The communication protocol used by the client. The value is AMQP.

Examples

The following command retrieves a status summary for the AMQP channel named MYAMQP:

```
dis chstatus(MYAMQP) chltype(AMQP) all
```

The command outputs the following status:

```
AMQ8417: Display Channel Status details.
   CHANNEL(MYAMQP)                         CHLTYPE(AMQP)
   CONNECTIONS(1)                          STATUS(RUNNING)
```

The following command retrieves a full status for the AMQP channel named MYAMQP:

```
dis chstatus(*) chltype(AMQP) clientid(*) all
```

The command outputs the following status:

```
AMQ8417: Display Channel Status details.
   CHANNEL(MYAMQP)                         CHLTYPE(AMQP)
   CLIENTID(recv_cc2022b)                  STATUS(RUNNING)
   CONNAME(192.168.60.1)                   AMQPKA(0)
   MCAUSER(matt)                           CLNTUSER( )
   MSGSNT(0)                               MSGRCVD(0)
   LSTMSGDA( )                             LSTMSGTI( )
   CHSTADA(2015-09-18)                     CHSTATI(06.23.30)
   PROTOCOL(AMQP)
```

DISPLAY CHSTATUS (MQTT): 

Use the MQSC command DISPLAY CHSTATUS (MQTT) to display the status of one or more MQ Telemetry channels.

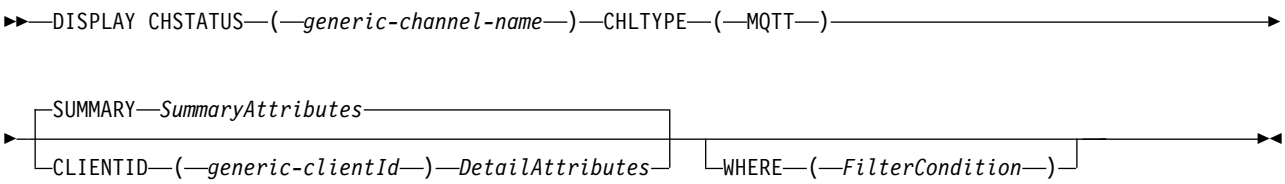
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Parameter descriptions for DISPLAY CHSTATUS" on page 808
- "Summary attributes" on page 809

Synonym: DIS CHS

DISPLAY CHSTATUS (MQTT)



SummaryAttributes:

DetailAttributes:

Notes:

- The default behavior is for **RUNMQSC** to return a summary of the connections to the channel. If **CLIENTID** is specified then **RUNMQSC** returns details of each client connected to the channel.
- Either **CLIENTID**, **SUMMARY**, or neither may be specified, but not both at the same time.
- The **DISPLAY CHSTATUS** command for MQ Telemetry has the potential to return a far larger number of responses than if the command was run for an IBM MQ channel. For this reason, the MQ Telemetry server does not return more responses than fit on the reply-to queue. The number of responses is limited to the value of MAXDEPTH parameter of the SYSTEM.MQSC.REPLY.QUEUE queue. When RUNMQSC processes an MQ Telemetry command that is truncated by the MQ Telemetry server, the AMQ8492 message is displayed specifying how many responses are returned based on the size of MAXDEPTH.
- You can use this command to list disconnected clients. As these clients are not associated with a particular channel, you list them using the wildcard character. For example,

  ```
  DIS CHS(*) CHLTYPE(MQTT) CLIENTID(*) WHERE(STATUS EQ DISCONNECTED).
  ```

  You should take care if using this command as there could be a large number of disconnected clients.

Parameter descriptions for DISPLAY CHSTATUS

You must specify the name of the channel for which you want to display status information. This parameter can be a specific channel name or a generic channel name. By using a generic channel name, you can display either the status information for all channels, or status information for one or more channels that match the specified name.

(*generic-channel-name* **)**

The name of the channel definition for which status information is to be displayed. A trailing asterisk (*) matches all channel definitions with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all channel definitions. A value is required for all channel types.

WHERE

Specify a filter condition to display status information for those channels that satisfy the selection criterion of the filter condition.

The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
The parameter to be used to display attributes for this DISPLAY command.

Status information for channels of a type for which the filter keyword is not valid is not displayed.

operator
This is used to determine whether a channel satisfies the filter value on the filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use this operator to display objects the attributes of which contain the specified item.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use this operator to display objects the attributes of which do not contain the specified item.

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this value can be:

- An explicit value, that is a valid value for the attribute that is being tested.

  You can use operators LT, GT, EQ, NE, LE, or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value SDR on the CHLTYPE parameter), you can use EQ or NE only.
- A generic value. This value is a character string with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

- An item in a list of values. Use CT or EX as the operator. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed.

ALL

Specify this parameter to display all the status information for each relevant instance.

If this parameter is specified, any parameters that request specific status information which are also specified have no effect; all the information is displayed.

Summary attributes

When SUMMARY is added to the MQSC command DISPLAY CHSTATUS (MQTT), the number of connections is displayed as the CONNECTIONS attribute. The following attributes display a summary for each channel.

ALL

Specify this parameter to display all the status information for each relevant instance. This attribute is the default value if no attributes are requested.

This parameter is valid for MQTT channels.

If this parameter is specified, any specified parameters that are requesting specific status information have no effect; and all the information is displayed.

CONNECTIONS

The number of current connections to this channel.

STATUS

The status of this channel.

Client details mode

STATUS

The status of the client.

CLNTUSER

The user ID that the client provided when it connected.

CONNAME

The name of the remote connection (IP address)

KAINT

The client's keep alive interval.

MCAUSER

The user ID that the client is using to access IBM MQ resources. This is the client user ID selected by the process described in MQTT client identity and authorization.

MSGSNT

Number of messages sent by the client since it connected last.

MSGRCVD

Number of messages received by the client since it connected last.

INDOUBTIN

Number of in doubt, inbound messages to the client.

INDOUBTOUT

Number of in doubt, outbound messages to the client.

PENDING

Number of outbound pending messages.

PROTOCOL

The communication protocol used by the client. This is, MQTT V3, HTTP, or MQTTV311.

LMSGDATE

Date last message was received or sent.

LMSGTIME

Time last message was received or sent.

CHLSDATE

Date channel started.

CHLSTIME

Time channel was started.

DISPLAY CLUSQMGR:

Use the MQSC command **DISPLAY CLUSQMGR** to display information about cluster channels for queue managers in a cluster.

Using MQSC commands

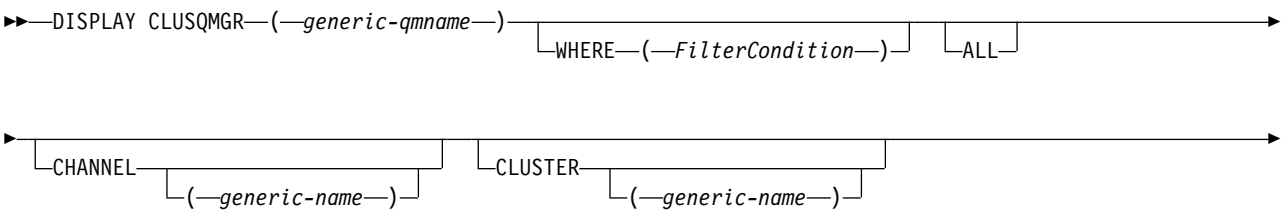
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes" on page 813
- "Parameter descriptions for DISPLAY CLUSQMGR" on page 813
- "Requested parameters" on page 814
- "Channel parameters" on page 816

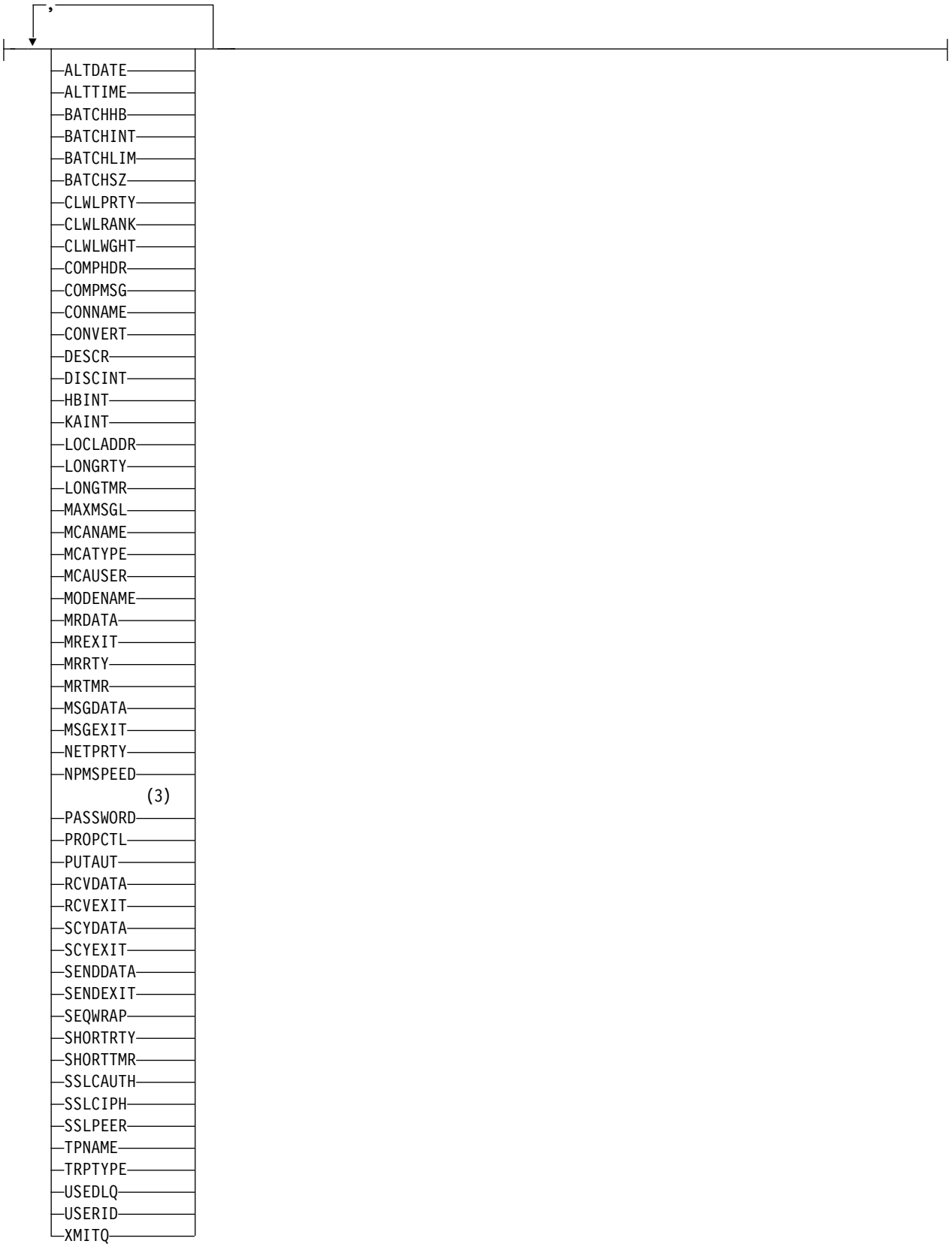
Synonym : DIS CLUSQMGR

DISPLAY CLUSQMGR



Requested attributes:

Channel attributes:



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Not valid on z/OS.

Usage notes

Unlike the **DISPLAY CHANNEL** command, this command includes information about cluster channels that are auto-defined, and the status of cluster channels.

Note: On z/OS, the command fails if the channel initiator is not started.

Parameter descriptions for DISPLAY CLUSQMGR

(*generic-qmgr-name*)
The name of the cluster queue manager for which information is to be displayed.

A trailing asterisk "*" matches all cluster queue managers with the specified stem followed by zero or more characters. An asterisk "*" on its own specifies all cluster queue managers.

WHERE
Specify a filter condition to display only those cluster channels that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this **DISPLAY** command. However, you cannot use the CMDSCOPE or MCANAME parameters as filter keywords. You cannot use CHANNEL or CLUSTER as filter keywords if you use them to select cluster queue managers.

operator
The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use CT to display objects the attributes of which contain the specified item.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use EX to display objects the attributes of which do not contain the specified item.

CTG Contains an item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use CTG to display objects the attributes of which match the generic string.

EXG Does not contain any item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use EXG to display objects the attributes of which do not match the generic string.

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, *filter-value* can be:
- An explicit value, that is a valid value for the attribute being tested.

You can use operators LT, GT, EQ, NE, LE,, or GE only. If the attribute value is a value from a possible set of values, you can use only EQ or NE. For example, the value STARTING on the **STATUS** parameter.

- A generic value. *filter-value* is a character string. An example is ABC*. If the operator is LK, all items where the attribute value begins with the string, ABC in the example, are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.
- An item in a list of values. The value can be explicit or, if it is a character value, it can be explicit or generic. If it is explicit, use CT or EX as the operator. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed. If it is generic, use CTG or EXG as the operator. If ABC* is specified with the operator CTG, all items where one of the attribute values begins with ABC are listed.

ALL Specify ALL to display all the parameters. If this parameter is specified, any parameters that are also requested specifically have no effect; all parameters are still displayed.

ALL is the default if you do not specify a generic name and do not request any specific parameters.

z/OS On z/OS ALL is also the default if you specify a filter condition using the WHERE parameter, but on other platforms, only requested attributes are displayed.

CHANNEL (*generic-name*)
This is optional, and limits the information displayed to cluster channels with the specified channel name. The value can be a generic name.

CLUSTER (*generic-name*)
This is optional, and limits the information displayed to cluster queue managers with the specified cluster name. The value can be a generic name.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

'' The command runs on the queue manager on which it was entered. '' is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered. You can enter a different queue manager name, if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

Some parameters are relevant only for cluster channels of a particular type or types. Attributes that are not relevant for a particular type of channel cause no output, and do not cause an error.

CLUSDATE
The date on which the definition became available to the local queue manager, in the form `yyyy-mm-dd`.

CLUSTIME
The time at which the definition became available to the local queue manager, in the form `hh.mm.ss`.

DEFTYPE
How the cluster channel was defined:

CLUSSDR
As a cluster-sender channel from an explicit definition.

CLUSSDRA
As a cluster-sender channel by auto-definition alone.

CLUSSDRB
As a cluster-sender channel by auto-definition and an explicit definition.

CLUSRCVR
As a cluster-receiver channel from an explicit definition.

QMID
The internally generated unique name of the cluster queue manager.

QMTYPE
The function of the cluster queue manager in the cluster:

REPOS
Provides a full repository service.

NORMAL
Does not provide a full repository service.

STATUS
The status of the channel for this cluster queue manager is one of the following values:

STARTING
The channel was started and is waiting to become active.

BINDING
The channel is performing channel negotiation and is not yet ready to transfer messages.

INACTIVE
The channel is not active.

INITIALIZING
The channel initiator is attempting to start a channel.

z/OS On z/OS, `INITIALIZING` is displayed as `INITIALIZI`.

RUNNING
The channel is either transferring messages at this moment, or is waiting for messages to arrive on the transmission queue so that they can be transferred.

STOPPING
The channel is stopping, or received a close request.

RETRYING
A previous attempt to establish a connection failed. The MCA attempts to connect again after the specified time interval.

PAUSED
The channel is waiting for the message-retry interval to complete before trying an MQPUT operation again.

STOPPED
This state can be caused by one of the following events:

- Channel manually stopped.

  A user entered a stop channel command for this channel.
- The number of attempts to establish a connection reached the maximum number of attempts allowed for the channel.

  No further attempt is made to establish a connection automatically.

A channel in this state can be restarted only by issuing the **START CHANNEL** command, or starting the MCA program in an operating-system dependent manner.

REQUESTING
A local requester channel is requesting services from a remote MCA.

SWITCHING
The channel is switching transmission queues.

SUSPEND
Specifies whether this cluster queue manager is suspended from the cluster or not (as a result of the **SUSPEND QMGR** command). The value of SUSPEND is either YES or NO.

VERSION
The version of the IBM MQ installation that the cluster queue manager is associated with.

The version has the format VVRRMMFF:

- VV: Version
- RR: Release
- MM: Maintenance level
- FF: Fix level

XMITQ
The cluster transmission queue.

Channel parameters

ALTDATE
The date on which the definition or information was last altered, in the form yyyy-mm-dd

ALTTIME
The time at which the definition or information was last altered, in the form hh.mm.ss

BATCHHB
The batch heartbeat value being used.

BATCHINT
Minimum batch duration.

BATCHLIM
Batch data limit.

The limit of the amount of data that can be sent through a channel.

BATCHSZ
Batch size.

CLWLPRTY
The priority of the channel for the purposes of cluster workload distribution.

CLWLRANK
The rank of the channel for the purposes of cluster workload distribution.

CLWLWGHT
The weighting of the channel for the purposes of cluster workload distribution.

COMPHDR
The list of header data compression techniques supported by the channel.

COMPMSG
The list of message data compression techniques supported by the channel.

CONNAME
Connection name.

CONVERT
Specifies whether the sender converts application message data.

DESCR
Description.

DISCINT
Disconnection interval.

HBINT
Heartbeat interval.

KAINT
KeepAlive timing for the channel.

LOCLADDR
Local communications address for the channel.

LONGRTY
Limit of number of attempts to connect using the long duration timer.

LONGTMR
Long duration timer.

MAXMSGL
Maximum message length for channel.

MCANAME
Message channel agent name.

You cannot use MCANAME as a filter keyword.

MCATYPE
Specifies whether the message channel agent runs as a separate process or a separate thread.

MCAUSER
Message channel agent user identifier.

MODENAME
LU 6.2 mode name.

MRDATA
Channel message-retry exit user data.

MREXIT
Channel message-retry exit name.

MRRTY
Channel message-retry count.

MRTMR
Channel message-retry time.

MSGDATA
Channel message exit user data.

MSGEXIT
Channel message exit names.

NETPRTY
The priority for the network connection.

NPMSPEED
Nonpersistent message speed.

PASSWORD
Password for initiating LU 6.2 session (if nonblank, `PASSWORD` is displayed as asterisks).

PROPCTL
Message property control.

PUTAUT
Put authority.

RCVDATA
Channel receive exit user data.

RCVEXIT
Channel receive exit names.

SCYDATA
Channel security exit user data.

SCYEXIT
Channel security exit name.

SENDDATA
Channel send exit user data.

SENDEXIT
Channel send exit names.

SEQWRAP
Sequence number wrap value.

SHORTRTY
Limit of number of attempts to connect using the short duration timer.

SHORTTMR
Short duration timer.

SSLCAUTH
Specifies whether TLS client authentication is required.

SSLCIPH
Cipher specification for the TLS connection.

SSLPEER
Filter for the Distinguished Name from the certificate of the peer queue manager or client at the other end of the channel.

TRPTYPE
Transport type.

TPNAME
LU 6.2 transaction program name.

USEDLQ
Determines whether the dead-letter queue is used when messages cannot be delivered by channels.

USERID
User identifier for initiating LU 6.2 session.

For more information about channel parameters, see “DEFINE CHANNEL” on page 575

DISPLAY CMDSERV on z/OS: z/OS

Use the MQSC command DISPLAY CMDSERV to display the status of the command server.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 12CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- “Usage notes for DISPLAY CMDSERV”

Synonym: DIS CS

DISPLAY CMDSERV

Usage notes for DISPLAY CMDSERV

1. The command server takes messages from the system command input queue, and commands using CMDSCOPE, and processes them. DISPLAY CMDSERV displays the status of the command server.
2. The response to this command is a message showing the current status of the command server, which is one of the following:

   ENABLED
   Available to process commands

   DISABLED
   Not available to process commands

   STARTING
   START CMDSERV in progress

   STOPPING
   STOP CMDSERV in progress

   STOPPED
   STOP CMDSERV completed

   RUNNING
   Available to process commands, currently processing a message

   WAITING
   Available to process commands, currently waiting for a message

DISPLAY COMMINFO on Multiplatforms: Multi

Use the MQSC command DISPLAY COMMINFO to display the attributes of a communication information object.

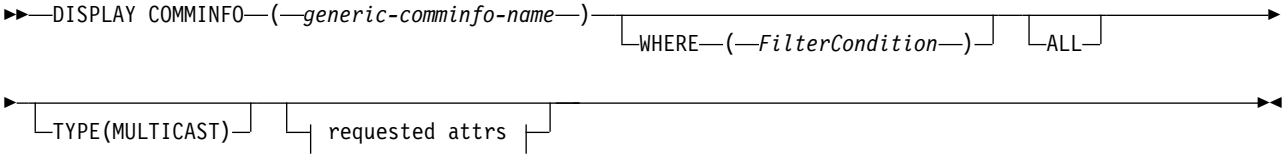
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Parameter descriptions for DISPLAY COMMINFO"
- "Requested parameters" on page 821

Synonym: DIS COMMINFO

DISPLAY COMMINFO



Requested attrs:

Parameter descriptions for DISPLAY COMMINFO

You must specify the name of the communication information object you want to display. This can be a specific communication information object name or a generic communication information object name. By using a generic communication information object name, you can display either:

- All communication information object definitions
- One or more communication information objects that match the specified name

(generic-comminfo-name)

The name of the communication information object definition to be displayed (see Rules for naming IBM MQ objects). A trailing asterisk (*) matches all communication information objects

with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all communication information objects. The names must all be defined to the local queue manager.

WHERE
Specify a filter condition to display only those communication information object definitions that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this DISPLAY command.

operator
This is used to determine whether a communication information object definition satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value DISABLED on the COMMEV parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

ALL Specify this to display all the parameters. If this parameter is specified, any parameters that are requested specifically have no effect; all parameters are still displayed.

TYPE Indicates the type of namelist to be displayed.

MULTICAST
Displays multicast communication information objects. This is the default.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

The default, if no parameters are specified (and the ALL parameter is not specified) is that the object names and TYPE parameters are displayed.

ALTDATE
The date on which the definition was last altered, in the form `yyyy-mm-dd`

ALTTIME
The time at which the definition was last altered, in the form `hh.mm.ss`

BRIDGE
Multicast bridging

CCSID
The coded character set identifier that messages are transmitted on.

COMMEV
Whether event messages are generated for Multicast.

DESCR(*string* **)**
Description

ENCODING
The encoding that the messages are transmitted in.

GRPADDR
The group IP address or DNS name.

MCHBINT
Multicast heartbeat interval.

MCPROP
Multicast property control

MONINT
Monitoring frequency.

MSGHIST
The amount of message history in kilobytes that is kept by the system to handle retransmissions in the case of NACKs (negative acknowledgments).

NSUBHIST
How much history a new subscriber joining a publication stream receives.

PORT The port number to transmit on.

DISPLAY CONN:

Use the MQSC command **`DISPLAY CONN`** to display connection information about the applications connected to the queue manager. This is a useful command because it enables you to identify applications with long-running units of work.

Using MQSC commands

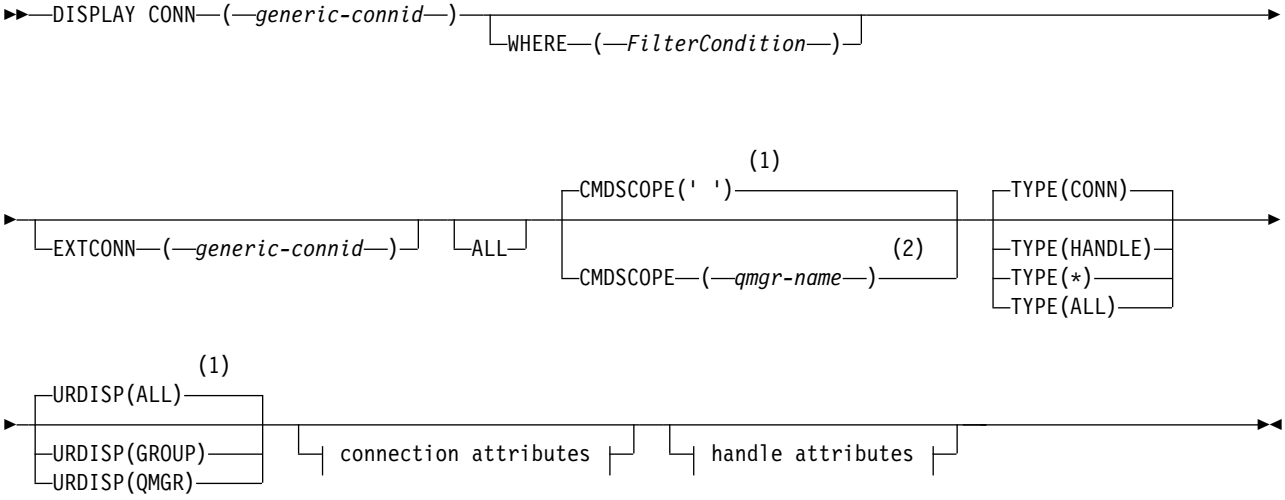
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources `2CR`. For an explanation of the source symbols, see Using commands on z/OS.

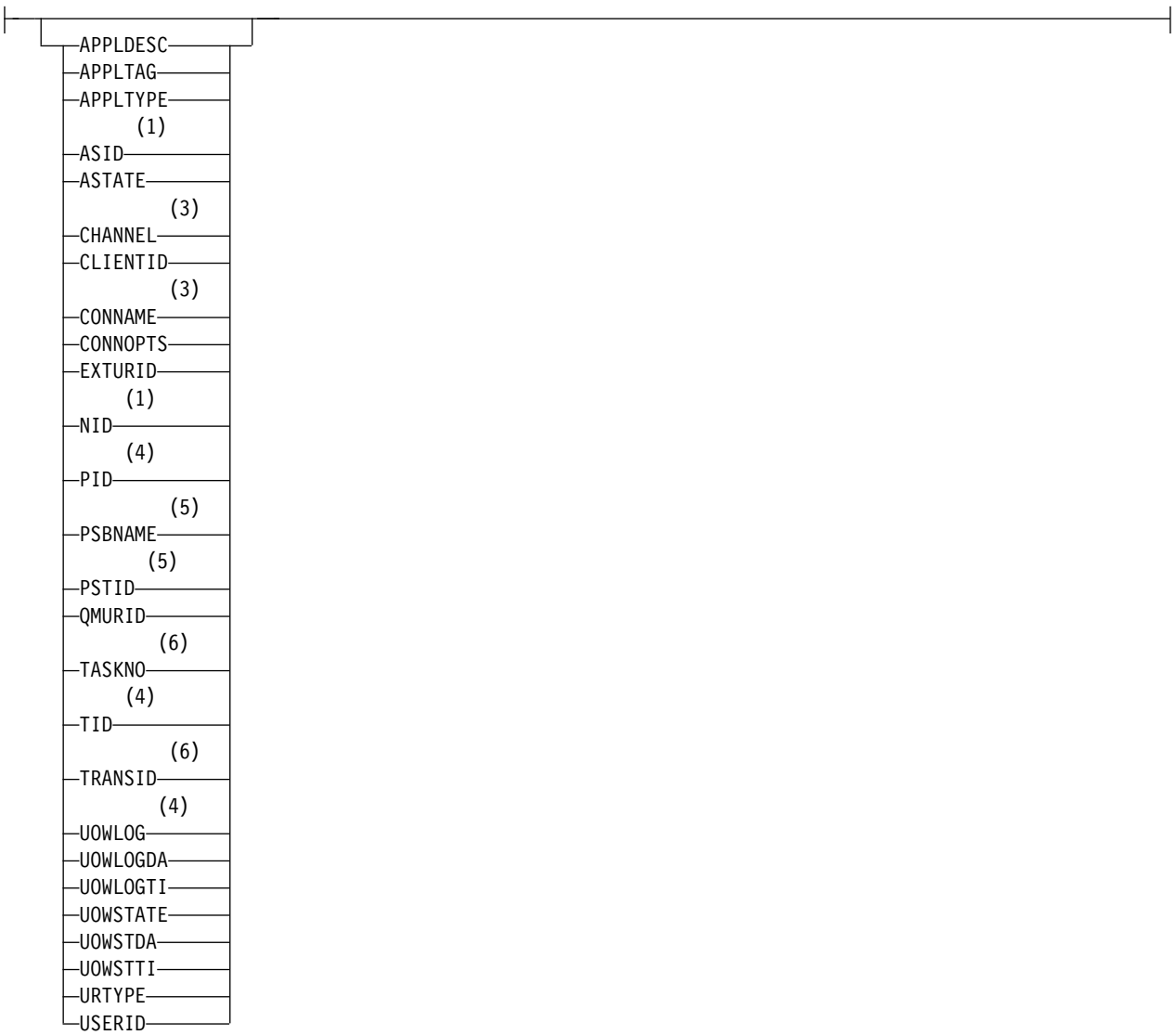
- "Usage notes for DISPLAY CONN" on page 825
- "Parameter descriptions for DISPLAY CONN" on page 825
- "Connection attributes" on page 827
- "Handle attributes" on page 832
- "Full attributes" on page 835

Synonym: DIS CONN

DISPLAY CONN



Connection attributes:



Handle attributes:

Notes:

1 Valid only on z/OS.

2 Valid only when the queue manager is a member of a queue-sharing group.

3 Valid only when the connection is associated with a channel.

4 Not valid on z/OS.

5 IMS only.

6 CICS for z/OS only.

Usage notes for DISPLAY CONN

1. z/OS This command is issued internally by IBM MQ on z/OS when taking a checkpoint, and when the queue manager is starting and stopping, so that a list of units of work that are in doubt at the time is written to the z/OS console log.
2. The TOPICSTR parameter might contain characters that cannot be translated into printable characters when the command output is displayed.

   z/OS On z/OS, these non-printable characters will be displayed as blanks.

   Multi On Multiplatforms platforms using **runmqsc**, these non-printable characters will be displayed as dots.
3. The state of asynchronous consumers, ASTATE, reflects that of the server-connection proxy on behalf of the client application; it does not reflect the client application state.

Parameter descriptions for DISPLAY CONN

You must specify a connection for which you want to display information. This can be a specific connection identifier or a generic connection identifier. A single asterisk (*) can be used as a generic connection identifier to display information for all connections.

(*generic-connid*)
The identifier of the connection definition for which information is to be displayed. A single asterisk (*) specifies that information for all connection identifiers is to be displayed.

When an application connects to IBM MQ, it is given a unique 24-byte connection identifier (ConnectionId). The value for CONN is formed by converting the last eight bytes of the ConnectionId to its 16 -character hexadecimal equivalent.

WHERE
Specify a filter condition to display only those connections that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this **DISPLAY** command. However, you cannot use the **CMDSCOPE**, **EXTCONN**, **QSGDISP**, **TYPE**, and **EXTURID** parameters as filter keywords.

operator
This is used to determine whether a connection satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which contain the specified item. You cannot use the **CONNOPTS** value MQCNO_STANDARD_BINDING with this operator.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not contain the specified item. You cannot use the **CONNOPTS** value MQCNO_STANDARD_BINDING with this operator.

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value NONE on the **UOWSTATE** parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string in the **APPLTAG** parameter) with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.
- An item in a list of values. Use CT or EX as the operator. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed.

ALL Specify this to display all the connection information of the requested type for each specified connection. This is the default if you do not specify a generic identifier, and do not request any specific parameters.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which it was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use **CMDSCOPE** as a filter keyword.

EXTCONN
The value for **EXTCONN** is based on the first sixteen bytes of the ConnectionId converted to its 32 -character hexadecimal equivalent.

Connections are identified by a 24-byte connection identifier. The connection identifier comprises a prefix, which identifies the queue manager, and a suffix which identifies the connection to that

queue manager. By default, the prefix is for the queue manager currently being administered, but you can specify a prefix explicitly by using the **EXTCONN** parameter. Use the **CONN** parameter to specify the suffix.

When connection identifiers are obtained from other sources, specify the fully qualified connection identifier (both **EXTCONN** and **CONN**) to avoid possible problems related to non-unique **CONN** values.

Do not specify both a generic value for **CONN** and a non-generic value for **EXTCONN**.

You cannot use **EXTCONN** as a filter keyword.

TYPE Specifies the type of information to be displayed. Values are:

CONN
Connection information for the specified connection.

z/OS On z/OS, this includes threads which may be logically or actually disassociated from a connection, together with those that are in-doubt and for which external intervention is needed to resolve them. These latter threads are those that **DIS THREAD TYPE(INDOUBT)** would show.

HANDLE
Information relating to any objects opened by the specified connection.

***** Display all available information relating to the connection.

ALL Display all available information relating to the connection.

z/OS On z/OS, if you specify **TYPE**(*ALL*|*) and **WHERE**(*xxxxx*) you only get CONN or HANDLE information returned, based on the **WHERE** specification. That is, if the *xxxxx* is a condition relating to handle attributes then only handle attributes for the connection are returned.

URDISP
Specifies the unit of recovery disposition of connections to be displayed. Values are:

ALL Display all connections. This is the default option.

GROUP
Display only those connections with a GROUP unit of recovery disposition.

QMGR
Display only those connections with a QMGR unit of recovery disposition.

Connection attributes

If **TYPE** is set to CONN, the following information is always returned for each connection that satisfies the selection criteria, except where indicated:

- Connection identifier (**CONN** parameter)
- Type of information returned (**TYPE** parameter)

The following parameters can be specified for **TYPE(CONN)** to request additional information for each connection. If a parameter is specified that is not relevant for the connection, operating environment, or type of information requested, that parameter is ignored.

APPLDESC
A string containing a description of the application connected to the queue manager, where it is known. If the application is not recognized by the queue manager the description returned is blank.

APPLTAG

A string containing the tag of the application connected to the queue manager. It is one of the following:

- z/OS z/OS batch job name
- z/OS TSO USERID
- CICS APPLID
- z/OS IMS region name
- Channel initiator job name
- IBM i IBM i job name
- UNIX UNIX process

  Notes:

  – HP-UX On HP-UX, if the process name exceeds 14 characters, only the first 14 characters are shown.

  – Linux Solaris On Linux and Solaris, if the process name exceeds 15 characters, only the first 15 characters are shown.

  – AIX On AIX, if the process name exceeds 28 characters, only the first 28 characters are shown.
- Windows Windows process

  Note: This consists of the full program path and executable file name. If it is more than 28 characters long, only the last 28 characters are shown.
- Internal queue manager process name

APPLTYPE

A string indicating the type of the application that is connected to the queue manager. It is one of the following:

BATCH
Application using a batch connection

RRSBATCH
RRS-coordinated application using a batch connection

CICS CICS transaction

IMS IMS transaction

CHINIT
Channel initiator

IBM i **OS400**
An IBM i application

SYSTEM
Queue manager

SYSTEMEXT
Application performing an extension of function that is provided by the queue manager

UNIX **UNIX**
A UNIX application

USER A user application

Windows **WINDOWSNT**
A Windows application

z/OS **ASID**
A 4-character address-space identifier of the application identified by **APPLTAG**. It distinguishes duplicate values of **APPLTAG**.

This parameter is returned only on z/OS when the **APPLTYPE** parameter does not have the value SYSTEM.

This parameter is valid only on z/OS.

ASTATE
The state of asynchronous consumption on this connection handle.

Possible values are:

SUSPENDED
An MQCTL call with the Operation parameter set to MQOP_SUSPEND has been issued against the connection handle so that so that asynchronous message consumption is temporarily suspended on this connection.

STARTED
An MQCTL call with the Operation parameter set to MQOP_START has been issued against the connection handle so that asynchronous message consumption can proceed on this connection.

STARTWAIT
An MQCTL call with the Operation parameter set to MQOP_START_WAIT has been issued against the connection handle so that asynchronous message consumption can proceed on this connection.

STOPPED
An MQCTL call with the Operation parameter set to MQOP_STOP has been issued against the connection handle so that asynchronous message consumption cannot currently proceed on this connection.

NONE
No MQCTL call has been issued against the connection handle. Asynchronous message consumption cannot currently proceed on this connection.

CHANNEL
The name of the channel that owns the connection. If there is no channel associated with the connection, this parameter is blank.

CLIENTID
The client ID of the client that is using the connection. If there is no client ID associated with the connection, this parameter is blank.

CONNAME
The connection name associated with the channel that owns the connection. If there is no channel associated with the connection, this parameter is blank.

CONNOPTS
The connect options currently in force for this application connection. Possible values are:

- MQCNO_HANDLE_SHARE_BLOCK
- MQCNO_HANDLE_SHARE_NO_BLOCK
- MQCNO_HANDLE_SHARE_NONE
- MQCNO_SHARED_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_ISOLATED_BINDING

- MQCNO_FASTPATH_BINDING
- MQCNO_SERIALIZE_CONN_TAG_Q_MGR
- MQCNO_SERIALIZE_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_Q_MGR
- MQCNO_RESTRICT_CONN_TAG_QSG
- MQCNO_ACCOUNTING_Q_ENABLED
- MQCNO_ACCOUNTING_Q_DISABLED
- MQCNO_ACCOUNTING_MQI_ENABLED
- MQCNO_ACCOUNTING_MQI_DISABLED

You cannot use the value MQCNO_STANDARD_BINDING as a filter value with the `CT` and `EX` operators on the **WHERE** parameter.

EXTURID
The external unit of recovery identifier associated with this connection. Its format is determined by the value of **URTYPE**.

You cannot use **EXTURID** as a filter keyword.

z/OS **NID**
Origin identifier, set only if the value of **UOWSTATE** is `UNRESOLVED`. This is a unique token identifying the unit of work within the queue manager. It is of the form `origin-node.origin-urid` where
- `origin-node` identifies the originator of the thread, except in the case where **APPLTYPE** is set to `RRSBATCH`, when it is omitted.
- `origin-urid` is the hexadecimal number assigned to the unit of recovery by the originating system for the specific thread to be resolved.

This parameter is valid only on z/OS.

PID Number specifying the process identifier of the application that is connected to the queue manager.

z/OS This parameter is not valid on z/OS.

z/OS **PSBNAME**
The 8-character name of the program specification block (PSB) associated with the running IMS transaction. You can use the **PSBNAME** and **PSTID** to purge the transaction using IMS commands. It is valid on z/OS only.

This parameter is returned only when the **APPLTYPE** parameter has the value IMS.

z/OS **PSTID**
The 4-character IMS program specification table (PST) region identifier for the connected IMS region. It is valid on z/OS only.

This parameter is returned only when the **APPLTYPE** parameter has the value IMS.

QMURID
The queue manager unit of recovery identifier.

z/OS On z/OS, this is an 8-byte log RBA, displayed as 16 hexadecimal characters.

Multi On Multiplatforms, this is an 8-byte transaction identifier, displayed as `m.n` where `m` and `n` are the decimal representation of the first and last 4 bytes of the transaction identifier.

z/OS You can use **QMURID** as a filter keyword. On z/OS, you must specify the filter value as a hexadecimal string.

Multi On platforms other than z/OS, you must specify the filter value as a pair of decimal numbers separated by a period (.). You can only use the `EQ`, `NE`, `GT`, `LT`, `GE`, or `LE` filter

operators.

z/OS However, on z/OS, if log shunting has taken place, as indicated by message CSQR026I, instead of the RBA you have to use the URID from the message.

z/OS **TASKNO**
A 7-digit CICS task number. This number can be used in the CICS command "CEMT SET TASK(taskno) PURGE" to end the CICS task. This parameter is valid on z/OS only.

This parameter is returned only when the **APPLTYPE** parameter has the value CICS.

TID Number specifying the thread identifier within the application process that has opened the specified queue.

z/OS This parameter is not valid on z/OS.

z/OS **TRANSID**
A 4-character CICS transaction identifier. This parameter is valid only on z/OS.

This parameter is returned only when the **APPLTYPE** parameter has the value CICS.

Multi **UOWLOG**
The file name of the extent to which the transaction associated with this connection first wrote.

Multi This parameter is valid only on Multiplatforms.

UOWLOGDA
The date that the transaction associated with the current connection first wrote to the log.

UOWLOGTI
The time that the transaction associated with the current connection first wrote to the log.

UOWSTATE
The state of the unit of work. It is one of the following:

NONE
There is no unit of work.

ACTIVE
The unit of work is active.

PREPARED
The unit of work is in the process of being committed.

z/OS **UNRESOLVED**
The unit of work is in the second phase of a two-phase commit operation. IBM MQ holds resources on its behalf and external intervention is required to resolve it. This might be as simple as starting the recovery coordinator (such as CICS, IMS, or RRS) or it might involve a more complex operation such as using the **RESOLVE INDOUBT** command. The UNRESOLVED value can occur only on z/OS.

UOWSTDA
The date that the transaction associated with the current connection was started.

UOWSTTI
The time that the transaction associated with the current connection was started.

URTYPE
The type of unit of recovery as seen by the queue manager. It is one of the following:

- z/OS CICS (valid only on z/OS)
- XA
- z/OS RRS (valid only on z/OS)
- z/OS IMS (valid only on z/OS)

- QMGR

URTYPE identifies the **EXTURID** type and not the type of the transaction coordinator. When **URTYPE** is QMGR, the associated identifier is in **QMURID** (and not **EXTURID**).

USERID
The user identifier associated with the connection.

This parameter is not returned when **APPLTYPE** has the value SYSTEM.

Handle attributes

If **TYPE** is set to HANDLE, the following information is always returned for each connection that satisfies the selection criteria, except where indicated:

- Connection identifier (**CONN** parameter)
- Read ahead status (**DEFREADA** parameter)
- Type of information returned (**TYPE** parameter)
- Handle status (**HSTATE**)
- Object name (**OBJNAME** parameter)
- Object type (**OBJTYPE** parameter)

The following parameters can be specified for **TYPE(HANDLE)** to request additional information for each queue. If a parameter is specified that is not relevant for the connection, operating environment, or type of status information requested, that parameter is ignored.

ASTATE
The state of the asynchronous consumer on this object handle.

Possible values are:

ACTIVE
An MQCB call has set up a function to call back to process messages asynchronously and the connection handle has been started so that asynchronous message consumption can proceed.

INACTIVE
An MQCB call has set up a function to call back to process messages asynchronously but the connection handle has not yet been started, or has been stopped or suspended, so that asynchronous message consumption cannot currently proceed.

SUSPENDED
The asynchronous consumption callback has been suspended so that asynchronous message consumption cannot currently proceed on this object handle. This can be either because an MQCB call with Operation MQOP_SUSPEND has been issued against this object handle by the application, or because it has been suspended by the system. If it has been suspended by the system, as part of the process of suspending asynchronous message consumption the callback function will be called with the reason code that describes the problem resulting in suspension. This will be reported in the Reason field in the MQCBC structure that is passed to the callback function.

For asynchronous message consumption to proceed, the application must issue an MQCB call with the Operation parameter set to MQOP_RESUME.

SUSPTEMP
The asynchronous consumption callback has been temporarily suspended by the system so that asynchronous message consumption cannot currently proceed on this object handle. As part of the process of suspending asynchronous message consumption, the

callback function will be called with the reason code that describes the problem resulting in suspension. This will be reported in the Reason field in the MQCBC structure passed to the callback function.

The callback function will be called again when asynchronous message consumption is resumed by the system, when the temporary condition has been resolved.

NONE
An MQCB call has not been issued against this handle, so no asynchronous message consumption is configured on this handle.

DEST The destination queue for messages that are published to this subscription. This parameter is only relevant for handles of subscriptions to topics. It is not returned for other handles.

DESTQMGR
The destination queue manager for messages that are published to this subscription. This parameter is relevant only for handles of subscriptions to topics. It is not returned for other handles. If DEST is a queue that is hosted on the local queue manager, this parameter will contain the local queue manager name. If DEST is a queue that is hosted on a remote queue manager, this parameter will contain the name of the remote queue manager.

HSTATE
The state of the handle.

Possible values are:

ACTIVE
An API call from this connection is currently in progress for this object. If the object is a queue, this condition can arise when an MQGET WAIT call is in progress.

If there is an MQGET SIGNAL outstanding, then this does not mean, by itself, that the handle is active.

INACTIVE
No API call from this connection is currently in progress for this object. If the object is a queue, this condition can arise when no MQGET WAIT call is in progress.

OBJNAME
The name of an object that the connection has open.

OBJTYPE
The type of the object that the connection has open. If this handle is that of a subscription to a topic, then the **SUBID** parameter identifies the subscription. You can then use the **DISPLAY SUB** command to find all the details about the subscription.

It is one of the following:
- QUEUE
- PROCESS
- QMGR
- z/OS STGCLASS (valid only on z/OS)
- NAMELIST
- CHANNEL
- AUTHINFO
- TOPIC

OPENOPTS
The open options currently in force for the connection for the object. This parameter is not returned for a subscription. Use the value in the **SUBID** parameter and the **DISPLAY SUB** command to find the details about the subscription.

Possible values are:

MQOO_INPUT_AS_Q_DEF
Open queue to get messages using queue-defined default.

MQOO_INPUT_SHARED
Open queue to get messages with shared access.

MQOO_INPUT_EXCLUSIVE
Open queue to get messages with exclusive access.

MQOO_BROWSE
Open queue to browse messages.

MQOO_OUTPUT
Open queue or topic to put messages.

MQOO_INQUIRE
Open queue to inquire attributes.

MQOO_SET
Open queue to set attributes.

MQOO_BIND_ON_OPEN
Bind handle to destination when queue is found.

MQOO_BIND_NOT_FIXED
Do not bind to a specific destination.

MQOO_SAVE_ALL_CONTEXT
Save context when message retrieved.

MQOO_PASS_IDENTITY_CONTEXT
Allow identity context to be passed.

MQOO_PASS_ALL_CONTEXT
Allow all context to be passed.

MQOO_SET_IDENTITY_CONTEXT
Allow identity context to be set.

MQOO_SET_ALL_CONTEXT
Allow all context to be set.

MQOO_ALTERNATE_USER_AUTHORITY
Validate with specified user identifier.

MQOO_FAIL_IF_QUIESCING
Fail if queue manager is quiescing.

z/OS **QSGDISP**
Indicates the disposition of the object. It is valid on z/OS only. The value is one of the following:

QMGR
The object was defined with **QSGDISP(QMGR)**.

COPY The object was defined with **QSGDISP(COPY)**.

SHARED
The object was defined with **QSGDISP(SHARED)**.

You cannot use **QSGDISP** as a filter keyword.

READA
The read ahead connection status.

Possible values are:

NO Read ahead of non-persistent messages is not enabled for this object.

YES Read ahead of non-persistent message is enabled for this object and is being used efficiently.

BACKLOG
Read ahead of non-persistent messages is enabled for this object. Read ahead is not being used efficiently because the client has been sent a large number of messages which are not being consumed.

INHIBITED
Read ahead was requested by the application but has been inhibited because of incompatible options specified on the first MQGET call.

SUBID
The internal, all-time unique identifier of the subscription. This parameter is relevant only for handles of subscriptions to topics. It is not returned for other handles.

Not all subscriptions show up in **DISPLAY CONN**; only those that have current handles open to the subscription show up. You can use the **DISPLAY SUB** command to see all subscriptions.

SUBNAME
The application's unique subscription name that is associated with the handle. This parameter is relevant only for handles of subscriptions to topics. It is not returned for other handles. Not all subscriptions will have a subscription name.

TOPICSTR
The resolved topic string. This parameter is relevant for handles with **OBJTYPE(TOPIC)**. For any other object type, this parameter is not returned.

Full attributes

If **TYPE** is set to *, or ALL, both Connection attributes and Handle attributes are returned for each connection that satisfies the selection criteria.

DISPLAY ENTAUTH on Multiplatforms: Multi

Use the MQSC command DISPLAY ENTAUTH to display the authorizations an entity has to a specified object.

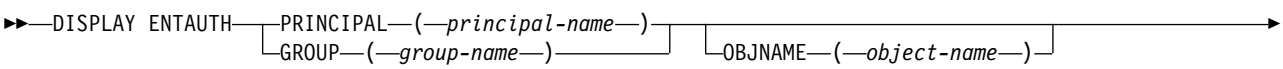
Using MQSC commands

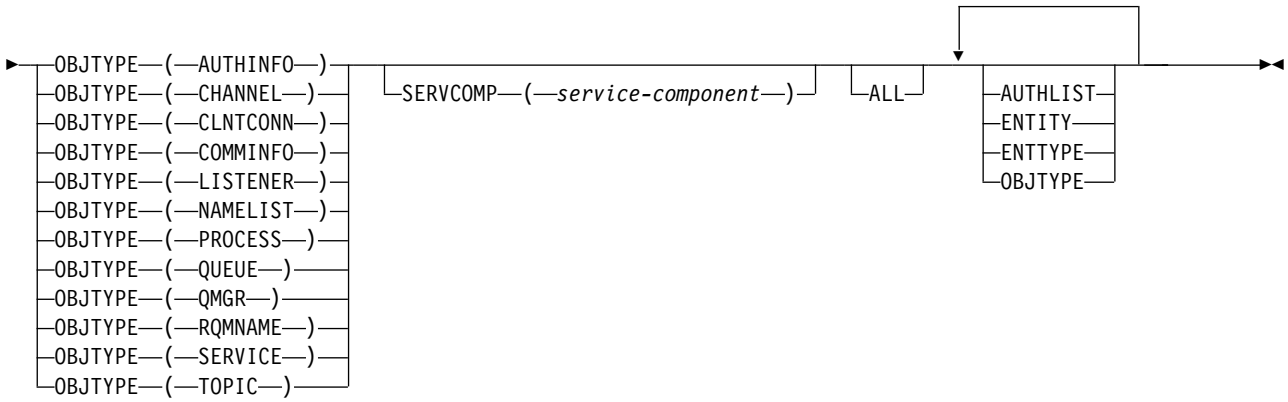
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Parameter descriptions" on page 836
- "Requested parameters" on page 837

Synonym: DIS ENTAUTH

DISPLAY ENTAUTH





Parameter descriptions

PRINCIPAL(*principal-name*)
A principal name. This is the name of a user for whom to retrieve authorizations to the specified object. On IBM MQ for Windows, the name of the principal can optionally include a domain name, specified in this format: `user@domain`.

You must specify either PRINCIPAL or GROUP.

GROUP(*group-name*)
A group name. This is the name of the user group on which to make the inquiry. You can specify one name only and it must be the name of an existing user group.

Windows For IBM MQ for Windows only, the group name can optionally include a domain name, specified in the following formats:

```
GroupName@domain
domain\GroupName
```

You must specify either PRINCIPAL or GROUP.

OBJNAME(*object-name*)
The name of the object or generic profile for which to display the authorizations.

This parameter is required unless the OBJTYPE parameter is QMGR. This parameter can be omitted if the OBJTYPE parameter is QMGR.

OBJTYPE
The type of object referred to by the profile. Specify one of the following values:

AUTHINFO
Authentication information record

CHANNEL
Channel

CLNTCONN
Client connection channel

COMMINFO
Communication information object

LISTENER
Listener

NAMELIST
Namelist

PROCESS
Process

QUEUE
Queue

QMGR
Queue manager

RQMNAME
Remote queue manager

SERVICE
Service

TOPIC
Topic

SERVCOMP(*service-component*)
The name of the authorization service for which information is to be displayed.

If you specify this parameter, it specifies the name of the authorization service to which the authorizations apply. If you omit this parameter, the inquiry is made to the registered authorization services in turn in accordance with the rules for chaining authorization services.

ALL
Specify this value to display all of the authorization information available for the entity and the specified profile.

Requested parameters

You can request the following information about the authorizations:

AUTHLIST
Specify this parameter to display the list of authorizations.

ENTITY
Specify this parameter to display the entity name.

ENTTYPE
Specify this parameter to display the entity type.

OBJTYPE
Specify this parameter to display the object type.

DISPLAY GROUP on z/OS: z/OS

Use the MQSC command DISPLAY GROUP to display information about the queue-sharing group to which the queue manager is connected. This command is valid only when the queue manager is a member of a queue-sharing group.

Using MQSC commands

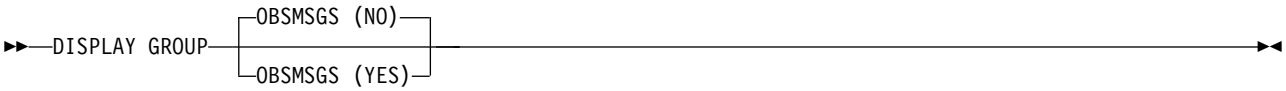
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for DISPLAY GROUP"
- "Parameter descriptions for DISPLAY GROUP"

Synonym: DIS GROUP

DISPLAY GROUP



Usage notes for DISPLAY GROUP

1. The response to the DISPLAY GROUP command is a series of messages containing information about the queue-sharing group to which the queue manager is connected.

   The following information is returned:

   - The name of the queue-sharing group
   - Whether all the queue managers that belong to the group are active or inactive
   - The names of all the queue managers that belong to the group.
   - If you specify OBSMSGS (YES), whether queue managers in the group contain obsolete messages in Db2

Parameter descriptions for DISPLAY GROUP

OBSMSGS

Specifies whether the command additionally looks for obsolete messages in Db2. This is optional. Possible values are:

NO Obsolete messages in Db2 are not looked for. This is the default value.

YES Obsolete messages in Db2 are looked for and messages containing information about any found are returned.

DISPLAY LISTENER on Multiplatforms: Multi

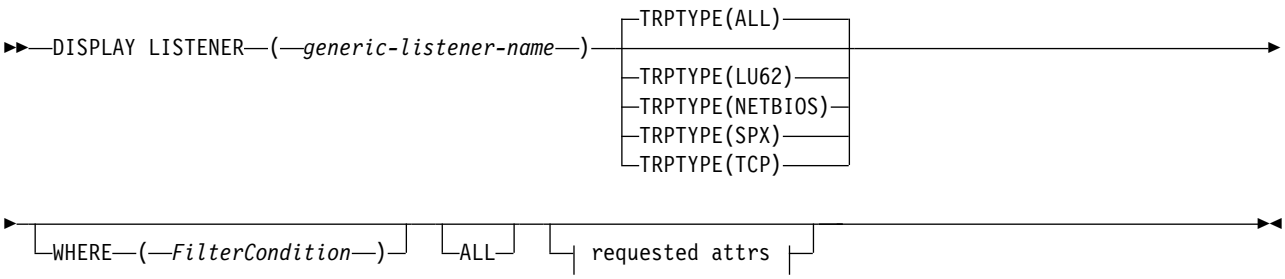
Use the MQSC command DISPLAY LISTENER to display information about a listener.

Using MQSC commands

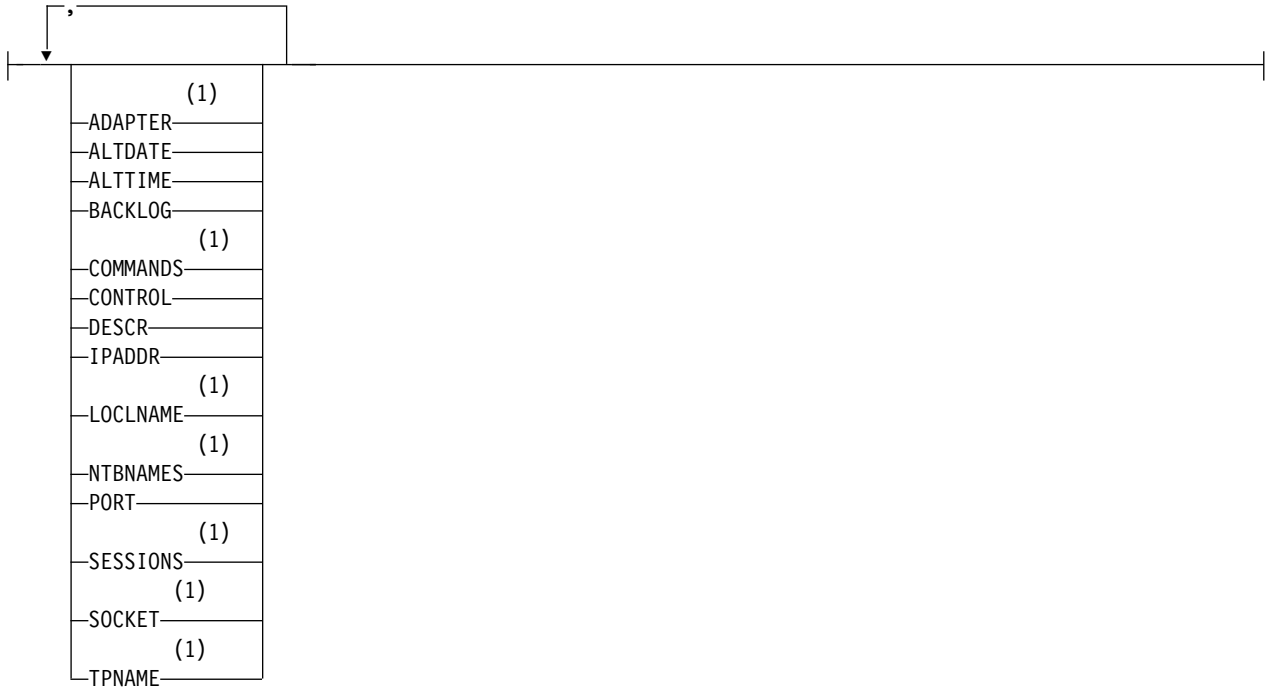
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

Synonym: DIS LSTR

DISPLAY LISTENER



Requested attrs:



Notes:

1 Valid only on Windows.

Usage notes

The values displayed describe the current definition of the listener. If the listener has been altered since it was started, the currently running instance of the listener object may not have the same values as the current definition.

Keyword and parameter descriptions for DISPLAY LISTENER

You must specify a listener for which you want to display information. You can specify a listener by using either a specific listener name or a generic listener name. By using a generic listener name, you can display either:

- Information about all listener definitions, by using a single asterisk (*), or
- Information about one or more listeners that match the specified name.

(*generic-listener-name*)
The name of the listener definition for which information is to be displayed. A single asterisk (*) specifies that information for all listener identifiers is to be displayed. A character string with an asterisk at the end matches all listeners with the string followed by zero or more characters.

TRPTYPE
Transmission protocol. If you specify this parameter, it must follow directly after the *generic-listener-name* parameter. If you do not specify this parameter, a default of ALL is assumed. Values are:

ALL This is the default value and displays information for all listeners.

LU62 Displays information for all listeners defined with a value of LU62 in their TRPTYPE parameter.

NETBIOS
Displays information for all listeners defined with a value of NETBIOS in their TRPTYPE parameter.

SPX Displays information for all listeners defined with a value of SPX in their TRPTYPE parameter.

TCP Displays information for all listeners defined with a value of TCP in their TRPTYPE parameter.

WHERE
Specify a filter condition to display information for those listeners that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

`filter-keyword`
Any parameter that can be used to display attributes for this DISPLAY command.

`operator`
This is used to determine whether a listener satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.
- A generic value. This is a character string. with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

ALL Specify this to display all the listener information for each specified listener. If this parameter is specified, any parameters that are requested specifically have no effect; all parameters are still displayed.

This is the default if you do not specify a generic identifier, and do not request any specific parameters.

Requested parameters

Specify one or more attributes that define the data to be displayed. The attributes can be specified in any order. Do not specify the same attribute more than once.

ADAPTER
The adapter number on which NetBIOS listens.

ALTDATE
The date on which the definition was last altered, in the form `yyyy-mm-dd`.

ALTTIME
The time at which the definition was last altered, in the form `hh.mm.ss`.

BACKLOG
The number of concurrent connection requests that the listener supports.

COMMANDS
The number of commands that the listener can use.

CONTROL
How the listener is to be started and stopped:

MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by use of the START LISTENER and STOP LISTENER commands.

QMGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY
The listener is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

DESCR
Descriptive comment.

IPADDR
The listener's IP address.

LOCLNAME
The NetBIOS local name that the listener uses.

NTBNAMES
The number of names that the listener can use.

PORT The port number for TCP/IP.

SESSIONS
The number of sessions that the listener can use.

SOCKET
SPX socket.

TPNAME
The LU6.2 transaction program name.

For more information on these parameters, see "DEFINE LISTENER on Multiplatforms" on page 642.

DISPLAY LOG on z/OS: z/OS

Use the MQSC command DISPLAY LOG to display log system parameters and information.

Using MQSC commands

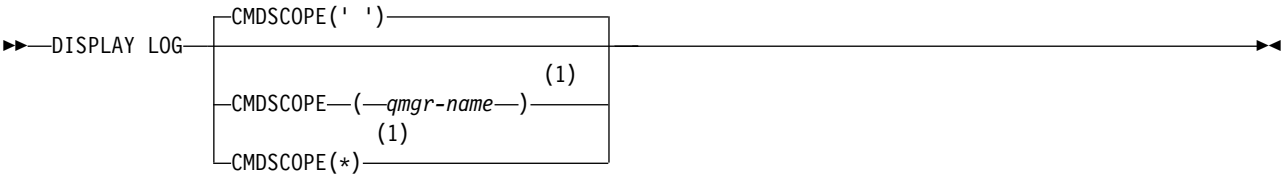
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 12CR. For an explanation of the source symbols, see Using commands on z/OS.

- "Usage notes for DISPLAY LOG"
- "Parameter descriptions for DISPLAY LOG" on page 843

Synonym: DIS LOG

DISPLAY LOG



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Usage notes for DISPLAY LOG

1. DISPLAY LOG returns a report that shows the initial log parameters, and the current values as changed by the SET LOG command:
   - Whether log compression is active (COMPLOG).
   - Whether zHyperWrite feature is being used (ZHYWRITE)
   - Length of time that an allowed archive read tape unit remains unused before it is deallocated (DEALLCT).

- Size of input buffer storage for active and archive log data sets (INBUFF).
- Size of output buffer storage for active and archive log data sets (OUTBUFF).
- Maximum number of dedicated tape units that can be set to read archive log tape volumes (MAXRTU).
- Maximum number of archive log volumes that can be recorded (MAXARCH).
- Maximum number of concurrent log offload tasks (MAXCNOFF)
- Whether archiving is on or off (OFFLOAD).
- Whether single or dual active logging is being used (TWOACTV).
- Whether single or dual archive logging is being used (TWOARCH).
- Whether single or dual BSDS is being used (TWOBSDS).
- Number of output buffers to be filled before they are written to the active log data sets (WRTHRSH).

It also returns a report about the status of the logs.

2. This command is issued internally by IBM MQ at the end of queue manager startup.

Parameter descriptions for DISPLAY LOG

CMDSCOPE

This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE cannot be used for commands issued from the first initialization input data set CSQINP1.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

DISPLAY LSSTATUS on Multiplatforms: 

Use the MQSC command DISPLAY LSSTATUS to display status information for one or more listeners.

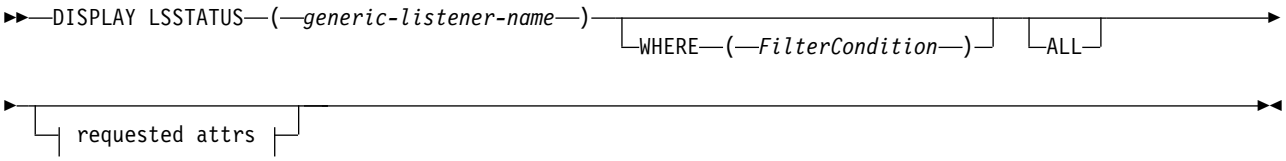
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

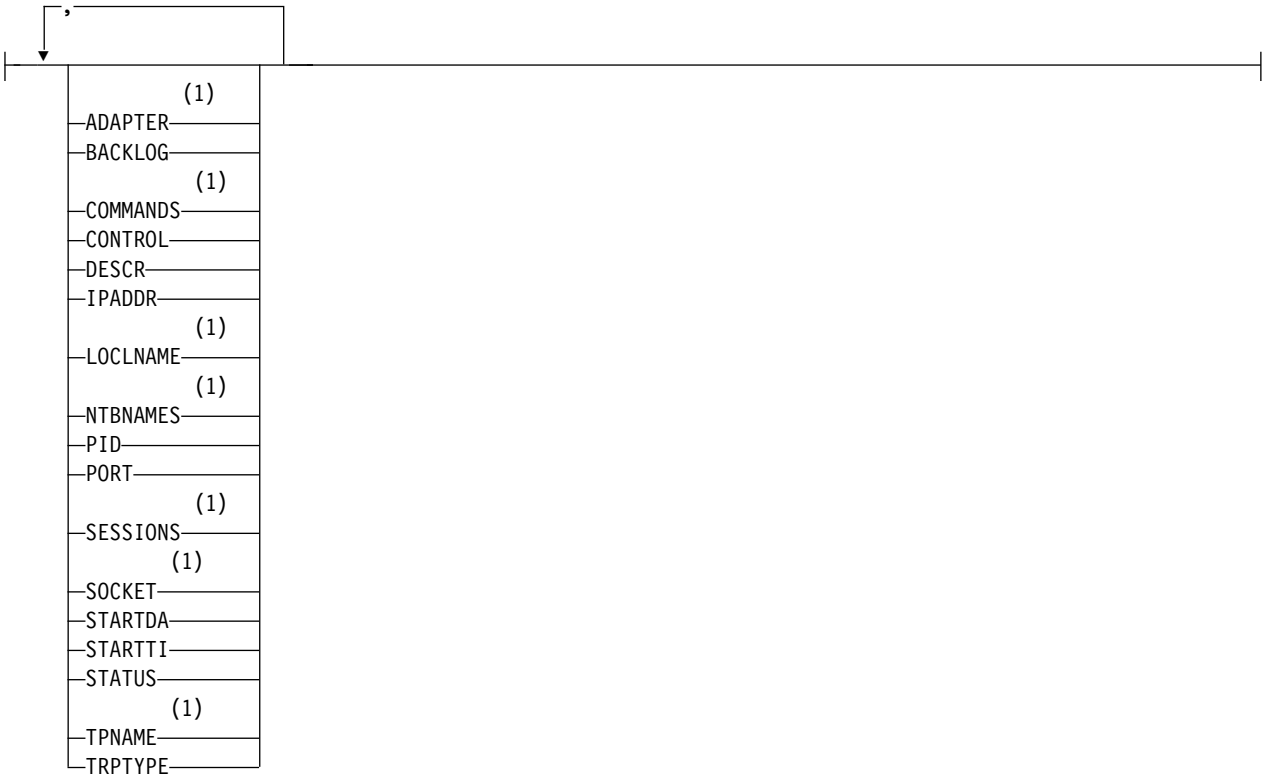
- “Keyword and parameter descriptions for DISPLAY LSSTATUS” on page 844
- “Requested parameters” on page 845

Synonym: DIS LSSTATUS

DISPLAY LSSTATUS



Requested attrs:



Notes:

1 Valid only on Windows.

Keyword and parameter descriptions for DISPLAY LSSTATUS

You must specify a listener for which you want to display status information. You can specify a listener by using either a specific listener name or a generic listener name. By using a generic listener name, you can display either:

- Status information for all listener definitions, by using a single asterisk (*), or
- Status information for one or more listeners that match the specified name.

(*generic-listener-name*)

The name of the listener definition for which status information is to be displayed. A single asterisk (*) specifies that information for all connection identifiers is to be displayed. A character string with an asterisk at the end matches all listeners with the string followed by zero or more characters.

WHERE

Specify a filter condition to display information for those listeners that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Any parameter that can be used to display attributes for this DISPLAY command.

operator
This is used to determine whether a listener satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.
- A generic value. This is a character string. with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

ALL Display all the status information for each specified listener. This is the default if you do not specify a generic name, and do not request any specific parameters.

Requested parameters

Specify one or more attributes that define the data to be displayed. The attributes can be specified in any order. Do not specify the same attribute more than once.

ADAPTER
The adapter number on which NetBIOS listens.

BACKLOG
The number of concurrent connection requests that the listener supports.

CONTROL
How the listener is to be started and stopped:

MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by use of the START LISTENER and STOP LISTENER commands.

QMGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY
The listener is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

DESCR
Descriptive comment.

IPADDR
The listener's IP address.

LOCLNAME
The NetBIOS local name that the listener uses.

NTBNAMES
The number of names that the listener can use.

PID The operating system process identifier associated with the listener.

PORT The port number for TCP/IP.

SESSIONS
The number of sessions that the listener can use.

SOCKET
SPX socket.

STARTDA
The date on which the listener was started.

STARTTI
The time at which the listener was started.

STATUS
The current status of the listener. It can be one of:

RUNNING
The listener is running.

STARTING
The listener is in the process of initializing.

STOPPING
The listener is stopping.

TPNAME
The LU6.2 transaction program name.

TRPTYPE
Transport type.

For more information on these parameters, see “DEFINE LISTENER on Multiplatforms” on page 642.

DISPLAY MAXSMSGS on z/OS: z/OS

Use the MQSC command DISPLAY MAXSMSGS to see the maximum number of messages that a task can get or put within a single unit of recovery.

Using MQSC commands

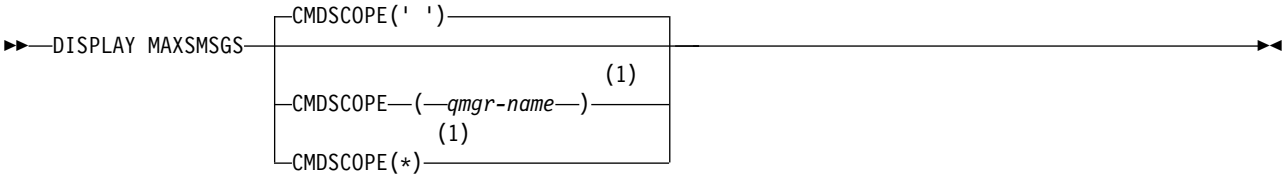
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes"
- "Parameter descriptions for DISPLAY MAXSMSGS"

Synonym: DIS MAXSM

DISPLAY MAXSMSGS



Notes:

1 Valid only on full function IBM MQ for z/OS when the queue manager is a member of a queue-sharing group.

Usage notes

This command is valid only on z/OS and is retained for compatibility with earlier releases, although it can no longer be issued from the CSQINP1 initialization data set. You should use the MAXUMSGS parameter of the DISPLAY QMGR command instead.

Parameter descriptions for DISPLAY MAXSMSGS

CMDSCOPE

This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

''
The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

*
The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

DISPLAY NAMELIST:

Use the MQSC command DISPLAY NAMELIST to display the names in a namelist.

Using MQSC commands

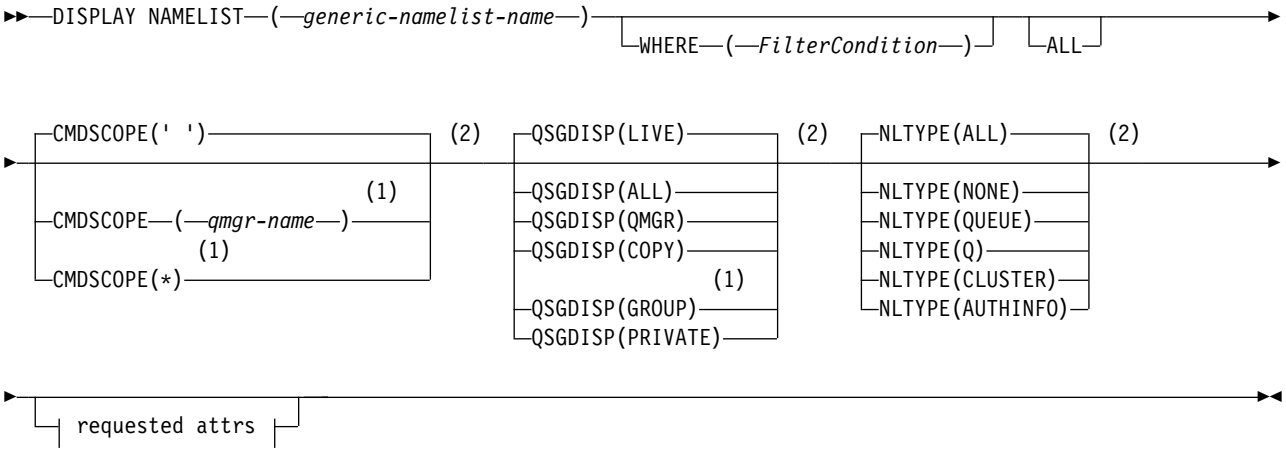
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for DISPLAY NAMELIST"
- "Requested parameters" on page 851

Synonym: DIS NL

DISPLAY NAMELIST



Requested attrs:

Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Parameter descriptions for DISPLAY NAMELIST

You must specify the name of the namelist definition you want to display. This can be a specific namelist name or a generic namelist name. By using a generic namelist name, you can display either:

- All namelist definitions
- One or more namelists that match the specified name

(*generic-namelist-name*)

The name of the namelist definition to be displayed (see Rules for naming IBM MQ objects). A trailing asterisk (*) matches all namelists with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all namelists.

WHERE

Specify a filter condition to display only those namelists that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

`filter-keyword`

Almost any parameter that can be used to display attributes for this DISPLAY command. However, you cannot use the CMDSCOPE or QSGDISP parameters as filter keywords. You cannot use NLTYPE as a filter keyword if you also use it to select namelists.

`operator`

This is used to determine whether a namelist satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which contain the specified item.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not contain the specified item.

CTG Contains an item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use this to display objects the attributes of which match the generic string.

EXG Does not contain any item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not match the generic string.

`filter-value`

The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value NONE on the NLTYPE parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. The characters must be valid for the attribute you are testing. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

- An item in a list of values. The value can be explicit or, if it is a character value, it can be explicit or generic. If it is explicit, use CT or EX as the operator. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed. If it is generic, use CTG or EXG as the operator. If ABC* is specified with the operator CTG, all items where one of the attribute values begins with ABC are listed.

ALL Specify this to display all the parameters. If this parameter is specified, any parameters that are requested specifically have no effect; all the parameters are displayed.

This is the default if you do not specify a generic name, and do not request any specific parameters.

z/OS On z/OS this is also the default if you specify a filter condition using the WHERE parameter, but on other platforms only requested attributes are displayed.

z/OS **CMDSCOPE**

This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

z/OS **QSGDISP**

Specifies the disposition of the objects for which information is to be displayed. Values are:

LIVE This is the default value and displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

ALL Displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with QSGDISP(GROUP).

If QSGDISP(ALL) is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

In a shared queue manager environment, use

```
DISPLAY NAMELIST(name) CMDSCOPE(*) QSGDISP(ALL)
```

to list ALL objects matching

```
name
```

in the queue-sharing group without duplicating those in the shared repository.

COPY Display information only for objects defined with QSGDISP(COPY).

GROUP
Display information only for objects defined with QSGDISP(GROUP). This is allowed only if there is a shared queue manager environment.

PRIVATE
Display information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY). Note that QSGDISP(PRIVATE) displays the same information as QSGDISP(LIVE).

QMGR
Display information only for objects defined with QSGDISP(QMGR).

QSGDISP displays one of the following values:

QMGR
The object was defined with QSGDISP(QMGR).

GROUP
The object was defined with QSGDISP(GROUP).

COPY The object was defined with QSGDISP(COPY).

You cannot use QSGDISP as a filter keyword.

z/OS **NLTYPE**
Indicates the type of namelist to be displayed.

This parameter is valid only on z/OS.

ALL
Displays namelists of all types. This is the default.

NONE
Displays namelists of type NONE.

QUEUE or Q
Displays namelists that hold lists of queue names.

CLUSTER
Displays namelists that are associated with clustering.

AUTHINFO
Displays namelists that contain lists of authentication information object names.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

The default, if no parameters are specified (and the ALL parameter is not specified) is that the object names, and, on z/OS, their NLTYPEs and QSGDISP are displayed.

ALTDATE
The date on which the definition was last altered, in the form `yyyy-mm-dd`

ALTTIME
The time at which the definition was last altered, in the form `hh.mm.ss`

DESCR
Description

NAMCOUNT
Number of names in the list

NAMES
List of names

See “DEFINE NAMELIST” on page 648 for more information about the individual parameters.

DISPLAY POLICY on Multiplatforms: Multi

Use the MQSC command DISPLAY POLICY to display a security policy.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- “Parameter descriptions for DISPLAY POLICY”

DISPLAY POLICY

Parameter descriptions for DISPLAY POLICY

(generic-policy-name)
Specifies the policy name, or names, to be displayed.

You can specify wildcard characters so that you can display multiple policy names.

The name of the policy, or policies (or part of the policy name or names) to display are the same as the name of the queue, or queues, that the policies control.

DISPLAY PROCESS:

Use the MQSC command DISPLAY PROCESS to display the attributes of one or more IBM MQ processes.

Using MQSC commands

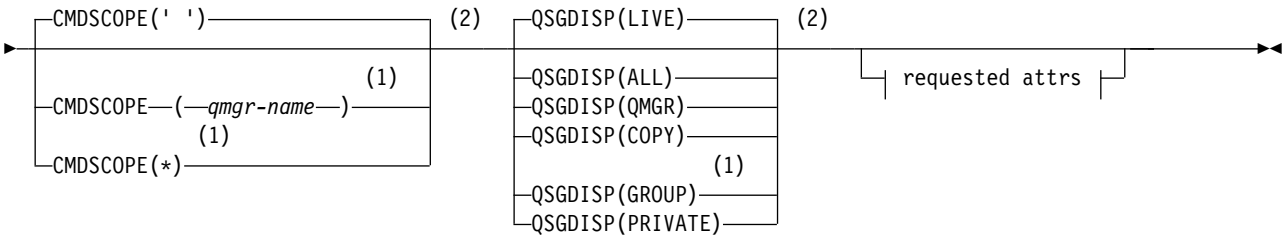
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- “Parameter descriptions for DISPLAY PROCESS” on page 853
- “Requested parameters” on page 855

Synonym: DIS PRO

DISPLAY PROCESS



Requested attrs:

Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Parameter descriptions for DISPLAY PROCESS

You must specify the name of the process you want to display. This can be a specific process name or a generic process name. By using a generic process name, you can display either:

- All process definitions
- One or more processes that match the specified name

(generic-process-name)
The name of the process definition to be displayed (see Rules for naming IBM MQ objects). A trailing asterisk (*) matches all processes with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all processes. The names must all be defined to the local queue manager.

WHERE
Specify a filter condition to display only those process definitions that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

`filter-keyword`
Almost any parameter that can be used to display attributes for this DISPLAY command.

z/OS However, you cannot use the CMDSCOPE or QSGDISP parameters as filter keywords.

`operator`
This is used to determine whether a process definition satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

`filter-value`
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value DEF on the APPLTYPE parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

ALL Specify this to display all the parameters. If this parameter is specified, any parameters that are requested specifically have no effect; all parameters are still displayed.

On AIX, HP-UX, Linux, IBM i, Solaris, and Windows z/OS , and z/OS, this is the default if you do not specify a generic name and do not request any specific parameters.

z/OS On z/OS this is also the default if you specify a filter condition using the WHERE parameter, but on other platforms only requested attributes are displayed.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

z/OS **QSGDISP**
Specifies the disposition of the objects for which information is to be displayed. Values are:

LIVE This is the default value and displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

ALL Displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with QSGDISP(GROUP).

If QSGDISP(LIVE) is specified or defaulted, or if QSGDISP(ALL) is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

COPY Display information only for objects defined with QSGDISP(COPY).

GROUP
Display information only for objects defined with QSGDISP(GROUP). This is allowed only if there is a shared queue manager environment.

PRIVATE
Display information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY). Note that QSGDISP(PRIVATE) displays the same information as QSGDISP(LIVE).

QMGR
Display information only for objects defined with QSGDISP(QMGR).

QSGDISP displays one of the following values:

QMGR
The object was defined with QSGDISP(QMGR).

GROUP
The object was defined with QSGDISP(GROUP).

COPY The object was defined with QSGDISP(COPY).

You cannot use QSGDISP as a filter keyword.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

The default, if no parameters are specified (and the ALL parameter is not specified) is that the object names and, on z/OS only, QSGDISP are displayed.

ALTDATE
The date on which the definition was last altered, in the form `yyyy-mm-dd`

ALTTIME
The time at which the definition was last altered, in the form `hh.mm.ss`

APPLICID
Application identifier

APPLTYPE
Application type. In addition to the values listed for this parameter in “Parameter descriptions for DEFINE PROCESS” on page 652, the value SYSTEM can be displayed. This indicates that the application type is a queue manager.

DESCR
Description

ENVRDATA
Environment data

USERDATA
User data

See “DEFINE PROCESS” on page 651 for more information about individual parameters.

DISPLAY PUBSUB:

Use the MQSC command DISPLAY PUBSUB to display publish/subscribe status information for a queue manager.

Using MQSC commands

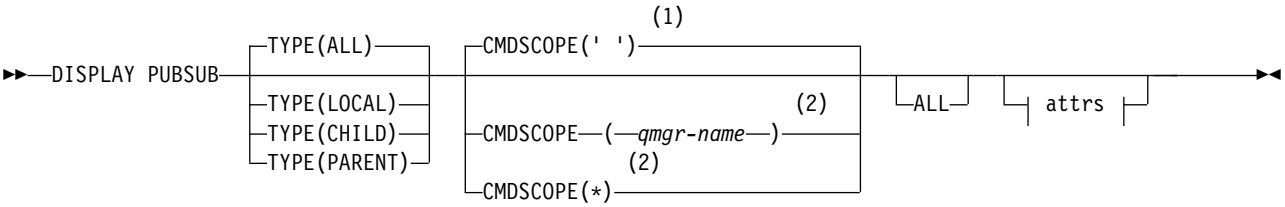
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- “Parameter descriptions for DISPLAY PUBSUB”
- “Returned parameters” on page 857

Synonym: None

DISPLAY PUBSUB



Attrs:

Notes:

1 Valid only on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

Parameter descriptions for DISPLAY PUBSUB

TYPE The type of publish/subscribe connections.

ALL Display the publish/subscribe status for this queue manager and for parent and child hierarchical connections.

CHILD
Display the publish/subscribe status for child connections.

LOCAL
Display the publish/subscribe status for this queue manager.

PARENT
Display the publish/subscribe status for the parent connection.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

' '
The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

*
The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

Returned parameters

A group of parameters is returned, containing the attributes TYPE, QMNAME, STATUS, SUBCOUNT, and TPCOUNT. This group is returned for the current queue manager if you set TYPE to LOCAL or ALL, for the parent queue manager if you set TYPE to PARENT or ALL, and for each child queue manager if you set TYPE to CHILD or ALL.

TYPE

CHILD
A child connection.

LOCAL
Information for this queue manager.

PARENT
The parent connection.

QMNAME
The name of the current queue manager or the remote queue manager connected as a parent or a child.

STATUS
The status of the publish/subscribe engine or the hierarchical connection. The publish/subscribe engine is initializing and is not yet operational. If the queue manager is a member of a cluster (has at least one CLUSRCVR defined), it remains in this state until the cluster cache is available.

z/OS On IBM MQ for z/OS, this requires that the Channel Initiator is running.

When TYPE is CHILD, the following values can be returned:

ACTIVE
The connection with the child queue manager is active.

ERROR
This queue manager is unable to initialize a connection with the child queue manager because of a configuration error. A message is produced in the queue manager logs to indicate the specific error. If you receive error message AMQ5821 or on z/OS systems CSQT821E, possible causes include:

- Transmit queue is full.
- Transmit queue put is disabled.

If you receive error message AMQ5814 or on z/OS systems CSQT814E, take the following actions:

- Check that the child queue manager is correctly specified.
- Ensure that broker is able to resolve the queue manager name of the child broker.

To resolve the queue manager name, at least one of the following resources must be configured:

- A transmission queue with the same name as the child queue manager name.
- A queue manager alias definition with the same name as the child queue manager name.
- A cluster with the child queue manager a member of the same cluster as this queue manager.
- A cluster queue manager alias definition with the same name as the child queue manager name.
- A default transmission queue.

After you have set up the configuration correctly, modify the child queue manager name to blank. Then set with the child queue manager name.

STARTING
Another queue manager is attempting to request that this queue manager become its parent.

If the child status remains in STARTING without progressing to ACTIVE, take the following actions:

- Check that the sender channel to child queue manager is running
- Check that the receiver channel from child queue manager is running

STOPPING
The queue manager is disconnecting.

If the child status remains in STOPPING, take the following actions:

- Check that the sender channel to child queue manager is running
- Check that the receiver channel from child queue manager is running

When TYPE is LOCAL, the following values can be returned:

ACTIVE
The publish/subscribe engine and the queued publish/subscribe interface are running. It is therefore possible to publish or subscribe using the application programming interface and the queues that are monitored by the queued publish/subscribe interface.

COMPAT
The publish/subscribe engine is running. It is therefore possible to publish or subscribe by using the application programming interface. The queued publish/subscribe interface is not running. Therefore, any message that is put to the queues that are monitored by the queued publish/subscribe interface are not acted upon by IBM MQ.

ERROR
The publish/subscribe engine has failed. Check your error logs to determine the reason for the failure.

INACTIVE
The publish/subscribe engine and the queued publish/subscribe interface are not running. It is therefore not possible to publish or subscribe using the application

programming interface. Any publish/subscribe messages that are put to the queues that are monitored by the queued publish/subscribe interface are not acted upon by IBM MQ.

If inactive and you want to start the publish/subscribe engine use the command **ALTER QMGR PSMODE(ENABLED)**.

STARTING
The publish/subscribe engine is initializing and is not yet operational. If the queue manager is a member of a cluster, that is, it has at least one CLUSRCVR defined, it remains in this state until the cluster cache is available.

z/OS On IBM MQ for z/OS, this requires that the Channel Initiator is running.

STOPPING
The publish/subscribe engine is stopping.

When TYPE is PARENT, the following values can be returned:

ACTIVE
The connection with the parent queue manager is active.

ERROR
This queue manager is unable to initialize a connection with the parent queue manager because of a configuration error. A message is produced in the queue manager logs to indicate the specific error. If you receive error message AMQ5821, z/OS or on z/OS systems CSQT821E, possible causes include:

- Transmit queue is full.
- Transmit queue put is disabled.

If you receive error message AMQ5814, z/OS or error message CSQT814E on z/OS systems, take the following actions:

- Check that the parent queue manager is correctly specified.
- Ensure that broker is able to resolve the queue manager name of the parent broker.

To resolve the queue manager name, at least one of the following resources must be configured:

- A transmission queue with the same name as the parent queue manager name.
- A queue manager alias definition with the same name as the parent queue manager name.
- A cluster with the parent queue manager a member of the same cluster as this queue manager.
- A cluster queue manager alias definition with the same name as the parent queue manager name.
- A default transmission queue.

After you have set up the configuration correctly, modify the parent queue manager name to blank. Then set with the parent queue manager name.

REFUSED
The connection has been refused by the parent queue manager. This might be caused by the following:

- The parent queue manager already has a child queue manager with the same name as this queue manager.
- The parent queue manager has used the command RESET QMGR TYPE(PUBSUB) CHILD to remove this queue manager as one of its children.

STARTING
The queue manager is attempting to request that another queue manager become its parent.

If the parent status remains in STARTING without progressing to ACTIVE, take the following actions:

- Check that the sender channel to parent queue manager is running
- Check that the receiver channel from parent queue manager is running

STOPPING
The queue manager is disconnecting from its parent.

If the parent status remains in STOPPING, take the following actions:

- Check that the sender channel to parent queue manager is running
- Check that the receiver channel from parent queue manager is running

SUBCOUNT
When TYPE is LOCAL, the total number of subscriptions against the local tree is returned. When TYPE is CHILD or PARENT, queue manager relations are not inquired and the value `NONE` is returned.

TPCOUNT
When TYPE is LOCAL, the total number of topic nodes in the local tree is returned. When TYPE is CHILD or PARENT, queue manager relations are not inquired and the value `NONE` is returned.

DISPLAY QMGR:

Use the MQSC command **DISPLAY QMGR** to display the queue manager parameters for this queue manager.

Using MQSC commands

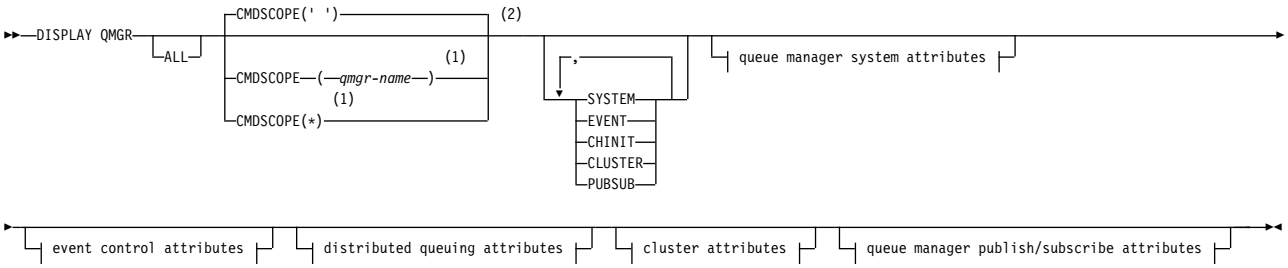
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources `2CR`. For an explanation of the source symbols, see Using commands on z/OS.

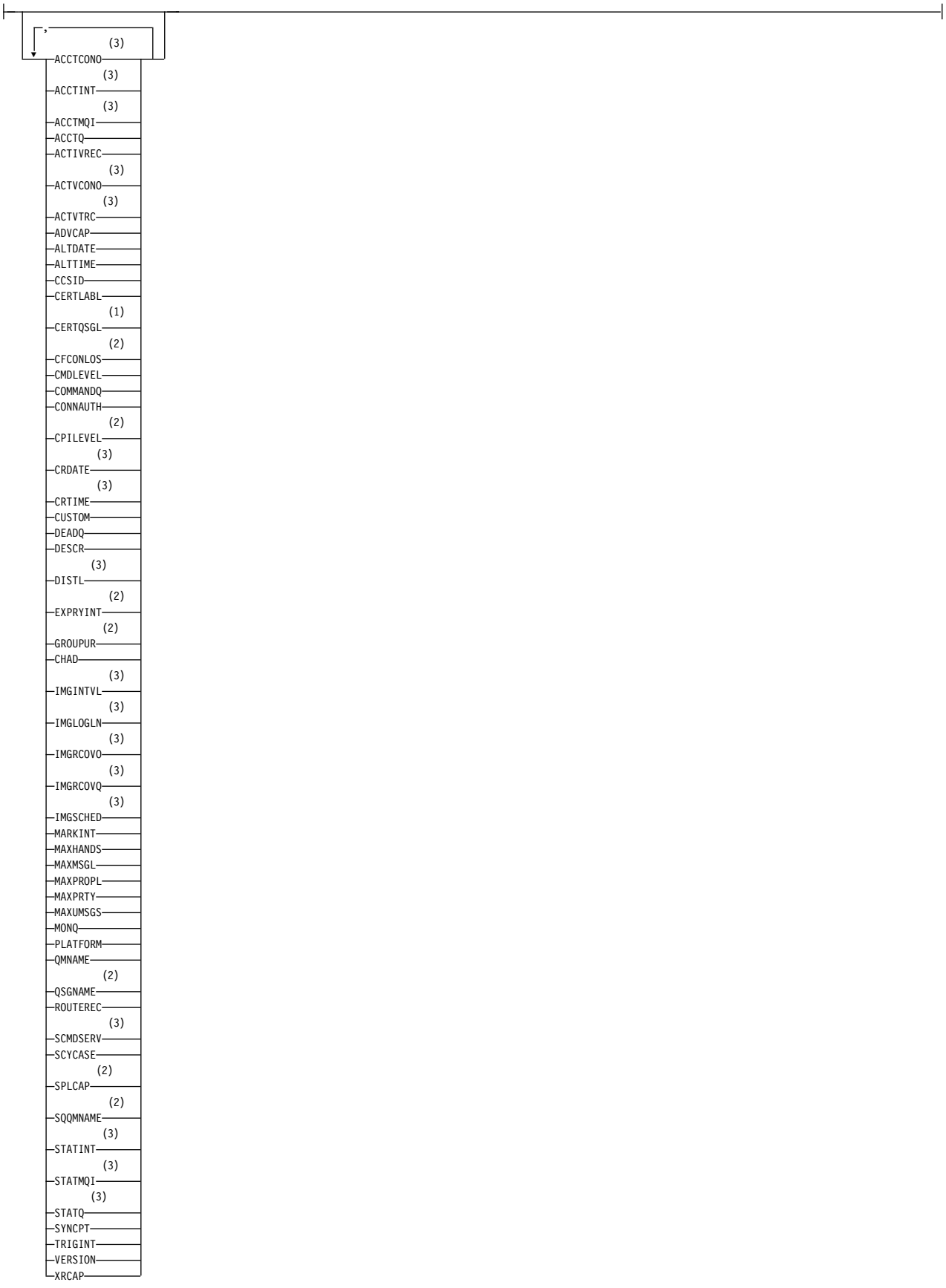
- Syntax diagram
- "Parameter descriptions for DISPLAY QMGR" on page 862
- "Requested parameters" on page 863

Synonym: **DIS QMGR**

DISPLAY QMGR



Queue manager system attributes:



Event control attributes:

Cluster attributes:

Queue manager publish/subscribe attributes:

Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Not valid on z/OS.

Parameter descriptions for DISPLAY QMGR

ALL Specify this parameter to display all the parameters. If this parameter is specified, any parameters that are requested specifically are ineffective; all parameters are still displayed.

Multi On Multiplatforms, this parameter is the default if you do not request any specific parameters.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This command is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of running this command is the same as entering the command on every queue manager in the queue-sharing group.

SYSTEM

Specify this parameter to display the set of queue manager system attributes that are available in the Queue manager system attrs list. See "Requested parameters" for information about these parameters.

If you specify this parameter, any request you make to display individual parameters within this set is ineffective.

EVENT

Specify this parameter to display the set of event control attributes that are available in the Event control attrs list. See "Requested parameters" for information about these parameters.

If you specify this parameter, any request you make to display individual parameters within this set is ineffective.

CHINIT

Specify this parameter to display the set of attributes relating to distributed queuing that are available in the Distributed queuing attrs list. You can also specify DQM to display the same set of attributes. See "Requested parameters" for information about these parameters.

If you specify this parameter, any request you make to display individual parameters within this set is ineffective.

CLUSTER

Specify this parameter to display the set of attributes relating to clustering that are available in the Cluster attrs list. See "Requested parameters" for information about these parameters.

If you specify this parameter, any request you make to display individual parameters within this set is ineffective.

PUBSUB

Specify this parameter to display the set of attributes relating to publish/subscribe that are available in the Queue manager pub/sub attrs list. See "Requested parameters" for information about these parameters.

If you specify this parameter, any request you make to display individual parameters within this set is ineffective.

Requested parameters

Note: If no parameters are specified (and the **ALL** parameter is not specified or defaulted), the queue manager name is returned.

You can request the following information for the queue manager:

Multi **ACCTCONO**
Whether the settings of the **ACCTQMQI** and **ACCTQ** queue manager parameters can be overridden. This parameter is valid only on Multiplatforms.

Multi **ACCTINT**
The interval at which intermediate accounting records are written. This parameter is valid only on Multiplatforms.

Multi **ACCTMQI**
Whether accounting information is to be collected for MQI data. This parameter is valid only on Multiplatforms.

ACCTQ
Whether accounting data collection is to be enabled for queues.

z/OS **ACTCHL**
The maximum number of channels that can be active at any time.

This parameter is valid only on z/OS.

ACTIVREC
Whether activity reports are to be generated if requested in the message.

Multi **ACTVCONO**
Whether the settings of the **ACTVTRC** queue manager parameter can be overridden. This parameter is valid only on Multiplatforms.

Multi **ACTVTRC**
Whether IBM MQ MQI application activity tracing information is to be collected. See Setting ACTVTRC to control collection of activity trace information. This parameter is valid only on Multiplatforms.

z/OS **ADOPTCHK**
Which elements are checked to determine whether an MCA is adopted when a new inbound channel is detected with the same name as an already active MCA.

This parameter is valid only on z/OS.

z/OS **ADOPTMCA**
Whether an orphaned MCA instance is to be restarted when a new inbound channel request matching the **ADOPTCHK** parameters is detected.

This parameter is valid only on z/OS.

MQ Adv. **ADVCAP**
Whether IBM MQ Advanced extended capabilities are available for a queue manager.

z/OS V 9.0.4 On z/OS, from IBM MQ Version 9.0.4, the queue manager sets the value to be ENABLED, only if the value of **QMGRPROD** is ADVANCEDVUE. For any other value of **QMGRPROD**, or if **QMGRPROD** is not set, the queue manager sets the value to DISABLED. If **ADVCAP** is ENABLED you must be entitled to IBM MQ Advanced for z/OS, Value Unit Edition (VUE). See "START QMGR on z/OS" on page 1037 and Installing IBM MQ Advanced for z/OS, Value Unit Edition for more information.

Multi V 9.0.5 On other platforms, from IBM MQ Version 9.0.5, the queue manager sets the value to be ENABLED, only if you have installed Managed File Transfer, XR, Advanced Message Security or RDQM. If you have not installed Managed File Transfer, XR, Advanced Message Security or RDQM, **ADVCAP** is set to DISABLED. If **ADVCAP** is ENABLED, you must be entitled

to IBM MQ Advanced. The list of installable components that enable **ADVCAP** might change in future releases. for more information, see IBM MQ components and features and Installing IBM MQ Advanced for Multiplatforms.

ALTDATE
The date on which the definition was last altered, in the form `yyyy-mm-dd`.

ALTTIME
The time at which the definition was last altered, in the form `hh.mm.ss`.

AUTHOREV
Whether authorization events are generated.

z/OS **BRIDGEEV**
On z/OS only, whether IMS bridge events are generated.

CCSID
Coded character set identifier. This parameter applies to all character string fields defined by the application programming interface (API), including the names of objects, and the creation date and time of each queue. It does not apply to application data carried as the text of messages.

CERTLABL
Specifies the certificate label that this queue manager used.

z/OS **CERTQSGL**
Specifies the queue-sharing group (QSG) certificate label.

This parameter is valid only on z/OS.

ULW **CERTVPOL**
Specifies which TLS certificate validation policy is used to validate digital certificates received from remote partner systems. This attribute can be used to control how strictly the certificate chain validation conforms to industry security standards. For more information about certificate validation policies, see Certificate validation policies in IBM MQ.

This parameter is valid only on UNIX, Linux, and Windows.

z/OS **CFCONLOS**
Specifies the action to be taken when the queue manager loses connectivity to the administration structure, or any CF structure with **CFCONLOS** set to `ASQMGR`.

This parameter is valid only on z/OS.

Multi **CHAD**
Whether auto-definition of receiver and server-connection channels is enabled.

z/OS This parameter is not valid on z/OS.

Multi **CHADEV**
Whether auto-definition events are enabled.

z/OS This parameter is not valid on z/OS.

CHADEXIT
The name of the channel auto-definition exit.

z/OS **CHIADAPS**
The number of adapter subtasks to use to process IBM MQ calls.

This parameter is valid only on z/OS.

z/OS **CHIDISPS**
The number of dispatchers to use for the channel initiator.

This parameter is valid only on z/OS.

CHISERVP
This field is reserved for IBM use only.

CHLAUTH
Whether channel authentication records are checked.

CHLEV
Whether channel events are generated.

CLWLEXIT
The name of the cluster workload exit.

CLWLDATA
The data passed to the cluster workload exit.

UNIX Windows z/OS **CLWLLEN**
The maximum number of bytes of message data that is passed to the cluster workload exit.

Linux This parameter is not valid on Linux.

CLWLMRUC
The maximum number of outbound cluster channels.

CLWLUSEQ
The behavior of MQPUTs for queues where **CLWLUSEQ** has a value of QMGR.

CMDEV
Whether command events are generated.

CMDLEVEL
Command level. This indicates the level of system control commands supported by the queue manager.

COMMANDQ
The name of the system-command input queue. Suitably authorized applications can put commands on this queue.

CONFIGEV
Whether configuration events are generated.

CONNAUTH
The name of an authentication information object that is used to provide the location of user ID and password authentication.

CPILEVEL
Reserved, this value has no significance.

CRDATE
The date on which the queue manager was created (in the form *yyyy-mm-dd*).

CRTIME
The time at which the queue manager was created (in the form *hh.mm.ss*).

CUSTOM
This attribute is reserved for the configuration of new features before separate attributes have been introduced. It can contain the values of zero or more attributes as pairs of attribute name and value in the form NAME(VALUE).

DEADQ
The name of the queue to which messages are sent if they cannot be routed to their correct destination (the dead-letter queue or undelivered-message queue). The default is blanks.

For example, messages are put on this queue when:

- A message arrives at a queue manager, destined for a queue that is not yet defined on that queue manager
- A message arrives at a queue manager, but the queue for which it is destined cannot receive it because, possibly:
  - The queue is full
  - The queue is inhibited for puts
  - The sending node does not have authority to put the message on the queue
- An exception message must be generated, but the queue named is not known to that queue manager

Note: Messages that have passed their expiry time are not transferred to this queue when they are discarded.

If the dead-letter queue is not defined, or full, or unusable for some other reason, a message that would have been transferred to it by a message channel agent is retained instead on the transmission queue.

If a dead-letter queue or undelivered-message queue is not specified, all blanks are returned for this parameter.

DEFCLXQ

The **DFTCLXQ** attribute controls which transmission queue is selected by default by cluster-sender channels to get messages from, to send the messages to cluster-receiver channels.

SCTQ

All cluster-sender channels send messages from SYSTEM.CLUSTER.TRANSMIT.QUEUE. The correlID of messages placed on the transmission queue identifies which cluster-sender channel the message is destined for.

SCTQ is set when a queue manager is defined. This behavior is implicit in versions of IBM WebSphere MQ, earlier than Version 7.5. In earlier versions, the queue manager attribute **DEFCLXQ** was not present.

CHANNEL

Each cluster-sender channel sends messages from a different transmission queue. Each transmission queue is created as a permanent dynamic queue from the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE.

If the queue manager attribute, **DEFCLXQ**, is set to CHANNEL,the default configuration is changed to cluster-sender channels being associated with individual cluster transmission queues. The transmission queues are permanent-dynamic queues created from the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE. Each transmission queue is associated with one cluster-sender channel. As one cluster-sender channel services a cluster transmission queue, the transmission queue contains messages for only one queue manager in one cluster. You can configure clusters so that each queue manager in a cluster contains only one cluster queue. In this case, the message traffic from a queue manager to each cluster queue is transferred separately from messages to other queues.

DEFXMITQ

Default transmission queue name. This parameter is the transmission queue on which messages, destined for a remote queue manager, are put if there is no other suitable transmission queue defined.

DESCR

Description.

Multi **DISTL**

Whether distribution lists are supported by the queue manager.

z/OS This parameter is not valid on z/OS.

z/OS **DNSGROUP**
This parameter is no longer used. Refer to WLM/DNS no longer supported. This parameter is valid only on z/OS

z/OS **DNSWLM**
This parameter is no longer used. Refer to WLM/DNS no longer supported. This parameter is valid only on z/OS.

z/OS **EXPRYINT**
On z/OS only, the approximate interval between scans for expired messages.

z/OS **GROUPUR**
On z/OS only, whether XA client applications are allowed to connect to this queue manager with a GROUP unit of recovery disposition.

V 9.0.2 **IMGINTVL**
The target frequency with which the queue manager automatically writes media images.

z/OS This parameter is not valid on z/OS.

V 9.0.2 **IMGLOGLN**
The target amount of recovery log written by which the queue manager automatically writes media images.

z/OS This parameter is not valid on z/OS.

V 9.0.2 **IMGRCOVO**
Whether specified objects are recoverable from a media image, if linear logging is being used.

z/OS This parameter is not valid on z/OS.

V 9.0.2 **IMGRCOVQ**
Whether a local or permanent dynamic queue object is recoverable from a media image, if linear logging is being used.

z/OS This parameter is not valid on z/OS.

V 9.0.2 **IMGSCHED**
Whether the queue manager automatically writes media images.

z/OS This parameter is not valid on z/OS.

z/OS **IGQ**
On z/OS only, whether intra-group queuing is to be used.

z/OS **IGQAUT**
On z/OS only, displays the type of authority checking used by the intra-group queuing agent.

z/OS **IGQUSER**
On z/OS only, displays the user ID used by the intra-group queuing agent.

INHIBTEV
Whether inhibit events are generated.

IPADDRV
Whether to use an IPv4 or IPv6 IP address for a channel connection in ambiguous cases.

LOCALEV
Whether local error events are generated.

Multi **LOGGEREV**
Whether recovery log events are generated. This parameter is valid only on Multiplatforms.

z/OS **LSTRTMR**
The time interval, in seconds, between attempts by IBM MQ to restart the listener after an APPC or TCP/IP failure.

This parameter is valid only on z/OS.

z/OS **LUGROUP**
The generic LU name to be used by the LU 6.2 listener that handles inbound transmissions for the queue-sharing group.

This parameter is valid only on z/OS.

z/OS **LUNAME**
The name of the LU to use for outbound LU 6.2 transmissions.

This parameter is valid only on z/OS.

z/OS **LU62ARM**
The suffix of the APPCPM member of SYS1.PARMLIB. This suffix nominates the LUADD for this channel initiator. When automatic restart manager (ARM) restarts the channel initiator, the z/OS command SET APPC= xx is issued.

This parameter is valid only on z/OS.

z/OS **LU62CHL**
The maximum number of channels that can be current, or clients that can be connected, that use the LU 6.2 transmission protocol. If the value of LU62CHL is zero, the LU 6.2 transmission protocol is not used.

This parameter is valid only on z/OS.

MARKINT
The mark browse interval in milliseconds.

z/OS **MAXCHL**
The maximum number of channels that can be current (including server-connection channels with connected clients).

This parameter is valid only on z/OS.

MAXHANDS
The maximum number of open handles that any one connection can have at any one time.

MAXMSGL
The maximum message length that can be handled by the queue manager. Individual queues or channels might have a smaller maximum than the value of this parameter.

MAXPROPL (*integer*)
The maximum length of property data in bytes that can be associated with a message.

MAXPRTY
The maximum priority. This value is 9.

MAXUMSGS
Maximum number of uncommitted messages within one sync point. The default value is 10000.

MAXUMSGS has no effect on MQ Telemetry. MQ Telemetry tries to batch requests to subscribe, unsubscribe, send, and receive messages from multiple clients into batches of work within a transaction.

MONACLS
Whether online monitoring data is to be collected for auto-defined cluster-sender channels, and, if so, the rate of data collection.

MONCHL
Whether online monitoring data is to be collected for channels, and, if so, the rate of data collection.

MONQ
Whether online monitoring data is to be collected for queues, and, if so, the rate of data collection.

z/OS **OPORTMAX**
The maximum value in the range of port numbers to be used when binding outgoing channels.

This parameter is valid only on z/OS.

z/OS **OPORTMIN**
The minimum value in the range of port numbers to be used when binding outgoing channels.

This parameter is valid only on z/OS.

PARENT
The name of the queue manager to which this queue manager is connected hierarchically as its child.

PERFMEV
Whether performance-related events are generated.

PLATFORM
The architecture of the platform on which the queue manager is running. The value of this parameter is:

- z/OS `MVS` (for z/OS platforms)
- `NSK`
- `OS2`
- `OS400`
- `APPLIANCE`
- `UNIX`
- `WINDOWSNT`

PSCLUS
Controls whether this queue manager participates in publish subscribe activity across any clusters in which it is a member. No clustered topic objects can exist in any cluster when modifying from `ENABLED` to `DISABLED`.

PSMODE
Controls whether the publish/subscribe engine and the queued publish/subscribe interface are running, and therefore controls whether applications can publish or subscribe by using the application programming interface and the queues that are monitored by the queued publish/subscribe interface.

PSNPMSG
If the queued publish/subscribe interface cannot process a non-persistent input message it might attempt to write the input message to the dead-letter queue (depending on the report options of the input message). If the attempt to write the input message to the dead-letter queue fails, and the MQRO_DISCARD_MSG report option was specified on the input message or PSNPMSG=DISCARD, the broker discards the input message. If PSNPMSG=KEEP is specified, the interface only discards the input message if the MQRO_DISCARD_MSG report option was set in the input message.

PSNPRES
If the queued publish/subscribe interface attempts to generate a response message in response to a non-persistent input message, and the response message cannot be delivered to the reply-to queue, this attribute indicates whether the interface tries to write the undeliverable message to the dead-letter queue or whether to discard the message.

PSRTYCNT
When the queued publish/subscribe interface fails to process a command message under sync point (for example a publish message that cannot be delivered to a subscriber because the subscriber queue is full and it is not possible to put the publication on the dead letter queue), the unit of work is backed out and the command tries this number of times again before the broker attempts to process the command message according to its report options instead.

PSSYNCPT
If this attribute is set to `IFPER`, when the queued publish/subscribe interface reads a publish or delete publication messages from a stream queue during normal operation then it specifies MQGMO_SYNCPOINT_IF_PERSISTENT. This value makes the queued pubsub daemon receive non-persistent messages outside sync point. If the daemon receives a publication outside sync point, the daemon forwards that publication to subscribers known to it outside sync point.

QMID
The internally generated unique name of the queue manager.

QMNAME
The name of the local queue manager. See Rules for naming IBM MQ objects.

z/OS **QSGNAME**
The name of the queue-sharing group to which the queue manager belongs, or blank if the queue manager is not a member of a queue-sharing group. You can use queue-sharing groups only on z/OS.

z/OS **RCVTIME**
The approximate length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to an inactive state. The value of this parameter is the numeric value qualified by **RCVTTYPE**.

This parameter is valid only on z/OS.

z/OS **RCVTMIN**
The minimum length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to an inactive state.

This parameter is valid only on z/OS.

z/OS **RCVTTYPE**
The qualifier to apply to the value in **RCVTIME**.

This parameter is valid only on z/OS.

REMOTEEV
Whether remote error events are generated.

REPOS
The name of a cluster for which this queue manager is to provide a repository manager service.

REPOSNL
The name of a list of clusters for which this queue manager is to provide a repository manager service.

REVDNS
Whether reverse lookup of the host name from a Domain Name Server (DNS) is done for the IP address from which a channel has connected.

ROUTEREC
Whether trace-route information is to be recorded if requested in the message.

Multi **SCHINIT**
Whether the channel initiator is to be started automatically when the queue manager starts.

z/OS This parameter is not valid on z/OS.

Multi **SCMDSERV**
Whether the command server is to be started automatically when the queue manager starts.

z/OS This parameter is not valid on z/OS.

z/OS **SCYCASE**
Whether the security profiles are uppercase or mixed case.

This parameter is valid only on z/OS.

If this parameter has been altered but the **REFRESH SECURITY** command has not yet been issued, the queue manager might not be using the case of profiles you expect. Use **DISPLAY SECURITY** to verify which case of profiles is actually in use.

SPLCAP

Indicates if Advanced Message Security (AMS) capabilities are available to the queue manager. If the AMS component is installed for the version of IBM MQ that the queue manager is running under, the attribute has a value `ENABLED (MQCAP_SUPPORTED)`. If the AMS component is not installed, the value is `DISABLED (MQCAP_NOT_SUPPORTED)`.

z/OS **SQQMNAME**
When a queue manager makes an MQOPEN call for a shared queue and the queue manager that is specified in the **ObjectQmgrName** parameter of the MQOPEN call is in the same queue-sharing group as the processing queue manager, the **SQQMNAME** attribute specifies whether the **ObjectQmgrName** is used or whether the processing queue manager opens the shared queue directly.

This parameter is valid only on z/OS.

SSLCRLNL
Indicates the namelist of AUTHINFO objects being used for the queue manager for certificate revocation checking.

Only authentication information objects with types of `LDAPCRL` or `OCSP` are allowed in the namelist referred to by **SSLCRLNL**. Any other type results in an error message when the list is processed and is subsequently ignored.

ULW **SSLCRYP**
Indicates the name of the parameter string being used to configure the cryptographic hardware present on the system. The PKCS #11 password appears as `xxxxxx`. This is valid only on UNIX, Linux, and Windows.

SSLEV
Whether TLS events are generated.

SSLFIPS
Whether only FIPS-certified algorithms are to be used if cryptography is processed in IBM MQ rather than in the cryptographic hardware itself.

SSLKEYR
Indicates the name of the Secure Sockets Layer key repository.

SSLRKEYC
Indicates the number of bytes to be sent and received within an TLS conversation before the secret key is renegotiated.

z/OS **SSLTASKS**
On z/OS only, indicates the number of server subtasks to use for processing TLS calls.

STATACLS
Whether statistics data is to be collected for auto-defined cluster-sender channels, and, if so, the rate of data collection.

STATCHL
It determines whether statistics data is to be collected for channels, and, if so, the rate of data collection.

Multi **STATINT**
The interval at which statistics monitoring data is written to the monitoring queue. This parameter is valid only on Multiplatforms.

Multi **STATMQI**
Whether statistics monitoring data is to be collected for the queue manager. This parameter is valid only on Multiplatforms.

Multi **STATQ**
Whether statistics data is to be collected for queues. This parameter is valid only on Multiplatforms.

STRSTPEV
Whether start and stop events are generated.

SUITEB
Whether Suite B compliant cryptography is used. For more information about Suite B configuration and its effect on TLS channels, see NSA Suite B Cryptography in IBM MQ.

SYNCPT
Whether sync point support is available with the queue manager.

z/OS **TCPCHL**
The maximum number of channels that can be current, or clients that can be connected, that use the TCP/IP transmission protocol. If zero, the TCP/IP transmission protocol is not used.

This parameter is valid only on z/OS.

z/OS **TCPKEEP**
Whether the KEEPALIVE facility is to be used to check that the other end of the connection is still available. If it is unavailable, the channel is closed.

This parameter is valid only on z/OS.

z/OS **TCPNAME**
The name of the preferred TCP/IP stack to be used in a CINET multiple stack environment. In INET single stack environments the channel initiator uses the only available TCP/IP stack.

This parameter is valid only on z/OS.

z/OS **TCPSTACK**
Whether the channel initiator uses only the TCP/IP stack specified in TCPNAME, or can optionally bind to any of the TCP/IP stacks defined in a CINET multiple stack environment.

This parameter is valid only on z/OS.

z/OS **TRAXSTR**
Whether channel initiator trace starts automatically.

This parameter is valid only on z/OS.

z/OS **TRAXTBL**
The size, in megabytes, of the trace data space of the channel initiator.

This parameter is valid only on z/OS.

TREELIFE
The lifetime of non-administrative topics.

TRIGINT
The trigger interval.

VERSION
The version of the IBM MQ installation that the queue manager is associated with. The version has the format `VVRRMMFF`:

`VV`: Version

`RR`: Release

`MM`: Maintenance level

`FF`: Fix level

XRCAP
Whether MQ Telemetry capability is supported by the queue manager.

For more information about these parameters, see "ALTER QMGR" on page 458.

Related information:

Working with queue managers

DISPLAY QMSTATUS on Multiplatforms: Multi

Use the MQSC command DISPLAY QMSTATUS to display status information associated with this queue manager.

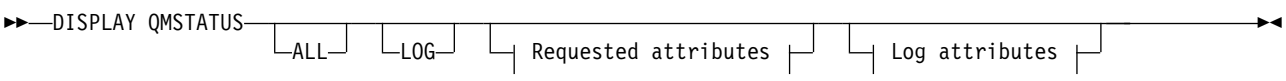
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Parameter descriptions for DISPLAY QMSTATUS" on page 875
- "Requested parameters" on page 875

Synonym: DIS QMSTATUS

DISPLAY QMSTATUS



Requested attributes:

Log attributes:

Parameter descriptions for DISPLAY QMSTATUS

ALL Specify this parameter to display all the parameters. If this parameter is specified, any parameters that are requested specifically have no effect; all parameters are still displayed.

This parameter is the default if you do not request any specific parameters.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

V 9.0.2 **ARCHLOG** *(x)*

Name of the oldest log extent for which the queue manager is waiting for archive notification.

This parameter is:

- Available only on queue managers using archive log management.
- Blank, if the queue manager is not using archive log management, or if the queue manager has no extents waiting for notification.

V 9.0.2 **ARCHSZ** *(x)*

The amount of space occupied, in megabytes, by log extents no longer required for restart or media recovery, but waiting to be archived.

Note that this value impacts the total space used by the queue manager for log extents.

This parameter is available only on queue managers using archive log management. If the queue manager is not using archive log management, this parameter is zero.

This attribute is not valid on IBM i.

CHINIT
The status of the channel initiator reading SYSTEM.CHANNEL.INITQ. It is one of the following:

STOPPED
The channel initiator is not running.

STARTING
The channel initiator is in the process of initializing and is not yet operational.

RUNNING
The channel initiator is fully initialized and is running.

STOPPING
The channel initiator is stopping.

CMDSERV
The status of the command server. It is one of the following:

STOPPED
The command server is not running.

STARTING
The command server is in the process of initializing and is not yet operational.

RUNNING
The command server is fully initialized and is running.

STOPPING
The command server is stopping.

CONNS
The current number of connections to the queue manager.

CURRLOG
The name of the log extent being written to at the time that the DISPLAY QMSTATUS command is processed. If the queue manager is using circular logging, and this parameter is explicitly requested, a blank string is displayed.

INSTDESC
Description of the installation associated with the queue manager. This parameter is not valid on IBM i.

INSTNAME
Name of the installation associated with the queue manager. This parameter is not valid on IBM i.

INSTPATH
Path of the installation associated with the queue manager. This parameter is not valid on IBM i.

LDAPCONN
The status of the connection to the LDAP server. It is one of the following:

CONNECTED
The queue manager currently has a connection to the LDAP server.

ERROR
The queue manager attempted to make a connection to the LDAP server and failed.

INACTIVE
The queue manager is not configured to use an LDAP server or has not yet made a connection to the LDAP server.

V 9.0.2 **LOG**

Specify this parameter to display all the LOG parameters. If this parameter is specified, any LOG parameters that are requested specifically have no effect; all parameters are still displayed.

V 9.0.2 **LOGINUSE** *(x)*

The percentage of the primary log space in use for restart recovery at this point in time.

A value of 100 or greater indicates the queue manager might have allocated, and be using, secondary log files, probably due to long-lived transactions at this point in time.

This attribute is not valid on IBM i.

V 9.0.2 **LOGPATH** *(x)*

Identifies the directory where log files are created by the queue manager.

V 9.0.2 **LOGUTIL** *(x)*

A percentage estimate of how well the queue manager workload is contained within the primary log space.

If the value is consistently above 100 you might want to investigate whether there are long-lived transactions, or if the number of primary files is not sufficient for the workload.

If the utilization continues to rise, eventually requests for most further operations requiring log activity will be refused, together with an MQRC_RESOURCE_PROBLEM return code being returned to the application. Transactions might be backed out.

This attribute is not valid on IBM i.

MEDIALOG

The name of the oldest log extent required by the queue manager to perform media recovery. If the queue manager is using circular logging, and this parameter is explicitly requested, a blank string is displayed.

V 9.0.2 **MEDIASZ** *(x)*

Size of the log data required for media recovery in megabytes.

This value shows how much log that must be read for media recovery and directly impacts the time taken for this operation.

This is zero for a circular logging queue manager. The size is typically reduced by taking more frequent media images of objects.

This attribute is not valid on IBM i.

QMNAME

The name of the queue manager. This parameter is always returned.

RECLOG

The name of the oldest log extent required by the queue manager to perform restart recovery. If the queue manager is using circular logging, and this parameter is explicitly requested, a blank string is displayed.

V 9.0.2 **RECSZ** *(x)*

Size of the log data required for restart recovery in megabytes.

This value shows how much log that must be read for restart recovery and directly impacts the time taken for this operation.

This attribute is not valid on IBM i.

V 9.0.2 **REUSESZ** *(x)*

This attribute is valid only on automatic or archive log management queue managers.

The amount of space occupied, in megabytes, by log extents available to be reused.

This value impacts the total space used by the queue manager for log extents.

The size is automatically managed by the queue manager, but if necessary you can request reductions using the **RESET QMGR TYPE(REDUCELOG)** command.

This attribute is not valid on IBM i.

STANDBY
Whether a standby instance is permitted. It is one of the following:

NOPERMIT
Standby instances are not permitted.

PERMIT
Standby instances are permitted.

STATUS
The status of the queue manager. It is one of the following:

STARTING
The queue manager is in the process of initializing.

RUNNING
The queue manager is fully initialized and is running.

QUIESCING
The queue manager is quiescing.

STARTDA
The date on which the queue manager was started (in the form yyyy-mm-dd).

STARTTI
The time at which the queue manager was started (in the form hh.mm.ss).

DISPLAY QSTATUS:

Use the MQSC command DISPLAY QSTATUS to display the status of one or more queues.

Using MQSC commands

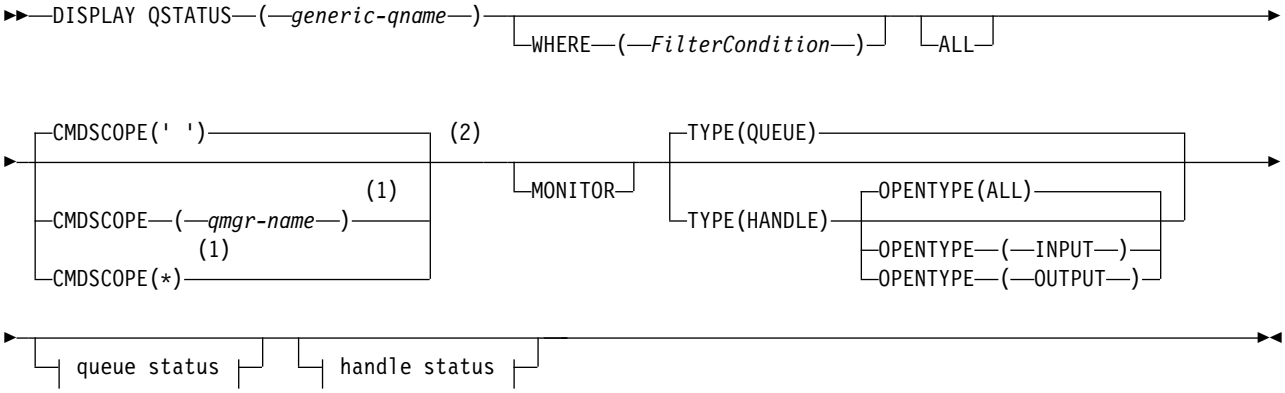
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

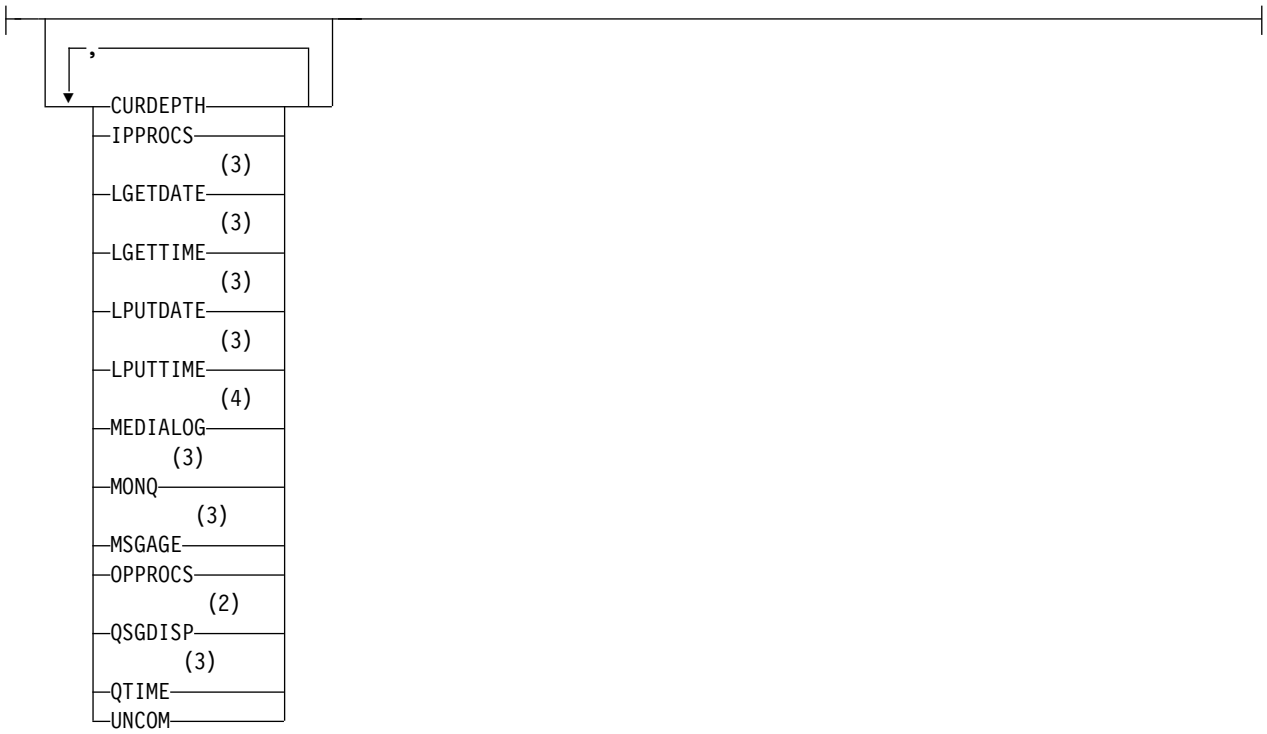
- Syntax diagram
- "Usage notes for DISPLAY QSTATUS" on page 880
- "Parameter descriptions for DISPLAY QSTATUS" on page 881
- "Queue status" on page 883
- "Handle status" on page 885

Synonym: DIS QS

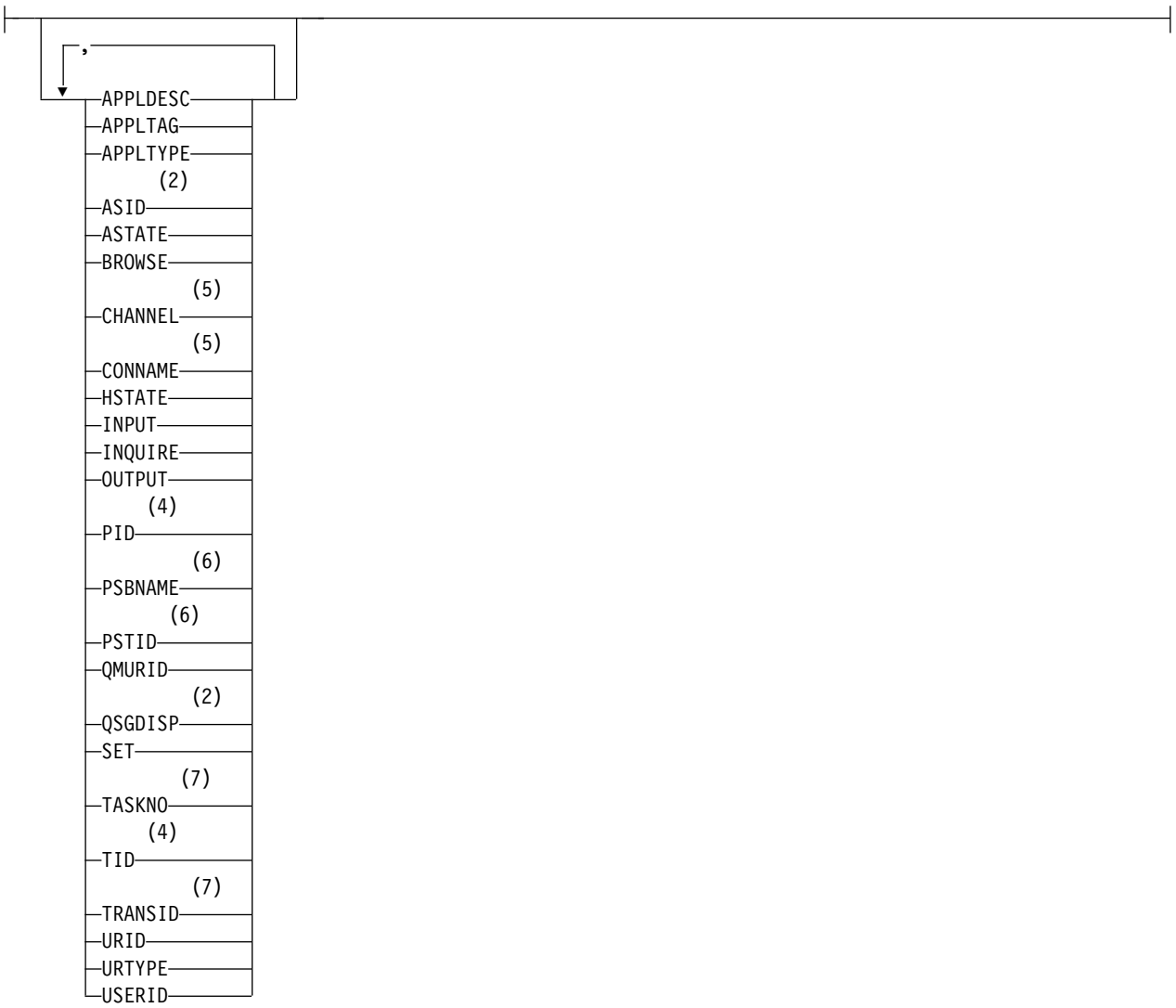
DISPLAY QSTATUS



Queue status:



Handle status:



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid on z/OS only.

3 Also displayed by selection of the MONITOR parameter.

4 Not valid on z/OS.

5 Channel initiator only

6 IMS only

7 CICS only

Usage notes for DISPLAY QSTATUS

The state of asynchronous consumers, ASTATE, reflects that of the server-connection proxy on behalf of the client application; it does not reflect the client application state.

Parameter descriptions for DISPLAY QSTATUS

You must specify the name of the queue for which you want to display status information. This name can either be a specific queue name or a generic queue name. By using a generic queue name you can display either:

- Status information for all queues, or
- Status information for one or more queues that match the specified name and other selection criteria

You must also specify whether you want status information about:

- Queues
- Handles that are accessing the queues

Note: You cannot use the DISPLAY QSTATUS command to display the status of an alias queue or remote queue. If you specify the name of one of these types of queue, no data is returned. You can, however, specify the name of the local queue or transmission queue to which the alias queue or remote queue resolves.

(*generic-qname*)
The name of the queue for which status information is to be displayed. A trailing asterisk (*) matches all queues with the specified stem followed by zero or more characters. An asterisk (*) on its own matches all queues.

WHERE
Specify a filter condition to display status information for queues that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

`filter-keyword`
Almost any parameter that can be used to display attributes for this DISPLAY command. However, you cannot use the CMDSCOPE, MONITOR, OPENTYPE, QSGDISP, QTIME, TYPE, or URID parameters as filter keywords.

`operator`
The operator is used to determine whether a queue satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use this filter to display objects whose attributes contain the specified item.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use this filter to display objects whose attributes do not contain the specified item.

`filter-value`
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this value can be:
- An explicit value, that is a valid value for the attribute being tested.

You can use operators LT, GT, EQ, NE, LE, or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value NO on the UNCOM parameter), you can only use EQ or NE.

- A generic value. This value is a character string (such as the character string in the APPLTAG parameter) with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.
- An item in a list of values. The operator must be CT or EX. If it is a character value, it can be explicit or generic. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed. If ABC* is specified, all items where one of the attribute values begins with ABC are listed.

ALL Display all the status information for each specified queue.

This value is the default if you do not specify a generic name, and do not request any specific parameters.

z/OS On z/OS, this value is also the default if you specify a filter condition using the WHERE parameter, but on other platforms only the requested attributes are displayed.

z/OS **CMDSCOPE**

This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group. It is valid on z/OS only.

' ' The command runs on the queue manager on which it was entered. This value is the default.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this value is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

MONITOR

Specify this value to return the set of online monitoring parameters. These are LGETDATE, LGETTIME, LPUTDATE, LPUTTIME, MONQ, MSGAGE, and QTIME. If you specify this parameter, any of the monitoring parameters that you request specifically have no effect; all monitoring parameters are still displayed.

OPENTYPE

Restricts the queues selected to queues which have handles with the specified type of access:

ALL Selects queues that are open with any type of access. This value is the default if the OPENTYPE parameter is not specified.

INPUT
Selects queues that are open for input only. This option does not select queues that are open for browse.

OUTPUT
Selects queues that are open only for output.

The OPENTYPE parameter is valid only if TYPE(HANDLE) is also specified.

You cannot use OPENTYPE as a filter keyword.

TYPE Specifies the type of status information required:

QUEUE
Status information relating to queues is displayed. This value is the default if the TYPE parameter is not specified.

HANDLE
Status information relating to the handles that are accessing the queues is displayed.

You cannot use TYPE as a filter keyword.

Queue status

For queue status, the following information is always returned for each queue that satisfies the selection criteria, except where indicated:

- Queue name
- Type of information returned (TYPE parameter)
- Current queue depth (CURDEPTH parameter) z/OS on platforms other than z/OS
- z/OS On z/OS only, the queue-sharing group disposition (QSGDISP parameter)

The following parameters can be specified for TYPE(QUEUE) to request additional information for each queue. If a parameter is specified that is not relevant for the queue, operating environment, or type of status information requested, that parameter is ignored.

CURDEPTH
The current depth of the queue, that is, the number of messages on the queue, including both committed messages and uncommitted messages.

IPPROCS
The number of handles that are currently open for input for the queue (either input-shared or input-exclusive). This number does not include handles that are open for browse.

For shared queues, the number returned applies only to the queue manager generating the reply. The number is not the total for all the queue managers in the queue-sharing group.

LGETDATE
The date on which the last message was retrieved from the queue since the queue manager started. A message being browsed does not count as a message being retrieved. When no get date is available, perhaps because no message has been retrieved from the queue since the queue manager was started, the value is shown as a blank.

z/OS For queues with QSGDISP(SHARED), the value shown is for measurements collected on this queue manager only.
This parameter is also displayed when you specify the MONITOR parameter.
A value is only displayed for this parameter if MONQ is set to a value other than OFF for this queue.

LGETTIME
The time at which the last message was retrieved from the queue since the queue manager started. A message being browsed does not count as a message being retrieved. When no get time is available, perhaps because no message has been retrieved from the queue since the queue manager was started, the value is shown as a blank.

z/OS For queues with QSGDISP(SHARED), the value shown is for measurements collected on this queue manager only.

This parameter is also displayed when you specify the MONITOR parameter.
A value is only displayed for this parameter if MONQ is set to a value other than OFF for this queue.

LPUTDATE
The date on which the last message was put to the queue since the queue manager started. When no put date is available, perhaps because no message has been put to the queue since the queue manager was started, the value is shown as a blank.

z/OS For queues with QSGDISP(SHARED), the value shown is for measurements collected on this queue manager only.
This parameter is also displayed when you specify the MONITOR parameter.
A value is only displayed for this parameter if MONQ is set to a value other than OFF for this queue.

LPUTTIME
The time at which the last message was put to the queue since the queue manager started. When no put time is available, perhaps because no message has been put to the queue since the queue manager was started, the value is shown as a blank.

z/OS For queues with QSGDISP(SHARED), the value shown is for measurements collected on this queue manager only.
This parameter is also displayed when you specify the MONITOR parameter.
A value is only displayed for this parameter if MONQ is set to a value other than OFF for this queue.

Note: Moving the system clock backwards should be avoided in case the LPUTTIME is being used to monitor the messages. The LPUTTIME of a queue is only updated when a message that arrives on the queue has a PutTime greater than the existing value of LPUTTIME. Because the PutTime of the message is less than the existing LPUTTIME of the queue in this case, the time is left unchanged.

Multi **MEDIALOG**
The log extent or journal receiver needed for media recovery of the queue. On queue managers on which circular logging is in place, MEDIALOG is returned as a null string.

This parameter is valid only on Multiplatforms.

MONQ
Current level of monitoring data collection for the queue.

This parameter is also displayed when you specify the MONITOR parameter.

MSGAGE
Age, in seconds, of the oldest message on the queue. The maximum displayable value is 999999999; if the age exceeds this value, 999999999 is displayed.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONQ is set to a value other than OFF for this queue.

OPPROCS
This is the number of handles that are currently open for output for the queue.

For shared queues, the number returned applies only to the queue manager generating the reply. The number is not the total for all the queue managers in the queue-sharing group.

z/OS **QSGDISP**
Indicates the disposition of the queue. The value displayed is one of the following:

QMGR
The object was defined with QSGDISP(QMGR).

COPY The object was defined with QSGDISP(COPY).

SHARED
The object was defined with QSGDISP(SHARED).

This parameter is valid on z/OS only.

For shared queues, if the CF structure used by the queue is unavailable or has failed, the status information might be unreliable.

You cannot use QSGDISP as a filter keyword.

QTIME
Interval, in microseconds, between messages being put on the queue and then being destructively read. The maximum displayable value is 999999999; if the interval exceeds this value, 999999999 is displayed.

The interval is measured from the time that the message is placed on the queue until it is destructively retrieved by an application and, therefore, includes any interval caused by a delay in committing by the putting application.

Two values are displayed and these are recalculated only when messages are processed:

- A value based on the last few messages processed
- A value based on a larger sample of the recently processed messages

These values depend on the configuration and behavior of your system, as well as the levels of activity within it, and serve as an indicator that your system is performing normally. A significant variation in these values might indicate a problem with your system. For queues with QSGDISP(SHARED), the values shown are for measurements collected on this queue manager only.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONQ is set to a value other than OFF for this queue.

UNCOM
Indicates whether there are any uncommitted changes (puts and gets) pending for the queue. The value displayed is one of the following:

YES

On z/OS, there are one or more uncommitted changes pending.

NO There are no uncommitted changes pending.

n Multi On Multiplatforms, an integer value indicating how many uncommitted changes are pending.

For shared queues, the value returned applies only to the queue manager generating the reply. The value does not apply to all the queue managers in the queue-sharing group.

Handle status

For handle status, the following information is always returned for each queue that satisfies the selection criteria, except where indicated:

- Queue name
- Type of information returned (TYPE parameter)
- Multi User identifier (USERID parameter) - not returned for APPLTYPE(SYSTEM)

- Multi Process ID (PID parameter)
- Multi Thread ID (TID parameter)
- Multi Application tag (APPLTAG parameter)
- Application type (APPLTYPE parameter)
- Multi Whether the handle provides input access (INPUT parameter)
- Multi Whether the handle provides output access (OUTPUT parameter)
- Multi Whether the handle provides browse access (BROWSE parameter)
- Multi Whether the handle provides inquire access (INQUIRE parameter)
- Multi Whether the handle provides set access (SET parameter)

The following parameters can be specified for TYPE(HANDLE) to request additional information for each queue. If a parameter that is not relevant is specified for the queue, operating environment, or type of status information requested, that parameter is ignored.

APPLDESC
A string containing a description of the application connected to the queue manager, where it is known. If the application is not recognized by the queue manager the description returned is blank.

APPLTAG
A string containing the tag of the application connected to the queue manager. It is one of the following:

- z/OS z/OS batch job name
- z/OS TSO USERID
- CICS APPLID
- IMS region name
- Channel initiator job name
- IBM i IBM i job name
- UNIX UNIX process
- Windows Windows process

  Note: The returned value consists of the full program path and executable file name. If it is more than 28 characters long, only the first 28 characters are shown.
- Internal queue manager process name

Application name represents the name of the process or job that has connected to the queue manager. In the instance that this process or job is connected via a channel, the application name represents the remote process or job rather than the local channel process or job name.

APPLTYPE
A string indicating the type of the application that is connected to the queue manager. It is one of the following:

BATCH
Application using a batch connection

RRSBATCH
RRS-coordinated application using a batch connection

CICS CICS transaction

IMS IMS transaction

CHINIT
Channel initiator

SYSTEM
Queue manager

SYSTEMEXT
Application performing an extension of function that is provided by the queue manager

USER A user application

z/OS **ASID**
A four-character address-space identifier of the application identified by APPLTAG. It distinguishes duplicate values of APPLTAG.

This parameter is returned only when the queue manager owning the queue is running on z/OS, and the APPLTYPE parameter does not have the value SYSTEM.

ASTATE
The state of the asynchronous consumer on this queue.

Possible values are:

ACTIVE
An MQCB call has set up a function to call back to process messages asynchronously and the connection handle has been started so that asynchronous message consumption can proceed.

INACTIVE
An MQCB call has set up a function to call back to process messages asynchronously but the connection handle has not yet been started, or has been stopped or suspended, so that asynchronous message consumption cannot currently proceed.

SUSPENDED
The asynchronous consumption call-back has been suspended so that asynchronous message consumption cannot currently proceed on this queue. This can be either because an MQCB call with Operation MQOP_SUSPEND has been issued against this object handle by the application, or because it has been suspended by the system. If it has been suspended by the system, as part of the process of suspending asynchronous message consumption the call-back function is initiated with the reason code that describes the problem resulting in suspension. This code is reported in the Reason field in the MQCBC structure that is passed to the call-back function.

For asynchronous message consumption to proceed, the application must issue an MQCB call with the Operation parameter set to MQOP_RESUME.

SUSPTEMP
The asynchronous consumption call-back has been temporarily suspended by the system so that asynchronous message consumption cannot currently proceed on this queue. As part of the process of suspending asynchronous message consumption, the call-back function is called with the reason code that describes the problem resulting in suspension. This code is reported in the Reason field in the MQCBC structure passed to the call-back function.

The call-back function is initiated again when asynchronous message consumption is resumed by the system, when the temporary condition has been resolved.

NONE
An MQCB call has not been issued against this handle, so no asynchronous message consumption is configured on this handle.

BROWSE
Indicates whether the handle is providing browse access to the queue. The value is one of the following:

YES The handle is providing browse access.

NO The handle is not providing browse access.

CHANNEL
The name of the channel that owns the handle. If there is no channel associated with the handle, this parameter is blank.

This parameter is returned only when the handle belongs to the channel initiator.

CONNAME
The connection name associated with the channel that owns the handle. If there is no channel associated with the handle, this parameter is blank.

This parameter is returned only when the handle belongs to the channel initiator.

HSTATE
Whether an API call is in progress.

Possible values are:

ACTIVE
An API call from a connection is currently in progress for this object. For a queue, this condition can arise when an MQGET WAIT call is in progress.

If there is an MQGET SIGNAL outstanding, then this value does not mean, by itself, that the handle is active.

INACTIVE
No API call from a connection is currently in progress for this object. For a queue, this condition can arise when no MQGET WAIT call is in progress.

INPUT
Indicates whether the handle is providing input access to the queue. The value is one of the following:

SHARED
The handle is providing shared-input access.

EXCL The handle is providing exclusive-input access.

NO The handle is not providing input access.

INQUIRE
Indicates whether the handle currently provides inquire access to the queue. The value is one of the following:

YES The handle provides inquire access.

NO The handle does not provide inquire access.

OUTPUT
Indicates whether the handle is providing output access to the queue. The value is one of the following:

YES The handle is providing output access.

NO The handle is not providing output access.

PID Number specifying the process identifier of the application that has opened the specified queue.

z/OS This parameter is not valid on z/OS.

z/OS **PSBNAME**
The eight characters long name of the program specification block (PSB) associated with the running IMS transaction. You can use the PSBNAME and PSTID to purge the transaction using IMS commands. It is valid on z/OS only.

This parameter is returned only when the APPLTYPE parameter has the value IMS.

z/OS **PSTID**
The four character IMS program specification table (PST) region identifier for the connected IMS region. It is valid on z/OS only.

This parameter is returned only when the APPLTYPE parameter has the value IMS.

QMURID
The queue manager unit of recovery identifier. On z/OS, this value is an 8-byte log RBA, displayed as 16 hexadecimal characters. On platforms other than z/OS, this value is an 8-byte transaction identifier, displayed as `m.n` where `m` and `n` are the decimal representation of the first and last 4 bytes of the transaction identifier.

You can use QMURID as a filter keyword. On z/OS, you must specify the filter value as a hexadecimal string. On platforms other than z/OS, you must specify the filter value as a pair of decimal numbers separated by a period (.). You can only use the EQ, NE, GT, LT, GE, or LE filter operators.

z/OS **QSGDISP**
Indicates the disposition of the queue. It is valid on z/OS only. The value is one of the following:

QMGR
The object was defined with QSGDISP(QMGR).

COPY The object was defined with QSGDISP(COPY).

SHARED
The object was defined with QSGDISP(SHARED).

You cannot use QSGDISP as a filter keyword.

SET Indicates whether the handle is providing set access to the queue. The value is one of the following:

YES The handle is providing set access.

NO The handle is not providing set access.

z/OS **TASKNO**
A seven-digit CICS task number. This number can be used in the CICS command "CEMT SET TASK(taskno) PURGE" to end the CICS task. This parameter is valid on z/OS only.

This parameter is returned only when the APPLTYPE parameter has the value CICS.

TID Number specifying the thread identifier within the application process that has opened the specified queue.

z/OS This parameter is not valid on z/OS.

An asterisk indicates that this queue was opened using a shared connection.

For further information about shared connections see Shared (thread independent) connections with MQCONNX.

z/OS **TRANSID**
A four-character CICS transaction identifier. This parameter is valid on z/OS only.

This parameter is returned only when the APPLTYPE parameter has the value CICS.

URID The external unit of recovery identifier associated with the connection. It is the recovery identifier known in the external syncpoint coordinator. Its format is determined by the value of URTYPE.

You cannot use URID as a filter keyword.

URTYPE

The type of unit of recovery as seen by the queue manager. It is one of the following:

- CICS (valid only on z/OS)
- XA
- RRS (valid only on z/OS)
- IMS (valid only on z/OS)
- QMGR

URTYPE identifies the EXTURID type and not the type of the transaction coordinator. When URTYPE is QMGR, the associated identifier is in QMURID (and not URID).

USERID

The user identifier associated with the handle.

This parameter is not returned when APPLTYPE has the value SYSTEM.

DISPLAY QUEUE:

Use the MQSC command **DISPLAY QUEUE** to display the attributes of one or more queues of any type.

Using MQSC commands

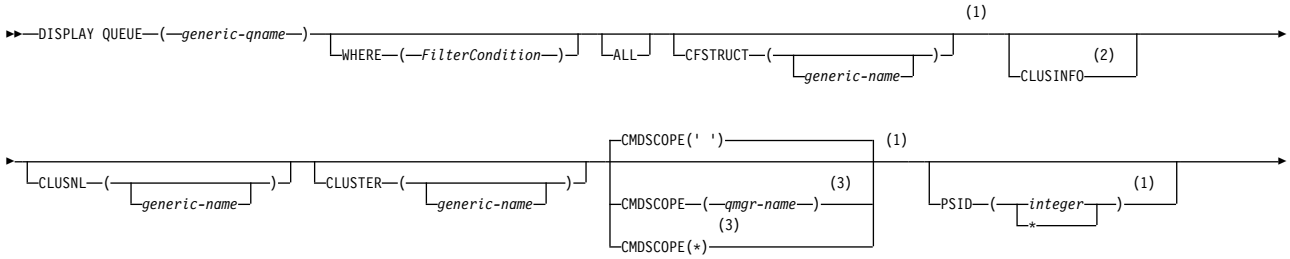
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

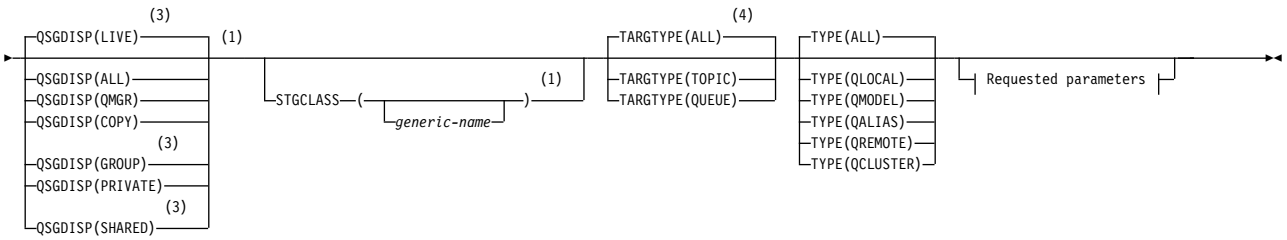
z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes" on page 893
- "Parameter descriptions for DISPLAY QUEUE" on page 893
- "Requested parameters" on page 897

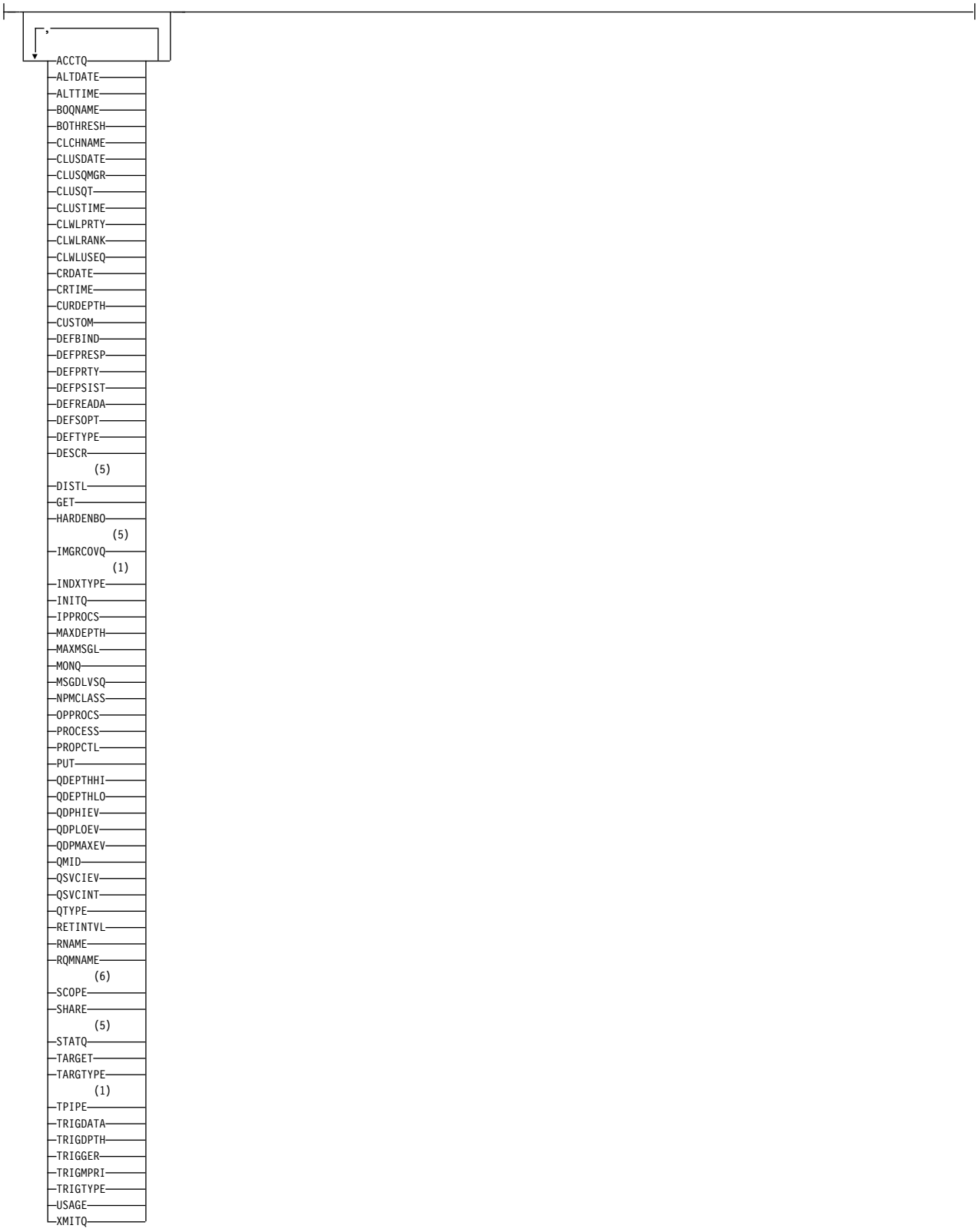
Synonym: **DIS Q**

DISPLAY QUEUE





Requested parameters:



Notes:

1 Valid only on z/OS.

2 On z/OS, you cannot issue this from CSQINP2.

3 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

4 Valid only on an alias queue.

5 Not valid on z/OS.

6 Not valid on z/OS or IBM i.

Usage notes

1. You can use the following commands (or their synonyms) as an alternative way to display these attributes.
   - **DISPLAY QALIAS**
   - **DISPLAY QCLUSTER**
   - **DISPLAY QLOCAL**
   - **DISPLAY QMODEL**
   - **DISPLAY QREMOTE**

   These commands produce the same output as the DISPLAY QUEUE TYPE(*queue-type*) command. If you enter the commands this way, do not use the **TYPE** parameter.
2. z/OS On z/OS, the channel initiator must be running before you can display information about cluster queues (using TYPE(QCLUSTER) or the CLUSINFO parameter).
3. The command might not show every clustered queue in the cluster when issued on a partial repository, because the partial repository only knows about a queue once it has tried to use it.

Parameter descriptions for DISPLAY QUEUE

You must specify the name of the queue definition you want to display. This can be a specific queue name or a generic queue name. By using a generic queue name, you can display either:

- All queue definitions
- One or more queues that match the specified name

queue-name
The local name of the queue definition to be displayed (see Rules for naming IBM MQ objects). A trailing asterisk * matches all queues with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all queues.

WHERE
Specify a filter condition to display only those queues that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this **DISPLAY** command. However, you cannot use the z/OS CMDSCOPE, QDPHIEV, QDPLOEV, QDPMAXEV, z/OS QSGDISP, or QSVCIEV parameters as filter keywords. You cannot use z/OS CFSTRUCT, CLUSTER, z/OS PSID, z/OS STGCLASS,, or CLUSNLif these are also used to select queues. Queues of a type for which the filter keyword is not a valid attribute are not displayed.

operator
This is used to determine whether a queue satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

`filter-value`
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value QALIAS on the CLUSQT parameter), you can only use EQ or NE. For the parameters HARDENBO, SHARE, and TRIGGER, use either EQ YES or EQ NO.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

ALL Specify this to display all the attributes. If this parameter is specified, any attributes that are also requested specifically have no effect; all attributes are still displayed.

On all platforms, this is the default if you do not specify a generic name and do not request any specific attributes.

z/OS On z/OS this is also the default if you specify a filter condition using the `WHERE` parameter, but on other platforms only requested attributes are displayed.

z/OS **CFSTRUCT (*generic-name)***
This parameter is optional and limits the information displayed to those queues where the value of the coupling facility structure is specified in brackets.

The value can be a generic name. If you do not enter a value for this parameter, **`CFSTRUCT`** is treated as a requested parameter.

CLUSINFO
This requests that, in addition to information about attributes of queues defined on this queue manager, information about these and other queues in the cluster that match the selection criteria is displayed. In this case, there might be multiple queues with the same name displayed. The cluster information is obtained from the repository on this queue manager.

z/OS Note that, on z/OS, you cannot issue `DISPLAY QUEUE CLUSINFO` commands from `CSQINP2`.

CLUSNL (*generic-name*)
This is optional, and limits the information displayed if entered with a value in brackets:

- For queues defined on the local queue manager, only those with the specified cluster list. The value can be a generic name. Only queue types for which **`CLUSNL`** is a valid parameter are restricted in this way; other queue types that meet the other selection criteria are displayed.
- For cluster queues, only those belonging to clusters in the specified cluster list if the value is not a generic name. If the value is a generic name, no restriction is applied to cluster queues.

If you do not enter a value to qualify this parameter, it is treated as a requested parameter, and cluster list information is returned about all the queues displayed.

Note: z/OS If the disposition requested is SHARED, CMDSCOPE must be blank or the local queue manager.

CLUSTER (*generic-name*)
This is optional, and limits the information displayed to queues with the specified cluster name if entered with a value in brackets. The value can be a generic name. Only queue types for which **CLUSTER** is a valid parameter are restricted in this way by this parameter; other queue types that meet the other selection criteria are displayed.

If you do not enter a value to qualify this parameter, it is treated as a requested parameter, and cluster name information is returned about all the queues displayed.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP or SHARED.

'' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use **CMDSCOPE** as a filter keyword.

z/OS **PSID (*integer*)**
The identifier of the page set where a queue resides. This is optional. Specifying a value limits the information displayed to queues that have an active association to the specified page set. The value consists of two numeric characters, in the range 00 - 99. An asterisk * on its own specifies all page set identifiers. If you do not enter a value, page set information is returned about all the queues displayed.

The page set identifier is displayed only if there is an active association of the queue to a page set, that is, after the queue has been the target of an MQPUT request. The association of a queue to a page set is not active when:

- The queue is just defined
- The STGCLASS attribute of the queue is altered, and there is no subsequent MQPUT request to the queue
- The queue manager is restarted and there are no messages on the queue

This parameter is valid only on z/OS.

z/OS **QSGDISP**
Specifies the disposition of the objects for which information is to be displayed. Values are:

LIVE This is the default value and displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY). If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, also display information for objects defined with QSGDISP(SHARED).

ALL Display information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with QSGDISP(GROUP) or QSGDISP(SHARED).

In a shared queue manager environment:

```
DISPLAY QUEUE(name) CMDSCOPE(*) QSGDISP(ALL)
```

The command lists objects matching name in the queue-sharing group, without duplicating those in the shared repository.

COPY Display information only for objects defined with QSGDISP(COPY).

GROUP
Display information only for objects defined with QSGDISP(GROUP). This is allowed only if there is a shared queue manager environment.

PRIVATE
Display information only for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

QMGR
Display information only for objects defined with QSGDISP(QMGR).

SHARED
Display information only for objects defined with QSGDISP(SHARED). This is allowed only in a shared queue manager environment.

Note: For cluster queues, this is always treated as a requested parameter. The value returned is the disposition of the real queue that the cluster queue represents.

If QSGDISP(LIVE) is specified or defaulted, or if QSGDISP(ALL) is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions) .

Note: In the QSGDISP(LIVE) case, this occurs only where a shared and a non-shared queue have the same name; such a situation should not occur in a well-managed system.

QSGDISP displays one of the following values:

QMGR
The object was defined with QSGDISP(QMGR).

GROUP
The object was defined with QSGDISP(GROUP).

COPY The object was defined with QSGDISP(COPY).

SHARED
The object was defined with QSGDISP(SHARED).

You cannot use **QSGDISP** as a filter keyword.

z/OS **STGCLASS (*generic-name)***
This is optional, and limits the information displayed to queues with the storage class specified if entered with a value in brackets. The value can be a generic name.

If you do not enter a value to qualify this parameter, it is treated as a requested parameter, and storage class information is returned about all the queues displayed.

This parameter is valid only on z/OS.

TARGTYPE (*target-type*)
This is optional and specifies the target type of the alias queue you want to be displayed.

TYPE (*queue-type*)

This is optional, and specifies the type of queues you want to be displayed. If you specify ALL, which is the default value, all queue types are displayed; this includes cluster queues if CLUSINFO is also specified.

As well as ALL, you can specify any of the queue types allowed for a **DEFINE** command: QALIAS, QLOCAL, QMODEL, QREMOTE, or their synonyms, as follows:

QALIAS
Alias queues

QLOCAL
Local queues

QMODEL
Model queues

QREMOTE
Remote queues

You can specify a queue type of QCLUSTER to display only cluster queue information. If QCLUSTER is specified, any selection criteria specified by the CFSTRUCT, STGCLASS, or PSID parameters are ignored. Note that you cannot issue **DISPLAY QUEUE TYPE(QCLUSTER)** commands from CSQINP2.

Multi On Multiplatforms, QTYPE (*type*) can be used as a synonym for this parameter.

The queue name and queue type z/OS (and, on z/OS, the queue disposition) are always displayed.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

Most parameters are relevant only for queues of a particular type or types. Parameters that are not relevant for a particular type of queue cause no output, nor is an error raised.

The following table shows the parameters that are relevant for each type of queue. There is a brief description of each parameter after the table, but for more information, see the **DEFINE** command for each queue type.

Table 114. Parameters that can be returned by the **DISPLAY QUEUE** *command.*

Cross-tabulation of queue parameters and queue types. If the parameter applies to the queue type, the cell contains a check mark.

	Local queue	Model queue	Alias queue	Remote queue	Cluster queue
ACCTQ	✓	✓	N/A	N/A	N/A
ALTDATE	✓	✓	✓	✓	✓
ALTTIME	✓	✓	✓	✓	✓
BOQNAME	✓	✓	N/A	N/A	N/A
BOTHRESH	✓	✓	N/A	N/A	N/A
CFSTRUCT	✓	✓	N/A	N/A	N/A

Table 114. Parameters that can be returned by the **DISPLAY QUEUE** *command (continued).*

Cross-tabulation of queue parameters and queue types. If the parameter applies to the queue type, the cell contains a check mark.

	Local queue	Model queue	Alias queue	Remote queue	Cluster queue
CLCHNAME	✓	✓	N/A	N/A	N/A
CLUSDATE	N/A	N/A	N/A	N/A	✓
CLUSNL	✓	N/A	✓	✓	N/A
CLUSQMGR	N/A	N/A	N/A	N/A	✓
CLUSQT	N/A	N/A	N/A	N/A	✓
CLUSTER	✓	N/A	✓	✓	✓
CLUSTIME	N/A	N/A	N/A	N/A	✓
CLWLPRTY	✓	N/A	✓	✓	✓
CLWLRANK	✓	N/A	✓	✓	✓
CLWLUSEQ	✓	N/A	N/A	N/A	N/A
CRDATE	✓	✓	N/A	N/A	N/A
CRTIME	✓	✓	N/A	N/A	N/A
CURDEPTH	✓	N/A	N/A	N/A	N/A
CUSTOM	✓	✓	✓	✓	✓
DEFBIND	✓	N/A	✓	✓	✓
DEFPRESP	✓	✓	✓	✓	✓
DEFPRTY	✓	✓	✓	✓	✓
DEFPSIST	✓	✓	✓	✓	✓
DEFREADA	✓	✓	✓	N/A	N/A
DEFSOPT	✓	✓	N/A	N/A	N/A
DEFTYPE	✓	✓	N/A	N/A	N/A
DESCR	✓	✓	✓	✓	✓
DISTL	✓	✓	N/A	N/A	N/A

Table 114. Parameters that can be returned by the **DISPLAY QUEUE** *command (continued).*

Cross-tabulation of queue parameters and queue types. If the parameter applies to the queue type, the cell contains a check mark.

	Local queue	Model queue	Alias queue	Remote queue	Cluster queue
GET	✓	✓	✓	N/A	N/A
HARDENBO	✓	✓	N/A	N/A	N/A
V 9.0.2 IMGRCOV	✓	✓	N/A	N/A	N/A
INDXTYPE	✓	✓	N/A	N/A	N/A
INITQ	✓	✓	N/A	N/A	N/A
IPPROCS	✓	N/A	N/A	N/A	N/A
MAXDEPTH	✓	✓	N/A	N/A	N/A
MAXMSGL	✓	✓	N/A	N/A	N/A
MONQ	✓	✓	N/A	N/A	N/A
MSGDLVSQ	✓	✓	N/A	N/A	N/A
NPMCLASS	✓	✓	N/A	N/A	N/A
OPPROCS	✓		N/A	N/A	N/A
PROCESS	✓	✓	N/A	N/A	N/A
PROPCTL	✓	✓	✓	N/A	N/A
PSID	✓	N/A	N/A	N/A	N/A
PUT	✓	✓	✓	✓	✓
QDEPTHHI	✓	✓	N/A	N/A	N/A
QDEPTHLO	✓	✓	N/A	N/A	N/A
QDPHIEV	✓	✓	N/A	N/A	N/A
QDPLOEV	✓	✓	N/A	N/A	N/A
QDPMAXEV	✓	✓	N/A	N/A	N/A
QMID	N/A	N/A	N/A	N/A	✓
QSGDISP	✓	✓	✓	✓	✓

Table 114. Parameters that can be returned by the **DISPLAY QUEUE** *command (continued).*

Cross-tabulation of queue parameters and queue types. If the parameter applies to the queue type, the cell contains a check mark.

	Local queue	Model queue	Alias queue	Remote queue	Cluster queue
QSVCIEV	✓	✓	N/A	N/A	N/A
QSVCINT	✓	✓	N/A	N/A	N/A
QTYPE	✓	✓	✓	✓	✓
RETINTVL	✓	✓	N/A	N/A	N/A
RNAME	N/A	N/A	N/A	✓	N/A
RQMNAME	N/A	N/A	N/A	✓	N/A
SCOPE	✓	N/A	✓	✓	N/A
SHARE	✓	✓	N/A	N/A	N/A
STATQ	✓	✓	N/A	N/A	N/A
STGCLASS	✓	✓	N/A	N/A	N/A
TARGET	N/A	N/A	✓	N/A	N/A
TARGTYPE	N/A	N/A	✓	N/A	N/A
TPIPE	✓	N/A	N/A	N/A	N/A
TRIGDATA	✓	✓	N/A	N/A	N/A
TRIGDPTH	✓	✓	N/A	N/A	N/A
TRIGGER	✓	✓	N/A	N/A	N/A
TRIGMPRI	✓	✓	N/A	N/A	N/A
TRIGTYPE	✓	✓	N/A	N/A	N/A
USAGE	✓	✓	N/A	N/A	N/A
XMITQ	N/A	N/A	N/A	✓	N/A

ACCTQ
Whether accounting (on z/OS, thread-level and queue-level accounting) data collection is to be enabled for the queue.

ALTDATE
The date on which the definition or information was last altered, in the form `yyyy-mm-dd`.

ALTTIME
The time at which the definition or information was last altered, in the form hh.mm.ss.

BOQNAME
Backout requeue name.

BOTHRESH
Backout threshold.

CLCHNAME
CLCHNAME is the generic name of the cluster-sender channels that use this queue as a transmission queue. The attribute specifies which cluster-sender channels send messages to a cluster-receiver channel from this cluster transmission queue.

CLUSDATE
The date on which the definition became available to the local queue manager, in the form yyyy-mm-dd.

CLUSNL
The namelist that defines the cluster that the queue is in.

CLUSQMGR
The name of the queue manager that hosts the queue.

CLUSQT
Cluster queue type. This can be:

QALIAS
The cluster queue represents an alias queue.

QLOCAL
The cluster queue represents a local queue.

QMGR
The cluster queue represents a queue manager alias.

QREMOTE
The cluster queue represents a remote queue.

CLUSTER
The name of the cluster that the queue is in.

CLUSTIME
The time at which the definition became available to the local queue manager, in the form hh.mm.ss.

CLWLPRTY
The priority of the queue for the purposes of cluster workload distribution.

CLWLRANK
The rank of the queue for the purposes of cluster workload distribution.

CLWLUSEQ
Whether puts are allowed to other queue definitions apart from local ones.

CRDATE
The date on which the queue was defined (in the form yyyy-mm-dd).

CRTIME
The time at which the queue was defined (in the form hh.mm.ss).

CURDEPTH
Current depth of queue.

On z/OS, CURDEPTH is returned as zero for queues defined with a disposition of GROUP. It is also returned as zero for queues defined with a disposition of SHARED if the CF structure that they use is unavailable or has failed.

Messages put on a queue count toward the current depth as they are put. Messages got from a queue do not count toward the current depth. This is true whether operations are done under syncpoint or not. Commit has no effect on current depth. Therefore:

- Messages put under syncpoint (but not yet committed) are included in the current depth.
- Messages got under syncpoint (but not yet committed) are not included in the current depth.

CUSTOM
This attribute is reserved for the configuration of new features before separate attributes have been introduced. It can contain the values of zero or more attributes as pairs of attribute name and value in the form `NAME(VALUE)`.

DEFBIND
Default message binding.

DEFPRESP
Default put response; defines the behavior that should be used by applications when the put response type in the MQPMO options has been set to MQPMO_RESPONSE_AS_Q_DEF.

DEFPRTY
Default priority of the messages put on the queue.

DEFPSIST
Whether the default persistence of messages put on this queue is set to NO or YES. NO means that messages are lost across a restart of the queue manager.

DEFREADA
This specifies the default read ahead behavior for non-persistent messages delivered to the client.

DEFSOPT
Default share option on a queue opened for input.

DEFTYPE
Queue definition type. This can be:

- PREDEFINED (Predefined)

  The queue was created with a DEFINE command, either by an operator or by a suitably authorized application sending a command message to the service queue.

- PERMDYN (Permanent dynamic)

  Either the queue was created by an application issuing `MQOPEN` with the name of a model queue specified in the object descriptor (MQOD), or (if this is a model queue) this determines the type of dynamic queue that can be created from it.

  On z/OS the queue was created with `QSGDISP(QMGR)`.

- TEMPDYN (Temporary dynamic)

  Either the queue was created by an application issuing `MQOPEN` with the name of a model queue specified in the object descriptor (MQOD), or (if this is a model queue) this determines the type of dynamic queue that can be created from it.

  On z/OS the queue was created with `QSGDISP(QMGR)`.

- SHAREDYN

  A permanent dynamic queue was created when an application issued an `MQOPEN` API call with the name of this model queue specified in the object descriptor (MQOD).

  On z/OS, in a queue-sharing group environment, the queue was created with `QSGDISP(SHARED)`.

DESCR
Descriptive comment.

Multi **DISTL**
Whether distribution lists are supported by the partner queue manager. Supported only on Multiplatforms.

GET Whether the queue is enabled for gets.

HARDENBO
Whether the back out count is hardened to ensure that the count of the number of times that a message has been backed out is accurate.

Note: This parameter affects only IBM MQ for z/OS. It can be set and displayed on other platforms but has no effect.

V 9.0.2 **IMGRCOVQ**
Whether a local or permanent dynamic queue object is recoverable from a media image if linear logging is being used.

Note: This parameter is not valid on IBM MQ for z/OS.

INDXTYPE
Index type (supported only on z/OS).

INITQ
Initiation queue name.

IPPROCS
Number of handles indicating that the queue is open for input.

On z/OS, IPPROCS is returned as zero for queues defined with a disposition of GROUP. With a disposition of SHARED, only the handles for the queue manager sending back the information are returned, not the information for the whole group.

MAXDEPTH
Maximum depth of queue.

MAXMSGL
Maximum message length.

MONQ
Online monitoring data collection.

MSGDLVSQ
Message delivery sequence.

NPMCLASS
Level of reliability assigned to non-persistent messages that are put to the queue.

OPPROCS
Number of handles indicating that the queue is open for output.

On z/OS, OPPROCS is returned as zero for queues defined with a disposition of GROUP. With a disposition of SHARED, only the handles for the queue manager sending back the information are returned, not the information for the whole group.

PROCESS
Process name.

PROPCTL
Property control attribute.

This parameter is applicable to Local, Alias and Model queues.

This parameter is optional.

Specifies how message properties are handled when messages are retrieved from queues using the MQGET call with the MQGMO_PROPERTIES_AS_Q_DEF option.

Permissible values are:

ALL To contain all the properties of the message, except those contained in the message descriptor (or extension), select All. The All value enables applications that cannot be changed to access all the message properties from MQRFH2 headers.

COMPAT
If the message contains a property with a prefix of **mcd.**, **jms.**, **usr.**, or **mqext.**, all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except those contained in the message descriptor (or extension), are discarded and are no longer accessible to the application.

This is the default value; it allows applications which expect JMS related properties to be in an MQRFH2 header in the message data to continue to work unmodified.

FORCE
Properties are always returned in the message data in an MQRFH2 header regardless of whether the application specifies a message handle.

A valid message handle supplied in the MsgHandle field of the MQGMO structure on the MQGET call is ignored. Properties of the message are not accessible via the message handle.

NONE
All properties of the message, except those in the message descriptor (or extension), are removed from the message before the message is delivered to the application.

PUT Whether the queue is enabled for puts.

QDEPTHHI
Queue Depth High event generation threshold.

QDEPTHLO
Queue Depth Low event generation threshold.

QDPHIEV
Whether Queue Depth High events are generated.

You cannot use QDPHIEV as a filter keyword.

QDPLOEV
Whether Queue Depth Low events are generated.

You cannot use QDPLOEV as a filter keyword.

QDPMAXEV
Whether Queue Full events are generated.

You cannot use QDPMAXEV as a filter keyword.

QMID
The internally generated unique name of the queue manager that hosts the queue.

QSVCIEV
Whether service interval events are generated.

You cannot use QSVCIEV as a filter keyword.

QSVCINT
Service interval event generation threshold.

QTYPE
Queue type.

The queue type is always displayed.

Multi On Multiplatforms, TYPE(*type*) can be used as a synonym for this parameter.

RETINTVL
Retention interval.

RNAME
Name of the local queue, as known by the remote queue manager.

RQMNAME
Remote queue manager name.

SCOPE
Scope of queue definition (not supported on z/OS).

SHARE
Whether the queue can be shared.

STATQ
Whether statistics data information is to be collected.

STGCLASS
Storage class.

TARGET
This parameter requests that the base object name of an aliased queue is displayed.

TARGTYPE
This parameter requests that the target (base) type of an aliased queue is displayed.

TPIPE The TPIPE names used for communication with OTMA using the IBM MQ - IMS bridge if the bridge is active. This parameter is supported only on z/OS.

z/OS For more information about TPIPEs, see Controlling the IMS bridge.

TRIGDATA
Trigger data.

TRIGDPTH
Trigger depth.

TRIGGER
Whether triggers are active.

TRIGMPRI
Threshold message priority for triggers.

TRIGTYPE
Trigger type.

USAGE
Whether the queue is a transmission queue.

XMITQ
Transmission queue name.

For more details of these parameters, see “DEFINE queues” on page 658.

Related information:
Displaying default object attributes
Working with model queues

DISPLAY SBSTATUS:

Use the MQSC command **DISPLAY SBSTATUS** to display the status of a subscription.

Using MQSC commands

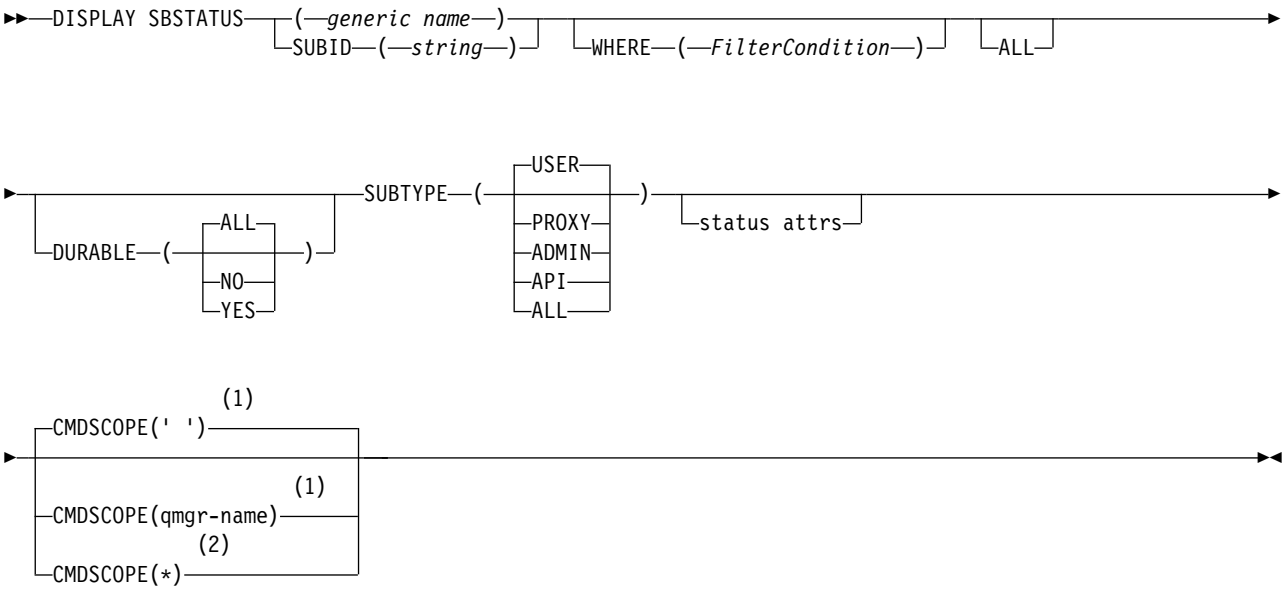
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for DISPLAY SBSTATUS" on page 907
- "Requested parameters" on page 909

Synonym: **DIS SBSTATUS**

DISPLAY SBSTATUS



Status attributes:

Notes:

1 Valid only on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Not valid on z/OS.

Parameter descriptions for DISPLAY SBSTATUS

You must specify the name of the subscription definition for which you want to display status information. This can be a specific subscription name or a generic subscription name. By using a generic subscription name, you can display either:

- All subscription definitions
- One or more subscriptions that match the specified name

(generic-name)
The local name of the subscription definition to be displayed. A trailing asterisk (*) matches all subscriptions with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all subscriptions.

WHERE
Specify a filter condition to display only those subscriptions that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this **DISPLAY** command.

z/OS However, you cannot use the **CMDSCOPE** parameter as a filter keyword.

Subscriptions of a type for which the filter keyword is not a valid attribute are not displayed.

operator
This is used to determine whether a subscription satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value USER on the **SUBTYPE** parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string you supply for the **SUBUSER** parameter) with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

ALL Display all the status information for each specified subscription definition. This is the default if you do not specify a generic name, and do not request any specific parameters.

z/OS On z/OS this is also the default if you specify a filter condition using the **WHERE** parameter, but on other platforms only, requested attributes are displayed.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if **QSGDISP** is set to GROUP.

'' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use **CMDSCOPE** as a filter keyword.

DURABLE
Specify this attribute to restrict the type of subscriptions which are displayed.

ALL Display all subscriptions.

NO Only information about nondurable subscriptions is displayed.

YES Only information about durable subscriptions is displayed.

SUBTYPE
Specify this attribute to restrict the type of subscriptions which are displayed.

USER Displays only **API** and **ADMIN** subscriptions.

PROXY
Only system created subscriptions relating to inter-queue manager subscriptions are selected.

ADMIN
Only subscriptions that have been created by an administration interface or modified by an administration interface are selected.

API Only subscriptions created by applications using an IBM MQ API call are selected.

ALL All subscription types are displayed (no restriction).

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

ACTCONN
Returns the *ConnId* of the *HConn* that currently has this subscription open.

DURABLE
A durable subscription is not deleted when the creating application closes its subscription handle.

NO The subscription is removed when the application that created it is closed or disconnected from the queue manager.

YES The subscription persists even when the creating application is no longer running or has been disconnected. The subscription is reinstated when the queue manager restarts.

LMSGDATE
The date on which a message was last published to the destination specified by this subscription.

LMSGTIME
The time on which a message was last published to the destination specified by this subscription.

MCASTREL
Indicator of the reliability of the multicast messages.

The values are expressed as a percentage. A value of 100 indicates that all messages are being delivered without problems. A value less than 100 indicates that some of the messages are experiencing network issues. To determine the nature of these issues you can enable event message generation, using the **COMMEV** parameter of the COMMINFO objects, and examine the generated event messages.

The following two values are returned:

- The first value is based on recent activity over a short period.
- The second value is based on activity over a longer period.

If no measurement is available the values are shown as blanks.

NUMMSGS
The number of messages put to the destination specified by this subscription since it was created, or since the queue manager was restarted, whichever is more recent. This number might not reflect the total number of messages that are, or have been, available to the consuming application. This is because it might also include publications that were partially processed but then undone by the queue manager due to a publication failure, or publications that were made within syncpoint that were rolled-back by the publishing application.

RESMDATE
The date of the most recent **MQSUB** API call that connected to the subscription.

RESMTIME
The time of the most recent **MQSUB** API call that connected to the subscription.

SUBID(*string*)
The internal, unique key identifying a subscription.

SUBUSER(*string*)
The owing user ID of the subscription.

SUBTYPE
Indicates how the subscription was created.

PROXY
An internally created subscription used for routing publications through a queue manager.

ADMIN
Created using the **DEF SUB** MQSC or PCF command. This **SUBTYPE** also indicates that a subscription has been modified using an administrative command.

API Created using an **MQSUB** API call.

TOPICSTR
Returns the fully resolved topic string of the subscription.

For more details of these parameters, see "DEFINE SUB" on page 698.

Related information:

Checking messages on a subscription

DISPLAY SECURITY on z/OS: z/OS

Use the MQSC command DISPLAY SECURITY to display the current settings for the security parameters.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

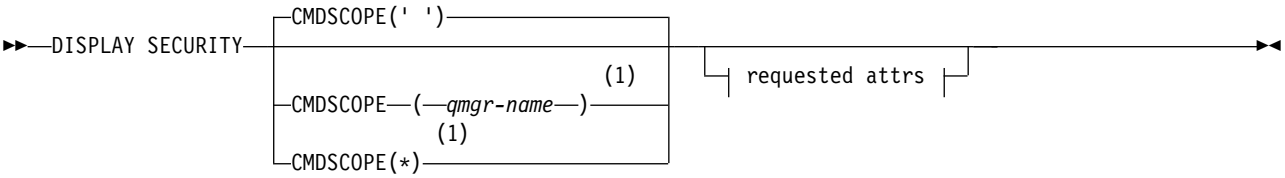
You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for DISPLAY SECURITY" on page 911

Note: From IBM WebSphere MQ Version 7.0 onwards, this command is no longer allowed to be issued from CSQINP1 or CSQINP2 on z/OS.

Synonym: DIS SEC

DISPLAY SECURITY



Requested attrs:

Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Parameter descriptions for DISPLAY SECURITY

CMDSCOPE
This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE cannot be used for commands issued from the first initialization input data set CSQINP1.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

ALL Display the TIMEOUT, INTERVAL, and SWITCHES parameters. This is the default if no requested parameters are specified.

The command also outputs an additional message, either CSQH037I or CSQH038I, stating whether security is currently using upper or mixed case security classes.

The command also outputs messages CSQH040I through CSQH042I showing the connection authentication settings currently in use.

INTERVAL
Time interval between checks.

SWITCHES
Display the current setting of the switch profiles.

If the subsystem security switch is off, no other switch profile settings are displayed.

TIMEOUT
Timeout value.

See "ALTER SECURITY on z/OS" on page 524 for details of the TIMEOUT and INTERVAL parameters.

Related information:
Displaying security status

DISPLAY SERVICE on Multiplatforms: Multi

Use the MQSC command DISPLAY SERVICE to display information about a service.

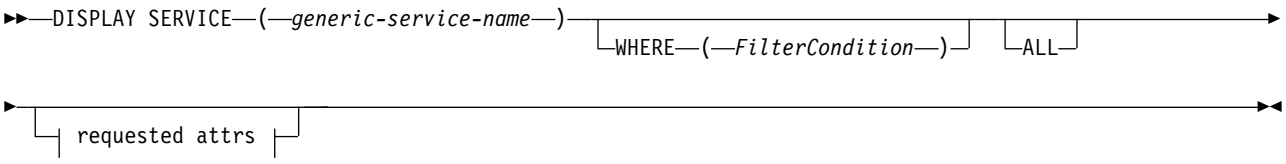
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

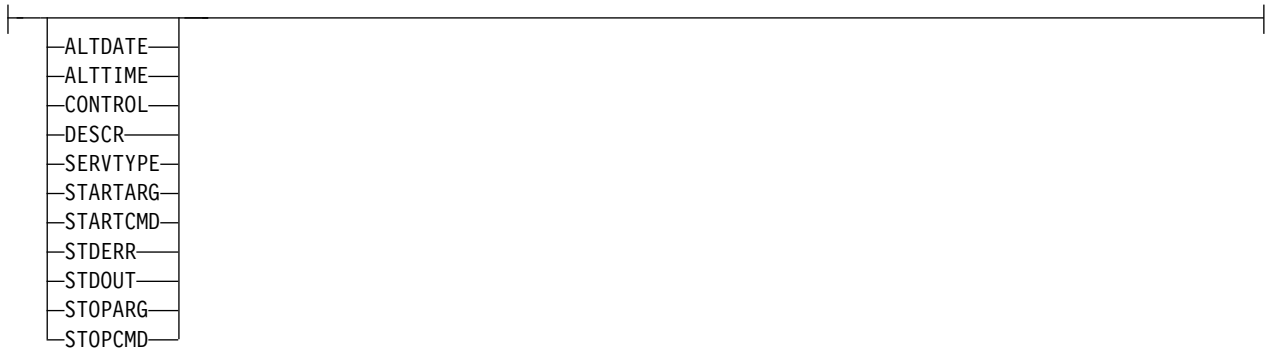
- Syntax diagram
- "Keyword and parameter descriptions for DISPLAY SERVICE"
- "Requested parameters" on page 913

Synonym:

DISPLAY SERVICE



Requested attrs:



Keyword and parameter descriptions for DISPLAY SERVICE

You must specify a service for which you want to display information. You can specify a service by using either a specific service name or a generic service name. By using a generic service name, you can display either:

- Information about all service definitions, by using a single asterisk (*), or
- Information about one or more service that match the specified name.

(*generic-service-name*)
The name of the service definition for which information is to be displayed. A single asterisk (*) specifies that information for all service identifiers is to be displayed. A character string with an asterisk at the end matches all services with the string followed by zero or more characters.

WHERE

Specify a filter condition to display information for those listeners that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword

Any parameter that can be used to display attributes for this DISPLAY command.

operator

This is used to determine whether a listener satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value

The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value MANUAL on the CONTROL parameter), you can only use EQ or NE. .
- A generic value. This is a character string. with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

ALL Specify this to display all the service information for each specified service. If this parameter is specified, any parameters that are requested specifically have no effect; all parameters are still displayed.

This is the default if you do not specify a generic identifier, and do not request any specific parameters.

On z/OS this is also the default if you specify a filter condition using the WHERE parameter, but on other platforms only requested attributes are displayed.

Requested parameters

Specify one or more attributes that define the data to be displayed. The attributes can be specified in any order. Do not specify the same attribute more than once.

ALTDATE

The date on which the definition was last altered, in the form `yyyy-mm-dd`.

ALTTIME

The time at which the definition was last altered, in the form `hh.mm.ss`.

CONTROL
How the service is to be started and stopped:

MANUAL
The service is not to be started automatically or stopped automatically. It is to be controlled by use of the START SERVICE and STOP SERVICE commands.

QMGR
The service is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY
The service is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

DESCR
Descriptive comment.

SERVTYPE
Specifies the mode in which the service is to run:

COMMAND
A command service object. Multiple instances of a command service object can be executed concurrently. You cannot monitor the status of command service objects.

SERVER
A server service object. Only one instance of a server service object can be executed at a time. The status of server service objects can be monitored using the DISPLAY SVSTATUS command.

STARTARG
Specifies the arguments to be passed to the user program at queue manager startup.

STARTCMD
Specifies the name of the program which is to run.

STDERR
Specifies the path to the file to which the standard error (stderr) of the service program is to be redirected.

STDOUT
Specifies the path to the file to which the standard output (stdout) of the service program is to be redirected.

STOPARG
Specifies the arguments to be passed to the stop program when instructed to stop the service.

STOPCMD
Specifies the name of the executable program to run when the service is requested to stop.

For more details of these parameters, see “DEFINE SERVICE on Multiplatforms” on page 691.

DISPLAY SMDS on z/OS: z/OS

Use the MQSC command DISPLAY SMDS to display the parameters of existing IBM MQ shared message data sets associated with a specified application structure.

Using MQSC commands

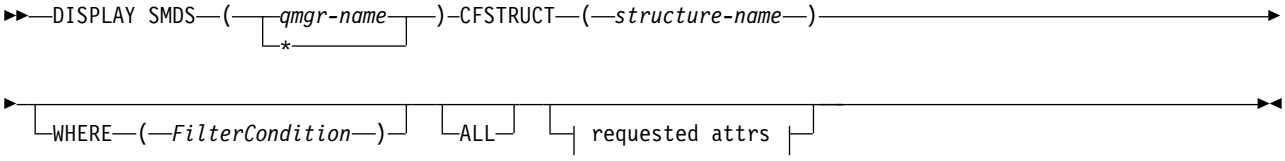
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for DISPLAY SMDS"
- "Usage notes for DISPLAY SMDSCONN" on page 918

Synonym:

DISPLAY SMDS



Requested attrs:

Notes:

1 z/OS For more information about this parameter, see Planning your coupling facility and offload storage environment.

Parameter descriptions for DISPLAY SMDS

The parameter descriptions for the DISPLAY SMDS command.

SMDS(*qmgr-name*|*)
Specifies the queue manager for which the shared message data set properties are to be displayed, or an asterisk to display the properties for all shared message data sets associated with the specified CFSTRUCT.

CFSTRUCT(*structure-name*)
Specify the coupling facility application structure for which the properties of one or more shared message data sets are to be displayed.

WHERE

Specify a filter condition to display only the SMDS information that satisfies the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword

Any parameter that can be used to display attributes for this DISPLAY command.

operator

This is used to determine whether a CF application structure satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value

The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use any of the operators except LK and NL. However, if the value is one from a possible set of values returnable on a parameter (for example, the value YES on the RECOVER parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. The characters must be valid for the attribute you are testing. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

  You can only use operators LK or NL for generic values on the DISPLAY SMDS command.

ALL Specify this keyword to display all attributes. If this keyword is specified, any attributes that are requested specifically have no effect; all attributes are still displayed.

This is the default behavior if you do not specify a generic name and do not request any specific attributes.

Requested parameters for DISPLAY SMDS

The following information is returned for each selected data set:

SMDS

The queue manager name which owns the shared message data set for which properties are being displayed.

CFSTRUCT

The coupling facility application structure name.

DSBUFS
Displays the override value for the number of buffers to be used by the owning queue manager for accessing shared message data sets for this structure, or DEFAULT if the group value from the CFSTRUCT definition is being used.

DSEXPAND
Displays the override value (YES or NO) for the data set expansion option, or DEFAULT if the group value from the CFSTRUCT definition is being used.

DISPLAY SMDSCONN on z/OS: z/OS

Use the MQSC command DISPLAY SMDSCONN to display status and availability information about the connection between the queue manager and the shared message data sets for the specified CFSTRUCT.

Using MQSC commands

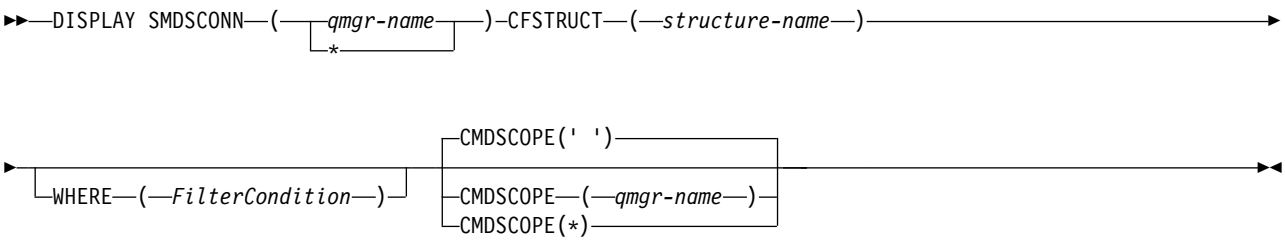
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for DISPLAY SMDSCONN"
- "Usage notes for DISPLAY SMDSCONN" on page 918

Synonym:

DISPLAY SMDSCONN



Parameter descriptions for DISPLAY SMDSCONN

The parameter descriptions for the DISPLAY SMDS command.

SMDSCONN(*qmgr-name*|*)
Specify the queue manager which owns the SMDS for which the connection information is to be displayed, or an asterisk to display the connection information for all shared message data sets associated with the specified CFSTRUCT.

CFSTRUCT(*structure-name*)
Specify the structure name for which the shared message data set connection information is required.

WHERE
Specify a filter condition to display only the SMDS connection information that satisfies the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Any parameter that can be used to display attributes for this DISPLAY command.

operator
This is used to determine whether a CF application structure satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use any of the operators except LK and NL. However, if the value is one from a possible set of values returnable on a parameter (for example, the value YES on the RECOVER parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. The characters must be valid for the attribute you are testing. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

  You can only use operators LK or NL for generic values on the DISPLAY SMDSCONN command.

CMDSCOPE
This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

'' The command runs on the queue manager on which it was entered.

This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group. You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

Usage notes for DISPLAY SMDSCONN

This command is only supported when the CFSTRUCT definition is currently using the option OFFLOAD(SMDS).

This information indicates whether the queue manager is currently able to allocate and open the data set.

The following results are returned for each selected connection:

SMDSCONN
The name of the queue manager which owns the shared message data set for this connection.

CFSTRUCT
The name of the coupling facility application structure.

OPENMODE
The mode in which the data set is currently open by this queue manager. This is one of the following:

NONE
The data set is not currently open.

READONLY
The data set is owned by another queue manager and is open for read-only access.

UPDATE
The data set is owned by this queue manager and is open for update access.

RECOVERY
The data set is open for recovery processing.

STATUS
The connection status as seen by this queue manager. This is one of the following:

CLOSED
This data set is not currently open.

OPENING
This queue manager is currently in the process of opening and validating this data set (including space map restart processing when necessary).

OPEN This queue manager has successfully opened this data set and it is available for normal use.

CLOSING
This queue manager is currently in the process of closing this data set, including quiescing normal I/O activity and storing the saved space map if necessary.

NOTENABLED
The SMDS definition is not in the ACCESS(ENABLED) state so the data set is not currently available for normal use. This status is only set when the SMDSCONN status does not already indicate some other form of failure.

ALLOCFAIL
This queue manager was unable to locate or allocate this data set.

OPENFAIL
This queue manager was able to allocate the data set but was unable to open it, so it has now been deallocated.

STGFAIL
The data set could not be used because the queue manager was unable to allocate associated storage areas for control blocks, or for space map or header record processing.

DATAFAIL
The data set was successfully opened but the data was found to be invalid or inconsistent, or a permanent I/O error occurred, so it has now been closed and deallocated.

This may result in the shared message data set itself being marked as STATUS(FAILED).

AVAIL
The availability of this data set connection as seen by this queue manager. This is one of the following:

NORMAL
The connection can be used and no error has been detected.

ERROR
The connection is unavailable because of an error.

The queue manager may try to enable access again automatically if the error may no longer be present, for example when recovery completes or the status is manually set to RECOVERED. Otherwise, it can be enabled again using the START SMDSCONN command in order to retry the action which originally failed.

STOPPED
The connection cannot be used because it has been explicitly stopped using the STOP SMDSCONN command. It can only be made available again by using a START SMDSCONN command to enable it.

EXPANDST
The data set automatic expansion status. This is one of the following:

NORMAL
No problem has been noted which would affect automatic expansion.

FAILED
A recent expansion attempt failed, causing the DSEXPAND option to be set to NO for this specific data set. This status is cleared when ALTER SMDS is used to set the DSEXPAND option back to YES or DEFAULT

MAXIMUM
The maximum number of extents has been reached, so future expansion is not possible (except by taking the data set out of service and copying it to larger extents).

Note, that the command works only if the structure is currently connected, that is, some shared queues allocated to that structure have been opened.

Related reference:

"START SMDSCONN on z/OS" on page 1040
Use the MQSC command START SMDSCONN to enable a previously stopped connection from this queue manager to the specified shared message data sets, allowing them to be allocated and opened again.

"STOP SMDSCONN on z/OS" on page 1058
Use the MQSC command STOP SMDSCONN to terminate the connection from this queue manager to one or more specified shared message data sets (causing them to be closed and deallocated) and to mark the connection as STOPPED.

DISPLAY STGCLASS on z/OS: z/OS

Use the MQSC command DISPLAY STGCLASS to display information about storage classes.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

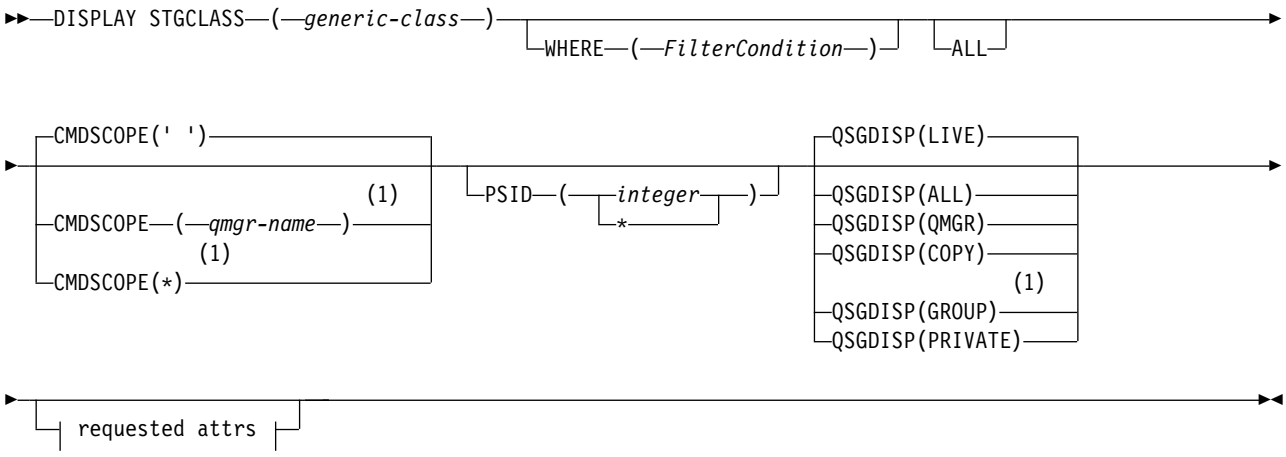
You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram

- "Parameter descriptions for DISPLAY STGCLASS"
- "Requested parameters" on page 923

Synonym: DIS STC

DISPLAY STGCLASS



Requested attrs:

Notes:

1 Valid only on IBM MQ for z/OS when the queue manager is a member of a queue-sharing group.

Parameter descriptions for DISPLAY STGCLASS

You use DISPLAY STGCLASS to show the page set identifiers that are associated with each storage class.

(generic-class)
Name of the storage class. This is required.

This is 1 through 8 characters. The first character is in the range A through Z; subsequent characters are A through Z or 0 through 9.

A trailing asterisk (*) matches all storage classes with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all storage classes.

WHERE
Specify a filter condition to display only those storage classes that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

`filter-keyword`
Almost any parameter that can be used to display attributes for this DISPLAY command.

However, you cannot use the CMDSCOPE or QSGDISP parameters as filter keywords. You cannot use PSID as a filter keyword if you also use it to select storage classes.

operator
This is used to determine whether a connection satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE or GE only. However, if the attribute value is one from a possible set of values on a parameter, you can only use EQ or NE.
- A generic value. This is a character string (such as the character string in the DESCR parameter) with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string ABC are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

ALL Specify this to display all the parameters. If this parameter is specified, any parameters that are also requested specifically have no effect; all parameters are still displayed.

This is the default if you do not specify a generic name, and do not request any specific parameters.

On z/OS this is also the default if you specify a filter condition using the WHERE parameter, but on other platforms only requested attributes are displayed.

CMDSCOPE
This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

If QSGDISP is set to GROUP, CMDSCOPE must be blank or the local queue manager.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue

manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

PSID(*integer*)
The page set identifier that a storage class maps to. This is optional.

The string consists of two numeric characters, in the range 00 through 99. An asterisk (*) on its own specifies all page set identifiers. See "DEFINE PSID on z/OS" on page 656.

QSGDISP
Specifies the disposition of the objects for which information is to be displayed. Values are:

LIVE This is the default value and displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

ALL Displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with QSGDISP(GROUP).

If QSGDISP(ALL) is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

In a shared queue manager environment, use

```
DISPLAY STGCLASS(generic-class) CMDSCOPE(*) QSGDISP(ALL)
```

to list ALL objects matching

```
name
```

in the queue-sharing group without duplicating those in the shared repository.

COPY Display information only for objects defined with QSGDISP(COPY).

GROUP
Display information only for objects defined with QSGDISP(GROUP). This is allowed only if there is a shared queue manager environment.

PRIVATE
Display information only for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

QMGR
Display information only for objects defined with QSGDISP(QMGR).

QSGDISP displays one of the following values:

QMGR
The object was defined with QSGDISP(QMGR).

GROUP
The object was defined with QSGDISP(GROUP).

COPY The object was defined with QSGDISP(COPY).

You cannot use QSGDISP as a filter keyword.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

The default, if no parameters are specified (and the ALL parameter is not specified) is the storage class names, their page set identifiers and queue-sharing group dispositions are displayed.

ALTDATE
The date on which the definition was last altered, in the form yyyy-mm-dd.

ALTTIME
The time at which the definition was last altered, in the form hh.mm.ss.

DESCR
Descriptive comment.

PASSTKTA
The application name used to authenticate IMS bridge passtickets. A blank value indicates that the default batch job profile name is to be used.

XCFGNAME
The name of the XCF group that IBM MQ is a member of.

XCFMNAME
The XCF member name of the IMS system within the XCF group specified in XCFGNAME.

For more details of these parameters, see "DEFINE STGCLASS on z/OS" on page 694.

DISPLAY SUB:

Use the MQSC command **DISPLAY SUB** to display the attributes associated with a subscription.

Using MQSC commands

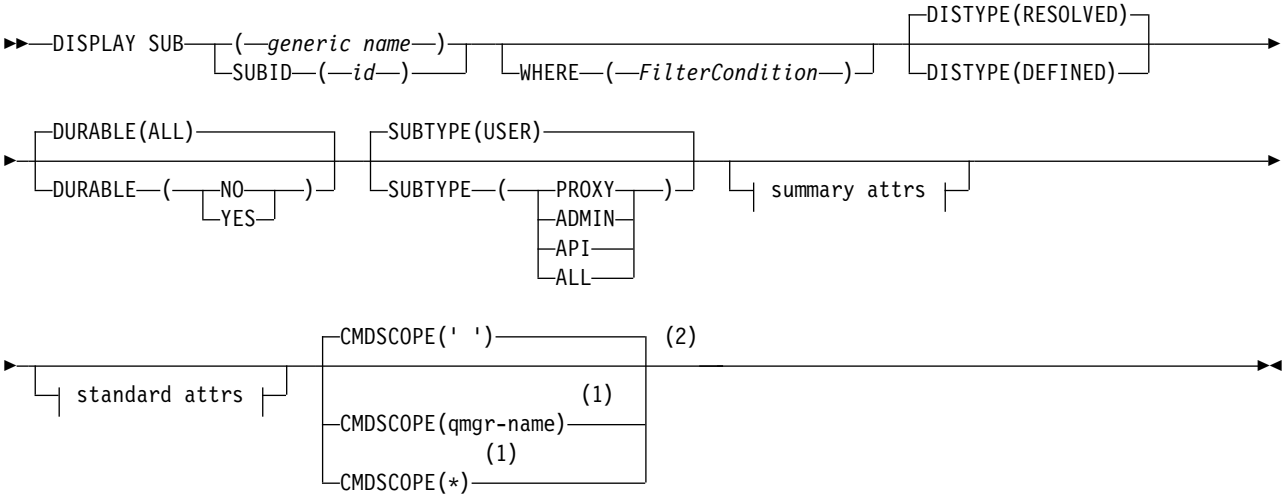
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for DISPLAY SUB" on page 925
- "Parameter descriptions for DISPLAY SUB" on page 926

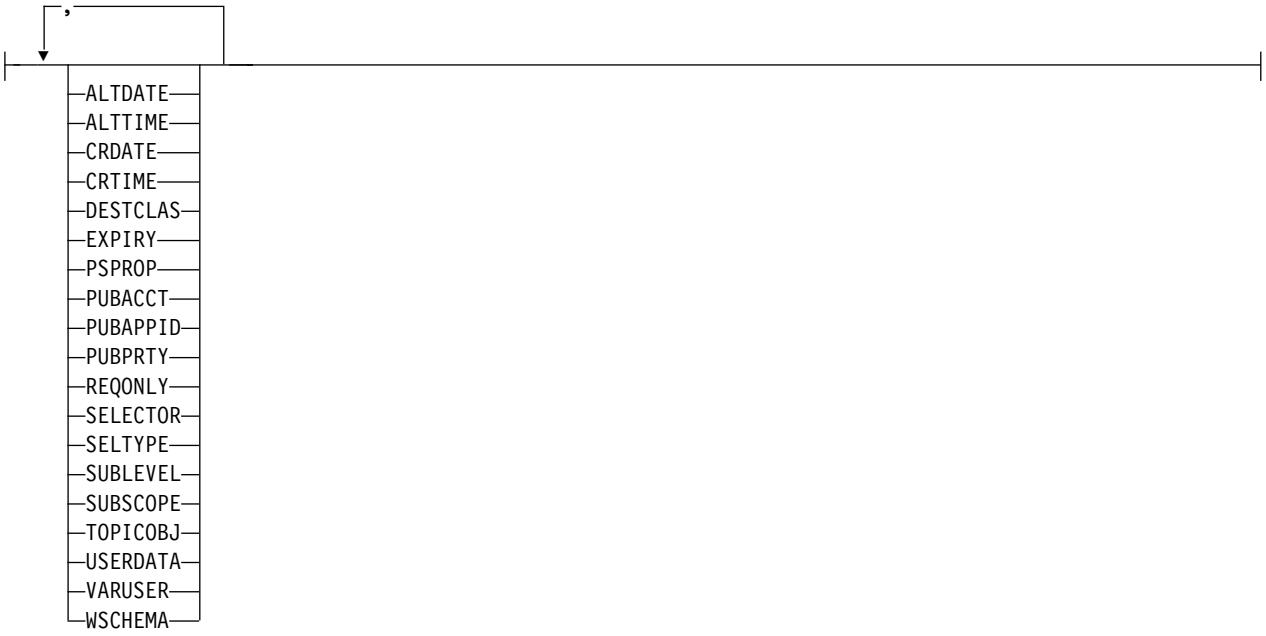
Synonym: **DIS SUB**

DISPLAY SUB



summary attributes:

standard attributes:



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes for DISPLAY SUB

The **TOPICSTR** parameter might contain characters that cannot be translated into printable characters when the command output is displayed.

z/OS On z/OS, these non-printable characters are displayed as blanks.

Multi On Multiplatforms using runmqsc, these non-printable characters are displayed as dots.

Parameter descriptions for DISPLAY SUB

You must specify either the name or the identifier of subscription you want to display. This can be a specific subscription name, or SUBID, or a generic subscription name. By using a generic subscription name, you can display either:

- All subscription definitions
- One or more subscriptions that match the specified name

The following forms are valid:

```
DIS SUB(xyz)
DIS SUB SUBID(123)
DIS SUB(xyz*)
```

(generic-name)
The local name of the subscription definition to be displayed. A trailing asterisk (*) matches all subscriptions with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all subscriptions.

WHERE
Specify a filter condition to display only those subscriptions that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this DISPLAY command. However, you cannot use the CMDSCOPE parameter as a filter keyword. Subscriptions of a type for which the filter keyword is not a valid attribute are not displayed.

operator
This is used to determine whether a subscription satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value QALIAS on the CLUSQT parameter), you can only use EQ or NE. For the parameters HARDENBO, SHARE, and TRIGGER, use either EQ YES or EQ NO.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

Note: z/OS On z/OS there is a 256 character limit for the filter-value of the MQSC WHERE clause. This limit is not in place for other platforms.

SUMMARY
Specify this to display the set of summary attributes that you want displayed.

ALL Specify this to display all the attributes.

If this parameter is specified, any attributes that are also requested specifically have no effect; all attributes are still displayed.

This is the default if you do not specify a generic name and do not request any specific attributes.

ALTDATE(*string*)
The date of the most recent **MQSUB** or **ALTER SUB** command that modified the properties of the subscription.

ALTTIME(*string*)
The time of the most recent **MQSUB** or **ALTER SUB** command that modified the properties of the subscription.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of setting this value is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

CRDATE(*string*)
The date of the first **MQSUB** or **DEF SUB** command that created this subscription.

CRTIME(*string*)
The time of the first **MQSUB** or **DEF SUB** command that created this subscription.

DEST(*string*)
The destination for messages published to this subscription; this parameter is the name of a queue.

DESTCLAS
System managed destination.

PROVIDED
The destination is a queue.

MANAGED
The destination is managed.

DESTCORL(*string*)
The **CorrelId** used for messages published to this subscription.

A blank value (default) results in a system generated correlation identifier being used.

If set to ' 000000000000000000000000000000000000000000000000 ' (48 zeros) the **CorrelId** set by the publishing application will be maintained in the copy of the message delivered to the subscription, unless messages are propagated across a publish/subscribe hierarchy.

DESTQMGR(*string*)
The destination queue manager for messages published to this subscription.

DISTYPE
Controls the output returned in the **TOPICSTR** and **TOPICOBJ** attributes.

RESOLVED
Returns the resolved (full) topic string in the **TOPICSTR** attribute. The value of the **TOPICOBJ** attribute is also returned. This is the default value.

DEFINED
Returns the values of the **TOPICOBJ** and **TOPICSTR** attributes provided when the subscription was created. The **TOPICSTR** attribute will contain the application part of the topic string only. You can use the values returned with **TOPICOBJ** and **TOPICSTR** to fully re-create the subscription by using **DISTYPE(DEFINED)**.

DURABLE
A durable subscription is not deleted when the creating application closes its subscription handle.

ALL Display all subscriptions.

NO The subscription is removed when the application that created it, is closed or disconnected from the queue manager.

YES The subscription persists even when the creating application is no longer running or has been disconnected. The subscription is reinstated when the queue manager restarts.

EXPIRY
The time to expiry of the subscription object from the creation date and time.

(*integer*)
The time to expiry, in tenths of a second, from the creation date and time.

UNLIMITED
There is no expiry time. This is the default option supplied with the product.

PSPROP
The manner in which publish subscribe related message properties are added to messages sent to this subscription.

NONE Do not add publish subscribe properties to the message.

COMPAT Publish/subscribe properties are added within an MQRFH version 1 header unless the message was published in PCF format.

MSGPROP
Publish/subscribe properties are added as message properties.

RFH2 Publish/subscribe properties are added within an MQRFH version 2 header.

PUBACCT(*string*)
Accounting token passed by the subscriber, for propagation into messages published to this subscription in the `AccountingToken` field of the MQMD.

PUBAPPID(*string*)
Identity data passed by the subscriber, for propagation into messages published to this subscription in the `ApplIdentityData` field of the MQMD.

PUBPRTY
The priority of the message sent to this subscription.

ASPUB Priority of the message sent to this subscription is taken from the priority supplied in the published message.

ASQDEF Priority of the message sent to this subscription is taken from the default priority of the queue defined as a destination.

(integer)
An integer providing an explicit priority for messages published to this subscription.

REQONLY
Indicates whether the subscriber polls for updates using the MQSUBRQ API call, or whether all publications are delivered to this subscription.

NO All publications on the topic are delivered to this subscription.

YES Publications are only delivered to this subscription in response to an MQSUBRQ API call.

This parameter is equivalent to the subscribe option MQSO_PUBLICATIONS_ON_REQUEST.

SELECTOR(*string*)
A selector that is applied to messages published to the topic.

SELTYPE
The type of selector string that has been specified.

NONE
No selector has been specified.

STANDARD
The selector references only the properties of the message, not its content, using the standard IBM MQ selector syntax. Selectors of this type are to be handled internally by the queue manager.

EXTENDED
The selector uses extended selector syntax, typically referencing the content of the message. Selectors of this type cannot be handled internally by the queue manager; extended selectors can be handled only by another program such as IBM Integration Bus.

SUB(*string*)
The application's unique identifier for a subscription.

SUBID(*string*)
The internal, unique key identifying a subscription.

SUBLEVEL(*integer*)
The level within the subscription hierarchy at which this subscription is made. The range is zero through 9.

SUBSCOPE
Determines whether this subscription is forwarded to other queue managers, so that the subscriber receives messages published at those other queue managers.

ALL The subscription is forwarded to all queue managers directly connected through a publish/subscribe collective or hierarchy.

QMGR The subscription forwards messages published on the topic only within this queue manager.

Note: Individual subscribers can only restrict **SUBSCOPE**. If the parameter is set to ALL at topic level, then an individual subscriber can restrict it to QMGR for this subscription. However, if the parameter is set to QMGR at topic level, then setting an individual subscriber to ALL has no effect.

SUBTYPE
Indicates how the subscription was created.

USER Displays only **API** and **ADMIN** subscriptions.

PROXY
An internally created subscription used for routing publications through a queue manager.

V 9.0.2 V 9.0.0.1 Subscriptions of type PROXY are not modified to ADMIN when alterations are attempted.

ADMIN
Created using **DEF SUB** MQSC or PCF command. This **SUBTYPE** also indicates that a subscription has been modified using an administrative command.

API Created using an **MQSUB** API request.

ALL All.

SUBUSER(*string*)
Specifies the user ID that is used for security checks that are performed to ensure that publications can be put to the destination queue associated with the subscription. This ID is either the user ID associated with the creator of the subscription or, if subscription takeover is permitted, the user ID that last took over the subscription. The length of this parameter must not exceed 12 characters.

TOPICOBJ(*string*)
The name of a topic object used by this subscription.

TOPICSTR(*string*)
Returns a topic string, that can contain wildcard characters to match a set of topic strings, for the subscription. The topic string is either the application provided portion only, or fully qualified, depending on the value of **DISTYPE**.

USERDATA(*string*)
Specifies the user data associated with the subscription. The string is a variable length value that can be retrieved by the application on an MQSUB API call and passed in a message sent to this subscription as a message property. The **USERDATA** is stored in the RFH2 header in the mqps folder with the key Sud.

V 9.0.2 V 9.0.0.2 An IBM MQ classes for JMS application can retrieve the subscription user data from the message by using the constant JMS_IBM_SUBSCRIPTION_USER_DATA. For more information, see Retrieval of user subscription data.

VARUSER
Specifies whether a user other than the subscription creator can connect to and take over ownership of the subscription.

ANY Any user can connect to and takeover ownership of the subscription.

FIXED Takeover by another USERID is not permitted.

WSCHEMA
The schema to be used when interpreting any wildcard characters in the topic string.

CHAR Wildcard characters represent portions of strings.

TOPIC Wildcard characters represent portions of the topic hierarchy.

Related information:
Displaying attributes of subscriptions

DISPLAY SVSTATUS on Multiplatforms: Multi

Use the MQSC command **DISPLAY SVSTATUS** to display status information for one or more services. Only services with a **SERVTYPE** of SERVER are displayed.

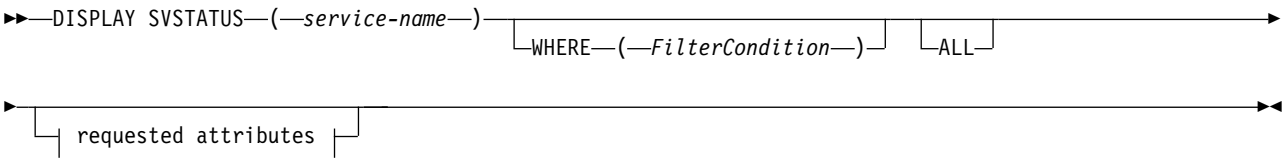
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

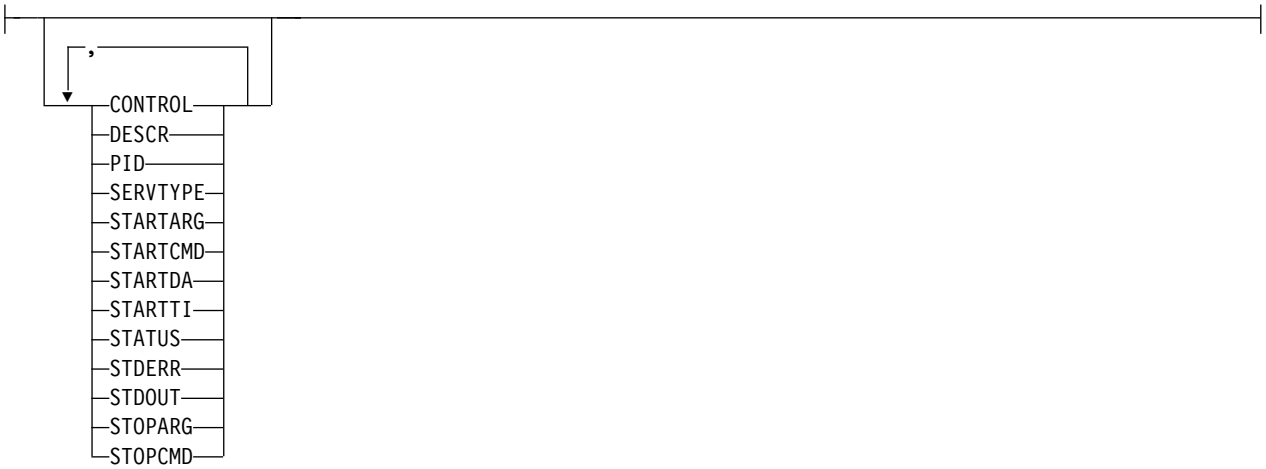
- Syntax diagram
- "Keyword and parameter descriptions for DISPLAY SVSTATUS"
- "Requested parameters" on page 932

Synonym:

DISPLAY SVSTATUS



Requested attributes:



Keyword and parameter descriptions for DISPLAY SVSTATUS

You must specify a service for which you want to display status information. You can specify a service by using either a specific service name or a generic service name. By using a generic service name, you can display either:

- Status information for all service definitions, by using a single asterisk (*), or
- Status information for one or more services that match the specified name.

(*generic-service-name*)
The name of the service definition for which status information is to be displayed. A single

asterisk (*) specifies that information for all connection identifiers is to be displayed. A character string with an asterisk at the end matches all services with the string followed by zero or more characters.

WHERE
Specify a filter condition to display status information for those services that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Any parameter that can be used to display attributes for this **DISPLAY** command.

operator
This is used to determine whether a service satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE, or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value MANUAL on the **CONTROL** parameter), you can only use EQ or NE.
- A generic value. This is a character string. with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

ALL Display all the status information for each specified service. This is the default if you do not specify a generic name, and do not request any specific parameters.

Requested parameters

Specify one or more attributes that define the data to be displayed. The attributes can be specified in any order. Do not specify the same attribute more than once.

CONTROL
How the service is to be started and stopped:

MANUAL
The service is not to be started automatically or stopped automatically. It is to be controlled by use of the **START SERVICE** and **STOP SERVICE** commands.

QMGR
The service is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY
The service is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

DESCR
Descriptive comment.

PID The operating system process identifier associated with the service.

SERVTYPE
The mode in which the service runs. A service can have a **SERVTYPE** of SERVER or COMMAND, but only services with **SERVTYPE(SERVER)** are displayed by this command.

STARTARG
The arguments passed to the user program at startup.

STARTCMD
The name of the program being run.

STARTDA
The date on which the service was started.

STARTTI
The time at which the service was started.

STATUS
The status of the process:

RUNNING
The service is running.

STARTING
The service is in the process of initializing.

STOPPING
The service is stopping.

STDERR
Destination of the standard error (stderr) of the service program.

STDOUT
Destination of the standard output (stdout) of the service program.

STOPARG
The arguments to be passed to the stop program when instructed to stop the service.

STOPCMD
The name of the executable program to run when the service is requested to stop.

For more information about these parameters, see “DEFINE SERVICE on Multiplatforms” on page 691.

Related information:
Working with services
Examples of using service objects

DISPLAY SYSTEM on z/OS: z/OS

Use the MQSC command DISPLAY SYSTEM to display general system parameters and information.

Using MQSC commands

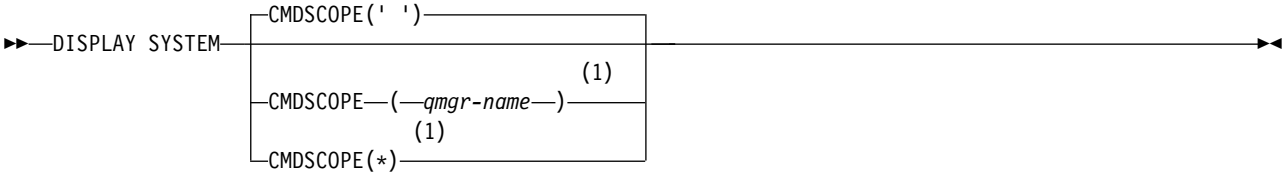
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 12CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for DISPLAY SYSTEM"
- "Parameter descriptions for DISPLAY SYSTEM" on page 935

Synonym: DIS SYSTEM

DISPLAY SYSTEM



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Usage notes for DISPLAY SYSTEM

1. DISPLAY SYSTEM returns a report that shows the initial values of the system parameters and the current values as changed by the SET SYSTEM command:
   - Default user ID for command security checks (CMDUSER).
   - Time in seconds for which queue manager exits can execute during each invocation (EXITLIM).
   - How many started server tasks to use to run queue manager exits (EXITTCB).
   - Number of log records written by IBM MQ between the start of one checkpoint and the next (LOGLOAD).
   - The Measured Usage Pricing property for this queue manager (MULCCAPT). This property is only displayed if the MULCCAPT property is set to REFINED.
   - CD The Operation Mode for this system (OPMODE). This is a list of three elements:
     - The first indicates whether the queue manager is operating in compatibility mode or new function mode.
     - The second shows the current compatibility level.
     - The third shows the level of new functions that are available.
   - The OTMA connection parameters (OTMACON).
   - Whether queue manager restart waits until all indexes are built, or completes before all indexes are built (QINDXBLD).

- Coded character set identifier for the queue manager (QMCCSID).
- The queue-sharing group parameters (QSGDATA).
- The RESLEVEL auditing parameter (RESAUDIT).
- The message routing code assigned to messages not solicited from a specific console (ROUTCDE).
- Whether SMF accounting data is collected when IBM MQ is started (SMFACCT).
- Whether SMF statistics are collected when IBM MQ is started (SMFSTAT).
- Default time, in minutes, between each gathering of statistics (STATIME).
- Whether tracing is started automatically (TRACSTR).
- Size of trace table, in 4 KB blocks, to be used by the global trace facility (TRACTBL).
- Time between scanning the queue index for WLM-managed queues (WLMTIME).
- WLMTIMU indicates whether WLMTIME is given in seconds or minutes.
- Whether batch jobs can currently be swapped out during some MQ API calls or not (CONNSWAP).
- A list of messages excluded from being written to any log (EXCLMSG).
- It might also return a report about system status.

2. This command is issued internally by IBM MQ at the end of queue manager startup.
3. Message CSQY101I includes the system parameter value for OPMODE that is being used in CSQ6SYSP. z/OS For more details, see Using CSQ6SYSP.

   CD In contrast, the OPMODE returned by the DISPLAY SYSTEM command has two further parameters containing the current compatibility and available function levels:

   - The compatibility level indicates which version the queue manager has been migrated from and therefore can fall back to if necessary providing the appropriate backward migration PTFs have been installed at that release.
   - The available function level indicates the available level of MQ new functions that are restricted by OPMODE. See OPMODE restrictions by version for more information on the functions that are restricted by OPMODE.

   This parameter is valid only on z/OS.

Parameter descriptions for DISPLAY SYSTEM

CMDSCOPE

This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE cannot be used for commands issued from the first initialization input data set CSQINP1.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect is the same as entering the command on every queue manager in the queue-sharing group.

DISPLAY TCLUSTER:

Use the MQSC command DISPLAY TCLUSTER to display the attributes of the IBM MQ cluster topic object.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

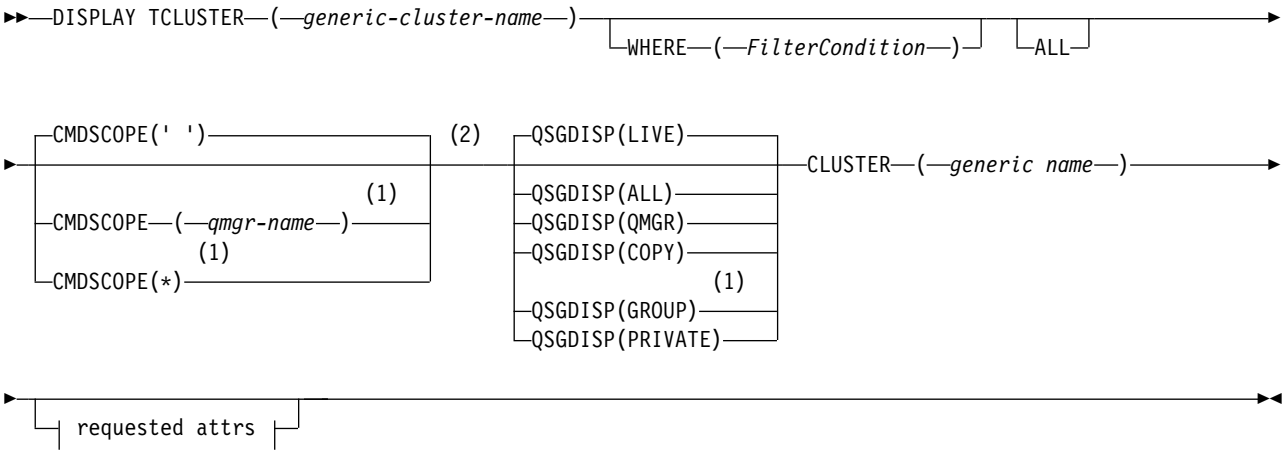
z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

The DISPLAY TCLUSTER command produces the same output as the DISPLAY TOPIC TYPE(CLUSTER) command.

See "DISPLAY TOPIC" on page 942 for further information.

Synonym: DIS TCLUSTER

DISPLAY TCLUSTER



Requested attrs:

Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Parameter descriptions for DISPLAY TCLUSTER

You must specify the name of the cluster topic definition you want to display. This name can be a specific cluster topic name or a generic cluster topic name. By using a generic topic name, you can display either:

(generic-cluster-name)
The name of the administrative cluster definition to be displayed (see Rules for naming IBM MQ objects). A trailing asterisk (*) matches all administrative topic objects with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all administrative topic objects.

WHERE
Specify a filter condition to display only those administrative topic object definitions that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

`filter-keyword`
Almost any parameter that can be used to display attributes for this DISPLAY command.

z/OS However, you cannot use the CMDSCOPE, or QSGDISP parameters as filter keywords.

`operator`
This part is used to determine whether a topic object satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

`filter-value`
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this value can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE, or GE only. However, if the attribute value is one from a possible set of values on a parameter, you can use only EQ or NE.
- A generic value. This value is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

Note: z/OS On z/OS there is a 256 character limit for the filter-value of the MQSC **WHERE** clause. This limit is not in place for other platforms.

ALL Specify this parameter to display all the attributes. If this parameter is specified, any attributes that are requested specifically have no effect; all attributes are still displayed.

This is the default if you do not specify a generic name, and do not request any specific attributes.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command runs on the queue manager on which it was entered. This value is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this process is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

z/OS **QSGDISP**

Specifies the disposition of the objects for which information is to be displayed. Values are:

LIVE LIVE is the default value and displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

ALL Display information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with QSGDISP(GROUP).

If QSGDISP(ALL) is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

In a shared queue manager environment, use

```
DISPLAY TOPIC(name) CMDSCOPE(*) QSGDISP(ALL)
```

to list ALL objects matching `name` in the queue-sharing group without duplicating those objects in the shared repository.

COPY Display information only for objects defined with QSGDISP(COPY).

GROUP
Display information only for objects defined with QSGDISP(GROUP). This is allowed only if there is a shared queue manager environment.

PRIVATE
Display information only for objects defined with QSGDISP(QMGR) or QSGDISP(COPY). QSGDISP(PRIVATE) displays the same information as QSGDISP(LIVE).

QMGR
Display information only for objects defined with QSGDISP(QMGR).

QSGDISP
QSGDISP displays one of the following values:

QMGR
The object was defined with QSGDISP(QMGR).

GROUP
The object was defined with QSGDISP(GROUP).

COPY The object was defined with QSGDISP(COPY).

You cannot use QSGDISP as a filter keyword.

CLUSTER
Displays topics with the specified cluster name. The value can be a generic name.

Requested attributes

CLROUTE
The routing behavior to use for topics in the cluster defined by the **CLUSTER** parameter.

CLSTATE
The current state of this topic in the cluster defined by the **CLUSTER** parameter. The values can be as follows:

ACTIVE
The cluster topic is correctly configured and being adhered to by this queue manager.

PENDING
Only seen by a hosting queue manager, this state is reported when the topic has been created but the full repository has not yet propagated it to the cluster. This might be because the host queue manager is not connected to a full repository, or because the full repository has deemed the topic to be invalid.

INVALID
This clustered topic definition conflicts with an earlier definition in the cluster and is therefore not currently active.

ERROR
An error has occurred with respect to this topic object.

This parameter is typically used to aid diagnosis when multiple definitions of the same clustered topic are defined on different queue managers, and the definitions are not identical. See Routing for publish/subscribe clusters: Notes on behavior.

CLUSDATE
The date on which the information became available to the local queue manager, in the form `yyyy-mm-dd`.

CLUSQMGR
The name of the queue manager that hosts the topic.

CLUSTIME
The time at which the information became available to the local queue manager, in the form `hh.mm.ss`.

QMID
The internally generated unique name of the queue manager that hosts the topic.

Usage notes for DISPLAY TCLUSTER

1. On z/OS, the channel initiator must be running before you can display information about cluster topics.
2. The TOPICSTR parameter might contain characters that cannot be translated into printable characters when the command output is displayed.

   z/OS On z/OS, these non-printable characters are displayed as blanks.

Multi On Multiplatforms using the **runmqsc** command, these non-printable characters are displayed as dots.

Related reference:

"DISPLAY TPSTATUS" on page 950
Use the MQSC command **DISPLAY TPSTATUS** to display the status of one or more topics in a topic tree.

"DISPLAY TOPIC" on page 942
Use the MQSC command **DISPLAY TOPIC** to display the attributes of one or more IBM MQ topic objects of any type.

DISPLAY THREAD on z/OS: z/OS

Use the MQSC command DISPLAY THREAD to display information about active and in-doubt threads.

Using MQSC commands

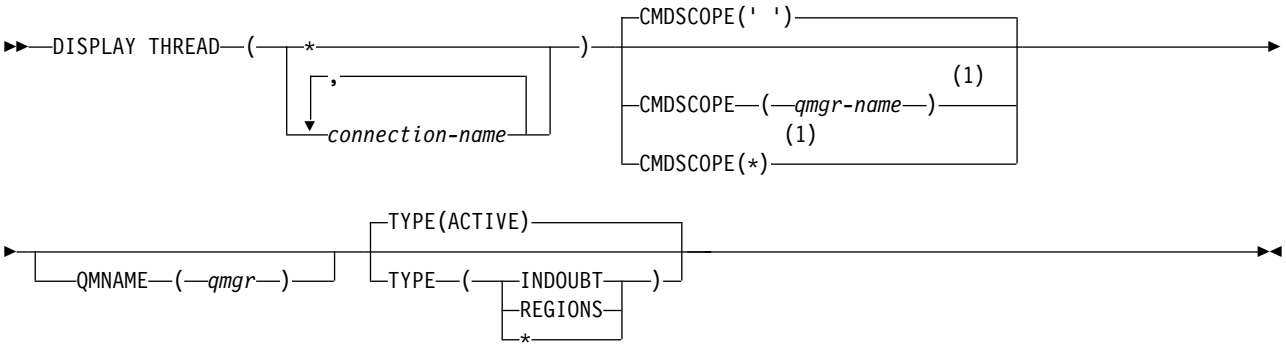
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes"
- "Parameter descriptions for DISPLAY THREAD" on page 941

Synonym: DIS THD

DISPLAY THREAD



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Usage notes

Threads shown as in doubt on one invocation of this command will probably be resolved for subsequent invocations.

This command is retained for compatibility with earlier release of IBM MQ. It has been superseded by the DISPLAY CONN command which is preferable to use.

Parameter descriptions for DISPLAY THREAD

(connection-name)

List of one or more *connection-name* s (of 1 through 8 characters each).

- For batch connections, this name is the batch job name
- For CICS connections, this name is the CICS applid
- For IMS connections, this name is the IMS job name
- For TSO connections, this name is the TSO user ID
- For RRS connections, this is RRSBATCH for all RRSBATCH-type connections, or the batch job name

Threads are selected from the address spaces associated with these connections only.

(*) Displays threads associated with all connections to IBM MQ.

CMDSCOPE

This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

TYPE The type of thread to display. This parameter is optional.

ACTIVE

Display only active threads.

An active thread is one for which a unit of recovery has started but not completed. Resources are held in IBM MQ on its behalf.

This is the default if TYPE is omitted.

INDOUBT

Display only in-doubt threads.

An in-doubt thread is one that is in the second phase of the two-phase commit operation. Resources are held in IBM MQ on its behalf. External intervention is needed to resolve the status of in-doubt threads. You might only have to start the recovery coordinator (CICS, IMS, or RRS), or you might need to do more. They might have been in doubt at the last restart, or they might have become in doubt since the last restart.

REGIONS

Display a summary of active threads for each active connection.

Note: Threads used internally by IBM MQ are excluded.

***** Display both active and in-doubt threads, but not regions.

If, during command processing, an active thread becomes in doubt, it might appear twice: once as active and once as in doubt.

QMNAME

Specifies that IBM MQ should check whether the designated queue manager is INACTIVE, and if so, report any shared units of work that were in progress on the designated and inactive queue manager.

This option is valid only for TYPE(INDOUBT).

z/OS For more information about the DISPLAY THREAD command and in-doubt recovery, see Recovering units of recovery on another queue manager in the queue-sharing group. Also, see messages CSQV401I through CSQV406I, and CSQV432I, in Agent services messages (CSQV...).

DISPLAY TOPIC:

Use the MQSC command **DISPLAY TOPIC** to display the attributes of one or more IBM MQ topic objects of any type.

Using MQSC commands

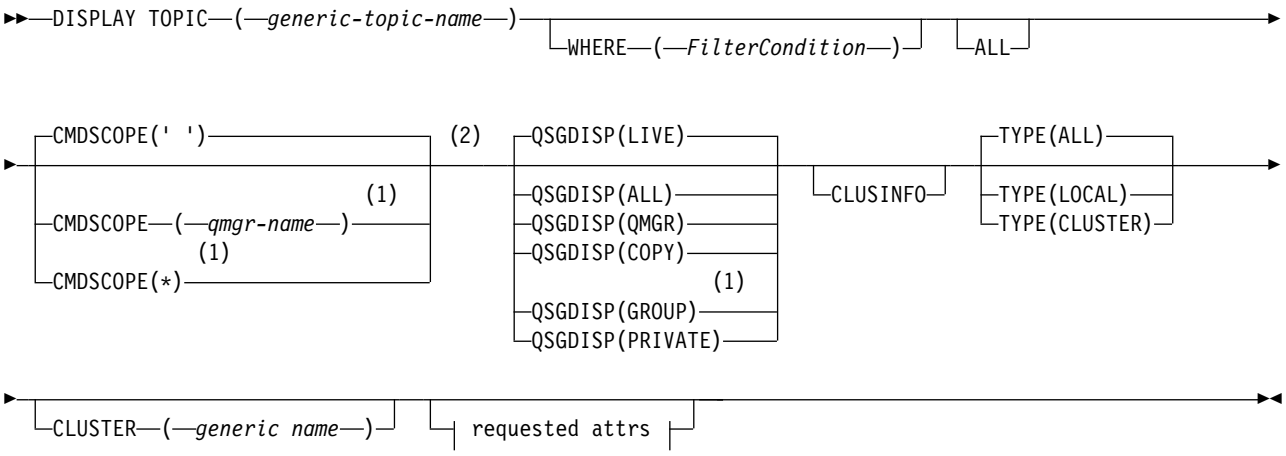
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

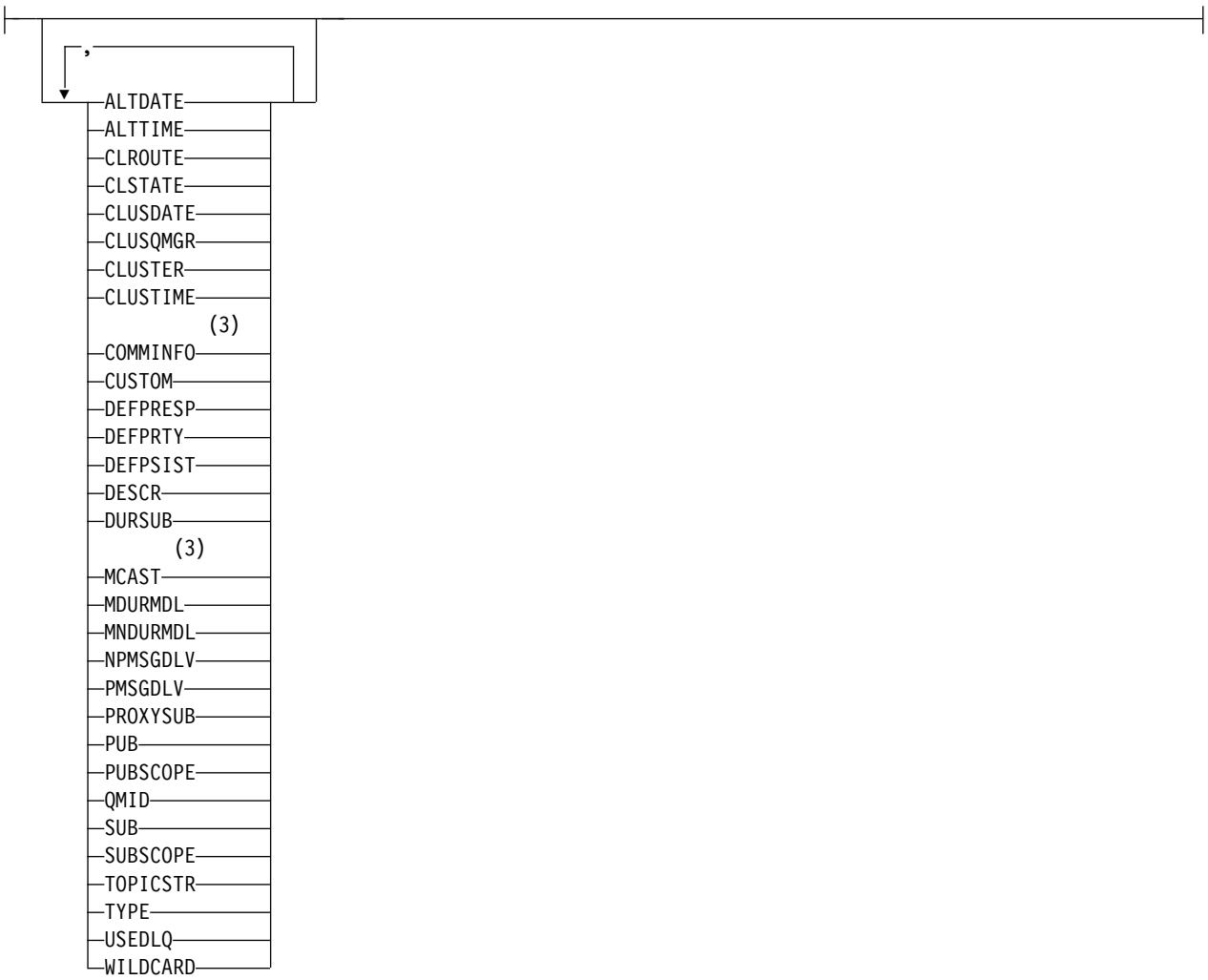
- Syntax diagram
- "Usage notes for DISPLAY TOPIC" on page 943
- "Parameter descriptions for DISPLAY TOPIC" on page 944
- "Requested parameters" on page 946

Synonym: DIS TOPIC

DISPLAY TOPIC



Requested attrs:



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Not valid on z/OS.

Usage notes for DISPLAY TOPIC

1. z/OS On z/OS, the channel initiator must be running before you can display information about cluster topics, using **TYPE(CLUSTER)** or the **CLUSINFO** parameter.
2. The **TOPICSTR** parameter might contain characters that cannot be translated into printable characters when the command output is displayed.

   z/OS On z/OS, these non-printable characters are displayed as blanks.

   Multi On Multiplatforms using the runmqsc command, these non-printable characters are displayed as dots
3. You can use the following command (or synonym) as an alternative way to display these attributes.

   ```
   DISPLAY TCLUSTER
   ```

   This command produces the same output as the following command:

   ```
   DISPLAY TOPIC TYPE(CLUSTER)
   ```

If you enter the command in this way, do not use the **TYPE** parameter.

Parameter descriptions for DISPLAY TOPIC

You must specify the name of the topic definition you want to display. This name can be a specific topic name or a generic topic name. By using a generic topic name, you can display either:

- All topic definitions
- One or more topic definitions that match the specified name

(generic-topic-name)
The name of the administrative topic definition to be displayed (see Rules for naming IBM MQ objects). A trailing asterisk (*) matches all administrative topic objects with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all administrative topic objects.

WHERE
Specify a filter condition to display only those administrative topic object definitions that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this **DISPLAY** command. However, you cannot use the **CMDSCOPE**, or **QSGDISP** parameters as filter keywords.

operator
This part is used to determine whether a topic object satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this value can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE, or GE only. However, if the attribute value is one from a possible set of values on a parameter, you can use only EQ or NE.
- A generic value. This value is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

Note: z/OS On z/OS there is a 256 character limit for the filter-value of the MQSC **WHERE** clause. This limit is not in place for other platforms.

ALL Specify this parameter to display all the attributes. If this parameter is specified, any attributes that are requested specifically have no effect; all attributes are still displayed.

This is the default if you do not specify a generic name, and do not request any specific attributes.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command runs on the queue manager on which it was entered. This value is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this process is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

z/OS **QSGDISP**

Specifies the disposition of the objects for which information is to be displayed. Values are:

LIVE LIVE is the default value and displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

ALL Display information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

If there is a shared queue manager environment, and the command is being processed on the queue manager where it was issued, this option also displays information for objects defined with QSGDISP(GROUP).

If QSGDISP(ALL) is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

In a shared queue manager environment, use

```
DISPLAY TOPIC(name) CMDSCOPE(*) QSGDISP(ALL)
```

to list ALL objects matching `name` in the queue-sharing group without duplicating those objects in the shared repository.

COPY Display information only for objects defined with QSGDISP(COPY).

GROUP
Display information only for objects defined with QSGDISP(GROUP). This is allowed only if there is a shared queue manager environment.

PRIVATE
Display information only for objects defined with QSGDISP(QMGR) or QSGDISP(COPY). QSGDISP(PRIVATE) displays the same information as QSGDISP(LIVE).

QMGR
Display information only for objects defined with QSGDISP(QMGR).

QSGDISP
QSGDISP displays one of the following values:

QMGR
The object was defined with QSGDISP(QMGR).

GROUP
The object was defined with QSGDISP(GROUP).

COPY The object was defined with QSGDISP(COPY).

You cannot use QSGDISP as a filter keyword.

CLUSINFO
Requests that, in addition to information about attributes of topics defined on this queue manager, information about these and other topics in the cluster, that match the selection criteria, is displayed. In this case, there might be multiple topics with the same topic string displayed. The cluster information is obtained from the repository on this queue manager.

z/OS On z/OS, the channel initiator must be running before you can use the CLUSINFO parameter to display information about cluster topics.

CLUSTER
Limits the information displayed to topics with the specified cluster name if entered with a value in brackets. The value can be a generic name.

If you do not enter a value to qualify this parameter, it is treated as a requested parameter, and cluster name information is returned about all the topics displayed.

z/OS On z/OS, the channel initiator must be running before you can use the CLUSINFO parameter to display information about cluster topics.

TYPE Specifies the type of topics that you want to be displayed. Values are:

ALL Display all topic types, including cluster topics if you also specify CLUSINFO.

LOCAL
Display locally defined topics.

CLUSTER
Display topics that are defined in publish/subscribe clusters. Cluster attributes include:

CLUSDATE
The date on which the definition became available to the local queue manager, in the form `yyyy-mm-dd`.

CLUSQMGR
The name of the queue manager hosting the topic.

CLUSTIME
The time at which the definition became available to the local queue manager, in the form `hh.mm.ss`.

QMID
The internally generated, unique name of the queue manager hosting the topic.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

Most of the parameters are relevant for both types of topics, but parameters that are not relevant for a particular type of topic cause no output, nor is an error raised.

The following table shows the parameters that are relevant for each type of topic. There is a brief description of each parameter after the table, but for more information, see “DEFINE TOPIC” on page 703

703.

Table 115. Parameters that can be returned by the DISPLAY TOPIC command

	Local topic	Cluster topic
ALTDATE	✓	✓
ALTTIME	✓	✓
CLROUTE	✓	✓
CLSTATE		✓
CLUSDATE		✓
CLUSQMGR		✓
CLUSTER	✓	✓
CLUSTIME		✓
COMMINFO	✓	
CUSTOM	✓	✓
DEFPRTY	✓	✓
DEFPSIST	✓	✓
DEFPRESP	✓	✓
DESCR	✓	✓
DURSUB	✓	✓
MCAST	✓	
MDURMDL	✓	✓
MNDURMDL	✓	✓
NPMSGDLV	✓	✓
PMSGDLV	✓	✓
PROXYSUB	✓	✓
PUB	✓	✓
PUBSCOPE	✓	✓
QMID		✓

Table 115. Parameters that can be returned by the DISPLAY TOPIC command (continued)

	Local topic	Cluster topic
SUB	✓	✓
SUBSCOPE	✓	✓
TOPICSTR	✓	✓
TYPE	✓	✓
USEDLQ	✓	
WILDCARD	✓	✓

ALTDATE
The date on which the definition or information was last altered, in the form `yyyy-mm-dd`.

ALTTIME
The time at which the definition or information was last altered, in the form `hh.mm.ss`.

CLROUTE
The routing behavior to use for topics in the cluster defined by the **CLUSTER** parameter.

CLSTATE
The current state of this topic in the cluster defined by the **CLUSTER** parameter. The values can be as follows:

ACTIVE
The cluster topic is correctly configured and being adhered to by this queue manager.

PENDING
Only seen by a hosting queue manager, this state is reported when the topic has been created but the full repository has not yet propagated it to the cluster. This might be because the host queue manager is not connected to a full repository, or because the full repository has deemed the topic to be invalid.

INVALID
This clustered topic definition conflicts with an earlier definition in the cluster and is therefore not currently active.

ERROR
An error has occurred with respect to this topic object.

This parameter is typically used to aid diagnosis when multiple definitions of the same clustered topic are defined on different queue managers, and the definitions are not identical. See Routing for publish/subscribe clusters: Notes on behavior.

CLUSDATE
The date on which the information became available to the local queue manager, in the form `yyyy-mm-dd`.

CLUSQMGR
The name of the queue manager that hosts the topic.

CLUSTER
The name of the cluster that the topic is in.

CLUSTIME
The time at which the information became available to the local queue manager, in the form `hh.mm.ss`.

COMMINFO
The communication information object name.

CUSTOM
This attribute is reserved for the configuration of new features before separate attributes have been introduced. It can contain the values of zero or more attributes as pairs of attribute name and value in the form `NAME(VALUE)`.

DEFPRTY
Default priority of the messages published to this topic.

DEFPSIST
Default persistence of messages published to this topic.

DEFPRESP
Default put response for this topic. This attribute defines the behavior that must be used by applications when the put response type in the MQPMO options has been set to MQPMO_RESPONSE_AS_TOPIC_DEF.

DESCR
Description of this administrative topic object.

DURSUB
Determines whether the topic permits durable subscriptions to be made.

MCAST
Specifies whether the topic is enabled for multicast.

MDURMDL
The name of the model queue for durable managed subscriptions.

MNDURMDL
The name of the model queue for non-durable managed subscriptions.

NPMSGDLV
The delivery mechanism for non-persistent messages.

PMSGDLV
The delivery mechanism for persistent messages.

PROXYSUB
Determines whether a proxy subscription is forced for this subscription, even if no local subscriptions exist.

PUB Determines whether the topic is enabled for publication.

PUBSCOPE
Determines whether this queue manager propagates publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster.

QMID
The internally generated unique name of the queue manager that hosts the topic.

SUB Determines whether the topic is enabled for subscription.

SUBSCOPE
Determines whether this queue manager propagates subscriptions to queue managers as part of a hierarchy or as part of a publish/subscribe cluster.

TOPICSTR
The topic string.

TYPE Specifies whether this object is a local topic or cluster topic.

USEDLQ
Determines whether the dead-letter queue is used when publication messages cannot be delivered to their correct subscriber queue.

WILDCARD
The behavior of wildcard subscriptions with respect to this topic.

For more details of these parameters, except the **CLSTATE** parameter, see "DEFINE TOPIC" on page 703.

Related reference:

"DISPLAY TPSTATUS"
Use the MQSC command **DISPLAY TPSTATUS** to display the status of one or more topics in a topic tree.

Related information:

Displaying administrative topic object attributes

Changing administrative topic attributes

DISPLAY TPSTATUS:

Use the MQSC command **DISPLAY TPSTATUS** to display the status of one or more topics in a topic tree.

Using MQSC commands

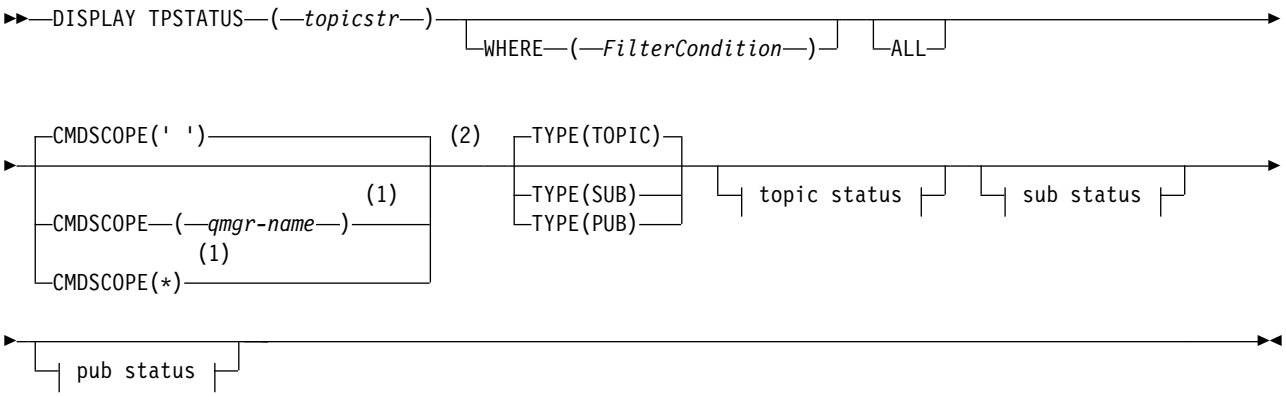
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

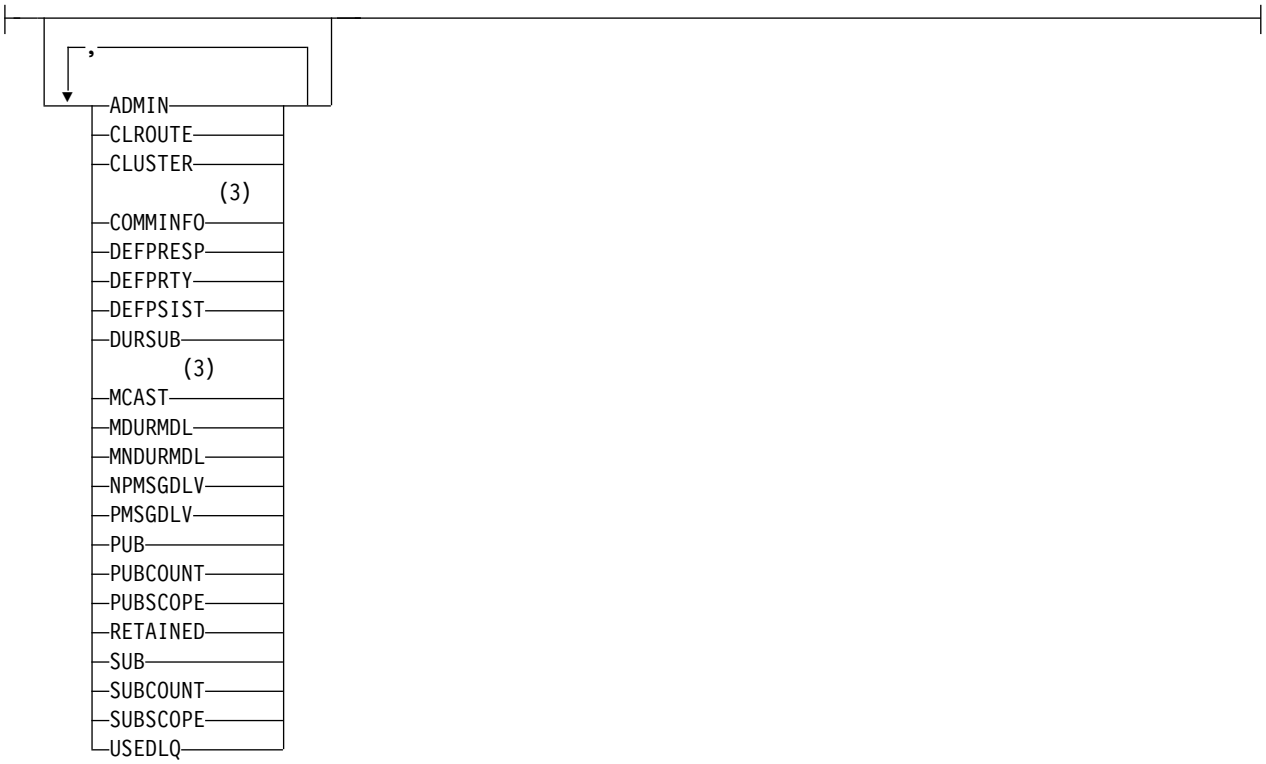
- Syntax diagram
- "Usage notes for DISPLAY TPSTATUS" on page 952
- "Parameter descriptions for DISPLAY TPSTATUS" on page 952
- "Topic status parameters" on page 954
- "Sub status parameters" on page 956
- "Pub status parameters" on page 957

Synonym: DIS TPS

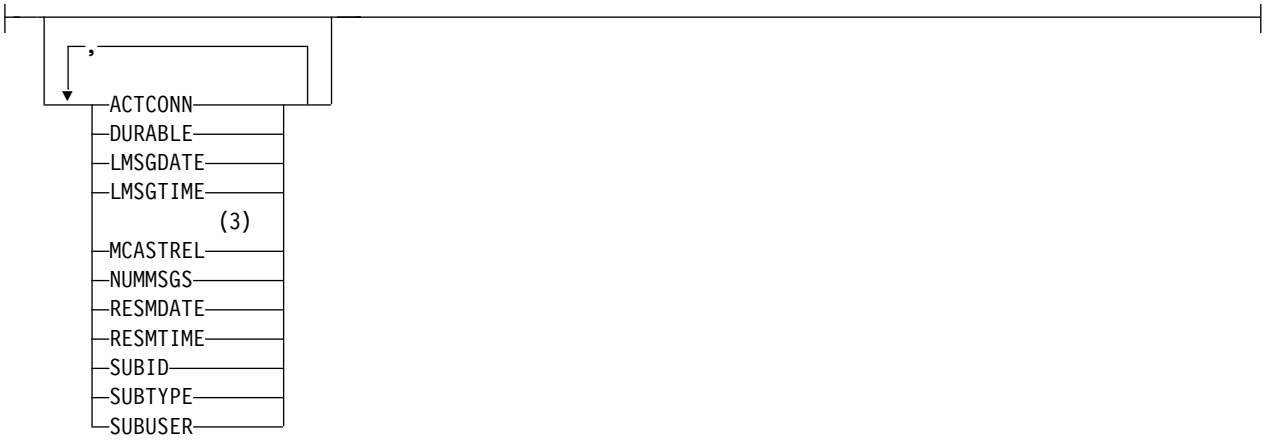
DISPLAY TPSTATUS



Topic status:



Sub status:



Pub status:

Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Not valid on z/OS.

Usage notes for DISPLAY TPSTATUS

1. The TOPICSTR parameter might contain characters that cannot be translated into printable characters when the command output is displayed.
   - Multi On Multiplatforms using the **runmqsc** command, these non-printable characters are displayed as dots.
   - z/OS On z/OS, these non-printable characters are displayed as blanks.
2. The topic-string input parameter on this command must match the topic you want to act on. Keep the character strings in your topic strings as characters that can be used from the location issuing the command. If you issue commands using MQSC, you have fewer characters available to you than if you are using an application that submits PCF messages, such as the IBM MQ Explorer.

Parameter descriptions for DISPLAY TPSTATUS

The **DISPLAY TPSTATUS** command requires a topic string value to determine which topic nodes the command returns.

topicstr)

The value of the topic string for which you want to display status information. You cannot specify the name of an IBM MQ topic object.

The topic string can have one of the following values:

- A specific topic string value. For example, `DIS TPS('Sports/Football')` returns just the 'Sports/Football' node.
- A topic string containing a "+" wildcard character. For example, `DIS TPS('Sports/Football/+')` returns all direct child nodes of the 'Sports/Football' node.
- A topic string containing a "#" wildcard character. For example, `DIS TPS('Sports/Football/#')` returns the 'Sports/Football' node and all its descendant nodes.
- A topic string containing more than one wildcard. For example, `DIS TPS('Sports/+/Teams/#')` returns any direct child node of 'Sports' that also has a 'teams' child, with all descendants of the latter nodes.

The **DISPLAY TPSTATUS** command does not support the '*' wildcard. For more information about using wildcards, see the related topic.

- To return a list of all root-level topics, use `DIS TPS('+')`
- To return a list of all topics in the topic tree, use `DIS TPS('#')`, but note that this command might return a large amount of data.

- To filter the list of topics returned, use the **WHERE** parameter. For example, `DIS TPS('Sports/Football/+') WHERE(TOPICSTR LK 'Sports/Football/L*')` returns all direct child nodes of the 'Sports/Football' node, that begin with the letter “L”.

WHERE

Specifies a filter condition to display only those administrative topic definitions that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword

Except for the CMDSCOPE parameter, any parameter that you can use with this DISPLAY command.

operator

Determines whether a topic string satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *topicstr*

NL Does not match a generic string that you provide as a *topicstr*

filter-value

The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this value can be:

- An explicit value that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE, or GE only. However, if the attribute value is one from a possible set of values on a parameter, you can use only EQ or NE.
- A generic value. This value is a character string with an asterisk at the end, for example ABC*. If the operator is LK, the command lists all topic nodes that begin with the string (ABC in the example). If the operator is NL, the command lists all topic nodes that do not begin with the string.

  You cannot use a generic *filter-value* for parameters with numeric values or with one of a set of values.

ALL Use this parameter to display all attributes.

If this parameter is specified, any attributes that you request specifically have no effect; the command displays all attributes.

This parameter is the default parameter if you do not specify a generic name, and do not request any specific attributes.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This value is the default value.

qmgr-name

The command runs on the named queue manager, if the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which you enter the command, but only if you are using a queue-sharing group environment and the command server is enabled.

***** The command runs on the local queue manager and on every active queue manager in the queue-sharing group. The effect of this option is equivalent to entering the command on every queue manager in the queue-sharing group.

TYPE

TOPIC
The command displays status information relating to each topic node, which is the default if you do not provide a **TYPE** parameter.

PUB The command displays status information relating to applications that have topic nodes open for publish.

SUB The command displays status information relating to applications that subscribe to the topic node or nodes. The subscribers that the command returns are not necessarily the subscribers that would receive a message published to this topic node. The value of **SelectionString** or **SubLevel** determines which subscribers receive such messages.

Topic status parameters

Topic status parameters define the data that the command displays. You can specify these parameters in any order but must not specify the same parameter more than once.

Topic objects can be defined with attributes that have a value of *ASPARENT*. Topic status shows the resolved values that result in finding the setting of the closest parent administrative topic object in the topic tree, and so will never display a value of *ASPARENT*.

ADMIN
If the topic node is an admin-node, the command displays the associated topic object name containing the node configuration. If the field is not an admin-node the command displays a blank.

CLROUTE
The routing behavior to use for topics in the cluster defined by the **CLUSTER** parameter. The values can be as follows:

DIRECT
A publication on this topic string, originating from this queue manager, is sent direct to any queue manager in the cluster with a matching subscription.

TOPICHOST
A publication on this topic string, originating from this queue manager, is sent to one of the queue managers in the cluster that hosts a definition of the corresponding clustered topic object, and from there to any queue manager in the cluster with a matching subscription.

NONE This topic node is not clustered.

CLUSTER
The name of the cluster to which this topic belongs.

' ' This topic does not belong to a cluster. Publications and subscriptions for this topic are not propagated to publish/subscribe cluster-connected queue managers.

COMMINFO
Displays the resolved value of the name of the communication information object to be used for ths topic node.

DEFPRESP
Displays the resolved default put response of messages published to the topic. The value can be *SYNC* or *ASYNC*

DEFPRTY
Displays the resolved default priority of messages published to the topic.

DEFPSIST
Displays the resolved default persistence for this topic string. The value can be *YES* or *NO*.

DURSUB
Displays the resolved value that shows whether applications can make durable subscriptions. The value can be *YES* or *NO*.

MCAST
Displays the resolved value that shows whether the topic could be transmittable via multicast or not. The value can be *ENABLED*, *DISABLED*, or *ONLY*.

MDURMDL
Displays the resolved value of the name of the model queue to be used for durable subscriptions.

MNDURMDL
Displays the resolved value of the name of the model queue used for non-durable subscriptions.

NPMSGDLV
Displays the resolved value for the delivery mechanism for non-persistent messages published to this topic. The value can be *ALL*, *ALLDUR*, or *ALLAVAIL*.

PMSGDLV
Displays the resolved value for the delivery mechanism for persistent messages published to this topic. The value can be *ALL*, *ALLDUR*, or *ALLAVAIL*.

PUB Displays the resolved value that shows whether publications are allowed for this topic. The values can be *ENABLED* or *DISABLED*.

PUBCOUNT
Displays the number of handles that are open for publish on this topic node.

PUBSCOPE
Determines whether this queue manager propagates publications, for this topic node, to other queue managers as part of a hierarchy or a cluster, or whether it restricts them to only subscriptions defined on the local queue manager. The value can be *QMGR* or *ALL*.

RETAINED
Displays whether there is a retained publication associated with this topic. The value can be *YES* or *NO*.

SUB Displays the resolved value that shows whether subscriptions are allowed for this topic. The values can be *ENABLED* or *DISABLED*.

SUBCOUNT
Displays the number of subscribers to this topic node, including durable subscribers that are not currently connected.

SUBSCOPE
Determines whether this queue manager propagates subscriptions, for this topic node, to other queue managers as part of a cluster or hierarchy, or whether it restricts the subscriptions to only the local queue manager. The value can be *QMGR* or *ALL*.

USEDLQ
Determines whether the dead-letter queue is used when publication messages cannot be delivered to their correct subscriber queue. The value can be *YES* or *NO*.

Sub status parameters

Sub status parameters define the data that the command displays. You can specify these parameters in any order but must not specify the same parameter more than once.

ACTCONN
Detects local publications, returning the currently active ConnectionId (CONNID) that opened this subscription.

DURABLE
Indicates whether a durable subscription is not deleted when the creating application closes its subscription handle, and persists over queue manager restart. The value can be *YES* or *NO*.

LMSGDATE
The date on which an MQPUT call last sent a message to this subscription. The MQPUT call updates the date field only when the call successfully puts a message to the destination specified by this subscription. An MQSUBRQ call causes an update to this value.

LMSGTIME
The time at which an MQPUT call last sent a message to this subscription. The MQPUT call updates the time field only when the call successfully puts a message to the destination specified by this subscription. An MQSUBRQ call causes an update to this value.

MCASTREL
Indicator of the reliability of the multicast messages.

The values are expressed as a percentage. A value of 100 indicates that all messages are being delivered without problems. A value less than 100 indicates that some of the messages are experiencing network issues. To determine the nature of these issues you can enable event message generation, use the **COMMEV** parameter of the COMMINFO objects, and examine the generated event messages.

The following two values are returned:

- The first value is based on recent activity over a short period.
- The second value is based on activity over a longer period.

If no measurement is available the values are shown as blanks.

NUMMSGS
Number of messages put to the destination specified by this subscription. An MQSUBRQ call causes an update to this value.

RESMDATE
Date of the most recent MQSUB call that connected to this subscription.

RESMTIME
Time of the most recent MQSUB call that connected to this subscription.

SUBID
An all time unique identifier for this subscription, assigned by the queue manager. The format of **SUBID** matches that of a CorrelId. For durable subscriptions, the command returns the **SUBID** even if the subscriber is not currently connected to the queue manager.

SUBTYPE
The type of subscription, indicating how it was created. The value can be *ADMIN*, *API*, or *PROXY*.

SUBUSER
The user ID that owns this subscription, which can be either the user ID associated with the creator of the subscription or, if subscription takeover is permitted, the user ID that last took over the subscription.

Pub status parameters

Pub status parameters define the data that the command displays. You can specify these parameters in any order but must not specify the same parameter more than once.

ACTCONN
The currently active ConnectionId (CONNID) associated with the handle that has this topic node open for publish.

LPUBDATE
The date on which this publisher last sent a message.

LPUBTIME
The time at which this publisher last sent a message.

MCASTREL
Indicator of the reliability of the multicast messages.

The values are expressed as a percentage. A value of 100 indicates that all messages are being delivered without problems. A value less than 100 indicates that some of the messages are experiencing network issues. To determine the nature of these issues you can enable event message generation, using the **COMMEV** parameter of the COMMINFO objects, and examine the generated event messages.

The following two values are returned:

- The first value is based on recent activity over a short period.
- The second value is based on activity over a longer period.

If no measurement is available the values are shown as blanks.

NUMPUBS
Number of publishes by this publisher. This value records the actual number of publishes, not the total number of messages published to all subscribers.

Related reference:

"DISPLAY TOPIC" on page 942
Use the MQSC command **DISPLAY TOPIC** to display the attributes of one or more IBM MQ topic objects of any type.

Related information:

Displaying administrative topic object attributes

DISPLAY TRACE on z/OS: z/OS

Use the MQSC command DISPLAY TRACE to display a list of active traces.

Using MQSC commands

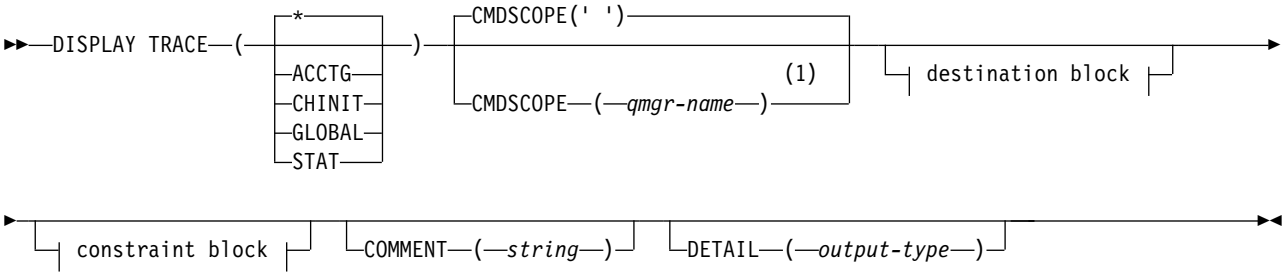
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 12CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for DISPLAY TRACE" on page 958
- "Destination block" on page 959
- "Constraint block" on page 959

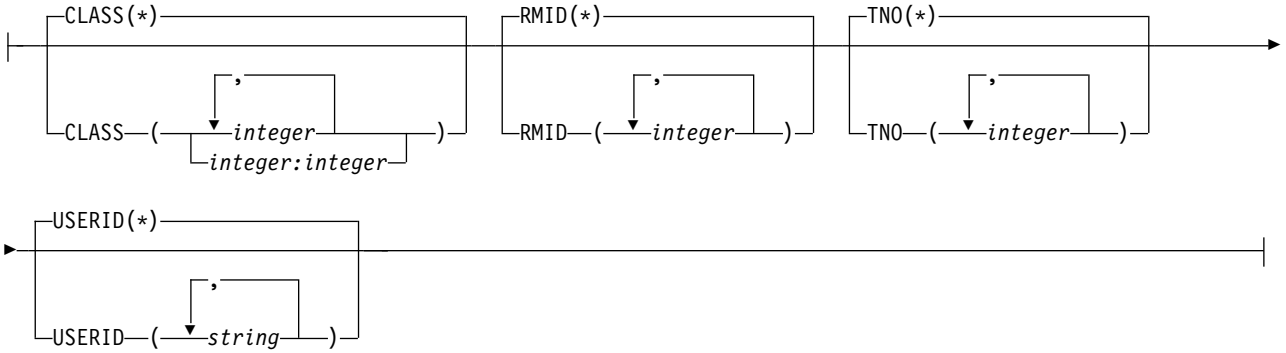
Synonym: DIS TRACE

DISPLAY TRACE



Destination block:

Constraint block:



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Parameter descriptions for DISPLAY TRACE

All parameters are optional. Each option that is used limits the effect of the command to active traces that were started using the same option, either explicitly or by default, with exactly the same parameter values.

***** Does not limit the list of traces. This is the default. The CLASS option cannot be used with DISPLAY TRACE(*).

Each remaining parameter in this section limits the list to traces of the corresponding type:

ACCTG
Accounting data (the synonym is A)

CHINIT
Service data from the channel initiator. The synonym is CHI or DQM.

GLOBAL
Service data from the entire queue manager except the channel initiator. The synonym is G.

STAT Statistical data (the synonym is S)

COMMENT(*string*)
Specifies a comment. This does not appear in the display, but it might be recorded in trace output.

DETAIL(*output-type*)
This parameter is ignored; it is retained only for compatibility with earlier releases.

Possible values for *output-type* are *, 1, or 2.

CMDSCOPE
This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE cannot be used for commands issued from the first initialization input data set CSQINP1.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

Destination block

DEST Limits the list to traces started for particular destinations. More than one value can be specified, but do not use the same value twice. If no value is specified, the list is not limited.

Possible values and their meanings are:

GTF The Generalized Trace Facility

RES A wraparound table residing in the ECSA (extended common service area)

SMF The System Management Facility

SRV A serviceability routine designed for IBM for problem diagnosis

Constraint block

CLASS(*integer*)
Limits the list to traces started for particular classes. See "START TRACE on z/OS" on page 1041 for a list of allowed classes.

The default is CLASS(*), which does not limit the list.

RMID(*integer*)
Limits the list to traces started for particular resource managers. See "START TRACE on z/OS" on page 1041 for a list of allowed resource manager identifiers. Do not use this option with the STAT or CHINIT trace type.

The default is RMID(*), which does not limit the list.

TNO(*integer*)
Limits the list to particular traces, identified by their trace number (0 to 32). Up to 8 trace numbers can be used. If more than one number is used, only one value for USERID can be used. The default is TNO(*), which does not limit the list.

0 is the trace that the channel initiator can start automatically. Traces 1 to 32 are those for queue manager or the channel initiator that can be started automatically by the queue manager, or manually, using the START TRACE command.

USERID(*string*)
Limits the list to traces started for particular user IDs. Up to 8 user IDs can be used. If more than one user ID is used, only one value can be used for TNO. Do not use this option with STAT. The default is USERID(*), which does not limit the list.

DISPLAY USAGE on z/OS: z/OS

Use the MQSC command DISPLAY USAGE to display information about the current state of a page set, to display information about the log data sets, or to display information about the shared message data sets.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

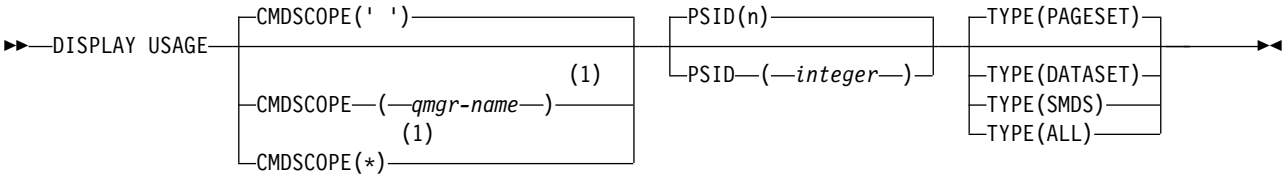
You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

Attention: From IBM MQ Version 8.0, the output of the DISPLAY USAGE command includes message CSQI065I `+MP11 Buffer pool attributes`, instead of message CSQP001I `+MG11 Buffer pool 0 has 25000 buffers`.

- Syntax diagram
- "Parameter descriptions for DISPLAY USAGE"

Synonym: DIS USAGE

DISPLAY USAGE



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Parameter descriptions for DISPLAY USAGE

CMDSCOPE
This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

'' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

PSID(*integer* **)**
The page-set identifier. This is optional.

This is a number, in the range 00 through 99. An asterisk (*) on its own specifies all page set identifiers.

The command fails if PSID has been specified together with TYPE(DATASET), or TYPE(SMDS).

If the command is running at the same time as an ALTER BUFFPOOL command the buffer pool attributes might not be entirely consistent. For example, the value of the location parameter might be BELOW, but the number of available buffers value might be more than can fit below the bar. If this occurs, run the display command again when the ALTER BUFFPOOL command has completed.

TYPE Defines the type of information to be displayed. Values are:

PAGESET
Display page set and buffer pool information. This is the default.

DATASET
Display data set information for log data sets. This returns messages containing 44-character data set names for the following:

- The log data set containing the BEGIN_UR record for the oldest incomplete unit of work for this queue manager, or if there are no incomplete units of work, the log data set containing the current highest written RBA.
- The log data set containing the oldest restart_RBA of any pageset owned by this queue manager.
- The log data set with a timestamp range that includes the timestamp of the last successful backup of any application structure known within the queue-sharing group.

SMDS
Display data set space usage information and buffer pool information for shared message data sets owned by this queue manager. Space usage information is only available when the data set is open. Buffer pool information is only available when the queue manager is connected to the structure. For more information about the displayed information, see the descriptions of messages CSQE280I and CSQE285I.

ALL
Display page set, data set, and SMDS information.

Note: This command is issued internally by IBM MQ:

- During queue manager shutdown so that the restart RBA is recorded on the z/OS console log.
- At queue manager startup so that page set information can be recorded.
- When DEFINE PSID is used to dynamically define the first page set in the queue manager that uses the buffer pool specified on the DEFINE PSID command.

Related reference:
"ALTER PSID on z/OS" on page 457
Use the MQSC command **ALTER PSID** to change the expansion method for a page set.

MOVE QLOCAL on z/OS: z/OS

Use the MQSC command MOVE QLOCAL to move all the messages from one local queue to another.

Using MQSC commands

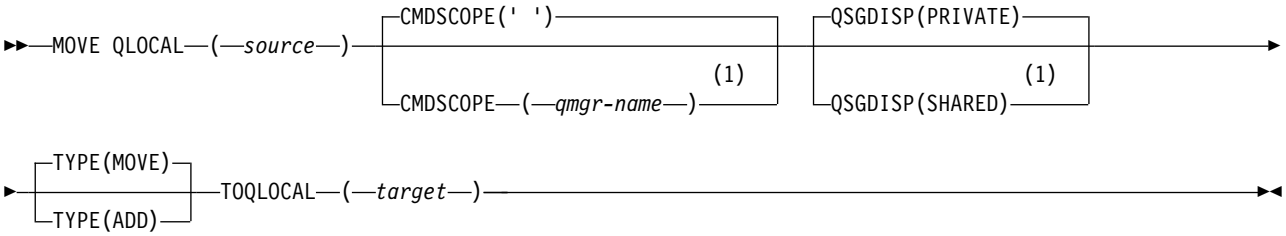
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for MOVE QLOCAL"
- "Parameter descriptions for MOVE QLOCAL" on page 963

Synonym: MOVE QL

MOVE QLOCAL



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Usage notes for MOVE QLOCAL

1. A typical use of the MOVE QLOCAL command is to move messages from a private queue to a shared queue when you are setting up a queue-sharing group environment.
2. The MOVE QLOCAL command ***moves*** messages; it does not copy them.
3. The MOVE QLOCAL command moves messages in a similar way to an application performing successive MQGET and MQPUT calls. However, the MOVE QLOCAL command does not physically delete logically-expired messages and, therefore, no expiration reports are generated.
4. The priority, context, and persistence of each message are not changed.
5. The command performs no data conversion and calls no exits.
6. Confirm-on-delivery (COD) report messages are not generated but confirm-on-arrival (COA) report messages are. This means that more than one COA report message can be generated for a message.
7. The MOVE QLOCAL command transfers the messages in batches. At COMMIT time, if the trigger conditions are met, trigger messages are produced. This might be at the end of the move operation.

   Note: Before the transfer of messages begins, this command verifies that the number of messages on the source queue, when added to the number of messages on the target queue, does not exceed MAXDEPTH on the target queue.

If the MAXDEPTH of the target queue were to be exceeded, no messages are moved.

8. The MOVE QLOCAL command can change the sequence in which messages can be retrieved. The sequence remains unchanged only if:
   - You specify `TYPE(MOVE)` and
   - The MSGDLVSQ parameter of the source and target queues is the same.
9. Messages are moved within one or more syncpoints. The number of messages in each syncpoint is determined by the queue manager.
10. If anything prevents the moving of one or more messages, the command stops processing. This can mean that some messages have already been moved, while others remain on the source queue. Some of the reasons that prevent a message being moved are:
    - The target queue is full.
    - The message is too long for the target queue.
    - The message is persistent, but the target queue cannot store persistent messages.
    - The page set is full.

Parameter descriptions for MOVE QLOCAL

You must specify the names of two local queues: the one you want to move messages from (the source queue) and the one you want to move the messages to (the target queue).

source The name of the local queue from which messages are moved. The name must be defined to the local queue manager.

The command fails if the queue contains uncommitted messages.

If an application has this queue open, or has open a queue that eventually resolves to this queue, the command fails. For example, the command fails if this queue is a transmission queue, and any queue that is, or resolves to, a remote queue that references this transmission queue, is open.

An application can open this queue while the command is in progress but the application waits until the command has completed.

CMDSCOPE

This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

QSGDISP

Specifies the disposition of the source queue.

PRIVATE

The queue is defined with QSGDISP(QMGR) or QSGDISP(COPY). This is the default value.

SHARED

The queue is defined with QSGDISP(SHARED). This is valid only in a queue-sharing group environment.

TYPE Specifies how the messages are moved.

MOVE
Move the messages from the source queue to the empty target queue.

The command fails if the target queue already contains one or more messages. The messages are deleted from the source queue. This is the default value.

ADD Move the messages from the source queue and add them to any messages already on the target queue.

The messages are deleted from the source queue.

target The name of the local queue to which messages are moved. The name must be defined to the local queue manager.

The name of the target queue can be the same as that of the source queue only if the queue exists as both a shared and a private queue. In this case, the command moves messages to the queue that has the opposite disposition (shared or private) from that specified for the source queue on the QSGDISP parameter.

If an application has this queue open, or has open a queue that eventually resolves to this queue, the command fails. The command also fails if this queue is a transmission queue, and any queue that is, or resolves to, a remote queue that references this transmission queue, is open.

No application can open this queue while the command is in progress.

If you specify TYPE(MOVE), the command fails if the target queue already contains one or more messages.

The DEFTYPE, HARDENBO, and USAGE parameters of the target queue must be the same as those of the source queue.

PING CHANNEL:

Use the MQSC command PING CHANNEL to test a channel by sending data as a special message to the remote queue manager, and checking that the data is returned. The data is generated by the local queue manager.

Using MQSC commands

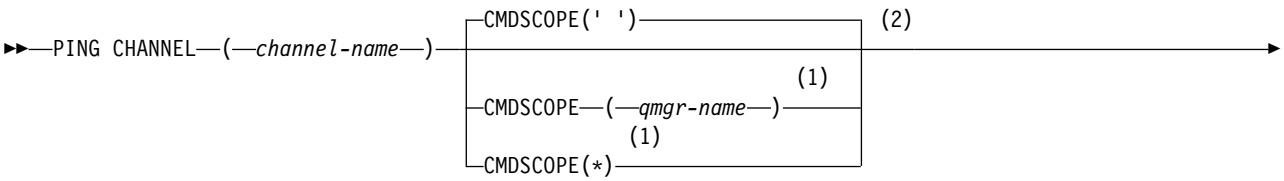
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

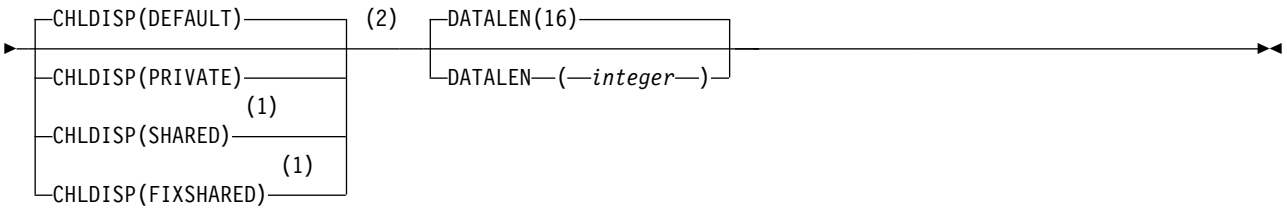
z/OS You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes" on page 965
- "Parameter descriptions for PING CHANNEL" on page 965

Synonym: PING CHL

PING CHANNEL





Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes

1. z/OS On z/OS, the command server and the channel initiator must be running.
2. Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel. If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the channel that was last added to the local queue manager's repository.
3. This command can be used only for sender (SDR), server (SVR), and cluster-sender (CLUSSDR) channels (including those that have been defined automatically). It is not valid if the channel is running; however, it is valid if the channel is stopped or in retry mode.

Parameter descriptions for PING CHANNEL

(channel-name)
The name of the channel to be tested. This is required.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

If CHLDISP is set to SHARED, CMDSCOPE must be blank or the local queue manager.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

Note: The '*' option is not permitted if CHLDISP is FIXSHARED.

z/OS **CHLDISP**
This parameter applies to z/OS only and can take the values of:

- DEFAULT
- PRIVATE
- SHARED
- FIXSHARED

If this parameter is omitted, then the DEFAULT value applies. This is the value of the default channel disposition attribute, DEFCDISP, of the channel object.

In conjunction with the various values of the CMDSCOPE parameter, this parameter controls two types of channel:

SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of SHARED.

PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than SHARED.

Note: This disposition is **not** related to the disposition set by the disposition of the queue-sharing group of the channel definition.

The combination of the CHLDISP and CMDSCOPE parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.
- On the most suitable queue manager in the group, determined automatically by the queue manager itself.

The various combinations of CHLDISP and CMDSCOPE are summarized in the following table.

Table 116. CHLDISP and CMDSCOPE for PING CHANNEL

CHLDISP	**CMDSCOPE( ) or CMDSCOPE (local-qmgr)**	**CMDSCOPE (qmgr-name)**	**CMDSCOPE(*)**
PRIVATE	Ping private channel on the local queue manager	Ping private channel on the named queue manager	Ping private channel on all active queue managers
SHARED	Ping a shared channel on the most suitable queue manager in the group This might automatically generate a command using CMDSCOPE and send it to the appropriate queue manager. If there is no definition for the channel on the queue manager to which the command is sent, or if the definition is unsuitable for the command, the command fails. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is actually run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted	Not permitted
FIXSHARED	Ping a shared channel on the local queue manager	Ping a shared channel on the named queue manager	Not permitted

DATALEN(*integer*)
The length of the data, in the range 16 through 32 768. This is optional.

PING QMGR on Multiplatforms: Multi

Use the MQSC command PING QMGR to test whether the queue manager is responsive to commands.

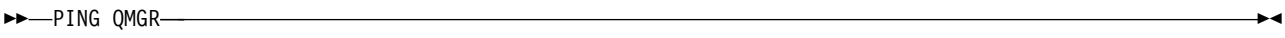
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Usage notes"

Synonym: PING QMGR

PING QMGR



Usage notes

If commands are issued to the queue manager by sending messages to the command server queue, this command causes a special message to be sent to it, consisting of a command header only, and checking that a positive reply is returned.

PURGE CHANNEL: AIX Linux Windows

Use the MQSC command PURGE CHANNEL to stop and purge a telemetry or AMQP channel. Purging a telemetry or AMQP channel disconnects all the MQTT or AMQP clients connected to it, cleans up the state of the MQTT or AMQP clients, and stops the telemetry or AMQP channel. Cleaning the state of a client deletes all the pending publications and removes all the subscriptions from the client.

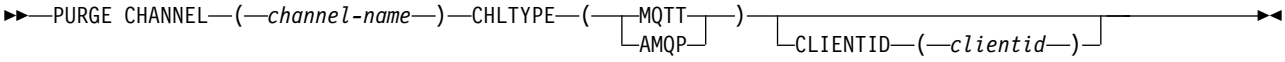
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Parameter descriptions for PURGE CHANNEL"

Synonym: None

PURGE CHANNEL



Parameter descriptions for PURGE CHANNEL

(channel name)
The name of the telemetry or AMQP channel to be stopped and purged. This parameter is required.

CHLTYPE (MQTT)

Channel type. This parameter is required. It must follow immediately after the (channel-name) parameter z/OS on all platforms except z/OS,

Multi On Multiplatforms, the value must be either `MQTT` or `AMQP`

z/OS On z/OS, the value must be `MQTT`

CLIENTID *(string)*

Client identifier. The client identifier is a 23 byte string that identifies an MQ Telemetry Transport or AMQP client. When the PURGE CHANNEL command specifies a CLIENTID, only the connection for the specified client identifier is purged. If the CLIENTID is not specified, all the connections on the channel are purged.

RECOVER CFSTRUCT on z/OS: z/OS

Use the MQSC command RECOVER CFSTRUCT to initiate recovery of CF application structures and associated shared message data sets. This command is valid only when the queue manager is a member of a queue-sharing group.

Using MQSC commands

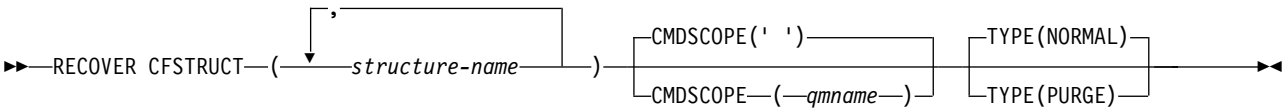
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources `CR`. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for RECOVER CFSTRUCT"
- "Keyword and parameter descriptions for RECOVER CFSTRUCT" on page 969

Synonym: REC CFSTRUCT

RECOVER CFSTRUCT



Usage notes for RECOVER CFSTRUCT

- The command fails if neither the specified application structure nor its associated shared message data sets are flagged as being in a FAILED state.
- If a data set is marked as FAILED but the corresponding structure is not, then the **RECOVER CFSTRUCT** command changes the structure state to FAILED, deleting the contents to perform recovery. This action deletes all nonpersistent messages stored in the structure and makes the structure unavailable until recovery is complete.
- For a structure with associated shared message data sets, the **RECOVER CFSTRUCT** command recovers the structure plus the offloaded message data for any data sets which are either already marked as FAILED or found to be empty or invalid when opened by recovery processing. Any data sets which are marked as ACTIVE and have valid headers are assumed not to require recovery.
- When recovery processing completes normally, all associated shared message data sets for the recovered structures (including data sets which did not need recovery) are marked as RECOVERED, indicating that the space map needs to be rebuilt.

- Following recovery, space map rebuild processing is performed for each affected data set, to map the space occupied by the recovered message data (ignoring any existing messages which were nonpersistent or backed out). When the space map has been rebuilt for each data set, it is marked as ACTIVE again.
- The command fails if any one of the specified structure names is not defined in the CFRM policy data set.
- The recovery process is both I/O and processor intensive, and can only run on a single z/OS image. It should therefore be run on the most powerful or least busy system in the queue-sharing group.
- The most likely failure is the loss of a complete CF and hence the simultaneous loss of all the application structures therein. If backup date and times are similar for each failed application structure, it is more efficient to recover them in a single **RECOVER CFSTRUCT** command.
- This command fails if any of the specified CF structures is defined with either a CFLEVEL of less than 3, or with RECOVER set to NO.
- To use TYPE(NORMAL), you must have taken a backup of the CF structures, using the **BACKUP CFSTRUCT** command.
- If backups of the requested CF structures have not been taken recently, using TYPE(NORMAL) may take a considerable amount of time.
- If a backup of the CF structure, or a required archive log, is not available, you can recover to an empty CF structure using TYPE(PURGE).
- The command **RECOVER CFSTRUCT(CSQSYSAPPL) TYPE(PURGE)** is prohibited. This is to prevent the accidental loss of queue manager internal objects.

Keyword and parameter descriptions for RECOVER CFSTRUCT

CFSTRUCT(*structure-names ...* **)**
Specify list of names of up to 256 structure names for which the coupling facility application structures are to be recovered, along with any associated shared message data sets which also need recovery. If resources for more than one structure need to be recovered, it is more efficient to recover them at the same time.

CMDSCOPE
This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

' '
The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

TYPE Specifies which variant of the **RECOVER** command is to be issued. Values are:

NORMAL
Perform true recovery by restoring data from a backup taken using the BACKUP CFSTRUCT command and reapplying logged changes since that time. Any nonpersistent messages are discarded.

This is the default.

PURGE
Reset the structure and associated shared message data sets to an empty state. This can be used to restore a working state when no backup is available, but results in the loss of all affected messages.

REFRESH CLUSTER:

Use the MQSC command REFRESH CLUSTER to discard all locally held cluster information and force it to be rebuilt. The command also processes any autodefined channels that are in doubt. After the command completes processing, you can perform a "cold-start" on the cluster.

Using MQSC commands

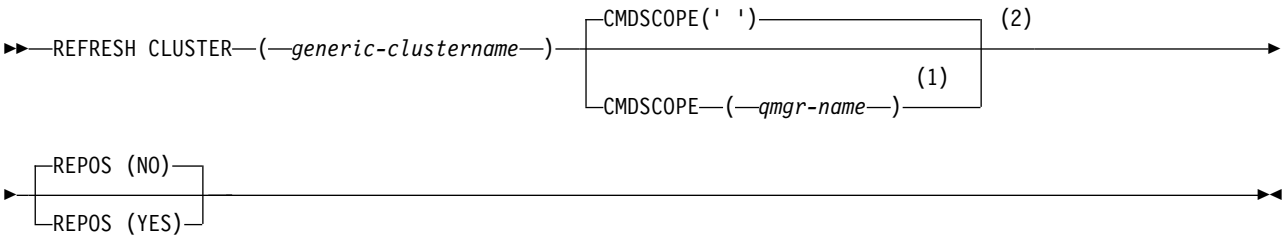
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for REFRESH CLUSTER"
- "Parameter descriptions for REFRESH CLUSTER" on page 972

Synonym: REF CLUSTER

REFRESH CLUSTER



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes for REFRESH CLUSTER

1. Issuing **REFRESH CLUSTER** is disruptive to the cluster. It might make cluster objects invisible for a short time until the **REFRESH CLUSTER** processing completes. Specifically, if an application is using a queue that it opened with MQOO_BIND_NOT_FIXED, it might receive the return code MQRC_NO_DESTINATIONS_AVAILABLE. If an application is publishing or subscribing on a cluster topic, that topic might become temporarily unavailable. The unavailability results in a break in the publication stream until the **REFRESH CLUSTER** command completes. If the command is issued on a full repository queue manager, **REFRESH CLUSTER** might make a large volume of messages flow.
2. For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.
3. Quiesce all publish/subscribe applications before issuing the **REFRESH CLUSTER** command, because issuing this command in a publish/subscribe cluster disrupts delivery of publications to and from other queue managers in the cluster, and might result in proxy subscriptions from other queue managers being canceled. If this happens, resynchronize the proxy subscriptions after the cluster has refreshed, and keep all publish/subscribe applications quiesced until after the proxy subscriptions have been resynchronized. See REFRESH CLUSTER considerations for publish/subscribe clusters.

4. When the command returns control to the user, it does not signify the command has completed. Activity on SYSTEM.CLUSTER.COMMAND.QUEUE indicates the command is still processing. See also the REFRESH CLUSTER step in Checking that async commands for distributed networks have finished.
5. If cluster-sender channels are running at the time **REFRESH CLUSTER** is issued, the refresh might not complete until the channels stop and restart. To hasten completion, stop all cluster-sender channels for the cluster before you run the **REFRESH CLUSTER** command. During the processing of the **REFRESH CLUSTER** command, if the channel is not in doubt, the channel state might be re-created.
6. If you select REPOS(YES), check that all cluster-sender channels in the relevant cluster are inactive or stopped before you issue the **REFRESH CLUSTER** command.

   If cluster-sender channels are running at the time you run the **REFRESH CLUSTER** REPOS(YES) command, those cluster-sender channels are ended during the operation and left in an INACTIVE state after the operation completes. Alternatively, you can force the channels to stop using the STOP CHANNEL command with MODE(FORCE).

   Stopping the channels ensures that the refresh can remove the channel state, and that the channel runs with the refreshed version after the refresh completes. If the state of a channel cannot be deleted, its state is not renewed after the refresh. If a channel was stopped, it does not automatically restart. The channel state cannot be deleted if the channel is in doubt, or because it is also running as part of another cluster.

   If you choose the option REPOS(YES) on full repository queue manager, you must alter it to be a partial repository. If it is the sole working repository in the cluster, the result is that there is no full repository left in the cluster. After the queue manager is refreshed, and restored to its status of a full repository, you must refresh the other partial repositories to restore a working cluster.

   If it is not the sole remaining repository, you do not need to refresh the partial repositories manually. Another working full repository in the cluster informs the other members of the cluster that the full repository running the **REFRESH CLUSTER** command resumed its role as a full repository.
7. It is not normally necessary to issue a **REFRESH CLUSTER** command except in one of the following circumstances:
   - Messages were removed from either the SYSTEM.CLUSTER.COMMAND.QUEUE, or from another a cluster transmission queue, where the destination queue is SYSTEM.CLUSTER.COMMAND.QUEUE on the queue manager in question.
   - Issuing a **REFRESH CLUSTER** command is recommended by IBM Service.
   - The CLUSRCVR channels were removed from a cluster, or their CONNAME s were altered on two or more full repository queue managers while they could not communicate.
   - The same name was used for a CLUSRCVR channel on more than one queue manager in a cluster. As a result, messages destined for one of the queue managers were delivered to another. In this case, remove the duplicates, and run a **REFRESH CLUSTER** command on the single remaining queue manager with a CLUSRCVR definition.
   - RESET CLUSTER ACTION(FORCEREMOVE) was issued in error.
   - The queue manager was restarted from an earlier point in time than it finished last time it was used; for example, by restoring backed up data.
8. Issuing **REFRESH CLUSTER** does not correct mistakes in cluster definitions, nor is it necessary to issue the command after such mistakes are corrected.
9. During **REFRESH CLUSTER** processing, the queue manager generates the message AMQ9875 followed by the message AMQ9442 or AMQ9404. The queue manager might also generate the message AMQ9420. If the cluster functionality is not affected, the message AMQ9420 can be ignored.
10. z/OS On z/OS, the command fails if the channel initiator is not started.
11. z/OS On z/OS, any errors are reported to the console on the system where the channel initiator is running. They are not reported to the system that issued the command.

Parameter descriptions for REFRESH CLUSTER

(*generic-clustername*)

The name of the cluster to be refreshed. Alternatively *generic-clustername* can be specified as "*". If "*" is specified, the queue manager is refreshed in all the clusters that it is a member of. If used with REPOS(YES), this forces the queue manager to restart its search for full repositories from the information in the local CLUSSDR definitions. It restarts its search, even if the CLUSSDR definitions connect the queue manager to several clusters.

The *generic-clustername* parameter is required.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

'' The command runs on the queue manager on which it was entered. '' is the default value.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered. If you do so, you must be using a queue-sharing group environment and the command server must be enabled.

REPOS

Specifies whether objects representing full repository cluster queue managers are also refreshed.

NO The queue manager retains knowledge of all cluster queue manager and cluster queues marked as locally defined. It also retains knowledge of all cluster queue managers that are marked as full repositories. In addition, if the queue manager is a full repository for the cluster, it retains knowledge of the other cluster queue managers in the cluster. Everything else is removed from the local copy of the repository and rebuilt from the other full repositories in the cluster. Cluster channels are not stopped if REPOS(NO) is used. A full repository uses its CLUSSDR channels to inform the rest of the cluster that it completed its refresh.

NO is the default.

YES Specifies that in addition to the REPOS(NO) behavior, objects representing full repository cluster queue managers are also refreshed. The REPOS(YES) option must not be used if the queue manager is itself a full repository. If it is a full repository, you must first alter it so that it is not a full repository for the cluster in question. The full repository location is recovered from the manually defined CLUSSDR definitions. After the refresh with REPOS(YES) is issued, the queue manager can be altered so that it is once again a full repository, if required.

z/OS On z/OS, N and Y are accepted synonyms of NO and YES.

Related information:
Clustering: Using REFRESH CLUSTER best practices

REFRESH QMGR:

Use the MQSC command REFRESH QMGR to perform special operations on queue managers.

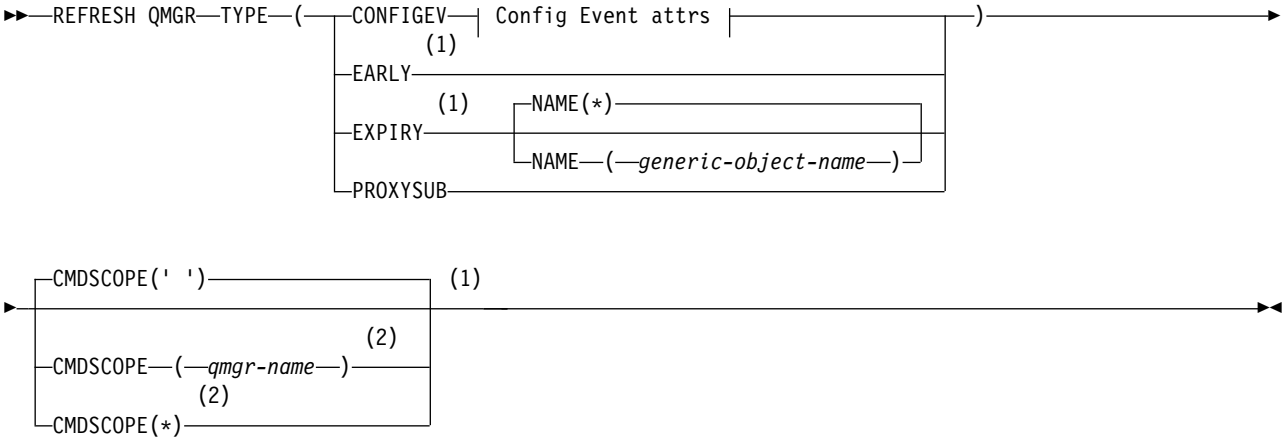
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

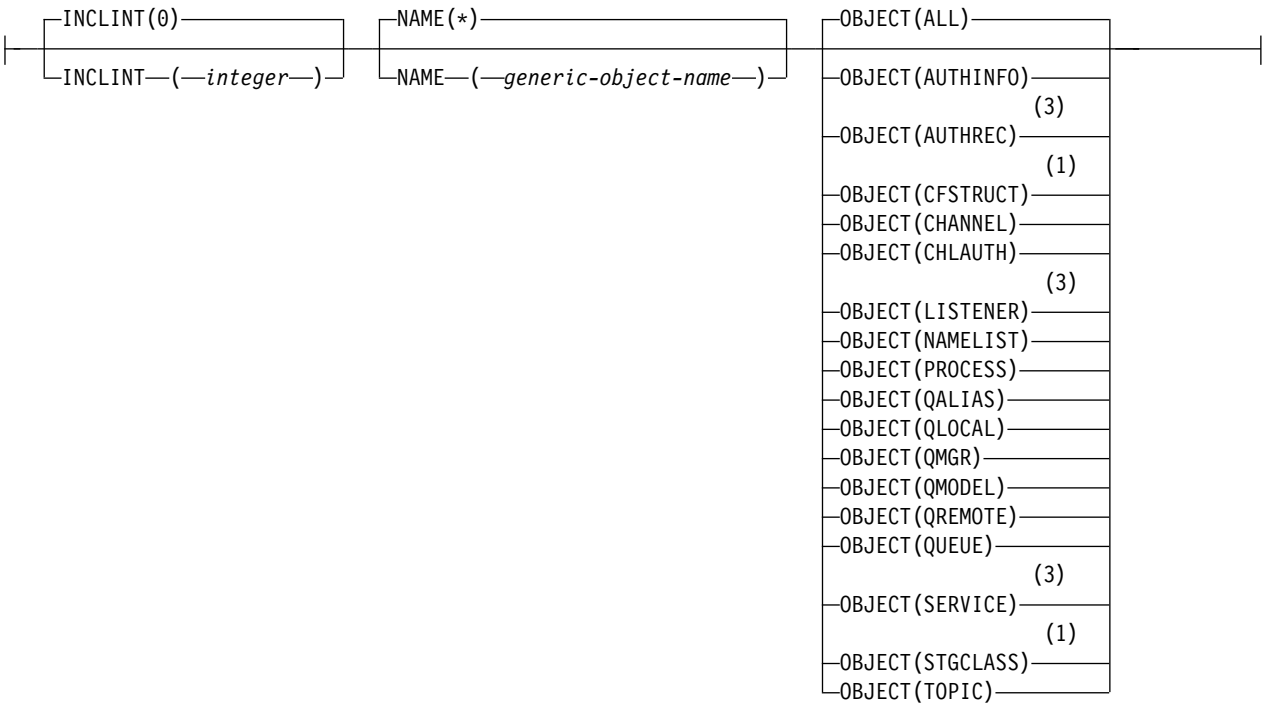
- Syntax diagram
- z/OS See "Using REFRESH QMGR on z/OS" on page 974
- "Usage Notes for REFRESH QMGR" on page 974
- "Parameter descriptions for REFRESH QMGR" on page 975

Synonym: None

REFRESH QMGR



Config Event attrs:



Notes:

1 Valid only on z/OS.

2 Valid only when the queue manager is a member of a queue-sharing group.

3 Not valid on z/OS.

z/OS

Using REFRESH QMGR on z/OS

REFRESH QMGR can be used on z/OS. Depending on the parameters used on the command, it may be issued from various sources. For an explanation of the symbols in this table, see “Using commands on z/OS” on page 365.

Command	Command Sources	Notes
REFRESH QMGR TYPE(CONFIGEV)	2CR	
REFRESH QMGR TYPE(EARLY)	C	Queue manager must not be active.
REFRESH QMGR TYPE(EXPIRY)	2CR	
REFRESH QMGR TYPE(PROXYSUB)	2CR	CHINIT must be active to complete the command.

Usage Notes for REFRESH QMGR

1. Issue this command with TYPE(CONFIGEV) after setting the CONFIGEV queue manager attribute to ENABLED, to bring the queue manager configuration up to date. To ensure that complete configuration information is generated, include all objects; if you have many objects, it might be preferable to use several commands, each with a different selection of objects, but such that all are included.
2. You can also use the command with TYPE(CONFIGEV) to recover from problems such as errors on the event queue. In such cases, use appropriate selection criteria, to avoid excessive processing time and event messages generation.

3. Issue the command with TYPE(EXPIRY) at any time when you believe that a queue could contain numbers of expired messages.
4. z/OS If TYPE(EARLY) is specified, no other keywords are allowed and the command can be issued only from the z/OS console and only if the queue manager is not active.
5. You are unlikely to use **REFRESH QMGR TYPE(PROXYSUB)** other than in exceptional circumstances. See Resynchronization of proxy subscriptions.
6. Successful completion of the **REFRESH QMGR TYPE(PROXYSUB)** command does not mean that the action completed. To check for true completion, see the REFRESH QMGR TYPE(PROXYSUB) step in Checking that async commands for distributed networks have finished.
7. z/OS If a **REFRESH QMGR TYPE(PROXYSUB)** command is issued on z/OS when the CHINIT is not running, the command is queued up and will be processed when the CHINIT starts.
8. Running the command REFRESH QMGR TYPE(CONFIGEV) OBJECT(ALL) includes authority records.

   You cannot specify the **INCLINT** and **NAME** parameters if you explicitly specify AUTHREC events. If you specify **OBJECT**(*ALL*) the **INCLINT** and **NAME** parameters are ignored.

Parameter descriptions for REFRESH QMGR

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

This parameter is not valid with TYPE(EARLY).

INCLINT ***(integer)***

Specifies a value in minutes defining a period immediately before the current time, and requests that only objects that have been created or changed within that period (as defined by the ALTDATE and ALTTIME attributes) are included. The value must be in the range zero through 999 999. A value of zero means there is no time limit (this is the default).

This parameter is valid only with TYPE(CONFIGEV).

NAME ***(generic-object-name)***

Requests that only objects with names that match the one specified are included. A trailing asterisk (*) matches all object names with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all objects (this is the default). NAME is ignored if OBJECT(QMGR) is specified.

This parameter is not valid with TYPE(EARLY).

OBJECT ***(objtype)***

Requests that only objects of the specified type are included. (Synonyms for object types, such as QL, can also be specified.) The default is ALL, to include objects of every type.

This parameter is valid only with TYPE(CONFIGEV).

TYPE This is required. Values are:

CONFIGEV
Requests that the queue manager generates a configuration event message for every object that matches the selection criteria specified by the OBJECT, NAME and INCLINT parameters. Matching objects defined with QSGDISP(QMGR) or QSGDISP(COPY) are always included. Matching objects defined with QSGDISP(GROUP) or QSGDISP(SHARED) are included only if the command is being executed on the queue manager where it is entered.

EARLY
Requests that the subsystem function routines (generally known as early code) for the queue manager replace themselves with the corresponding routines in the linkpack area (LPA).

You need to use this command only after you install new subsystem function routines (provided as corrective maintenance or with a new version or release of IBM MQ). This command instructs the queue manager to use the new routines.

z/OS See Task 3: Update the z/OS link list and LPA for more information about IBM MQ early code routines.

EXPIRY
Requests that the queue manager performs a scan to discard expired messages for every queue that matches the selection criteria specified by the NAME parameter. (The scan is performed regardless of the setting of the EXPRYINT queue manager attribute.)

PROXYSUB
Requests that the queue manager resynchronizes the proxy subscriptions that are held with, and on behalf of, queue managers that are connected in a hierarchy or publish/subscribe cluster.

You should only resynchronize the proxy subscriptions in exceptional circumstances. See Resynchronization of proxy subscriptions.

REFRESH SECURITY:

Use the MQSC command REFRESH SECURITY to perform a security refresh.

Using MQSC commands

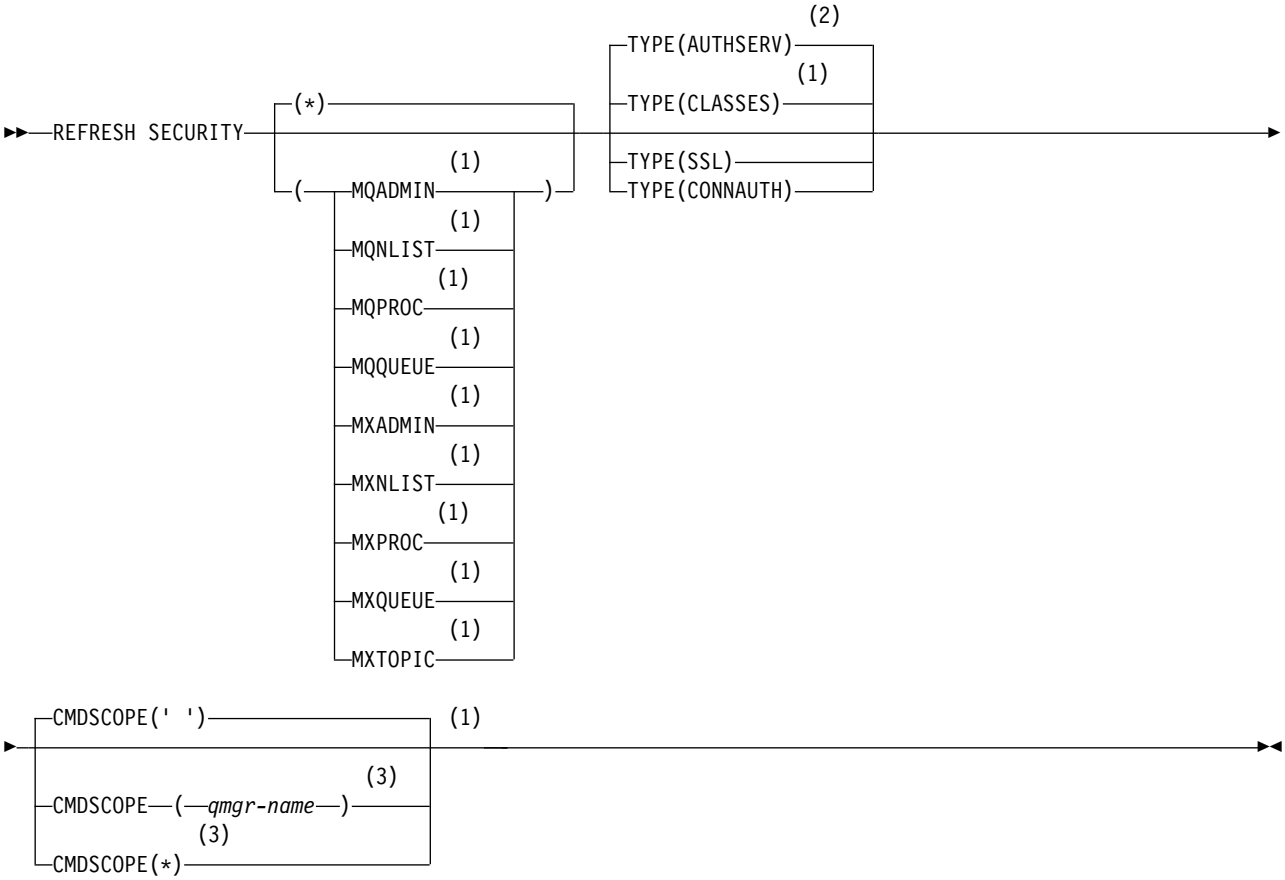
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- z/OS See "Using REFRESH SECURITY on z/OS" on page 977
- "Usage notes for REFRESH SECURITY" on page 978
- "Parameter descriptions for REFRESH SECURITY" on page 979

Synonym: REF SEC

REBUILD SECURITY is another synonym for REFRESH SECURITY.

REFRESH SECURITY



Notes:

1 Valid only on z/OS.

2 Not valid on z/OS.

3 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

z/OS

Using REFRESH SECURITY on z/OS

REFRESH SECURITY can be used on z/OS. Depending on the parameters used on the command, it may be issued from various sources. For an explanation of the symbols in this table, see "Using commands on z/OS" on page 365.

Command	Command Sources	Notes
REFRESH SECURITY TYPE(CLASSES)	CR	
REFRESH SECURITY TYPE(SSL)	CR	Not allowed from CSQINPT or CSQINP2. Channel initiator must be running.

Usage notes for REFRESH SECURITY

When you issue the REFRESH SECURITY TYPE(SSL) MQSC command, all running TLS channels are stopped and restarted. Sometimes TLS channels can take a long time to shut down and this means that the refresh operation takes some time to complete. There is a time limit of 10 minutes for a TLS refresh to complete z/OS (or 1 minute on z/OS), so it can potentially take 10 minutes for the command to finish. This can give the appearance that the refresh operation has "frozen". The refresh operation will fail with an MQSC error message of AMQ9710 or PCF error MQRCCF_COMMAND_FAILED if the timeout is exceeded before all channels have stopped. This is likely to happen if the following conditions are true:

- The queue manager has many TLS channels running simultaneously when the refresh command is invoked
- The channels are handling large numbers of messages

If a refresh fails under these conditions, retry the command later when the queue manager is less busy. In the case where many channels are running, you can choose to stop some of the channels manually before invoking the REFRESH command.

When using TYPE(SSL):

1. z/OS On z/OS, the command server and channel initiator must be running.
2. z/OS On z/OS, IBM MQ determines whether a refresh is needed due to one, or more, of the following reasons:
   - The contents of the key repository have changed
   - The location of the LDAP server to be used for Certification Revocation Lists has changed
   - The location of the key repository has changed

   If no refresh is needed, the command completes successfully and the channels are unaffected.
3. Multi On Multiplatforms, the command updates all TLS channels regardless of whether a security refresh is needed.
4. If a refresh is to be performed, the command updates all TLS channels currently running, as follows:
   - Sender, server and cluster-sender channels using TLS are allowed to complete the current batch. In general they then run the TLS handshake again with the refreshed view of the TLS key repository. However, you must manually restart a requester-server channel on which the server definition has no CONNAME parameter.
   - V 9.0.0 AMQP channels using TLS are restarted, with any currently connected clients being forcibly disconnected. The client receives an `amqp:connection:forced` AMQP error message.
   - All other channel types using TLS are stopped with a STOP CHANNEL MODE(FORCE) STATUS(INACTIVE) command. If the partner end of the stopped message channel has retry values defined, the channel retries and the new TLS handshake uses the refreshed view of the contents of the TLS key repository, the location of the LDAP server to be used for Certification Revocation Lists, and the location of the key repository. In the case of a server-connection channel, the client application loses its connection to the queue manager and has to reconnect in order to continue.

z/OS When using TYPE(CLASSES):

- Classes MQADMIN, MQNLIST, MQPROC, and MQQUEUE can only hold profiles defined in uppercase.
- Classes MXADMIN, MXNLIST, MXPROC, and MQXUEUE can hold profiles defined in mixed case.
- Class MXTOPIC can be refreshed whether using uppercase or mixed case classes. Although it is a mixed case class, it is the only mixed case class that can be active with either group of classes.
- The MQCMD and MQCONN classes cannot be specified, and are not included by REFRESH SECURITY CLASS(*).

  Security information from the MQCMD and MQCONN classes is not cached in the queue manager. See Refreshing queue manager security on z/OS for further information.

Notes:

1. Performing a `REFRESH SECURITY(*) TYPE(CLASSES)` operation is the only way to change the classes being used by your system from uppercase-only support to mixed case support.

   Do this by checking the queue manager attribute SCYCASE to see if it is set to UPPER or MIXED
2. It is your responsibility to ensure that you have copied, or defined, all the profiles you need in the appropriate classes before you carry out a `REFRESH SECURITY(*) TYPE(CLASSES)` operation.
3. A refresh of an individual class is allowed only if the classes currently being used are of the same type. For example, if MQPROC is in use, you can issue a refresh for MQPROC but not MXPROC.

Parameter descriptions for REFRESH SECURITY

The command qualifier allows you to indicate more precise behavior for a specific TYPE value. Select from:

*
A full refresh of the type specified is performed. z/OS This is the default value on z/OS systems.

z/OS **MQADMIN**
Valid only if TYPE is CLASSES. Specifies that Administration type resources are to be refreshed. Valid on z/OS only.

Note: If, when refreshing this class, it is determined that a security switch relating to one of the other classes has been changed, a refresh for that class also takes place.

z/OS **MQNLIST**
Valid only if TYPE is CLASSES. Specifies that Namelist resources are to be refreshed. Valid on z/OS only.

z/OS **MQPROC**
Valid only if TYPE is CLASSES. Specifies that Process resources are to be refreshed. Valid on z/OS only.

z/OS **MQQUEUE**
Valid only if TYPE is CLASSES. Specifies that Queue resources are to be refreshed. Valid on z/OS only.

z/OS **MXADMIN**
Valid only if TYPE is CLASSES. Specifies that administration type resources are to be refreshed. Valid on z/OS only.

Note: If, when refreshing this class, it is determined that a security switch relating to one of the other classes has been changed, a refresh for that class also takes place.

z/OS **MXNLIST**
Valid only if TYPE is CLASSES. Specifies that namelist resources are to be refreshed. Valid on z/OS only.

z/OS **MXPROC**
Valid only if TYPE is CLASSES. Specifies that process resources are to be refreshed. Valid on z/OS only.

z/OS **MXQUEUE**
Valid only if TYPE is CLASSES. Specifies that queue resources are to be refreshed. Valid on z/OS only.

z/OS **MXTOPIC**
Valid only if TYPE is CLASSES. Specifies that topic resources are to be refreshed. Valid on z/OS only.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

' '
The command runs on the queue manager on which it was entered. This is the default value z/OS for non-z/OS systems.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

*
The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

TYPE Specifies the type of refresh that is to be performed.

Multi **AUTHSERV**
The list of authorizations held internally by the authorization services component is refreshed.

This is the default value.

z/OS **CLASSES**

IBM MQ in-storage ESM (external security manager, for example RACF) profiles are refreshed. The in-storage profiles for the resources being requested are deleted. New entries are created when security checks for them are performed, and are validated when the user next requests access.

You can select specific resource classes for which to perform the security refresh.

This is valid only on z/OS where it is the default.

CONNAUTH

Refreshes the cached view of the configuration for connection authentication.

You must refresh the configuration before the queue manager recognizes the changes.

Multi On Multiplatforms, this is a synonym for AUTHSERV.

See Connection authentication for more information.

SSL Refreshes the cached view of the Secure Sockets Layer, or Transport Layer Security, key repository and allows updates to become effective on successful completion of the command. Also refreshed are the locations of:

- the LDAP servers to be used for Certified Revocation Lists
- the key repository

as well as any cryptographic hardware parameters specified through IBM MQ.

To refresh CHLAUTH use the "REFRESH QMGR" on page 973 command.

Related information:

z/OS Refreshing queue manager security on z/OS

RESET CFSTRUCT on z/OS: z/OS

Use the MQSC command RESET CFSTRUCT to modify the status of a specific application structure.

Using MQSC commands

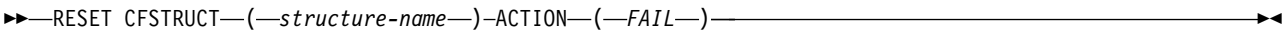
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Notes:"
- "Parameter descriptions for RESET CFSTRUCT"

Synonym: None.

RESET CFSTRUCT



Notes:

1. z/OS Valid only when the queue manager is a member of a queue-sharing group.
2. RESET CFSTRUCT requires CFLEVEL(5)

Parameter descriptions for RESET CFSTRUCT

CFSTRUCT(*structure-name*)
Specify the name of the coupling facility application structure that you want to reset.

ACTION(*FAIL*)
Specify this keyword to simulate a structure failure and set the status of the application structure to FAILED

RESET CHANNEL:

Use the MQSC command RESET CHANNEL to reset the message sequence number for an IBM MQ channel with, optionally, a specified sequence number to be used the next time that the channel is started.

Using MQSC commands

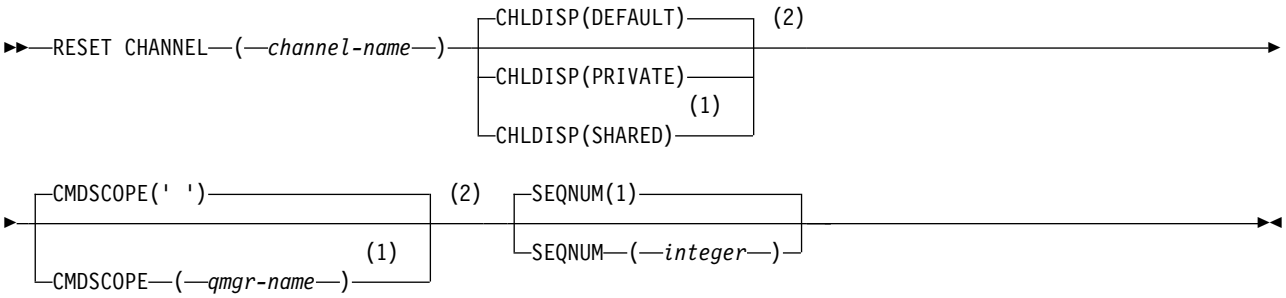
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes"
- "Parameter descriptions for RESET CHANNEL" on page 983

Synonym: RESET CHL

RESET CHANNEL



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes

1. z/OS On z/OS, the command server and channel initiator must be running.
2. This command can be issued to a channel of any type except SVRCONN and CLNTCONN channels, (including those that have been defined automatically). However, if it is issued to a sender or server channel, then in addition to resetting the value at the end at which the command is issued, the value at the other (receiver or requester) end is also reset to the same value the next time this channel is initiated (and resynchronized if necessary). Issuing this command on a cluster-sender channel might reset the message sequence number at either end of the channel. However, this is not significant because the sequence numbers are not checked on clustering channels.
3. If the command is issued to a receiver, requester, or cluster-receiver channel, the value at the other end is not reset as well; this must be done separately if necessary.
4. Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel. If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the channel that was last added to the local queue manager's repository.
5. If the message is non-persistent, and the RESET CHANNEL command is issued to the sender channel, reset data is sent and flows every time the channel starts.

Parameter descriptions for RESET CHANNEL

(channel-name)
The name of the channel to be reset. This is required.

z/OS **CHLDISP**
This parameter applies to z/OS only and can take the values of:

- DEFAULT
- PRIVATE
- SHARED

If this parameter is omitted, then the DEFAULT value applies. This is taken from the default channel disposition attribute, DEFCDISP, of the channel object.

In conjunction with the various values of the CMDSCOPE parameter, this parameter controls two types of channel:

SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of SHARED.

PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than SHARED.

Note: This disposition is **not** related to the disposition set by the disposition of the queue-sharing group of the channel definition.

The combination of the CHLDISP and CMDSCOPE parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.

The various combinations of CHLDISP and CMDSCOPE are summarized in the following table:

Table 117. CHLDISP and CMDSCOPE for RESET CHANNEL

CHLDISP	CMDSCOPE( ) or CMDSCOPE (local-qmgr)	CMDSCOPE (qmgr-name)
PRIVATE	Reset private channel on the local queue manager	Reset private channel on the named queue manager
SHARED	Reset a shared channel on all active queue managers. This might automatically generate a command using CMDSCOPE and send it to the appropriate queue managers. If there is no definition for the channel on the queue managers to which the command is sent, or if the definition is unsuitable for the command, the action fails there. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is actually run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

If CHLDISP is set to SHARED, CMDSCOPE must be blank or the local queue manager.

' '
The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name only if you are using a queue-sharing group environment and if the command server is enabled.

SEQNUM(*integer*)
The new message sequence number, which must be in the range 1 through 999 999 999. This is optional.

RESET CLUSTER:

Use the MQSC command **RESET CLUSTER** to perform special operations on clusters.

Using MQSC commands

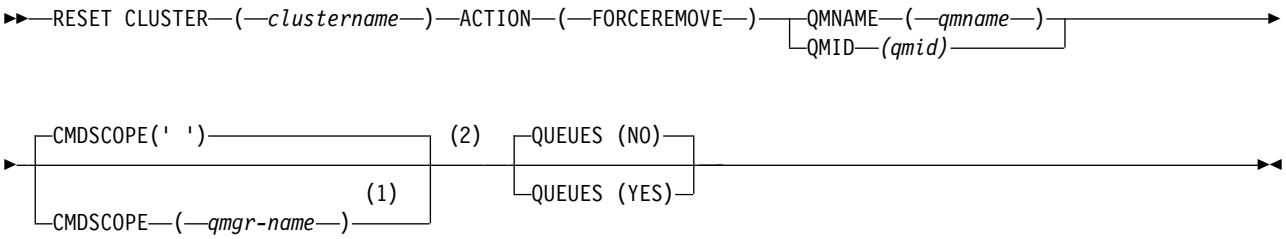
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for RESET CLUSTER"
- "Parameter descriptions for RESET CLUSTER" on page 985

Synonym: None

RESET CLUSTER



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes for RESET CLUSTER

- z/OS On z/OS, the command fails if the channel initiator has not been started.
- z/OS On z/OS, any errors are reported to the console on the system where the channel initiator is running; they are not reported to the system that issued the command.

- To avoid any ambiguity, it is preferable to use QMID rather than QMNAME. The queue manager identifier can be found by commands such as DISPLAY QMGR and DISPLAY CLUSQMGR.

  If QMNAME is used, and there is more than one queue manager in the cluster with that name, the command is not actioned.
- If you use characters other than those listed in Rules for naming IBM MQ objects in your object or variable names, for example in QMID, you must enclose the name in quotation marks.
- If you remove a queue manager from a cluster using this command, you can rejoin it to the cluster by issuing a **REFRESH CLUSTER** command. Wait at least 10 seconds before issuing a **REFRESH CLUSTER** command, because the repository ignores any attempt to rejoin the cluster within 10 seconds of a **RESET CLUSTER** command. If the queue manager is in a publish/subscribe cluster, you then need to reinstate any required proxy subscriptions. See REFRESH CLUSTER considerations for publish/subscribe clusters.

  Note: For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.
- Successful completion of the command does not mean that the action completed. To check for true completion, see the RESET CLUSTER step in Checking that async commands for distributed networks have finished.

Parameter descriptions for RESET CLUSTER

(clustername)
The name of the cluster to be reset. This is required.

ACTION(FORCEREMOVE)

Requests that the queue manager is forcibly removed from the cluster. This might be needed to ensure correct cleanup after a queue manager has been deleted.
This action can be requested only by a repository queue manager.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

' '
The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

QMID(*qmid*)
The identifier of the queue manager to be forcibly removed.

QMNAME(*qmname*)
The name of the queue manager to be forcibly removed.

QUEUES
Specifies whether cluster queues owned by the queue manager being force removed are removed from the cluster.

NO
Cluster queues owned by the queue manager being force removed are not removed from the cluster. This is the default.

YES
Cluster queues owned by the queue manager being force removed are removed from the

cluster in addition to the cluster queue manager itself. The cluster queues are removed even if the cluster queue manager is not visible in the cluster, perhaps because it was previously force removed without the QUEUES option.

z/OS On z/OS, **N** and **Y** are accepted synonyms of **NO** and **YES**.

RESET QMGR:

Use the MQSC command RESET QMGR as part of your backup and recovery procedures.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

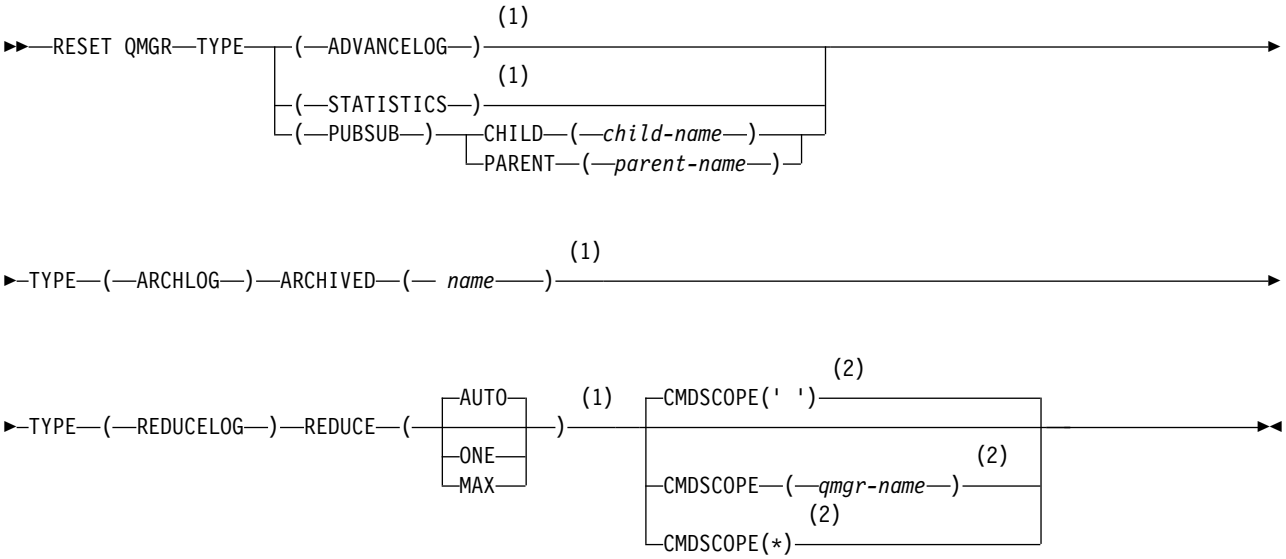
z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

Multi V 9.0.2 Use the **TYPE(ARCHLOG)** option to notify the queue manager that all log extents, up to the specified one, have been archived. If the log management type is not ARCHIVE, the command fails. Use the **TYPE(REDUCELOG)** option to request that the queue manager reduces the number of log extents, provided they are no longer required.

- Syntax diagram
- "Usage notes for RESET QMGR" on page 987
- "Parameter descriptions for RESET QMGR" on page 988

Synonym: None

RESET QMGR



Notes:

1 Not valid on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group

Usage notes for RESET QMGR

- You can use this command to request that the queue manager starts writing to a new log extent, making the previous log extent available for backup. See Updating a backup queue manager. Alternatively, you can use this command to request that the queue manager ends the current statistics collection period and writes the collected statistics. You can also use this command to forcibly remove a publish/subscribe hierarchical connection for which this queue manager is nominated as either the parent or the child in the hierarchical connection.
- The queue manager might refuse a request to advance the recovery log, if advancing the recovery log would cause the queue manager to become short of space in the active log.
- You are unlikely to use **RESET QMGR** TYPE(PUBSUB) other than in exceptional circumstances. Typically the child queue manager uses **ALTER QMGR** PARENT(' ') to remove the hierarchical connection.
- When you need to disconnect from a child or parent queue manager with which the queue manager has become unable to communicate, you must issue the **RESET QMGR** TYPE (PUBSUB) command from a queue manager. When using this command, the remote queue manager is not informed of the canceled connection. It might, therefore, be necessary to issue the **ALTER QMGR** PARENT(' ') command at the remote queue manager. If the child queue manager is not manually disconnected, it is forcibly disconnected and the parent status is set to REFUSED.
- If you are resetting the parent relationship, issue the **ALTER QMGR** PARENT(' ') command, otherwise the queue manager attempts to re-establish the connection when the publish/subscribe capability of the queue manager is later enabled.
- Successful completion of the **RESET QMGR** TYPE(PUBSUB) command does not mean that the action completed. To check for true completion, see the RESET QMGR TYPE(PUBSUB) step in Checking that async commands for distributed networks have finished.
- V 9.0.2 You must specify one only of **ADVANCELOG**, **STATISTICS**, **PUBSUB**, **ARCHLOG** or **REDUCELOG**.

Multi V 9.0.2

Usage notes for TYPE(ARCHLOG)

This option requires change authority on the queue manager object.

The command fails if the log extent is not recognized, or is the current log.

If, for some reason, the programmatic way that your enterprise notifies your log extents are archived is not working, and the disk is filling up with log extents, your administrator can use this command.

You need to determine yourself, the name to pass in from your archiving process, as to what has already been archived.

Multi V 9.0.2

Usage notes for TYPE(REDUCELOG)

This option requires change authority on the queue manager object.

You should not need this command in normal circumstances. In general, when using automatic management of log files, you should leave it up to the queue manager to reduce the number of log extents as necessary.

For circular logging, you can use this option to remove inactive secondary log extents. A growth in secondary log extents is usually noticed by an increase in disk usage, often due to some specific issue in the past.

Note: For circular logging the command might not be able reduce the log extents by the required number immediately. In that case, the command returns, and the reduction takes place asynchronously at some later point.

For linear logging this can remove log extents that are not required for recovery (and have been archived if you are using archive log management) as noticed by a high value for REUSESZ on the DISPLAY QMSTATUS command.

You should run this command only after some specific event that has caused the number of log extents to be extraordinarily large.

The command blocks until the chosen number of extents have been deleted. Note that the command does not return the number of extents that have been removed, but a queue manager error log message is written, indicating what has taken place.

Parameter descriptions for RESET QMGR

TYPE

ADVANCELOG
Requests that the queue manager starts writing to a new log extent, making the previous log extent available for backup. See Updating a backup queue manager. This command is accepted only if the queue manager is configured to use linear logging.

Multi V 9.0.2 **ARCHLOG**

ARCHIVED (*name*)
Notifies the queue manager that this extent, and all logically earlier ones, have been archived.

The extent name is, for example, S0000001.LOG or AMQA000001 on IBM i.

PUBSUB
Requests that the queue manager cancels the indicated publish/subscribe hierarchical connection. This value requires that one of the CHILD or PARENT attributes is specified:

CHILD
The name of the child queue manager for which the hierarchical connection is to be forcibly canceled. This attribute is used only with TYPE(PUBSUB). It cannot be used together with PARENT.

PARENT
The name of a parent queue manager for which the hierarchical connection is to be forcibly canceled. This attribute is used only with TYPE(PUBSUB). It cannot be used together with CHILD.

Multi V 9.0.2 **REDUCELOG**

REDUCE
Requests the queue manager to reduce the number of inactive or superfluous log extents and the way in which the log extents are reduced.

The value can be one of the following:

AUTO
Reduce the log extents by an amount chosen by the queue manager.

ONE Reduce the log extents by one extent, if possible.

MAX Reduce the log extents by the maximum number possible.

STATISTICS
Requests that the queue manager ends the current statistics collection period and writes the collected statistics.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' '
The command runs on the queue manager on which it was entered. This value is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

*
The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of setting this value is the same as entering the command on every queue manager in the queue-sharing group.

RESET QSTATS on z/OS: z/OS

Use the MQSC command RESET QSTATS to report performance data for a queue and then to reset that data.

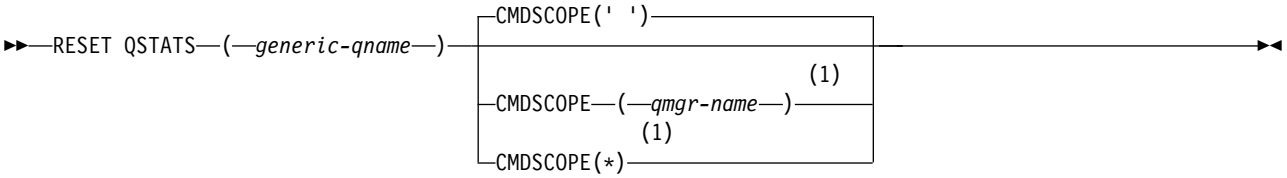
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for RESET QSTATS"
- "Parameter descriptions for RESET QSTATS" on page 990

Synonym: None



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Usage notes for RESET QSTATS

1. If there is more than one queue with a name that satisfies the *generic q-name*, all those queues are reset.

2. Issue this command from an application, and not the z/OS console or its equivalent, to ensure that the statistical information is recorded.
3. The following information is kept for all queues, both private and shared. For shared queues each queue manager keeps an independent copy of the information:

   MSGIN
   Incremented each time a message is put to the shared queue

   MSGOUT
   Incremented each time a message is removed from the shared queue

   HIQDEPTH
   Calculated by comparing the current value for HIQDEPTH held by this queue manager with the new queue depth obtained from the coupling facility during every put operation. The depth of the queue is affected by all queue managers putting messages to the queue or getting messages from it.

   To retrieve the information and obtain full statistics for a shared queue, specify **CMDSCOPE(*)** to broadcast the command to all queue managers in the queue-sharing group.

   The peak queue depth approximates to the maximum of all the returned HIQDEPTH values, the total MQPUT count approximates to the sum of all the returned MSGIN values, and the total MQGET count approximates to the sum of all the returned MSGOUT values.
4. If the PERFMEV attribute of the queue manager is DISABLED, the command fails.

Parameter descriptions for RESET QSTATS

generic-qname
The name of the local queue with a disposition of QMGR, COPY, or SHARED, but not GROUP, with performance data that is to be reset.

A trailing asterisk (*) matches all queues with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all queues.

The performance data is returned in the same format as parameters returned by DISPLAY commands. The data is:

QSTATS
The name of the queue

z/OS **QSGDISP**
The disposition of the queue, that is, QMGR, COPY, or SHARED.

RESETINT
The number of seconds since the statistics were last reset.

HIQDEPTH
The peak queue depth since the statistics were last reset.

MSGSIN
The number of messages that have been added to the queue by MQPUT and MQPUT1 calls since the statistics were last reset.

The count includes messages added to the queue in units of work that have not yet been committed, but the count is not decremented if the units of work are later backed out. The maximum displayable value is 999 999 999; if the number exceeds this value, 999 999 999 is displayed.

MSGSOUT
The number of messages removed from the queue by destructive (non-browse) MQGET calls since the statistics were last reset.

The count includes messages removed from the queue in units of work that have not yet been committed, but the count is not decremented if the units of work are subsequently backed out. The maximum displayable value is 999 999 999; if the number exceeds this value, 999 999 999 is displayed.

CMDSCOPE

This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

Example output

The following example, shows the output from the command on z/OS.

```
12.44.16 STC16696  CSQM201I !MQ13 CSQMDRTC  RESET QSTATS DETAILS  902
   902             QSTATS(CICS01.INITQ)
   902             QSGDISP(QMGR)
   902             RESETINT(43)
   902             HIQDEPTH(0)
   902             MSGSIN(0)
   902             MSGSOUT(0)
   902              END QSTATS DETAILS
12.44.16 STC16696  CSQM201I !MQ13 CSQMDRTC  RESET QSTATS DETAILS  903
   903             QSTATS(MQ13.DEAD.QUEUE)
   903             QSGDISP(QMGR)
   903             RESETINT(43)
   903             HIQDEPTH(0)
   903             MSGSIN(0)
   903             MSGSOUT(0)
   903              END QSTATS DETAILS
12.44.16 STC16696  CSQM201I !MQ13 CSQMDRTC  RESET QSTATS DETAILS  904
   904             QSTATS(SYSTEM.ADMIN.ACTIVITY.QUEUE)
   904             QSGDISP(QMGR)
   904             RESETINT(43)
   904             HIQDEPTH(0)
   904             MSGSIN(0)
   904             MSGSOUT(0)
```

RESET SMDS on z/OS: z/OS

Use the MQSC command RESET SMDS to modify availability or status information relating to one or more shared message data sets associated with a specific application structure.

Using MQSC commands

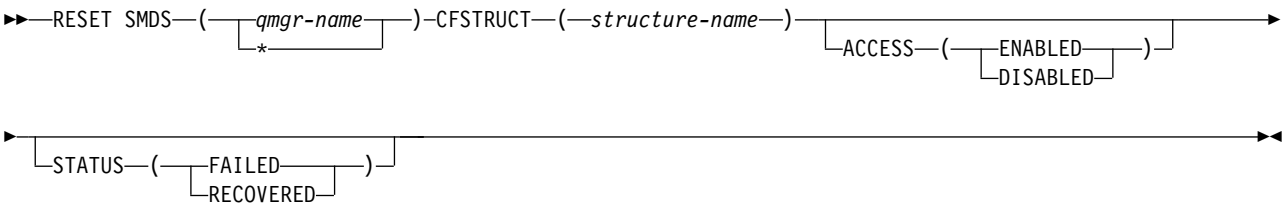
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for RESET SMDS"

Synonym:

RESET SMDS



Parameter descriptions for RESET SMDS

This command is only supported when the CFSTRUCT definition is currently using the option OFFLOAD(SMDS).

SMDS(*qmgr-name*|*)
Specify the queue manager for which the shared message data set availability or status information is to be modified, or an asterisk to modify the information for all data sets associated with the specified CFSTRUCT.

CFSTRUCT(*structure-name*)
Specify the coupling facility application structure for which the availability or status information for one or more shared message data sets is to be modified.

ACCESS(ENABLED|DISABLED)
This keyword is used to enable and disable access to a shared message data set, making it available or unavailable to the queue managers in the group.

This keyword is useful when a shared message data set is required to be temporarily unavailable, for example while moving it to a different volume. In this instance, the keyword would be used to mark the data set as ACCESS(DISABLED) causing all of the queue managers to close it normally and deallocate it. When the data set is ready to be used, it can be marked as ACCESS(ENABLED) allowing the queue managers to access it again.

ENABLED
Use the ENABLED parameter to enable access to the shared message data set after previously disabling access, or to retry access after an error has caused the availability state to be set to ACCESS(SUSPENDED).

DISABLED
Use the DISABLED parameter to indicate that the shared message data set cannot be

used until the access has been changed back to ENABLED. Any queue managers currently connected to the shared message data set are disconnected from it.

STATUS(FAILED | RECOVERED)

This keyword is used to specify that a shared message data set requires recovery/repair, or to reset the STATUS of the data set from FAILED.

If you have detected that a data set is in need of repair, this keyword can be used to manually mark the data set as STATUS(FAILED). If the queue manager detects that the data set requires repair, it automatically marks it as STATUS(FAILED). Then if RECOVER CFSTRUCT is used to successfully complete a repair to the data set, the queue manager automatically marks it as STATUS(RECOVERED). If another method is used to successfully repair the data set, this keyword can be used to manually mark the data set as STATUS(RECOVERED). It is not necessary to manually alter the ACCESS, as it is automatically changed to SUSPENDED while the STATUS is FAILED and then back to ENABLED when the STATUS is set to RECOVERED.

FAILED

Use the FAILED parameter to indicate that the shared message data set needs to be recovered or repaired, and should not be used until this has been completed. This is only allowed if the current state is STATUS(ACTIVE) or STATUS(RECOVERED). If the current availability state is ACCESS(ENABLED) and is not changed on the same command, this sets ACCESS(SUSPENDED) to prevent further attempts to use the shared message data set until it has been repaired. Any queue managers currently connected to the shared message data set are forced to disconnect from it, by closing and deallocating the data set. This status may be set automatically if a permanent I/O error occurs when accessing a shared message data set or if a queue manager determines that header information in the data set is invalid or is inconsistent with the current state of the structure.

RECOVERED

Use the RECOVERED parameter to reset the state from STATUS(FAILED) if the shared message data set does not actually need to be recovered, for example if it was merely temporarily unavailable. If the current availability state (after any change specified on the same command) is ACCESS(SUSPENDED), this sets ACCESS(ENABLED) to allow the owning queue manager to open the shared message data set and perform restart processing, after which the status is changed to STATUS(ACTIVE) and other queue managers can use it again.

RESET TPIPE on z/OS: z/OS

Use the MQSC command RESET TPIPE to reset the recoverable sequence numbers for an IMS Tpipe used by the IBM MQ - IMS bridge.

Using MQSC commands

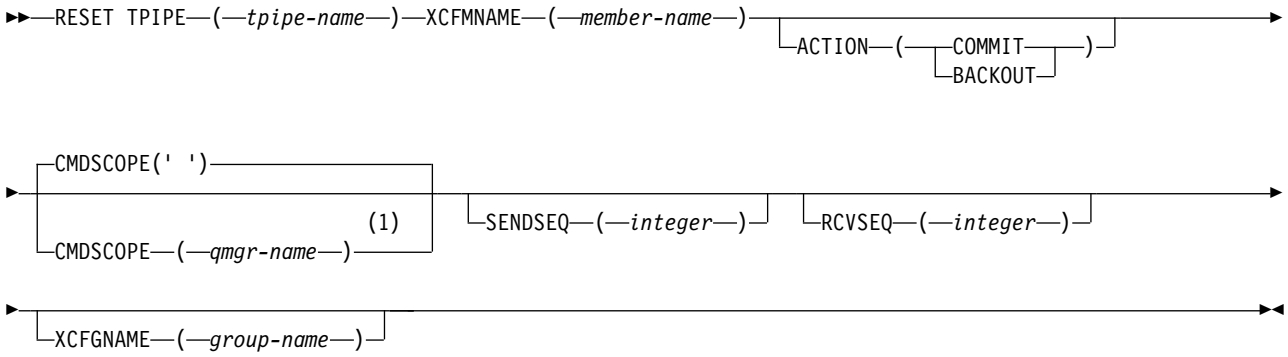
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes" on page 994
- "Parameter descriptions for RESET TPIPE" on page 994

Synonym: There is no synonym for this command.

RESET TPIPE



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Usage notes

1. This command is used in response to the resynchronization error reported in message CSQ2020E, and initiates resynchronization of the Tpipe with IMS.
2. The command fails if the queue manager is not connected to the specified XCF member.
3. The command fails if the queue manager is connected to the specified XCF member, but the Tpipe is open.

Parameter descriptions for RESET TPIPE

(*tpipe-name*)
The name of the Tpipe to be reset. This is required.

CMDSCOPE
This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

' '
The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

ACTION
Specifies whether to commit or back out any unit of recovery associated with this Tpipe. This is required if there is such a unit of recovery reported in message CSQ2020E; otherwise it is ignored.

COMMIT
The messages from IBM MQ are confirmed as having already transferred to IMS ; that is, they are deleted from the IBM MQ - IMS bridge queue.

BACKOUT
The messages from IBM MQ are backed out; that is, they are returned to the IBM MQ - IMS bridge queue.

SENDSEQ(*integer*)
The new recoverable sequence number to be set in the Tpipe for messages sent by IBM MQ and to be set as the partner's receive sequence number. It must be hexadecimal and can be up to 8 digits long, and can optionally be enclosed by `X' '`. It is optional; if omitted, the sequence number is not changed but the partner's receive sequence is set to the IBM MQ send sequence number.

RCVSEQ(*integer*)
The new recoverable sequence number to be set in the Tpipe for messages received by IBM MQ and to be set as the partner's send sequence number. It must be hexadecimal and can be up to 8 digits long, and can optionally be enclosed by `X' '`. It is optional; if omitted, the sequence number is not changed but the partner's send sequence is set to the IBM MQ receive sequence number.

XCFGNAME(*group-name*)
The name of the XCF group to which the Tpipe belongs. This is 1 through 8 characters long. It is optional; if omitted, the group name used is that specified in the OTMACON system parameter.

XCFMNAME(*member-name*)
The name of the XCF member within the group specified by XCFGNAME to which the Tpipe belongs. This is 1 through 16 characters long, and is required.

RESOLVE CHANNEL:

Use the MQSC command RESOLVE CHANNEL to request a channel to commit or back out in-doubt messages.

Using MQSC commands

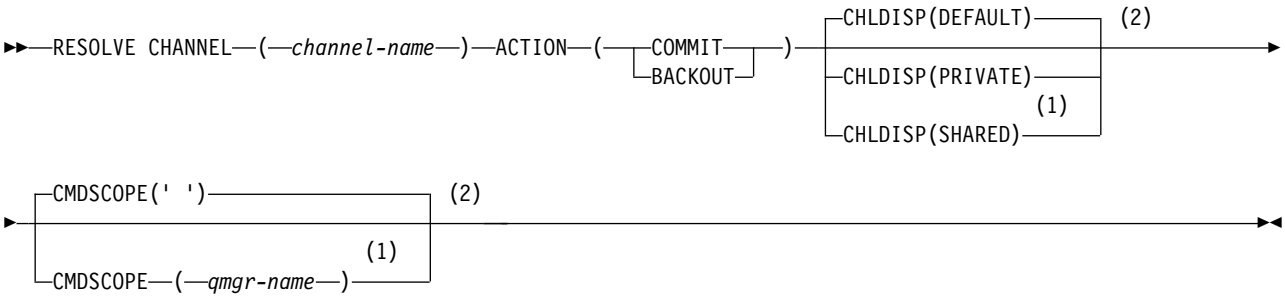
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources `CR`. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for RESOLVE CHANNEL" on page 996
- "Parameter descriptions for RESOLVE CHANNEL" on page 996

Synonym: RESOLVE CHL (RES CHL on z/OS)

RESOLVE CHANNEL



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes for RESOLVE CHANNEL

1. This command is used when the other end of a link fails during the confirmation period, and for some reason it is not possible to reestablish the connection.
2. In this situation the sending end remains in doubt as to whether the messages were received. Any outstanding units of work must be resolved by being backed out or committed.
3. If the resolution specified is not the same as the resolution at the receiving end, messages can be lost or duplicated.
4. z/OS On z/OS, the command server and the channel initiator must be running.
5. This command can be used only for sender (SDR), server (SVR), and cluster-sender (CLUSSDR) channels (including those that have been defined automatically).
6. Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel. If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the channel that was last added to the local queue manager's repository.
7. If the resolution specified is not the same as the resolution at the receiving end, messages can be lost or duplicated.

Parameter descriptions for RESOLVE CHANNEL

(channel-name)
The name of the channel for which in-doubt messages are to be resolved. This is required.

ACTION
Specifies whether to commit or back out the in-doubt messages (this is required):

COMMIT
The messages are committed, that is, they are deleted from the transmission queue

BACKOUT
The messages are backed out, that is, they are restored to the transmission queue

z/OS **CHLDISP**
This parameter applies to z/OS only and can take the values of:

- DEFAULT
- PRIVATE
- SHARED

If this parameter is omitted, then the DEFAULT value applies. This is taken from the default channel disposition attribute, DEFCDISP, of the channel object.

In conjunction with the various values of the CMDSCOPE parameter, this parameter controls two types of channel:

SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of SHARED.

PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than SHARED.

Note: This disposition is **not** related to the disposition set by the disposition of the queue-sharing group of the channel definition.

The combination of the CHLDISP and CMDSCOPE parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.

The various combinations of CHLDISP and CMDSCOPE are summarized in the following table:

Table 118. CHLDISP and CMDSCOPE for RESOLVE CHANNEL

CHLDISP	CMDSCOPE( ) or CMDSCOPE (local-qmgr)	CMDSCOPE (qmgr-name)
PRIVATE	Resolve private channel on the local queue manager	Resolve private channel on the named queue manager
SHARED	Resolve a shared channel on all active queue managers. This might automatically generate a command using CMDSCOPE and send it to the appropriate queue manager. If there is no definition for the channel on the queue manager to which the command is sent, or if the definition is unsuitable for the command, the command fails. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is actually run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

If CHLDISP is set to SHARED, CMDSCOPE must be blank or the local queue manager.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name only if you are using a queue-sharing group environment and if the command server is enabled.

RESOLVE INDOUBT on z/OS: z/OS

Use the MQSC command RESOLVE INDOUBT to resolve threads left in doubt because IBM MQ or a transaction manager could not resolve them automatically.

Using MQSC commands

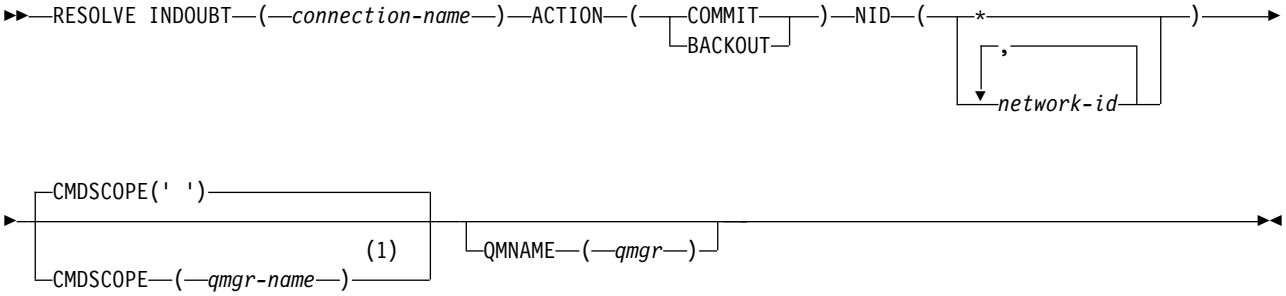
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes"
- "Parameter descriptions for RESOLVE INDOUBT"

Synonym: RES IND

RESOLVE INDOUBT



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Usage notes

This command does not apply to units of recovery associated with batch or TSO applications, unless you are using the RRS adapter.

Parameter descriptions for RESOLVE INDOUBT

(connection-name)
1 through 8 character connection name.

- For a CICS connection it is the CICS applid.
- For an IMS adapter connection, it is the IMS control region job name.
- For an IMS bridge connection, it is the IBM MQ queue manager name.
- For an RRS connection, it is RRSBATCH.

ACTION
Specifies whether to commit or back out the in-doubt threads:

COMMIT
Commits the threads

BACKOUT
Backs out the threads

CMDSCOPE

This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

NID Origin identifier. Specifies the thread or threads to be resolved.

(*origin-id*)

This is as returned by the DISPLAY CONN command, and is of the form *origin-node*.*origin-urid*, where:

- *origin-node* identifies the originator of the thread, except RRSBATCH where it is omitted.
- *origin-urid* is the hexadecimal number assigned to the unit of recovery by the originating system for the specific thread to be resolved.

When *origin-node* is present there must be a period (.) between it and *origin-urid*.

(*) Resolves all threads associated with the connection.

QMNAME

Specifies that if the designated queue manager is INACTIVE, IBM MQ should search information held in the coupling facility about units of work, performed by the indicated queue manager, that match the connection name and origin identifier.

Matching units of work are either committed or backed out according to the ACTION specified.

Only the shared portion of the unit of work are resolved by this command.

As the queue manager is necessarily inactive, local messages are unaffected and remain locked until the queue manager restarts, or after restarting, connects with the transaction manager.

Examples:

```
RESOLVE INDOUBT(CICSA) ACTION(COMMIT) NID(CICSA.ABCDEF0123456789)
RESOLVE INDOUBT(CICSA) ACTION(BACKOUT) NID(*)
```

RESUME QMGR:

Use the MQSC command RESUME QMGR to inform other queue managers in a cluster that the local queue manager is available again for processing and can be sent messages. It reverses the action of the SUSPEND QMGR command.

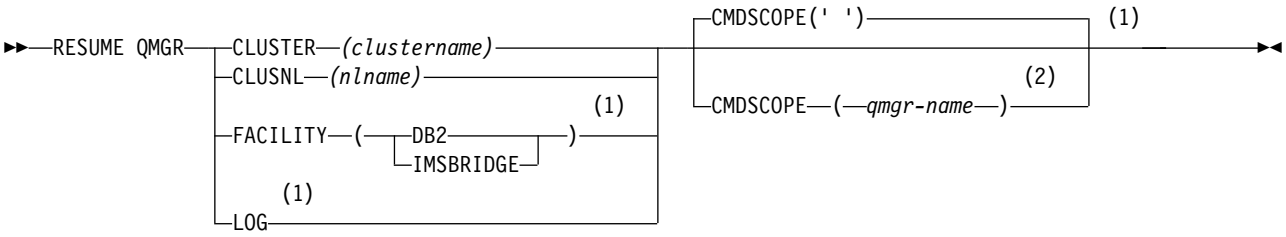
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- z/OS See "Using RESUME QMGR on z/OS"
- "Usage notes"
- "Parameter descriptions for RESUME QMGR" on page 1001

Synonym: None

RESUME QMGR



Notes:

1 Valid only on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

z/OS

Using RESUME QMGR on z/OS

RESUME QMGR can be used on z/OS. Depending on the parameters used on the command, it may be issued from various sources. For an explanation of the symbols in this table, see "Using commands on z/OS" on page 365.

Command	Command Sources	Notes
RESUME QMGR CLUSTER/CLUSNL	CR	Ensure the channel initiator is running
RESUME QMGR FACILITY	CR	
RESUME QMGR LOG	C	

Usage notes

1. UNIX Linux The command is valid only on UNIX and Linux.
2. z/OS On z/OS, if you define CLUSTER or CLUSNL:
   a. The command fails if the channel initiator has not been started.

b. Any errors are reported to the console on the system where the channel initiator is running; they are not reported to the system that issued the command.

3. z/OS On z/OS, you cannot issue RESUME QMGR CLUSTER *(clustername)* or RESUME QMGR FACILITY commands from CSQINP2.

4. z/OS This command, with the CLUSTER and CLUSNL parameters, is **not** available on the reduced function form of IBM MQ for z/OS supplied with WebSphere Application Server.

5. z/OS On z/OS, the SUSPEND QMGR and RESUME QMGR commands are supported through the console only. However, all the other SUSPEND and RESUME commands are supported through the console and command server.

Parameter descriptions for RESUME QMGR

CLUSTER ***(clustername)***
The name of the cluster for which availability is to be resumed.

CLUSNL ***(nlname)***
The name of the namelist specifying a list of clusters for which availability is to be resumed.

FACILITY
Specifies the facility to which connection is to be re-established.

Db2 Re-establishes connection to Db2.

IMSBRIDGE
Resumes normal IMS bridge activity.

This parameter is only valid on z/OS.

LOG Resumes logging and update activity for the queue manager that was suspended by a previous SUSPEND QMGR command. Valid on z/OS only. If LOG is specified, the command can be issued only from the z/OS console.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

RVERIFY SECURITY on z/OS: z/OS

Use the MQSC command RVERIFY SECURITY to set a reverification flag for all specified users. The user is reverified the next time that security is checked for that user.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

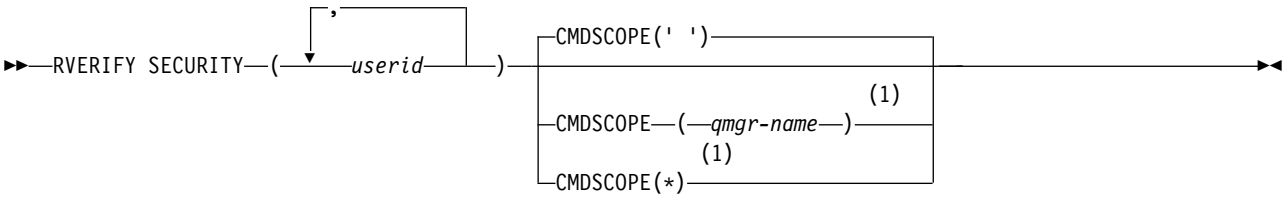
You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for RVERIFY SECURITY"

Synonym: REV SEC

REVERIFY SECURITY is another synonym for RVERIFY SECURITY

RVERIFY SECURITY



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Parameter descriptions for RVERIFY SECURITY

(userids...)
You must specify one or more user IDs. Each user ID specified is signed off and signed back on again the next time that a request is issued on behalf of that user that requires security checking.

CMDSCOPE
This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE cannot be used for commands issued from the first initialization input data set CSQINP1.

''
The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

*
The command runs on the local queue manager and is also passed to every active queue

manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

SET ARCHIVE on z/OS: z/OS

Use the MQSC command SET ARCHIVE to dynamically change certain archive system parameter values initially set by your system parameter module at queue manager startup.

Using MQSC commands

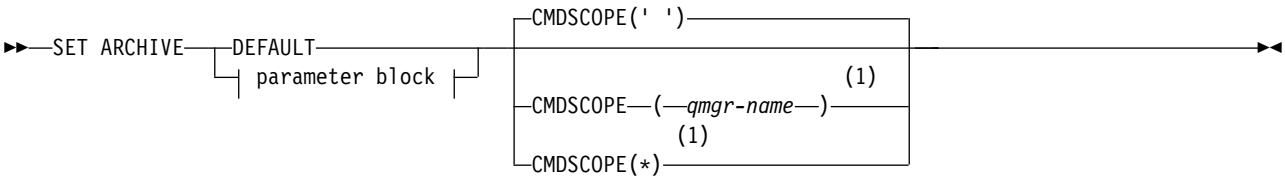
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 12CR. For an explanation of the source symbols, see Using commands on z/OS.

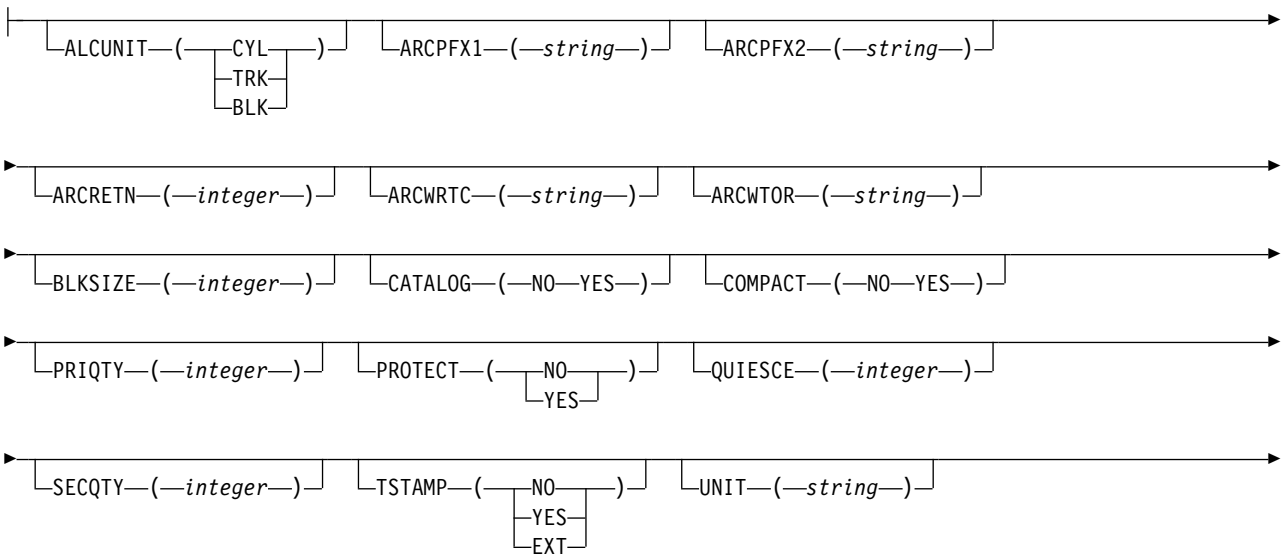
- Syntax diagram
- "Usage notes for SET ARCHIVE" on page 1004
- "Parameter descriptions for SET ARCHIVE" on page 1004
- "Parameter block" on page 1004

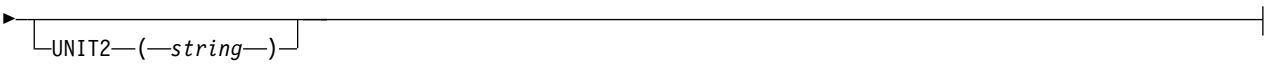
Synonym: SET ARC

SET ARCHIVE



Parameter Block:





Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Usage notes for SET ARCHIVE

1. The new values will be used at the next archive log offload.
2. The queue manager picks up the values in ZPARM, so the **SET ARCHIVE** values you used in the previous cycle are lost.

   To permanently change the values, either change the CSQ6SYSP parameters and regenerate the parameter module, or put the **SET ARCHIVE** commands into a data set in the CSQINP2 concatenation.

Parameter descriptions for SET ARCHIVE

CMDSCOPE

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE cannot be used for commands issued from the first initialization input data set CSQINP1.

'' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which it was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

You cannot use CMDSCOPE(*qmgr-name)* for commands issued from the first initialization input data set, CSQINP1.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE(*) for commands issued from CSQINP1.

DEFAULT

Resets all the archive system parameters to the values set at queue manager startup.

Parameter block

z/OS For a full description of these parameters, see Using CSQ6ARVP.

Parameter block is any one or more of the following parameters that you want to change:

ALCUNIT

Specifies the unit in which primary and secondary space allocations are made.

Specify one of:

CYL Cylinders

TRK Tracks

BLK Blocks

ARCPFX1
Specifies the prefix for the first archive log data set name.

See the TSTAMP parameter for a description of how the data sets are named and for restrictions on the length of ARCPFX1.

ARCPFX2
Specifies the prefix for the second archive log data set name.

See the TSTAMP parameter for a description of how the data sets are named and for restrictions on the length of ARCPFX2.

ARCRETN
Specifies the retention period, in days, to be used when the archive log data set is created.

The parameter must be in the range zero - 9999.

z/OS For more information about discarding archive log data sets, see Discarding archive log data sets.

ARCWRTC
Specifies the list of z/OS routing codes for messages about the archive log data sets to the operator.

Specify up to 14 routing codes, each with a value in the range 1 through 16. You must specify at least one code. Separate codes in the list by commas, not by blanks.

For more information about z/OS routing codes, see the *MVS Routing and Descriptor Codes* manual.

ARCWTOR
Specifies whether a message is to be sent to the operator and a reply received before attempting to mount an archive log data set.

Other IBM MQ users might be forced to wait until the data set is mounted, but they are not affected while IBM MQ is waiting for the reply to the message.

Specify either:

YES The device needs a long time to mount archive log data sets. For example, a tape drive. (The synonym is **Y**.)

NO The device does not have long delays. For example, DASD. (The synonym is **N**.)

BLKSIZE
Specifies the block size of the archive log data set. The block size you specify must be compatible with the device type you specify in the UNIT parameter.

The parameter must be in the range 4 097 through 28 672. The value you specify is rounded up to a multiple of 4 096.

This parameter is ignored for data sets that are managed by the storage management subsystem (SMS).

CATALOG
Specifies whether archive log data sets are cataloged in the primary integrated catalog facility (ICF) catalog.

Specify either:

NO Archive log data sets are not cataloged. (The synonym is **N**.)

YES Archive log data sets are cataloged. (The synonym is **Y**.)

COMPACT
Specifies whether data written to archive logs is to be compacted. This option applies only to a 3480 or 3490 device that has the improved data recording capability (IDRC) feature. When this feature is turned on, hardware in the tape control unit writes data at a much higher density than normal,

allowing for more data on each volume. Specify NO if you do not use a 3480 device with the IDRC feature or a 3490 base model, with the exception of the 3490E. Specify YES if you want the data to be compacted.

Specify either:

NO Do not compact the data sets. (The synonym is **N**.)

YES Compact the data sets. (The synonym is **Y**.)

PRIQTY
Specifies the primary space allocation for DASD data sets in ALCUNITs.

The value must be greater than zero.

This value must be sufficient for a copy of either the log data set or its corresponding BSDS, whichever is the larger.

PROTECT
Specifies whether archive log data sets are to be protected by discrete ESM (external security manager) profiles when the data sets are created.

Specify either:

NO Profiles are not created. (The synonym is **N**.)

YES Discrete data set profiles are created when logs are offloaded. (The synonym is **Y**.) If you specify YES:

- ESM protection must be active for IBM MQ.
- The user ID associated with the IBM MQ address space must have authority to create these profiles.
- The TAPEVOL class must be active if you are archiving to tape.

Otherwise, offloads will fail.

QUIESCE
Specifies the maximum time in seconds allowed for the quiesce when an ARCHIVE LOG command is issued with MODE QUIESCE specified.

The parameter must be in the range 1 through 999.

SECQTY
Specifies the secondary space allocation for DASD data sets in ALCUNITs.

The parameter must be greater than zero.

TSTAMP
Specifies whether the archive log data set name has a time stamp in it.

Specify either:

NO Names do not include a time stamp. (The synonym is **N**.) The archive log data sets are named:

arcpfxi.A *nnnnnnn*

Where *arcpfxi* is the data set name prefix specified by ARCPFX1 or ARCPFX2. *arcpfxi* can have up to 35 characters.

YES Names include a time stamp. (The synonym is **Y**.) The archive log data sets are named:

arcpfxi.cyyddd.T *hhmmsst*.A *nnnnnnn*

where *c* is 'D' for the years up to and including 1999 or 'E' for the year 2000 and later, and *arcpfxi* is the data set name prefix specified by ARCPFX1 or ARCPFX2. *arcpfxi* can have up to 19 characters.

EXT Names include a time stamp. The archive log data sets are named:
arcpfxi.D *yyyyddd*.T *hhmmsst*.A *nnnnnnn*

Where *arcpfxi* is the data set name prefix specified by ARCPFX1 or ARCPFX2. *arcpfxi* can have up to 17 characters.

UNIT
Specifies the device type or unit name of the device that is used to store the first copy of the archive log data set.

Specify a device type or unit name of 1 through 8 characters.

If you archive to DASD, you can specify a generic device type with a limited volume range.

UNIT2
Specifies the device type or unit name of the device that is used to store the second copy of the archive log data sets.

Specify a device type or unit name of 1 through 8 characters.

If this parameter is blank, the value set for the UNIT parameter is used.

SET AUTHREC on Multiplatforms: Multi

Use the MQSC command SET AUTHREC to set authority records associated with a profile name.

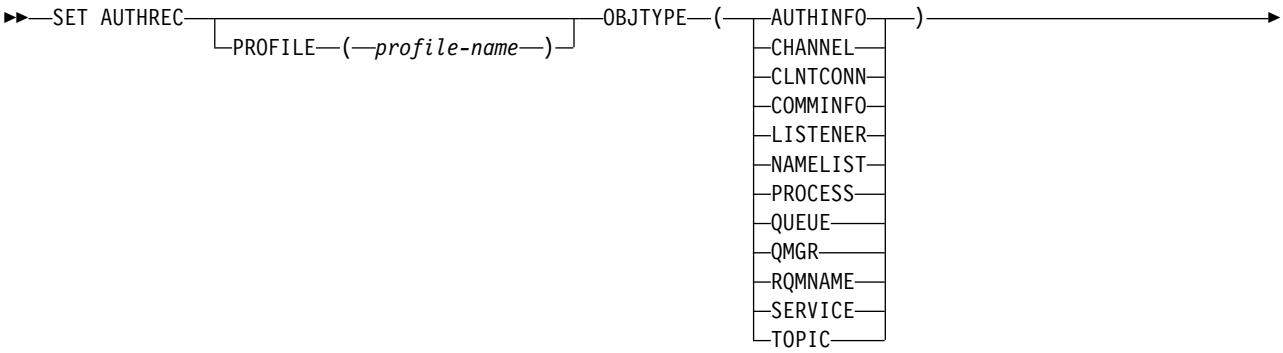
Using MQSC commands

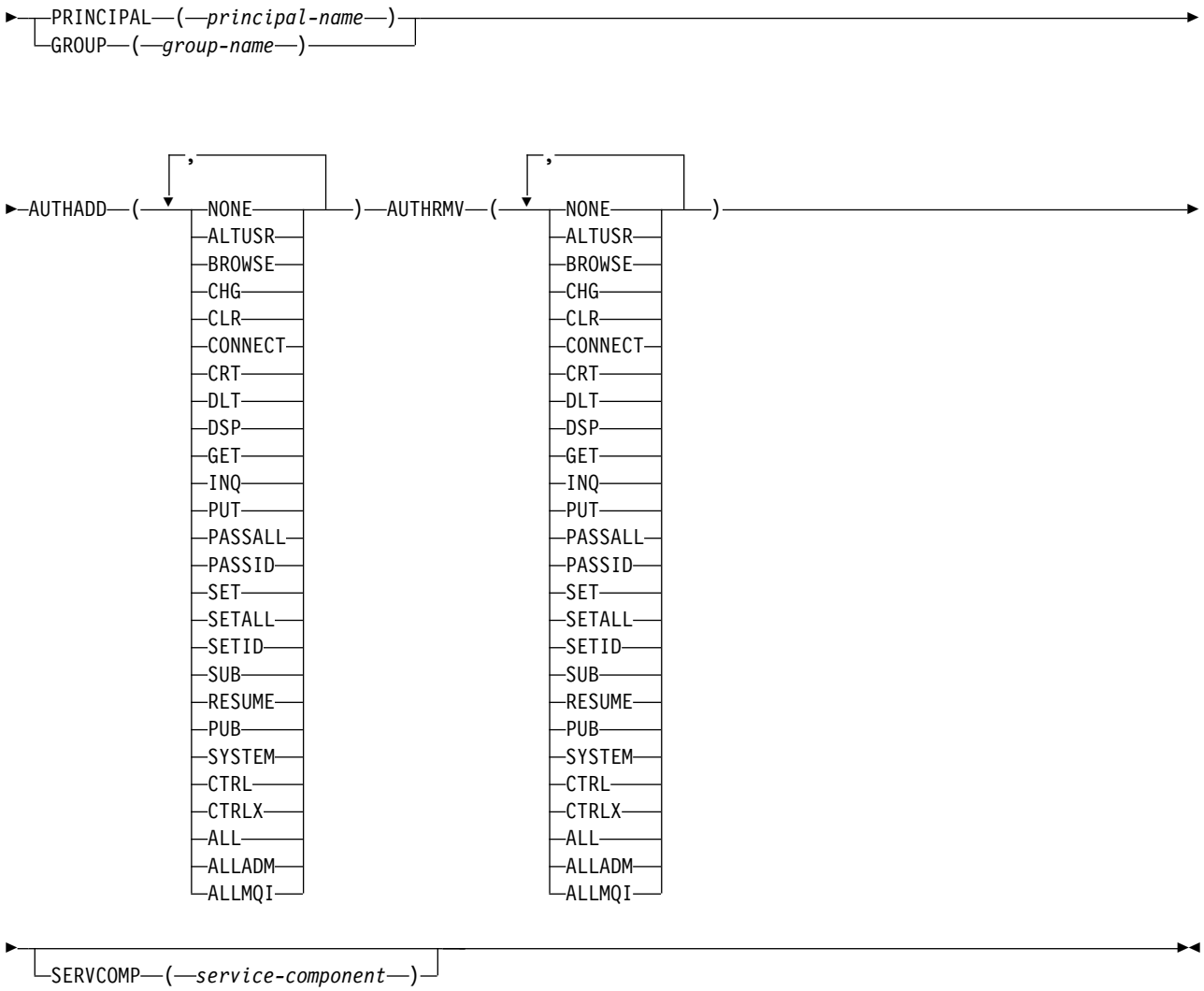
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Parameter descriptions" on page 1008
- Usage notes for SET AUTHREC

See "**setmqaut (grant or revoke authority)**" on page 321 for more information on the options that you can select.

SET AUTHREC





Parameter descriptions

PROFILE(*profile-name*)
The name of the object or generic profile for which to display the authority records. This parameter is required unless the **OBJTYPE** parameter is QMGR, in which case it can be omitted.

See Using OAM generic profiles on UNIX, Linux, and Windows for more information on generic profiles and wildcard characters.

OBJTYPE
The type of object referred to by the profile. Specify one of the following values:

AUTHINFO
Authentication information record

CHANNEL
Channel

CLNTCONN
Client connection channel

COMMINFO
Communication information object

LISTENER
Listener

NAMELIST
Namelist

PROCESS
Process

QUEUE
Queue

QMGR
Queue manager

RQMNAME
Remote queue manager

SERVICE
Service

TOPIC
Topic

PRINCIPAL(*principal-name*)
A principal name. This is the name of a user for whom to set authority records for the specified profile. On IBM MQ for Windows, the name of the principal can optionally include a domain name, specified in this format: `user@domain`.

You must specify either PRINCIPAL or GROUP.

GROUP(*group-name*)
A group name. This is the name of the user group for which to set authority records for the specified profile. You can specify one name only and it must be the name of an existing user group.

Windows For IBM MQ for Windows only, the group name can optionally include a domain name, specified in the following format:

```
GroupName@domain
```

You must specify either PRINCIPAL or GROUP.

AUTHADD
A list of authorizations to add in the authority records. Specify any combination of the following values:

NONE
No authorization

ALTUSR
Specify an alternative user ID on an MQI call

BROWSE
Retrieve a message from a queue by issuing an **MQGET** call with the `BROWSE` option

CHG Change the attributes of the specified object, using the appropriate command set

CLR Clear a queue or a topic

CONNECT
Connect an application to a queue manager by issuing an **MQCONN** call

CRT Create objects of the specified type using the appropriate command set

DLT Delete the specified object using the appropriate command set

DSP Display the attributes of the specified object using the appropriate command set

GET Retrieve a message from a queue by issuing an **MQGET** call

INQ Make an inquiry on a specific queue by issuing an **MQINQ** call

PUT Put a message on a specific queue by issuing an **MQPUT** call

PASSALL
Pass all context

PASSID
Pass the identity context

SET Set attributes on a queue by issuing an **MQSET** call

SETALL
Set all context on a queue

SETID
Set the identity context on a queue

SUB Create, alter, or resume a subscription to a topic using the **MQSUB** call

RESUME
Resume a subscription using the MQSUB call

PUB Publish a message on a topic using the **MQPUT** call

SYSTEM
Give authority to principals or groups, who are authorized to carry out privileged operations on the queue manager, for internal system operations.

CTRL Start and stop the specified channel, listener, or service, and ping the specified channel

CTRLX
Reset or resolve the specified channel

ALL Use all operations relevant to the object

`all` authority is equivalent to the union of the authorities `alladm`, `allmqi`, and `system` appropriate to the object type.

ALLADM
Perform all administration operations relevant to the object

ALLMQI
Use all MQI calls relevant to the object

AUTHRMV
A list of authorizations to remove from the authority records. Specify any combination of the following values:

NONE
No authorization

ALTUSR
Specify an alternative user ID on an MQI call

BROWSE
Retrieve a message from a queue by issuing an **MQGET** call with the `BROWSE` option

CHG Change the attributes of the specified object, using the appropriate command set

CLR Clear a queue or a topic

CONNECT
Connect an application to a queue manager by issuing an **MQCONN** call

CRT Create objects of the specified type using the appropriate command set

DLT Delete the specified object using the appropriate command set

DSP Display the attributes of the specified object using the appropriate command set

GET Retrieve a message from a queue by issuing an **MQGET** call

INQ Make an inquiry on a specific queue by issuing an **MQINQ** call

PUT Put a message on a specific queue by issuing an **MQPUT** call

PASSALL
Pass all context

PASSID
Pass the identity context

SET Set attributes on a queue by issuing an **MQSET** call

SETALL
Set all context on a queue

SETID
Set the identity context on a queue

SUB Create, alter, or resume a subscription to a topic using the **MQSUB** call

RESUME
Resume a subscription using the MQSUB call

PUB Publish a message on a topic using the **MQPUT** call

SYSTEM
Use queue manager for internal system operations

CTRL Start and stop the specified channel, listener, or service, and ping the specified channel

CTRLX
Reset or resolve the specified channel

ALL Use all operations relevant to the object

`all` authority is equivalent to the union of the authorities `alladm`, `allmqi`, and `system` appropriate to the object type.

ALLADM
Perform all administration operations relevant to the object

ALLMQI
Use all MQI calls relevant to the object

Note: To use SETID or SETALL authority, authorizations must be granted on both the appropriate queue object and also on the queue manager object.

SERVCOMP(*service-component*)
The name of the authorization service for which information is to be set.

If you specify this parameter, it specifies the name of the authorization service to which the authorizations apply. If you omit this parameter, the authority record is set using the registered authorization services in turn in accordance with the rules for chaining authorization services.

Usage notes for SET AUTHREC

The list of authorizations to add and the list of authorizations to remove must not overlap. For example, you cannot add display authority and remove display authority with the same command. This rule applies even if the authorities are expressed using different options. For example, the following command fails because DSP authority overlaps with ALLADM authority:

```
SET AUTHREC PROFILE(*) OBJTYPE(QUEUE) PRINCIPAL(PRINC01) AUTHADD(DSP) AUTHRMV(ALLADM)
```

The exception to this overlap behavior is with the ALL authority. The following command first adds ALL authorities then removes the SETID authority:

```
SET AUTHREC PROFILE(*) OBJTYPE(QUEUE) PRINCIPAL(PRINC01) AUTHADD(ALL) AUTHRMV(SETID)
```

The following command first removes ALL authorities then adds the DSP authority:

```
SET AUTHREC PROFILE(*) OBJTYPE(QUEUE) PRINCIPAL(PRINC01) AUTHADD(DSP) AUTHRMV(ALL)
```

Regardless of the order in which they are provided on the command, the ALL are processed first.

SET CHLAUTH:

Use the MQSC command SET CHLAUTH to create or modify a channel authentication record.

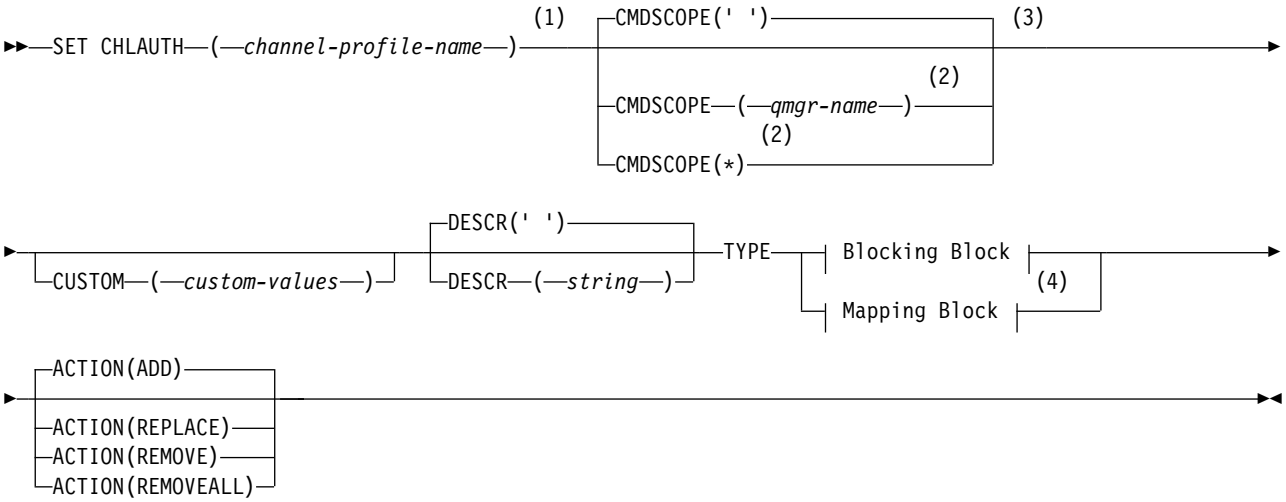
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

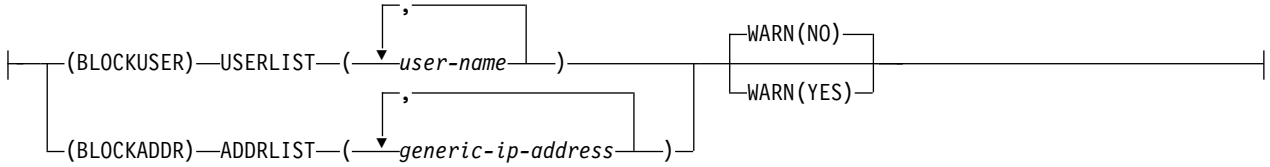
z/OS You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- Usage notes
- Parameters

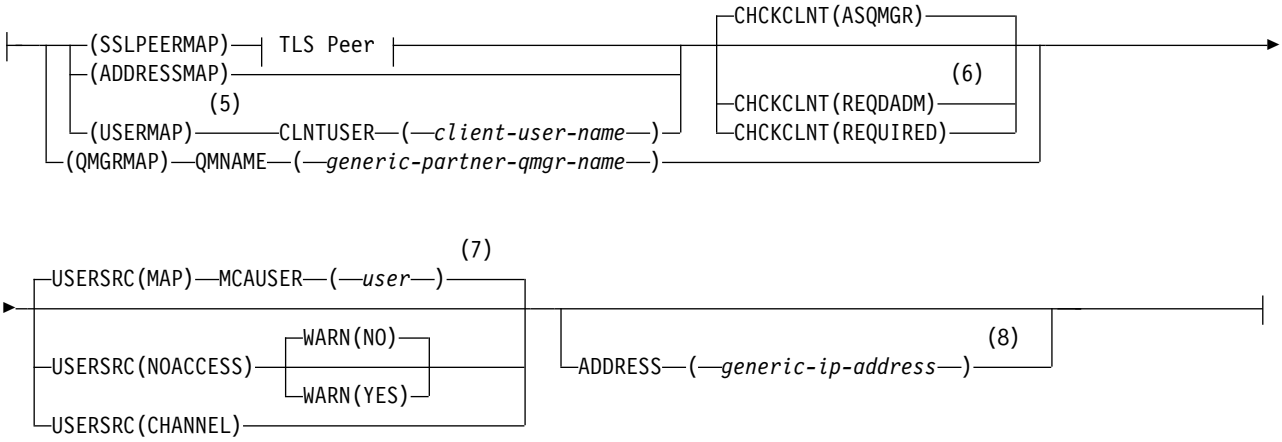
SET CHLAUTH



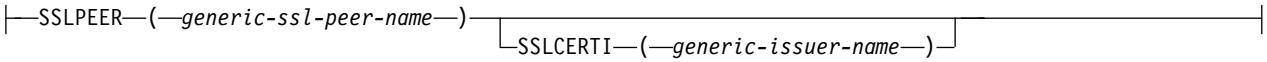
Blocking Block:



Mapping Block:



TLS Peer:



Notes:

1 The channel profile name must be '*' when TYPE is BLOCKADDR.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Select the appropriate value for TYPE, depending upon the option that you select from the two types of block.

5 USERMAP rules apply only to server connection channels.

6 Not valid on z/OS.

7 If you allow USERSRC to default to MAP, you must set a value for the parameter MCAUSER.

8 Mandatory when TYPE is ADDRESSMAP.

Usage notes

The following table shows which parameters are valid for each value of **ACTION**:

	Action		
Parameter	**ADD or REPLACE**	**REMOVE**	**REMOVEALL**
CHLAUTH	✓	✓	✓
TYPE	✓	✓	✓
z/OS CMDSCOPE	✓	✓	✓
ACTION	✓	✓	✓
ADDRESS	✓	✓	

	Action		
Parameter	**ADD or REPLACE**	**REMOVE**	**REMOVEALL**
ADDRLIST	✓	✓	
CHCKCLNT	✓		
CLNTUSER	✓	✓	
MCAUSER	✓		
QMNAME	✓	✓	
SSLCERTI	✓	✓	
SSLPEER	✓	✓	
USERLIST	✓	✓	
USERSRC	✓		
WARN	✓		
DESCR	✓		

Note the following:

- CHLAUTH rules can be used for any channels
- USERMAP rules are valid, only for server connection channels.
- Changes, such as mapping the MCAUSER of the channel, take effect only when starting a channel.

  Therefore, if a channel is already running, that channel must be stopped, and restarted, for the CHLAUTH rule changes to take effect.

Parameters

channel-profile-name

The name of the channel or set of channels for which you are setting channel authentication configuration. You can use one or more asterisks (*), in any position, as wildcards to specify a set of channels. If you set **TYPE** to BLOCKADDR, you must set the generic channel name to a single asterisk, which matches all channel names. On z/OS the generic-channel-name must be in quotes if it contains an asterisk.

TYPE

The **TYPE** parameter must follow the **channel-profile-name** parameter.

The type of channel authentication record for which to set allowed partner details or mappings to MCAUSER. This parameter is required. The following values can be used:

BLOCKUSER

This channel authentication record prevents a specified user or users from connecting. The BLOCKUSER parameter must be accompanied by a USERLIST.

BLOCKADDR

This channel authentication record prevents connections from a specified IP address or

addresses. The BLOCKADDR parameter must be accompanied by an ADDRLIST. BLOCKADDR operates at the listener before the channel name is known.

SSLPEERMAP
This channel authentication record maps TLS Distinguished Names (DNs) to MCAUSER values. The SSLPEERMAP parameter must be accompanied by an SSLPEER.

ADDRESSMAP
This channel authentication record maps IP addresses to MCAUSER values. The ADDRESSMAP parameter must be accompanied by an ADDRESS. ADDRESSMAP operates at the channel.

USERMAP
This channel authentication record maps asserted user IDs to MCAUSER values. The USERMAP parameter must be accompanied by a CLNTUSER.

QMGRMAP
This channel authentication record maps remote queue manager names to MCAUSER values. The QMGRMAP parameter must be accompanied by a QMNAME.

ACTION
The action to perform on the channel authentication record. The following values are valid:

ADD Add the specified configuration to a channel authentication record. This is the default value.

For types SSLPEERMAP, ADDRESSMAP, USERMAP and QMGRMAP, if the specified configuration exists, the command fails.

For types BLOCKUSER and BLOCKADDR, the configuration is added to the list.

REPLACE
Replace the current configuration of a channel authentication record.

For types SSLPEERMAP, ADDRESSMAP, USERMAP and QMGRMAP, if the specified configuration exists, it is replaced with the new configuration. If it does not exist it is added.

For types BLOCKUSER and BLOCKADDR, the configuration specified replaces the current list, even if the current list is empty. If you replace the current list with an empty list, this acts like REMOVEALL.

REMOVE
Remove the specified configuration from the channel authentication records. If the configuration does not exist the command fails. If you remove the last entry from a list, this acts like REMOVEALL.

REMOVEALL
Remove all members of the list and thus the whole record (for BLOCKADDR and BLOCKUSER) or all previously defined mappings (for ADDRESSMAP, SSLPEERMAP, QMGRMAP and USERMAP) from the channel authentication records. This option cannot be combined with specific values supplied in **ADDRLIST**, **USERLIST**, **ADDRESS**, **SSLPEER**, **QMNAME** or **CLNTUSER**. If the specified type has no current configuration the command still succeeds.

ADDRESS

Attention: Host names can be specified in this parameter, only on queue managers that have IBM MQ Version 8.0 new functions enabled with OPMODE.
The filter to be used to compare with the IP address or host name of the partner queue manager or client at the other end of the channel. Channel authentication records containing hostnames are only checked if the queue manager is configured to look them up with REVDNS(ENABLED). Details of the values that are allowed as host names are defined in the IETF documents RFC 952 and RFC 1123. Hostname matching is not case sensitive.

This parameter is mandatory with **TYPE(ADDRESSMAP)**

This parameter is also valid when **TYPE** is SSLPEERMAP, USERMAP, or QMGRMAP and **ACTION** is ADD, REPLACE, or REMOVE. You can define more than one channel authentication object with the same main identity, for example the same TLS peer name, with different addresses. However, you cannot define channel authentication records with overlapping address ranges for the same main identity. See "Generic IP addresses for channel authentication records" on page 1019 for more information about filtering IP addresses.

If the address is generic then it must be in quotes.

ADDRLIST
A list of up to 256 generic IP addresses which are banned from accessing this queue manager on any channel. This parameter is only valid with TYPE(BLOCKADDR). See "Generic IP addresses for channel authentication records" on page 1019 for more information about filtering IP addresses.

If the address is generic then it must be in quotes.

CHCKCLNT
Specifies whether the connection that matches this rule and is being allowed in with **USERSRC(CHANNEL)** or **USERSRC(MAP)**, must also specify a valid user ID and password. The password cannot contain single quotation marks (').

Attention: This parameter is valid only on queue managers that have IBM MQ Version 8.0 new functions enabled with OPMODE.

REQDADM
A valid user ID and password are required for the connection to be allowed if you are using a privileged user ID.

Any connections using a non-privileged user ID are not required to provide a user ID and password. The user ID and password are checked against the user repository details provided in an authentication information object and supplied on **ALTER QMGR** in the **CONNAUTH** field. If no user repository details are provided, so that user ID and password checking are not enabled on the queue manager, the connection is not successful.

z/OS This option is not valid on z/OS platforms.

REQUIRED
A valid user ID and password are required for the connection to be allowed. The password cannot contain single quotation marks (').

The user ID and password are checked against the user repository details provided in an authentication information object and supplied on **ALTER QMGR** in the **CONNAUTH** field. If no user repository details are provided, so that user ID and password checking are not enabled on the queue manager, the connection is not successful.

ASQMGR In order for the connection to be allowed, it must meet the connection authentication requirements defined on the queue manager.

If the **CONNAUTH** field provides an authentication information object, and the value of **CHCKCLNT** is REQUIRED, the connection fails unless a valid user ID and password are supplied. If the **CONNAUTH** field does not provide an authentication information object, or the value of **CHCKCLNT** is not REQUIRED, the user ID and password are not required.

Attention: If you select REQUIRED or REQADM on Multiplatforms and you have not set the CONNAUTH field on the queue manager, or if the value of CHCKCLNT is NONE, the connection fails. On Multiplatforms, you receive message AMQ9793. On z/OS, you receive message CSQX793E. This parameter is valid only with **TYPE(USERMAP)**, **TYPE(ADDRESSMAP)**, and **TYPE(SSLPEERMAP)** and only when **USERSRC** is not set to NOACCESS. It only applies to inbound connections which are SVRCONN channels.

Example rules that use this attribute:

- Anything in the defined network can use an asserted user ID if a valid password is supplied:

```
SET CHLAUTH('*.SVRCONN') +
    TYPE(ADDRESSMAP) ADDRESS('192.0.2.*') +
    USERSRC(CHANNEL) CHCKCLNT(REQUIRED)
```

- This rule ensures that SSL authentication must succeed before processing client authentication according to the policy set at the queue manager:

```
SET CHLAUTH('SSL.APP1.SVRCONN') +
    TYPE(SSLPEERMAP) SSLPEER('CN="Steve Smith", L="BankA"') +
    MCAUSER(SSMITH) CHCKCLNT(ASQMGR)
```

CLNTUSER

The client asserted user ID to be mapped to a new user ID, allowed through unchanged, or blocked.

This can be the user ID flowed from the client indicating the user ID the client side process is running under, or the user ID presented by the client on an MQCONNX call using MQCSP.

The maximum length of the string is MQ_CLIENT_USER_ID_LENGTH.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

'' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect is the same as entering the command on every queue manager in the queue-sharing group.

CUSTOM

Reserved for future use.

DESCR

Provides descriptive information about the channel authentication record, which is displayed when you issue the DISPLAY CHLAUTH command. It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: Use characters from the coded character set identifier (CCSID) for this queue manager. Other characters might be translated incorrectly if the information is sent to another queue manager.

MCAUSER

The user identifier to be used when the inbound connection matches the TLS DN, IP address, client asserted user ID or remote queue manager name supplied.

This parameter is mandatory with **USERSRC(MAP)** and is valid when **TYPE** is SSLPEERMAP, ADDRESSMAP, USERMAP, or QMGRMAP.

If you use lower case user IDs you must enclose them in quotation marks: For example:

```
SET CHLAUTH('SYSTEM.DEF.SVRCONN') TYPE(USERMAP) CLNTUSER('johndoe') +
    USERSRC(MAP) MCAUSER(JOHNDOE1) +
    ADDRESS('::FFFF:9.20.4.136') +
    DESCR('Client from z/Linux machine') +
    ACTION(REPLACE)
```

This allows the lower case user ID to use channel SYSTEM.DEF.SVRCONN on IP address ::FFFF:9.20.4.136. The MCA user for the connection is JOHNDOE1.

If you display the Channel Status (CHS) of the channel, the output is MCAUSER(JOHNDOE1).

This parameter can be used only when **ACTION** is ADD or REPLACE.

QMNAME
The name of the remote partner queue manager, or pattern that matches a set of queue manager names, to be mapped to a user ID or blocked.

This parameter is valid only with **TYPE(QMGRMAP)**.

If the queue manager name is generic then it must be in quotes.

SSLCERTI

Attention: This parameter is valid only on queue managers that have IBM MQ Version 8.0 new functions enabled with OPMODE.
This parameter is additional to the **SSLPEER** parameter.

SSLCERTI restricts matches to being within certificates issued by a particular Certificate Authority.

A blank **SSLCERTI** acts like a wildcard, matches any Issuer Distinguished Name.

SSLPEER

The filter to use to compare with the Subject Distinguished Name of the certificate from the peer queue manager or client at the other end of the channel.

The **SSLPEER** filter is specified in the standard form used to specify a Distinguished Name. See IBM MQ rules for SSLPEER values for details.

The maximum length of the parameter is 1024 bytes.

USERLIST
A list of up to 100 user IDs which are banned from use of this channel or set of channels. Use the special value *MQADMIN to mean privileged or administrative users. The definition of this value depends on the operating system, as follows:

- Windows On Windows, all members of the mqm group, the Administrators group and SYSTEM.
- UNIX Linux On UNIX and Linux, all members of the mqm group.
- IBM i On IBM i, the profiles (users) qmqm and qmqmadm and all members of the qmqmadm group, and any user defined with the *ALLOBJ special setting.
- z/OS On z/OS, the user ID that the channel initiator, queue manager and ⌂advanced message security address spaces are running under.

For more information about privileged users, see Privileged users.

This parameter is only valid with **TYPE(BLOCKUSER)**.

USERSRC
The source of the user ID to be used for MCAUSER at run time. The following values are valid:

MAP Inbound connections that match this mapping use the user ID specified in the **MCAUSER** attribute. This is the default value.

NOACCESS
Inbound connections that match this mapping have no access to the queue manager and the channel ends immediately.

CHANNEL
Inbound connections that match this mapping use the flowed user ID or any user defined on the channel object in the MCAUSER field.

Note that WARN and USERSRC(CHANNEL), or USERSRC(MAP) are incompatible. This is because channel access is never blocked in these cases, so there is never a reason to generate a warning.

WARN
Indicates whether this record operates in warning mode.

NO This record does not operate in warning mode. Any inbound connection that matches this record is blocked. This is the default value.

YES This record operates in warning mode. Any inbound connection that matches this record and would therefore be blocked is allowed access. An error message is written and, if channel events are configured, a channel event message is created showing the details of what would have been blocked, see Channel Blocked. The connection is allowed to continue. An attempt is made to find another record that is set to WARN(NO) to set the credentials for the inbound channel.

Related information:

Channel authentication records

Securing remote connectivity to the queue manager

Generic IP addresses for channel authentication records:

In the various commands that create and display channel authentication records, you can specify certain parameters as either a single IP address or a pattern to match a set of IP addresses.

When you create a channel authentication record, using the MQSC command **SET CHLAUTH** or the PCF command **Set Channel Authentication Record**, you can specify a generic IP address in various contexts. You can also specify a generic IP address in the filter condition when you display a channel authentication record using the commands **DISPLAY CHLAUTH** or **Inquire Channel Authentication Records**.

You can specify the address in any of the following ways:

- a single IPv4 address, such as 192.0.2.0
- a pattern based on an IPv4 address, including an asterisk (*) as a wildcard. The wildcard represents one or more parts of the address, depending on context. For example, the following values are all valid:
  - 192.0.2.*
  - 192.0.*
  - 192.0.*.2
  - 192.*.2
  - *
- a pattern based on an IPv4 address, including a hyphen (-) to indicate a range, for example 192.0.2.1-8
- a pattern based on an IPv4 address, including both an asterisk and a hyphen, for example 192.0.*.1-8
- a single IPv6 address, such as 2001:DB8:0:0:0:0:0:0
- a pattern based on an IPv6 address including an asterisk (*) as a wildcard. The wildcard represents one or more parts of the address, depending on context. For example, the following values are all valid:
  - 2001:DB8:0:0:0:0:0:*
  - 2001:DB8:0:0:0:*
  - 2001:DB8:0:0:0:*:0:1
  - 2001:*:1
  - *

- a pattern based on an IPv6 address, including a hyphen (-) to indicate a range, for example 2001:DB8:0:0:0:0:0:0-8
- a pattern based on an IPv6 address, including both an asterisk and a hyphen, for example 2001:DB8:0:0:0:*:0:0-8

If your system supports both IPv4 and IPv6, you can use either address format. IBM MQ recognizes IPv4 mapped addresses in IPv6.

Certain patterns are invalid:

- A pattern cannot have fewer than the required number of parts, unless the pattern ends with a single trailing asterisk. For example 192.0.2 is invalid, but 192.0.2.* is valid.
- A trailing asterisk must be separated from the rest of the address by the appropriate part separator (a dot (.) for IPv4, a colon (:) for IPv6). For example, 192.0* is not valid because the asterisk is not in a part of its own.
- A pattern may contain additional asterisks provided that no asterisk is adjacent to the trailing asterisk. For example, 192.*.2.* is valid, but 192.0.*.* is not valid.
- An IPv6 address pattern cannot contain a double colon and a trailing asterisk, because the resulting address would be ambiguous. For example, 2001::* could expand to 2001:0000:*, 2001:0000:0000:* and so on

Related information:

Mapping an IP address to an MCAUSER user ID

SET LOG on Multiplatforms: Multi V 9.0.2

On Multiplatforms, use the MQSC command SET LOG to notify the queue manager that archiving of a log extent is complete. If the log management type is not ARCHIVE, the command fails.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Parameter descriptions for SET LOG"
- "Usage notes"

Synonym: SET LOG

SET LOG

Parameter descriptions for SET LOG

ARCHIVED (*name*)
The extent name, for example, S0000001.LOG or AMQA000001 on IBM i.

Usage notes

This command requires change authority on the queue manager object.

The command fails if the log extent is not recognized, or is being written.

The command does not fail if the extent has already been marked as having been archived.

Extents prefixed with the letter R are extents that are waiting to be reused so these extents cannot be passed to **SET LOG ARCHIVED**.

Any extent (prefixed with S) can be archived and passed to **SET LOG ARCHIVED**, except for the current extent. So extents needed for restart or media recovery, or both, can be archived and passed to **SET LOG ARCHIVED** because the queue manager has finished writing to them.

Note that extents can be archived and passed to **SET LOG ARCHIVED** in any order - not necessarily in the order in which they were written.

A message is written to the error log if the queue manager is notified about an extent more than once, either from this command, or the "RESET QMGR" on page 986 command.

SET LOG on z/OS: z/OS

On z/OS, use the MQSC command SET LOG to dynamically change certain log system parameter values that were initially set by your system parameter module at queue manager startup.

Using MQSC commands

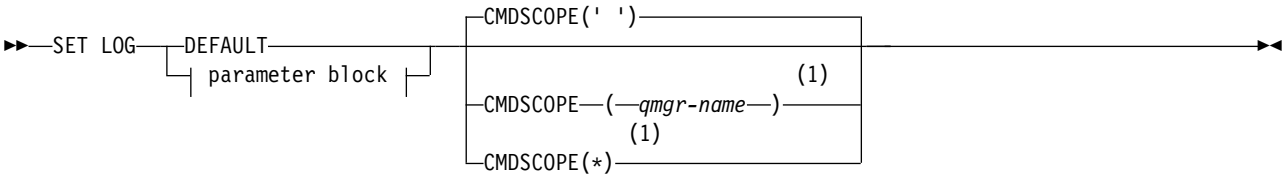
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 12CR. For an explanation of the source symbols, see Using commands on z/OS.

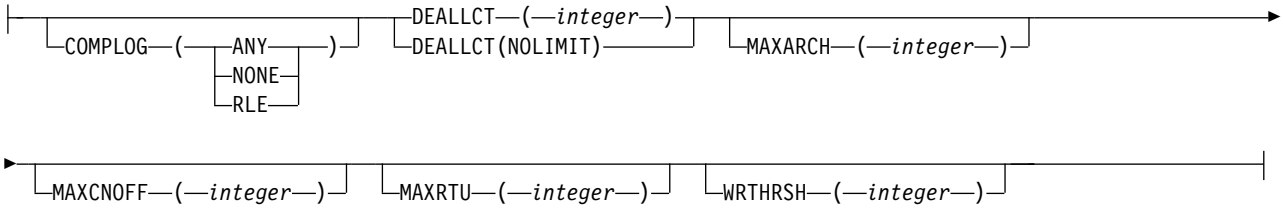
- Syntax diagram
- "Usage notes for SET LOG" on page 1022
- "Parameter descriptions for SET LOG" on page 1022
- "Parameter block" on page 1022

Synonym: SET LOG

SET LOG



Parameter Block:



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Usage notes for SET LOG

1. Any changes to WRTHRSH take immediate effect.
2. Any change to MAXARCH takes effect for the next scheduled offload (that is, not for any offload in progress at the time the command is issued).

Parameter descriptions for SET LOG

CMDSCOPE

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

" The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which it was entered, only if you are using a queue-sharing group environment and if the command server is enabled.You cannot use CMDSCOPE(*qmgr-name)* for commands issued from the first initialization input data set, CSQINP1.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE(*) for commands issued from CSQINP1.

DEFAULT

Reset all the log system parameters to the values specified at queue manager startup.

Parameter block

z/OS For a full description of these parameters, see Using CSQ6LOGP.

Parameter block is any one or more of the following parameters that you want to change:

COMPLOG

This parameter specifies whether compression is used by the queue manager when writing log records. Any compressed records are automatically decompressed irrespective of the current COMPLOG setting.

The possible values are:

ANY Enable the queue manager to select the compression algorithm that gives the greatest degree of log record compression. Using this option currently results in RLE compression.

NONE

No log data compression is used. This is the default value.

RLE Log data compression is performed using run-length encoding (RLE).

z/OS For more details about log compression, see Log compression.

DEALLCT

Specifies the length of time that an allocated archive read tape unit is allowed to remain unused before it is deallocated. You are recommended to specify the maximum possible values, within system constraints, for both options to achieve the optimum performance for reading archive tapes.

This, together with the MAXRTU parameter, allows IBM MQ to optimize archive log reading from tape devices.

The possible values are:

integer
: Specifies the maximum time in minutes, in the range 0 through 1439. Zero means that a tape unit is deallocated immediately.

NOLIMIT or 1440
: Indicates that the tape unit is never deallocated.

MAXARCH
Specifies the maximum number of archive log volumes that can be recorded in the BSDS. When this number is exceeded, recording begins again at the start of the BSDS.

Use a decimal number in the range 10 through 1000.

MAXCNOFF
Maximum number of concurrent log offload tasks.

Specify a decimal number between 1 and 31. If no value is specified the default of 31 applies.

Configure a number lower than the default if your archive logs are allocated on a tape device, and there are constraints on the number of such devices that can be concurrently allocated to the queue manager.

MAXRTU(*integer*)
Specifies the maximum number of dedicated tape units that can be allocated to read archive log tape volumes. This overrides the value for MAXRTU set by CSQ6LOGP in the archive system parameters.

This, together with the DEALLCT parameter, allows IBM MQ to optimize archive log reading from tape devices.

Note:

1. The integer value can be in the range 1 - 99.
2. If the number specified is greater than the current specification, the maximum number of tape units allowable for reading archive logs increases.
3. If the number specified is less than the current specification, tape units that are not being used are immediately deallocated to adjust to the new value. Active, or premounted, tape units remain allocated.
4. A tape unit is a candidate for deallocation because of a lowered value only if there is no activity for the unit.
5. When you are asked to mount an archive tape and you reply CANCEL, the MAXRTU value is reset to the current number of tape units.

   For example, if the current value is 10, but you reply CANCEL to the request for the seventh tape unit, the value is reset to six.

WRTHRSH
Specifies the number of 4 KB output buffers to be filled before they are written to the active log data sets.

The larger the number of buffers, the less often the write takes place, and this improves the performance of IBM MQ. The buffers might be written before this number is reached if significant events, such as a commit point, occur.

Specify the number of buffers in the range 1 through 256.

SET POLICY: Multi

Use the MQSC command SET POLICY to set a security policy.

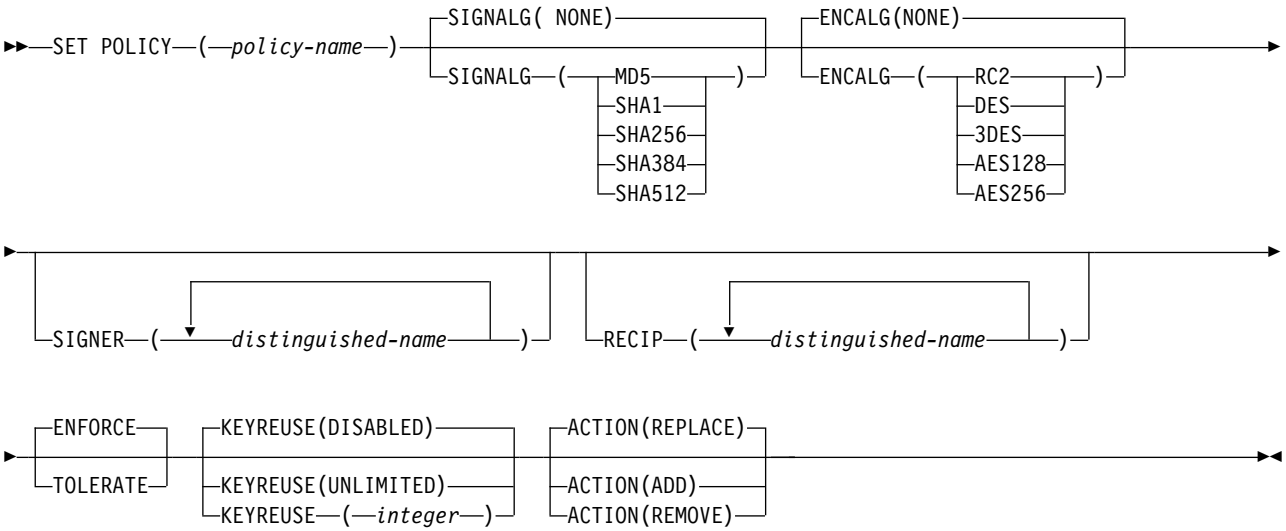
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Parameter descriptions for SET POLICY"

Important: You must have an Advanced Message Security (AMS) license installed to issue this command. If you attempt to issue the **SET POLICY** command without an AMS license installed, you receive message AMQ7155 - License file not found or not valid.

SET POLICY



Parameter descriptions for SET POLICY

(policy-name)
Name of the policy, required.

The policy name must match the name of the queue which is to be protected.

SIGNALG
Specifies the digital signature algorithm from one of the following values:

- NONE
- MD5
- SHA1
- SHA256
- SHA384
- SHA512

The default value is NONE.

ENCALG
Specifies the digital encryption algorithm from one of the following values:

- NONE
- RC2
- DES
- 3DES
- AES128
- AES256

The default value is NONE.

RECIP ***(distinguished-name)***
Specifies the message distinguished name (DN) of the recipient, that is, the certificate of a DN provided used to encrypt a given message.

Notes:

1. The attributes names for DNs must be provided in capital letters.
2. Commas must be used as a name separator.
3. You must specify at least one recipient, if you use any encryption algorithm other than NONE.

You can specify multiple **RECIP** parameters on the same policy.

SIGNER ***(distinguished-name)***
Specifies a signature DN that is validated during the message retrieval. Only messages signed by the user, with a DN provided, are accepted during retrieval.

Notes:

1. The attributes name for DNs must be provided in capital letters.
2. Commas must be used as a name separator.
3. You can specify signature DNs, only if you use any signature algorithm other than NONE.

You can specify multiple **SIGNER** parameters on the same policy.

ENFORCE
Specifies that all messages must be protected when retrieved from the queue.

Any unprotected message encountered is moved to the SYSTEM.PROTECTION.ERROR.QUEUE.

ENFORCE is the default value.

TOLERATE
Specifies that the messages that are not protected when retrieved from the queue can ignore the policy.

TOLERATE is optional and exists to facilitate staged implementation, where:

- Policies have been applied to queues, but those queues might already contain unprotected messages, or
- Queues might still receive messages from remote systems that do not yet have the policy set.

V 9.0.0 **KEYREUSE**
Specify the number of times that an encryption key can be re-used, in the range 1-9999999, or the special values *DISABLED* or *UNLIMITED*.

Note that this is a maximum number of times a key can be reused, therefore a value of *1* means, at most, two messages can use the same key.

DISABLED
Prevents a symmetric key from being reused

UNLIMITED
Allows a symmetric key to be reused any number of times.

DISABLED is the default value.

Attention: Key reuse is valid only for CONFIDENTIALITY policies, that is, **SIGNALG** set to *NONE* and **ENCALG** set to an algorithm value. For all other policy types, you must omit the parameter, or set the **KEYREUSE** value to *DISABLED*.

ACTION
Specify the action for the parameters supplied, as they apply to any existing policy, using one of the following values:

REPLACE
Has the effect of replacing any existing policy with the parameters supplied.

ADD Has the effect that signers and recipients parameters have an additive effect. That is, if a signer or recipient is specified, and does not already exist in a preexisting policy, the signer or recipient value is added to the existing policy definition.

REMOVE
Has the opposite effect of *ADD*. That is, if any of the signer or recipient values specified exist in a preexisting policy, those values are removed from the policy definition.

REPLACE is the default value.

SET SYSTEM on z/OS: z/OS

Use the MQSC command SET SYSTEM to dynamically change certain general system parameter values that were initially set from your system parameter module at queue manager startup. To permanently change these, either change the CSQ6SYSP parameters and regenerate the parameter module, or put the SET SYSTEM commands into a data set in the CSQINP2 concatenation.

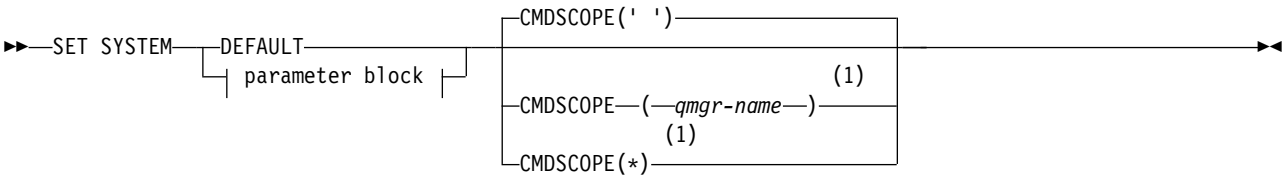
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

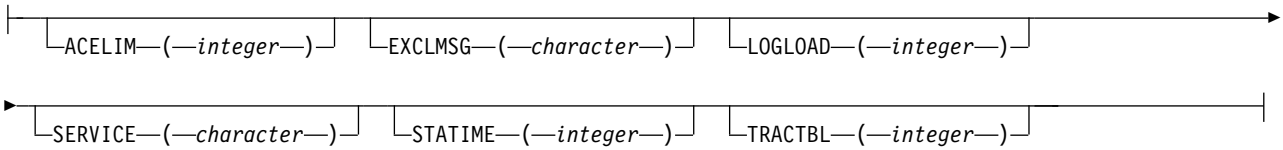
You can issue this command from sources 12CR. For an explanation of the source symbols, see Using commands on z/OS.

Synonym: None

SET SYSTEM



Parameter Block:



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

The CTHREAD, IDFORE, and IDBACK parameters are ignored in IBM WebSphere MQ Version 7.1 or later, but are still allowed for compatibility with earlier versions. Any attempt to change the value of one of these parameters sets it to a default value of 32767.

Usage notes for SET SYSTEM

The new values take immediate effect, with the possible exception of STATIME and TRACTBL.

Changes to STATIME take effect when the current interval expires, unless the new interval is less than the unexpired portion of the current interval, in which case statistics are gathered immediately and the new interval then takes effect.

For TRACTBL, if there is any trace currently in effect, the existing trace table continues to be used, and its size is unchanged. A new global trace table is only obtained for a new START TRACE command. If a new trace table is created with insufficient storage, the old trace table continues to be used, and the message CSQW153E is displayed.

Parameter descriptions for SET SYSTEM

CMDSCOPE
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which it was entered, only if you are using a queue-sharing group environment and if the command server is enabled. You cannot use CMDSCOPE(*qmgr-name)* for commands issued from the first initialization input data set, CSQINP1.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE(*) for commands issued from CSQINP1.

DEFAULT
Resets all the general system parameters to the values set at queue manager startup.

Parameter block

z/OS For a full description of these parameters, see Using CSQ6SYSP.

Parameter block is any one or more of the following parameters that you want to change:

ACELIM

Specifies the maximum size of the ACE storage pool in 1 KB blocks. The number must be in the range 0-999999. The default value of zero means no imposed constraint, beyond what is available in the system.

You should only set a value for ACELIM on queue managers that have been identified as using exorbitant quantities of ECSA storage. Limiting the ACE storage pool has the effect of limiting the number of connections in the system, and so, the amount of ECSA storage used by a queue manager.

Once the queue manager reaches the limit it is not possible for applications to obtain new connections. The lack of new connections causes failures in MQCONN processing, and applications coordinated through RRS are likely to experience failures in any IBM MQ API.

An ACE represents approximately 8% of the total ECSA required for the thread-related control blocks for a connection. So, for example, specifying ACELIM=5120 would be expected to cap the total amount of ECSA allocated by the queue manager (for thread-related control blocks) at approximately 64000K; that is 5120 multiplied by 12.5.

In order to cap the amount total amount of ECSA allocated by the queue manager, for thread-related control blocks at 5120K, an ACELIM value of 410 is required.

You can use SMF 115 subtype 5 records, produced by statistics CLASS(3) trace, to monitor the size of the 'ACE/PEB' storage pool, and hence set an appropriate value for ACELIM.

You can obtain the total amount of ECSA storage used by the queue manager, for control blocks, from SMF 115 subtype 7 records, written by statistics CLASS(2) trace; that is the first two elements in QSRSPHBT added together.

Note that, you should consider setting ACELIM as a mechanism to protect a z/OS image from a badly behaving queue manager, rather than as a means to control application connections to a queue manager.

EXCLMSG

Specify a list of message identifiers to be excluded from being written to any log. Messages in this list are not sent to the z/OS console and hardcopy log. As a result using the EXCLMSG parameter to exclude messages is more efficient from a CPU perspective than using z/OS mechanisms such as the message processing facility list and should be used instead where possible. This list is dynamic and is updated using the SET SYSTEM command.

The default value is an empty list ( ).

Message identifiers are supplied without the CSQ prefix and without the action code suffix (I-D-E-A). For example, to exclude message CSQX500I, add X500 to this list. This list can contain a maximum of 16 message identifiers.

To be eligible to be included in the list, the message must be issued after normal startup of the MSTR or CHIN address spaces and begin with the one of the following characters E, H, I, J, L, M, N, P, R, T, V, W, X, Y, 2 ,3, 5, 9.

Message identifiers that are issued as a result of processing commands can be added to the list, however are not excluded.

For example:

```
SET SYSTEM EXCLMSG(X511,X512)
```

suppresses the channel started and channel no longer active messages.

LOGLOAD
Specifies the number of log records that IBM MQ writes between the start of one checkpoint and the next. IBM MQ starts a new checkpoint after the number of records that you specify has been written.

Specify a value in the range 200 through 16 000 000.

SERVICE
This parameter is reserved for use by IBM.

STATIME
Specifies the interval, in minutes, between consecutive gatherings of statistics.

Specify a number in the range zero through 1440.

If you specify a value of zero, both statistics data and accounting data is collected at the SMF data collection broadcast.

TRACTBL
Specifies the default size, in 4 KB blocks, of trace table where the global trace facility stores IBM MQ trace records.

Specify a value in the range 1 through 999.

Note: Storage for the trace table is allocated in the ECSA. Therefore, you must select this value with care.

START CHANNEL:

Use the MQSC command START CHANNEL to start a channel.

Using MQSC commands

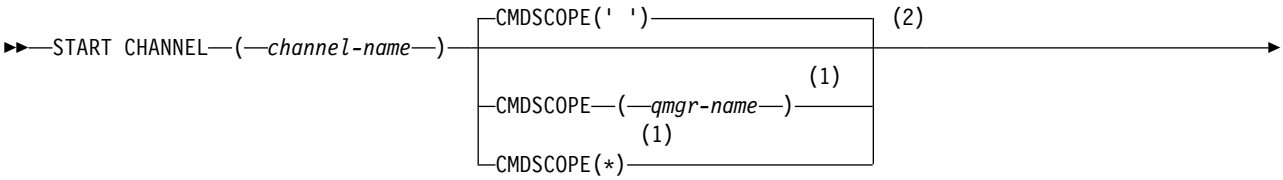
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes" on page 1030
- "Parameter descriptions for START CHANNEL" on page 1030

Synonym: STA CHL

START CHANNEL



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes

1. z/OS On z/OS, the command server and the channel initiator must be running.
2. This command can be issued to a channel of any type except CLNTCONN channels (including those that have been defined automatically). If, however, it is issued to a receiver (RCVR), server-connection (SVRCONN) or cluster-receiver (CLUSRCVR) channel, the only action is to enable the channel, not to start it.
3. Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel. If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the channel that was last added to the local queue manager's repository.

Parameter descriptions for START CHANNEL

(channel-name)
The name of the channel definition to be started. This is required for all channel types. The name must be that of an existing channel.

z/OS **CHLDISP**
This parameter applies to z/OS only and can take the values of:

- DEFAULT
- PRIVATE
- SHARED
- FIXSHARED

If this parameter is omitted, then the DEFAULT value applies. This is taken from the default channel disposition attribute, DEFCDISP, of the channel object.

In conjunction with the various values of the CMDSCOPE parameter, this parameter controls two types of channel:

SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of SHARED.

PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than SHARED.

Note: This disposition is not related to the disposition set by the disposition of the queue-sharing group of the channel definition.

The combination of the CHLDISP and CMDSCOPE parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.
- On every active queue manager in the group.
- On the most suitable queue manager in the group, determined automatically by the queue manager itself.

The various combinations of CHLDISP and CMDSCOPE are summarized in the following table:

Table 119. CHLDISP and CMDSCOPE for START CHANNEL

CHLDISP	**CMDSCOPE( ) or CMDSCOPE (local-qmgr)**	**CMDSCOPE (qmgr-name)**	**CMDSCOPE(*)**
PRIVATE	Start as a private channel on the local queue manager	Start as a private channel on the named queue manager	Start as a private channel on all active queue managers
SHARED	For a shared SDR, RQSTR, and SVR channel, start as a shared channel on the most suitable queue manager in the group. For a shared RCVR and SVRCONN channel, start the channel as a shared channel on all active queue managers. For a shared CLUSSDR or CLUSRCVR channel, this option is not permitted. This might automatically generate a command using CMDSCOPE and send it to the appropriate queue managers. If there is no definition for the channel on the queue managers to which the command is sent, or if the definition is unsuitable for the command, the action fails there. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is actually run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted	Not permitted
FIXSHARED	For a shared SDR, RQSTR, and SVR channel, with a nonblank CONNAME, start as a shared channel on the local queue manager. For all other types, this option is not permitted.	For a shared SDR, RQSTR, and SVR with a nonblank CONNAME, start as a shared channel on the named queue manager. For all other types, this option is not permitted.	Not permitted

Channels started with CHLDISP(FIXSHARED) are tied to the specific queue manager; if the channel initiator on that queue manager stops for any reason, the channels are not recovered by another queue manager in the group. For more information about SHARED and FIXSHARED channels, see Starting a shared channel.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

If CHLDISP is set to SHARED, CMDSCOPE must be blank or the local queue manager.

' '
: The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
: The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

  You can specify a queue manager name only if you are using a queue-sharing group environment and if the command server is enabled.

*
: The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

  This option is not permitted if CHLDISP is FIXSHARED.

START CHANNEL (MQTT): AIX Linux Windows

Use the MQSC command START CHANNEL to start an MQ Telemetry channel.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

The START CHANNEL (MQTT) command is only valid for MQ Telemetry channels. Supported platforms for MQ Telemetry are AIX, Linux, Windows.

Synonym: STA CHL

START CHANNEL

Parameter descriptions for START CHANNEL

(channel-name)
: The name of the channel definition to be started. The name must be that of an existing channel.

CHLTYPE
: Channel type. The value must be MQTT.

START CHINIT on z/OS: z/OS

Use the MQSC command START CHINIT to start a channel initiator.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

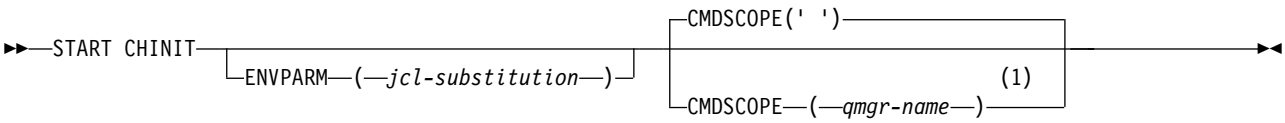
You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes"
- "Parameter descriptions for START CHINIT"

Synonym: STA CHI

Syntax diagram

START CHINIT



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

Usage notes

1. The command server must be running.
2. Although START CHINIT is permitted from CSQINP2, its processing is not complete (and the channel initiator is not available) until after CSQINP2 processing has finished. For these commands, consider using CSQINPX instead.

Parameter descriptions for START CHINIT

CMDSCOPE

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

'' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

ENVPARM(*jcl-substitution*)

The parameters and values to be substituted in the JCL procedure (xxxxCHIN, where xxxx is the queue manager name) that is used to start the channel initiator address space.

jcl-substitution

One or more character strings of the form `keyword=value` enclosed in single quotation marks. If you use more than one character string, separate the strings by commas and enclose the entire list in single quotation marks, for example ENVPARM('HLQ=CSQ,VER=520').

This parameter is valid only on z/OS.

INITQ(*string*)

The name of the initiation queue for the channel initiation process. This is the initiation queue that is specified in the definition of the transmission queue.

The initiation queue on z/OS is always SYSTEM.CHANNEL.INITQ).

Related information:

Command resource security checking for alias queues and remote queues

START CMDSERV on z/OS: z/OS

Use the MQSC command START CMDSERV to initialize the command server.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources `12C`. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for START CMDSERV"

Synonym: STA CS

START CMDSERV

Usage notes for START CMDSERV

1. START CMDSERV starts the command server and allows it to process commands in the system-command input queue (SYSTEM.COMMAND.INPUT), mover commands, and commands using CMDSCOPE.
2. If this command is issued through the initialization files or through the operator console before work is released to the queue manager (that is, before the command server is started automatically), it overrides any earlier STOP CMDSERV command and allows the queue manager to start the command server automatically by putting it into an ENABLED state.
3. If this command is issued through the operator console while the command server is in a STOPPED or DISABLED state, it starts the command server and allows it to process commands on the system-command input queue, mover commands, and commands using CMDSCOPE immediately.
4. If the command server is in a RUNNING or WAITING state (including the case when the command is issued through the command server itself), or if the command server has been stopped automatically because the queue manager is closing down, no action is taken, the command server remains in its current state, and an error message is returned to the command originator.
5. START CMDSERV can be used to restart the command server after it has been stopped, either because of a serious error in handling command messages, or commands using the CMDSCOPE parameter.

START LISTENER:

Use the MQSC command START LISTENER to start a channel listener.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

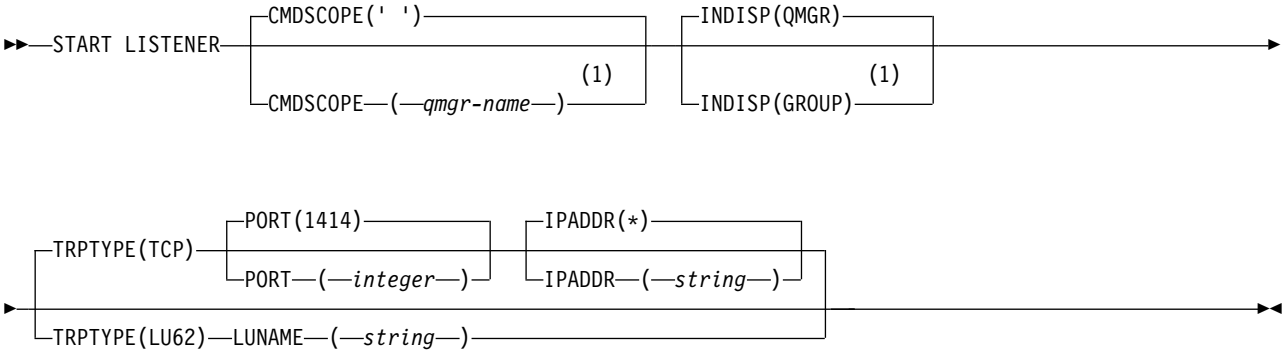
- z/OS Syntax diagram for IBM MQ for z/OS
- Syntax diagram for IBM MQ on other platforms
- "Usage notes"
- "Parameter descriptions for START LISTENER" on page 1036

Synonym: STA LSTR

z/OS

IBM MQ for z/OS

START LISTENER

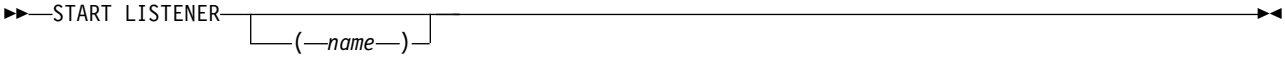


Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

IBM MQ on other platforms

START LISTENER



Usage notes

1. z/OS On z/OS:
   a. The command server and the channel initiator must be running.
   b. If IPADDR is not specified, the listener listens on all available IPv4 and IPv6 addresses.
   c. For TCP/IP, it is possible to listen on multiple addresses and port combinations.

d. For each START LISTENER for TCP/IP request, the address and port combination is added to the list of combinations upon which the listener is currently listening.

e. A START LISTENER for TCP/IP request fails if it specifies the same, or a subset or superset of an existing, combination of addresses and ports upon which a TCP/IP listener is currently listening.

f. If you are starting a listener on a specific address to provide a secure interface with a security product, for example a firewall, it is important to ensure there is no linkage to the other non-secure interfaces in the system.

   You should disable IP forwarding and routing from other non-secure interfaces so that packets arriving at the other interface do not get passed to this specific address.

   Consult the appropriate TCP/IP documentation for information on how to do this.

2. On IBM i, UNIX, and Windows, this command is valid only for channels for which the transmission protocol (TRPTYPE) is TCP.

Parameter descriptions for START LISTENER

(*name*)
Name of the listener to be started. If you specify this parameter, you cannot specify any other parameters.

If you do not specify a name z/OS (on platforms other than z/OS), the SYSTEM.DEFAULT.LISTENER.TCP is started.

z/OS This parameter is not valid on z/OS.

z/OS **CMDSCOPE**
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

z/OS **INDISP**
Specifies the disposition of the inbound transmissions that are to be handled. The possible values are:

QMGR
Listen for transmissions directed to the queue manager. This is the default.

GROUP
Listen for transmissions directed to the queue-sharing group. This is allowed only if there is a shared queue manager environment.

This parameter is valid only on z/OS.

z/OS **IPADDR**
IP address for TCP/IP specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric form. This is valid only if the transmission protocol (TRPTYPE) is TCP/IP.

This parameter is valid only on z/OS.

z/OS **LUNAME(*string*)**

The symbolic destination name for the logical unit as specified in the APPC side information data set. (This must be the same LU that was specified for the queue manager, using the LUNAME parameter of the ALTER QMGR command.)

This parameter is valid only for channels with a transmission protocol (TRPTYPE) of LU 6.2. A START LISTENER command that specifies TRPTYPE(LU62) must also specify the LUNAME parameter.

This parameter is valid only on z/OS.

z/OS **PORT(*port-number*)**

Port number for TCP. This is valid only if the transmission protocol (TRPTYPE) is TCP.

This parameter is valid only on z/OS.

z/OS **TRPTYPE**

Transport type to be used. This is optional.

TCP TCP. This is the default if TRPTYPE is not specified.

LU62 SNA LU 6.2.

This parameter is valid only on z/OS.

START QMGR on z/OS: z/OS

Use the MQSC command START QMGR to initialize the queue manager.

Using MQSC commands

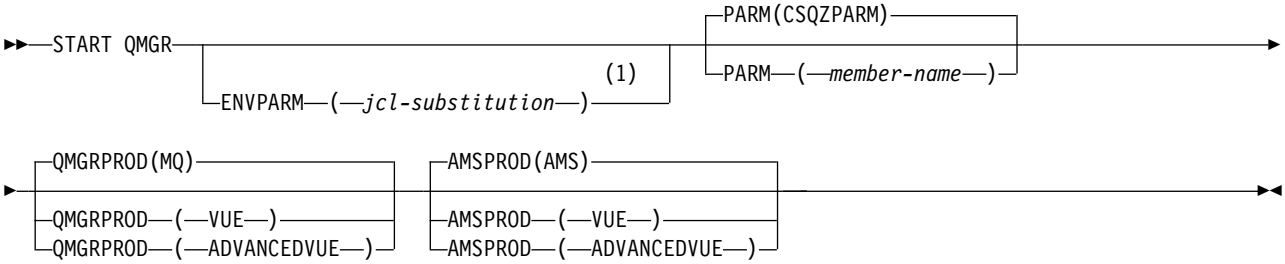
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from source C. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes" on page 1038
- "Parameter descriptions for START QMGR" on page 1038

Synonym: STA QMGR

START QMGR



Notes:

1 MSTR is accepted as a synonym for ENVPARM

Usage notes

When the command has been completed, the queue manager is active and available to CICS, IMS, batch, and TSO applications.

V 9.0.3 New startup parameters **QMGRPROD** and **AMSPROD** have been added to indicate against which product that component should have its usage recorded.

V 9.0.3 You can specify the attribute for the queue manager:

- As a parameter on the START QMGR command
- As a part of the PARM on the EXEC PGM statement in the MSTR JCL procedure
- As part of the compiled queue manager ZPARMS, using the CSQ6USGP macro
- As a default value if not specified elsewhere.

V 9.0.3 If you specify the attribute by more than one of the above mechanisms, the order of the items in the preceding list defines the order of precedence from highest to lowest. The default value is used if you do not explicitly specify an attribute.

V 9.0.3 If you specify an attribute that is not valid, an error message is issued and queue manager startup ends.

Parameter descriptions for START QMGR

These are optional.

ENVPARM(*jcl-substitution* **)**
The parameters and values to be substituted in the JCL procedure (xxxxMSTR, where xxxx is the queue manager name) that is used to start the queue manager address space.

jcl-substitution
One or more character strings of the form:

```
keyword=value
```

enclosed in single quotation marks. If you use more than one character string, separate the strings by commas and enclose the entire list in single quotation marks, for example ENVPARM('HLQ=CSQ,VER=520').

MSTR is accepted as a synonym for ENVPARM

PARM(*member-name* **)**
The load module that contains the queue manager initialization parameters. *member-name* is the name of a load module provided by the installation.

The default is CSQZPARM, which is provided by IBM MQ.

V 9.0.3 **QMGRPROD**
Specifies the type of product ID against which the queue manager usage is to be recorded. The value can be one of the following:

MQ The queue manager is a stand-alone IBM MQ for z/OS product, with product ID 5655␣MQ9. This is the default value if the IBM MQ for z/OS Value Unit Edition (VUE) is not installed.

VUE The queue manager is a stand-alone VUE product, with product ID 5655␣VU9. This is the default value if the IBM MQ for z/OS Value Unit Edition (VUE) is installed.

ADVANCEDVUE
The queue manager is part of an IBM MQ Advanced for z/OS, Value Unit Edition product, with product ID 5655⌂AV1.

V 9.0.3 **AMSPROD**
Specifies the type of product ID against which the queue manager usage is to be recorded. The value can be one of the following:

AMS Advanced Message Security (AMS) is a stand-alone Advanced Message Security for z/OS product, with product ID 5655⌂AM9. This is the default value, unless the attribute for the queue manager indicates IBM MQ Advanced for z/OS, Value Unit Edition.

ADVANCED
AMS is part of an IBM MQ Advanced for z/OS product, with product ID 5655⌂AV9.

ADVANCEDVUE
AMS is part of an IBM MQ Advanced for z/OS, Value Unit Edition product, with product ID 5655⌂AV1. This is the default value, if the attribute for the queue manager is also **ADVANCEDVUE**.

START SERVICE on Multiplatforms: Multi

Use the MQSC command **START SERVICE** to start a service. The identified service definition is started within the queue manager and inherits the environment and security variables of the queue manager.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Parameter descriptions for START SERVICE"

Synonym:

START SERVICE

Parameter descriptions for START SERVICE

(*service-name*)
The name of the service definition to be started. This is required. The name must that of an existing service on this queue manager.

If the service is already running, and the operating system task is active, an error is returned.

Related information:
Working with services
Managing services
Examples of using service objects

START SMDSCONN on z/OS: z/OS

Use the MQSC command START SMDSCONN to enable a previously stopped connection from this queue manager to the specified shared message data sets, allowing them to be allocated and opened again.

Using MQSC commands

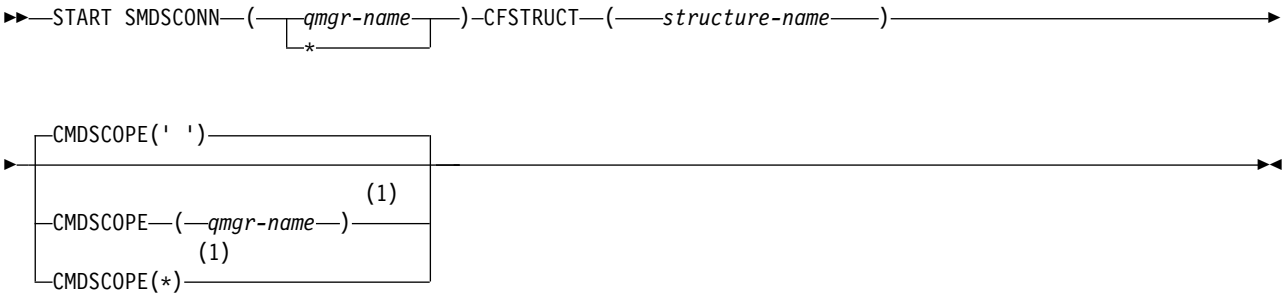
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for START SMDSCONN"

Synonym:

START SMDSCONN



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Parameter descriptions for START SMDSCONN

This command is used after connections have been put into the AVAIL(STOPPED) state by a previous STOP SMDSCONN command. It can also be used to signal to the queue manager to retry a connection which is in the AVAIL(ERROR) state after a previous error.

SMDSCONN(*qmgr-name* | *)
Specify the queue manager which owns the shared message data set for which the connection is to be started or an asterisk to start connections to all shared message data sets associated with the specified structure.

CFSTRUCT(*structure-name*)
Specify the structure name for which shared message data set connections are to be started.

CMDSCOPE
This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

START TRACE on z/OS: z/OS

Use the MQSC command START TRACE to start traces.

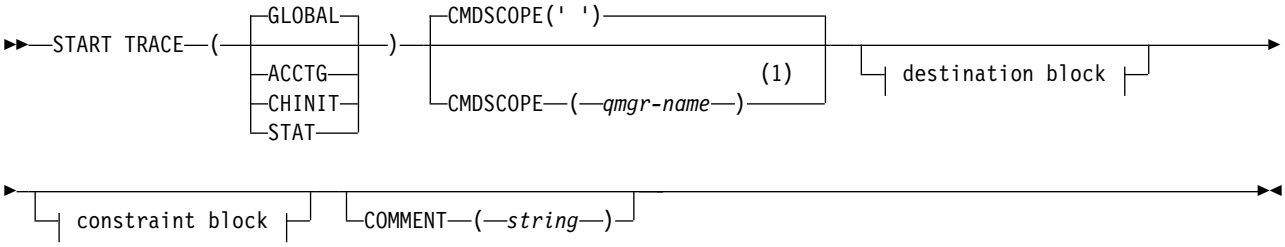
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 12CR. For an explanation of the source symbols, see Using commands on z/OS.

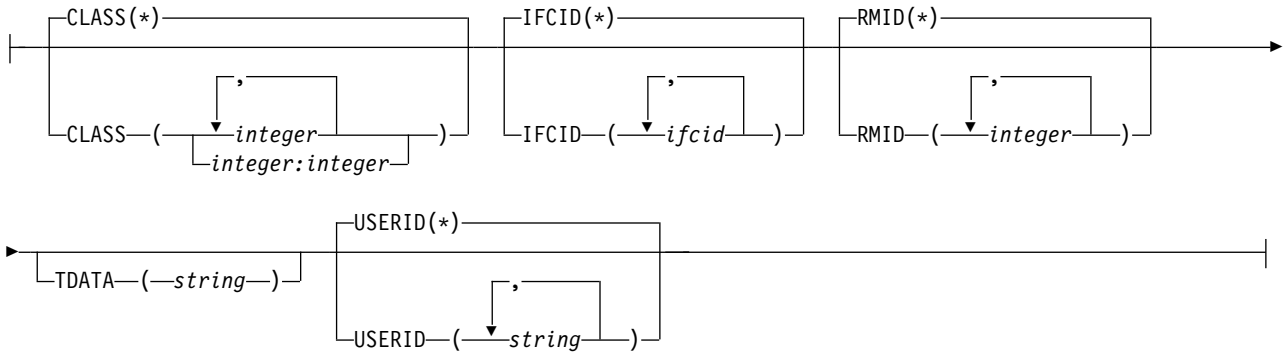
Synonym: STA TRACE

START TRACE



Destination block:

Constraint block:



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Usage notes

When you issue this command, a trace number is returned in message number CSQW130I. You can use this trace number (TNO) in ALTER TRACE, DISPLAY TRACE, and STOP TRACE commands.

Parameter descriptions for START TRACE

If you do not specify a trace type to be started, the default (GLOBAL) trace is started. The types are:

ACCTG
Collects accounting data that can be used to charge your customers for their use of your queue manager. The synonym is A.

Class(4) statistics are supported for channel accounting, that is, SMF116 subtype 10 records.

Note: Accounting data can be lost if the accounting trace is started or stopped while applications are running. For information about the conditions that must be satisfied for successful collection of accounting data, see Using IBM MQ trace.

CHINIT
This includes data from the channel initiator. The synonym is CHI or DQM. If tracing for the channel initiator is started, it stops if the channel initiator stops.

Note that you cannot issue START TRACE(CHINIT) if the command server or the channel initiator is not running.

GLOBAL
This includes data from the entire queue manager except the channel initiator. The synonym is G.

STAT Collects statistical data broadcast by various components of IBM MQ, at time intervals that can be chosen during installation. The synonym is S.

Class(4) statistics are supported for channel initiator statistics, that is, SMF115 subtype 231 records.

CMDSCOPE
This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE cannot be used for commands issued from the first initialization input data set CSQINP1.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

COMMENT(*string*)
Specifies a comment that is reproduced in the trace output record (except in the resident trace tables). It can be used to record why the command was issued.

string is any character string. It must be enclosed in single quotation marks if it includes a blank, comma, or special character.

Destination block

DEST Specifies where the trace output is to be recorded. More than one value can be specified, but do not use the same value twice.

The meaning of each value is as follows:

GTF The z/OS Generalized Trace Facility (GTF). If used, the GTF must be started and accepting user (USR) records before the START TRACE command is issued.

RES A wrap-around table residing in the ECSA, or a data space for CHINIT.

SMF The System Management Facility (SMF). If used, the SMF must be functioning before the START TRACE command is issued. The SMF record numbers used by IBM MQ are 115 and 116. For SMF record type 115, subtypes 1, 2, and 215 are provided for the performance statistics trace.

SRV A serviceability routine reserved for IBM use only; not for general use.

Note: If your IBM support center need you to use this destination for your trace data they will supply you with module CSQWVSER. If you try to use destination SRV without CSQWVSER an error message is produced at the z/OS console when you issue the START TRACE command.

Allowed values, and the default value, depend on the type of trace started, as shown in the following table:

Table 120. Destinations allowed for each trace type

Type	GTF	RES	SMF	SRV
GLOBAL	Allowed	Default	No	Allowed
STAT	No	No	Default	Allowed
ACCTG	Allowed	No	Default	Allowed
CHINIT	No	Default	No	Allowed

Constraint block

The constraint block places optional constraints on the kinds of data collected by the trace. The allowed constraints depend on the type of trace started, as shown in the following table:

Table 121. Constraints allowed for each trace type

Type	CLASS	IFCID	RMID	USERID
GLOBAL	Allowed	Allowed	Allowed	Allowed
STAT	Allowed	No	No	No
ACCTG	Allowed	No	No	No
CHINIT	Allowed	Allowed	No	No

CLASS

Introduces a list of classes of data gathered. The classes allowed, and their meaning, depend on the type of trace started:

(*) Starts a trace for all classes of data.

(*integer*)

Any number in the class column of the table that follows. You can use more than one of the classes that are allowed for the type of trace started. A range of classes can be specified as *m:n* (for example, CLASS(01:03)). If you do not specify a class, the default is to start class 1.

Table 122. Descriptions of trace events and classes.

Table showing the different trace events produced for the various trace classes.

Class	Description
	Global trace
01	Reserved for IBM service
02	User parameter error detected in a control block
03	User parameter error detected on entry to MQI
	User parameter error detected on exit from MQI
	User parameter error detected in a control block
04	Reserved for IBM service
	Statistics trace
01	Subsystem statistics
	Queue manager statistics
02	Queue manager storage summary statistics
03	Queue manager storage detail summary
04	Channel initiator statistics
	Accounting trace
01	The processor time spent processing MQI calls and a count of MQPUT, MQPUT1 and MQGET calls
03	Enhanced accounting and statistical data
04	Channel accounting data
	CHINIT trace
01	Reserved for IBM service
04	Reserved for IBM service

IFCID Reserved for IBM service.

RMID Introduces a list of specific resource managers for which trace information is gathered. You cannot use this option for STAT, ACCTG, or CHINIT traces.

(*) Starts a trace for all resource managers.

This is the default.

(*integer*)

The identifying number of any resource manager in the following table. You can use up to 8 of the allowed resource manager identifiers; do not use the same one twice.

Table 123. Resource Manager identifiers that are allowed

RMID	Resource manager
1	Initialization procedures
2	Agent services management
3	Recovery management
4	Recovery log management
6	Storage management
7	Subsystem support for allied memories
8	Subsystem support for subsystem interface (SSI) functions
12	System parameter management
16	Instrumentation commands, trace, and dump services
23	General command processing
24	Message generator
26	Instrumentation accounting and statistics
148	Connection manager
163	Topic Manager
197	CF manager
199	Functional recovery
200	Security management
201	Data management
211	Lock management
212	Message management
213	Command server
215	Buffer management
242	IBM MQ IMS - bridge
245	Db2 manager

TDATA

Reserved for IBM service.

USERID

Introduces a list of specific user IDs for which trace information is gathered. You cannot use this option for STAT, ACCTG, or CHINIT traces.

(*) Starts a trace for all user IDs. This is the default.

(*userid*)

Names a user ID. You can use up to 8 user IDs; a separate trace is started for each. The user ID is the primary authorization ID of the task, used by IBM MQ inside the queue manager. This is the userid displayed by the MQSC command DISPLAY CONN.

STOP CHANNEL:

Use the MQSC command **STOP CHANNEL** to stop a channel.

Using MQSC commands

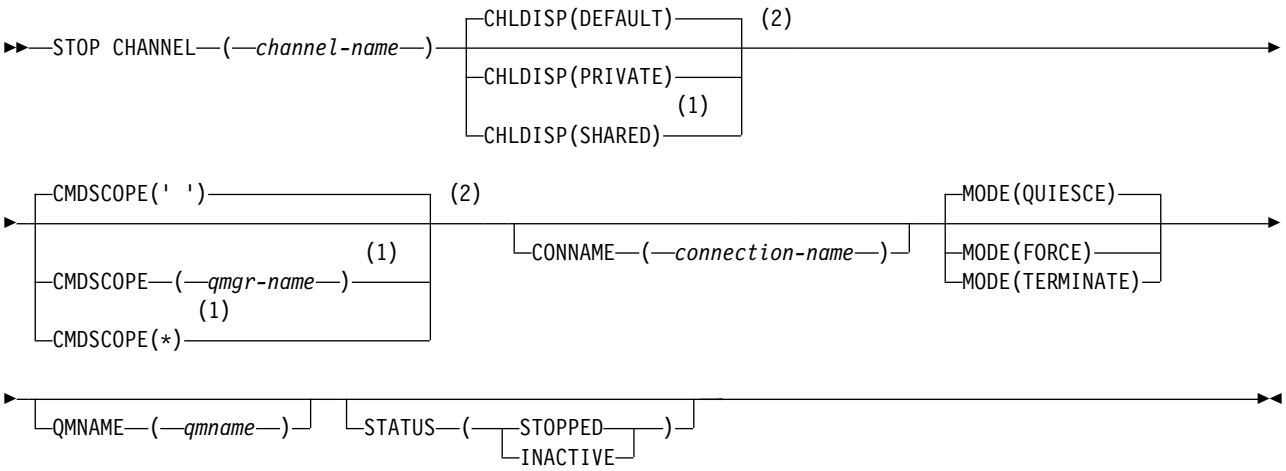
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

z/OS You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for STOP CHANNEL"
- "Parameter descriptions for STOP CHANNEL" on page 1047

Synonym: STOP CHL

STOP CHANNEL



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes for STOP CHANNEL

1. If you specify either QMNAME or CONNAME, STATUS must either be INACTIVE or not specified. Do not specify a QMNAME or CONNAME and STATUS(STOPPED). It is not possible to have a channel stopped for one partner but not for others. This sort of function can be provided by a channel security exit. For more information about channel exits, see Channel exit programs.
2. z/OS On z/OS, the command server and the channel initiator must be running.
3. Any channels in STOPPED state need to be started manually; they are not started automatically. See Restarting stopped channels for information about restarting stopped channels.
4. This command can be issued to a channel of any type except CLNTCONN channels (including those that have been defined automatically).

5. Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel. If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the channel that was last added to the local queue manager repository.

Parameter descriptions for STOP CHANNEL

(channel-name)
The name of the channel to be stopped. This parameter is required for all channel types.

z/OS **CHLDISP**
This parameter applies to z/OS only and can take the values of:

- DEFAULT
- PRIVATE
- SHARED

If this parameter is omitted, then the DEFAULT value applies. This is taken from the default channel disposition attribute, **DEFCDISP**, of the channel object.

In conjunction with the various values of the **CMDSCOPE** parameter, this parameter controls two types of channel:

SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of SHARED.

PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than SHARED.

Note: This disposition is not related to the disposition set by the disposition of the queue-sharing group of the channel definition.

The combination of the **CHLDISP** and **CMDSCOPE** parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.
- On every active queue manager in the group.
- On the most suitable queue manager in the group, determined automatically by the queue manager itself.

The various combinations of **CHLDISP** and **CMDSCOPE** are summarized in the following table:

Table 124. CHLDISP and CMDSCOPE for STOP CHANNEL

CHLDISP	CMDSCOPE( ) or CMDSCOPE (local-qmgr)	CMDSCOPE (qmgr-name)	CMDSCOPE(*)
PRIVATE	Stop as a private channel on the local queue manager.	Stop as a private channel on the named queue manager	Stop as a private channel on all active queue managers

Table 124. CHLDISP and CMDSCOPE for STOP CHANNEL (continued)

CHLDISP	CMDSCOPE( ) or CMDSCOPE (local-qmgr)	CMDSCOPE (qmgr-name)	CMDSCOPE(*)
SHARED	For RCVR and SVRCONN channels, stop as shared channel on all active queue managers. For SDR, RQSTR, and SVR channels, stop as a shared channel on the queue manager where it is running. If the channel is in an inactive state (not running), or if it is in RETRY state because the channel initiator on which it was running has stopped, a STOP request for the channel is issued on the local queue manager. This might automatically generate a command using **CMDSCOPE** and send it to the appropriate queue manager. If there is no definition for the channel on the queue manager to which the command is sent, or if the definition is unsuitable for the command, the command fails. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is actually run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted	Not permitted

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

If **CHLDISP** is set to SHARED, **CMDSCOPE** must be blank or the local queue manager.

' '
The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name only if you are using a queue-sharing group environment and if the command server is enabled.

*
The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

CONNAME *(connection-name)*

Connection name. Only channels matching the specified connection name are stopped.

When issuing the **STOP CHANNEL** command using a **CONNAME** parameter, ensure that the value specified in the **CONNAME** parameter is exactly as shown in "DISPLAY CHSTATUS" on page 783.

MODE

Specifies whether the current batch is allowed to finish in a controlled manner. This parameter is optional.

QUIESCE
This is the default.

Multi On Multiplatforms, allows the current batch to finish processing.

z/OS On z/OS, the channel stops after the current message has finished processing. (The batch is then ended and no more messages are sent, even if there are messages waiting on the transmission queue.)

For a receiving channel, if there is no batch in progress, the channel waits for either of the following to take place before it stops:

- The next batch to start
- The next heartbeat (if heartbeats are being used)

For server-connection channels, allows the current connection to end.
If you issue a `STOP CHANNEL` *`channelname`* `MODE (QUIESCE)` command on a server-connection channel, the IBM MQ client infrastructure becomes aware of the stop request in a timely manner. This time is dependent upon the speed of the network.
If a client application is using the server-connection channel and is performing either of the following operations at the time that the command is issued, then the MQPUT or MQGET operation fails:

- An MQPUT operation with the PMO option MQPMO_FAIL_IF_QUIESCE specified.
- An MQGET operation with the GMO option MQGMO_FAIL_IF_QUIESCE set.

The client application receives reason code MQRC_CONNECTION_QUIESCING.
If a client application is using the server-connection channel and is performing either of the following operations, then the client application is allowed to complete the MQPUT or MQGET operation:

- An MQPUT operation without the PMO option MQPMO_FAIL_IF_QUIESCE specified.
- An MQGET operation without the GMO option MQGMO_FAIL_IF_QUIESCE set.

The next time the application tries to use the server-connection channel, it receives reason code MQRC_CONNECTION_QUIESCING.
If the client application is not performing an MQ API call when the server-connection channel is stopped, it becomes aware of the stop request as a result of issuing a subsequent call to IBM MQ and receives return code MQRC_CONNECTION_QUIESCING.
After sending the MQRC_CONNECTION_QUIESCING return code to the client, and allowing any outstanding MQPUT or MQGET operations to complete if necessary, the server ends the client connections for the server-connection channel.
Due to the imprecise timing of network operations, the client application should not attempt further MQ API operations.

FORCE

For server-connection channels, breaks the current connection, returning MQRC_CONNECTION_BROKEN. For other channel types, terminates transmission of any current batch. This is likely to result in in-doubt situations.

z/OS On IBM MQ for z/OS, specifying **FORCE** interrupts any message reallocation in progress, which might leave BIND_NOT_FIXED messages partially reallocated or out of order.

TERMINATE

z/OS On z/OS, **TERMINATE** is synonymous with **FORCE**.

Multi On other platforms, **TERMINATE** terminates transmission of any current batch.

This allows the command to actually terminate the channel thread or process.

For server-connection channels, **TERMINATE** breaks the current connection, returning MQRC_CONNECTION_BROKEN.

z/OS On z/OS, specifying **TERMINATE** interrupts any message reallocation in progress, which might leave BIND_NOT_FIXED messages partially reallocated or out of order.

QMNAME *(qmname)*
Queue manager name. Only channels matching the specified remote queue manager are stopped.

STATUS
Specifies the new state of any channels stopped by this command. For more information about channels in STOPPED state, especially SVRCONN channels on z/OS, see Restarting stopped channels.

STOPPED
The channel is stopped. For a sender or server channel the transmission queue is set to **GET(DISABLED)** and NOTRIGGER.

This is the default if **QMNAME** or **CONNAME** are not specified.

INACTIVE
The channel is inactive.

This is the default if **QMNAME** or **CONNAME** are specified.

STOP CHANNEL (MQTT): AIX Linux Windows

Use the MQSC command STOP CHANNEL to stop an MQ Telemetry channel.

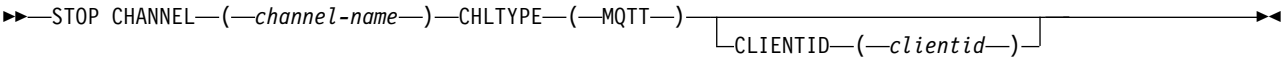
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

The STOP CHANNEL (MQTT) command is only valid for MQ Telemetry channels.

Synonym: STOP CHL

STOP CHANNEL



Usage notes for STOP CHANNEL

1. Any channels in STOPPED state need to be started manually; they are not started automatically.

Parameter descriptions for STOP CHANNEL

(channel-name)
The name of the channel to be stopped. This parameter is required for all channel types including MQTT channels.

CHLTYPE
Channel type. Tthe value must be MQTT.

CLIENTID *(string)*
Client identifier. The client identifier is a 23-byte string that identifies an MQ Telemetry Transport

client. When the STOP CHANNEL command specifies a CLIENTID, only the connection for the specified client identifier is stopped. If the CLIENTID is not specified, all the connections on the channel are stopped.

STOP CHINIT on z/OS: z/OS

Use the MQSC command STOP CHINIT to stop a channel initiator. The command server must be running.

Using MQSC commands

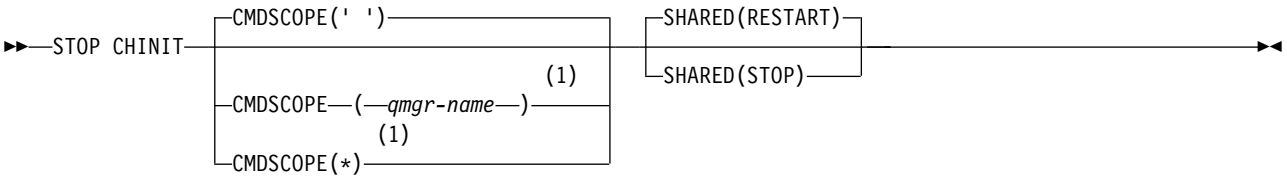
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for STOP CHINIT"
- "Parameter descriptions for STOP CHINIT"

Synonym: STOP CHI

STOP CHINIT



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Usage notes for STOP CHINIT

1. When you issue the STOP CHINIT command, IBM MQ stops any channels that are running in the following way:
   - Sender and server channels are stopped using STOP CHANNEL MODE(QUIESCE) STATUS(INACTIVE)
   - All other channels are stopped using STOP CHANNEL MODE(FORCE)

   See "STOP CHANNEL" on page 1046 for information about what this involves.
2. You might receive communications-error messages as a result of issuing the STOP CHINIT command.

Parameter descriptions for STOP CHINIT

CMDSCOPE

This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

'' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

SHARED

Specifies whether the channel initiator should attempt to restart any active sending channels, started with CHLDISP(SHARED), that it owns on another queue manager. The possible values are:

RESTART
Shared sending channels are to be restarted. This is the default.

STOP Shared sending channels are not to be restarted, so will become inactive.

(Active channels started with CHLDISP(FIXSHARED) are not restarted, and always become inactive.)

STOP CMDSERV on z/OS: z/OS

Use the MQSC command STOP CMDSERV to stop the command server.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 12C. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Usage notes for STOP CMDSERV"

Synonym: STOP CS

STOP CMDSERV

Usage notes for STOP CMDSERV

1. STOP CMDSERV stops the command server from processing commands in the system-command input queue (SYSTEM.COMMAND.INPUT), mover commands, and commands using CMDSCOPE.
2. If this command is issued through the initialization files or through the operator console before work is released to the queue manager (that is, before the command server is started automatically), it prevents the command server from starting automatically and puts it into a DISABLED state. It overrides an earlier START CMDSERV command.
3. If this command is issued through the operator console or the command server while the command server is in a RUNNING state, it stops the command server when it has finished processing its current command. When this happens, the command server enters the STOPPED state.
4. If this command is issued through the operator console while the command server is in a WAITING state, it stops the command server immediately. When this happens, the command server enters the STOPPED state.

5. If this command is issued while the command server is in a DISABLED or STOPPED state, no action is taken, the command server remains in its current state, and an error message is returned to the command originator.

STOP CONN on Multiplatforms: Multi

Use the MQSC command STOP CONN to break a connection between an application and the queue manager.

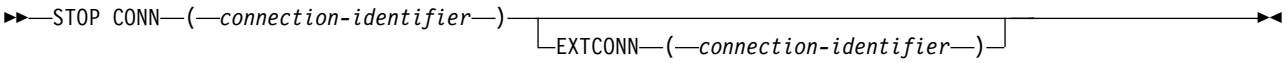
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Usage notes"
- "Parameter descriptions for STOP CONN"

Synonym: STOP CONN

STOP CONN



Usage notes

There might be circumstances in which the queue manager cannot implement this command when the success of this command cannot be guaranteed.

Parameter descriptions for STOP CONN

(*connection-identifier*)
The identifier of the connection definition for the connection to be broken.

When an application connects to IBM MQ, it is given a unique 24-byte connection identifier (ConnectionId). The value of CONN is formed by converting the last eight bytes of the ConnectionId to its 16-character hexadecimal equivalent.

EXTCONN
The value of EXTCONN is based on the first sixteen bytes of the ConnectionId converted to its 32-character hexadecimal equivalent.

Connections are identified by a 24-byte connection identifier. The connection identifier comprises a prefix, which identifies the queue manager, and a suffix which identifies the connection to that queue manager. By default, the prefix is for the queue manager currently being administered, but you can specify a prefix explicitly by using the EXTCONN parameter. Use the CONN parameter to specify the suffix.

When connection identifiers are obtained from other sources, specify the fully qualified connection identifier (both EXTCONN and CONN) to avoid possible problems related to non-unique CONN values.

Related reference:
"DISPLAY CONN" on page 822
Use the MQSC command **DISPLAY CONN** to display connection information about the applications connected to the queue manager. This is a useful command because it enables you to identify applications with long-running units of work.

STOP LISTENER:

Use the MQSC command STOP LISTENER to stop a channel listener.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

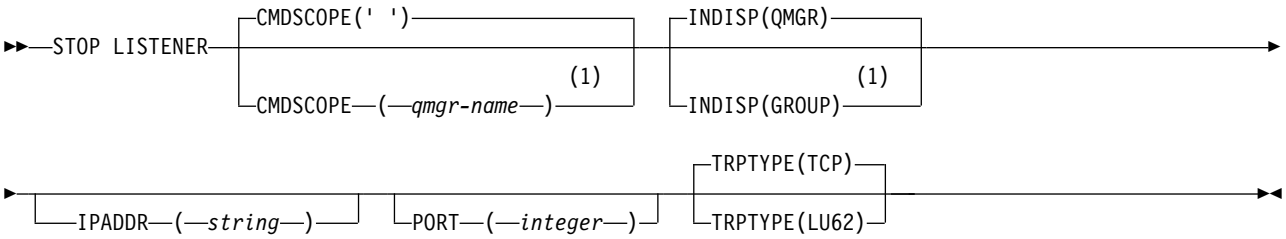
z/OS You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- z/OS Syntax diagram for IBM MQ for z/OS
- Syntax diagram for IBM MQ on other platforms
- z/OS "Usage notes"
- "Parameter descriptions for STOP LISTENER" on page 1055

Synonym: STOP LSTR

z/OS
z/OS

STOP LISTENER

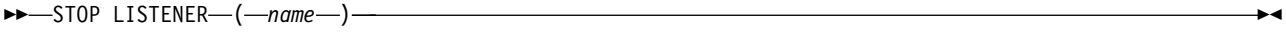


Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

Other platforms

STOP LISTENER



z/OS
Usage notes

On z/OS:

- The command server and the channel initiator must be running.

- If a listener is listening on multiple addresses or ports, only the address and port combinations with the address, or port, specified are stopped.
- If a listener is listening on all addresses for a particular port, a stop request for a specific IPADDR with the same port fails.
- If neither an address nor a port is specified, all addresses and ports are stopped and the listener task ends.

Parameter descriptions for STOP LISTENER

(*name*)
Name of the listener to be stopped. If you specify this parameter, you cannot specify any other parameters.

This parameter is required on all platforms z/OS other than z/OS where it is not a supported parameter.

z/OS **CMDSCOPE**
This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

This parameter is valid only on z/OS.

z/OS **INDISP**
Specifies the disposition of the inbound transmissions that the listener handles. The possible values are:

QMGR
Handling for transmissions directed to the queue manager. This is the default.

GROUP
Handling for transmissions directed to the queue-sharing group. This is allowed only if there is a shared queue manager environment.

This parameter is valid only on z/OS.

z/OS **IPADDR**
IP address for TCP/IP specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric form. This is valid only if the transmission protocol (TRPTYPE) is TCP/IP.

This parameter is valid only on z/OS.

z/OS **PORT**
The port number for TCP/IP. This is the port number on which the listener is to stop listening. This is valid only if the transmission protocol is TCP/IP.

This parameter is valid only on z/OS.

z/OS **TRPTYPE**
Transmission protocol used. This is optional.

TCP TCP. This is the default if TRPTYPE is not specified.

LU62 SNA LU 6.2.

This parameter is valid only on z/OS.

The listener stops in quiesce mode (it disregards any further requests).

STOP QMGR on z/OS: z/OS

Use the MQSC command STOP QMGR to stop the queue manager.

Using MQSC commands

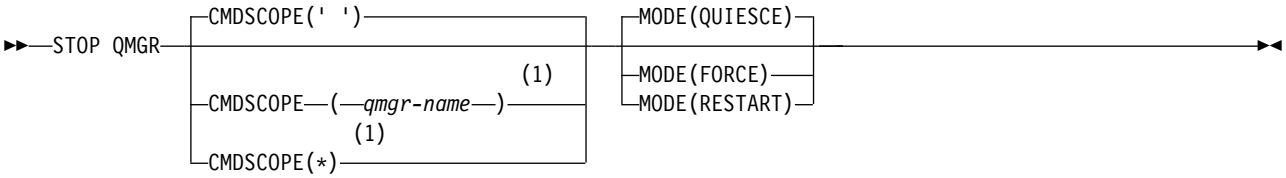
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for STOP QMGR"

Synonym: There is no synonym for this command.

STOP QMGR



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Parameter descriptions for STOP QMGR

The parameters are optional.

CMDSCOPE

This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

MODE

Specifies whether programs currently being executed are allowed to finish.

QUIESCE

Allows programs currently being executed to finish processing. No new program is allowed to start. This is the default.

This option means that all connections to other address spaces must terminate before the queue manager stops. The system operator can determine whether any connections remain by using the DISPLAY CONN command, and can cancel remaining connections using z/OS commands.

This option deregisters IBM MQ from the z/OS automatic restart manager (ARM).

FORCE

Terminates programs currently being executed, including utilities. No new program is allowed to start. This option might cause in-doubt situations.

This option might not work if all the active logs are full, and log archiving has not occurred. In this situation you must issue the z/OS command CANCEL to terminate.

This option deregisters IBM MQ from the z/OS automatic restart manager (ARM).

RESTART

Terminates programs currently being executed, including utilities. No new program is allowed to start. This option might cause in-doubt situations.

This option might not work if all the active logs are full, and log archiving has not occurred. In this situation you must issue the z/OS command CANCEL to terminate.

This option does not deregister IBM MQ from ARM, so the queue manager is eligible for immediate automatic restart.

STOP SERVICE on Multiplatforms: Multi

Use the MQSC command **STOP SERVICE** to stop a service.

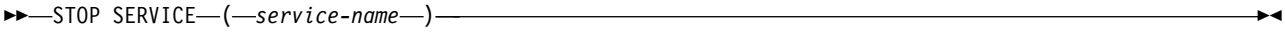
Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

- Syntax diagram
- "Usage notes"
- "Parameter descriptions for STOP SERVICE" on page 1058

Synonym:

STOP SERVICE



Usage notes

If the service is running, it is requested to stop. This command is processed asynchronously so might return before the service has stopped.

If the service that is requested to stop has no STOP command defined, an error is returned.

Parameter descriptions for STOP SERVICE

(*service-name*)
The name of the service definition to be stopped. This is required. The name must that of an existing service on this queue manager.

Related reference:

"ALTER SERVICE on Multiplatforms" on page 526
Use the MQSC command **ALTER SERVICE** to alter the parameters of an existing IBM MQ service definition.

"START SERVICE on Multiplatforms" on page 1039
Use the MQSC command **START SERVICE** to start a service. The identified service definition is started within the queue manager and inherits the environment and security variables of the queue manager.

Related information:

Working with services

Managing services

Examples of using service objects

STOP SMDSCONN on z/OS: z/OS

Use the MQSC command STOP SMDSCONN to terminate the connection from this queue manager to one or more specified shared message data sets (causing them to be closed and deallocated) and to mark the connection as STOPPED.

Using MQSC commands

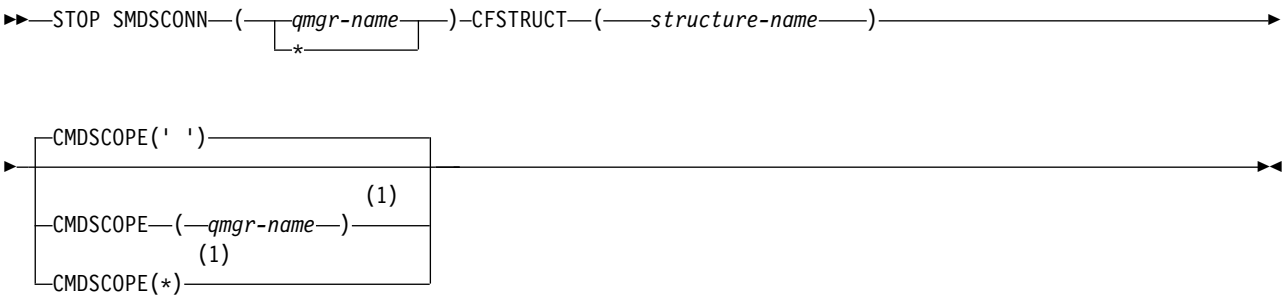
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

You can issue this command from sources 2CR. For an explanation of the source symbols, see Using commands on z/OS.

- "Syntax diagram for STOP SMDSCONN"
- "Parameter descriptions for STOP SMDSCONN" on page 1059

Syntax diagram for STOP SMDSCONN

Synonym:



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

Parameter descriptions for STOP SMDSCONN

SMDSCONN
Specify the queue manager which owns the shared message data set for which the connection is to be stopped, or an asterisk to stop connections to all shared message data sets associated with the specified structure.

CFSTRUCT
Specify the structure name for which shared message data set connections are to be stopped.

CMDSCOPE
This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

'' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

STOP TRACE on z/OS: z/OS

Use the MQSC command STOP TRACE to stop tracing.

Using MQSC commands

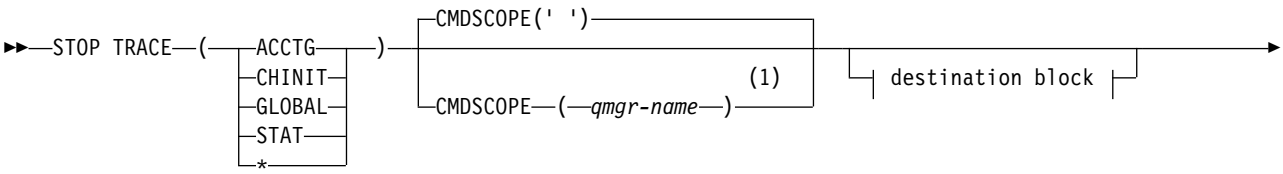
For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

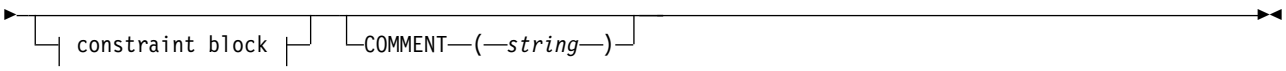
You can issue this command from sources 12CR. For an explanation of the source symbols, see Using commands on z/OS.

- Syntax diagram
- "Parameter descriptions for STOP TRACE" on page 1060
- "Destination block" on page 1061
- "Constraint block" on page 1061

Synonym: There is no synonym for this command.

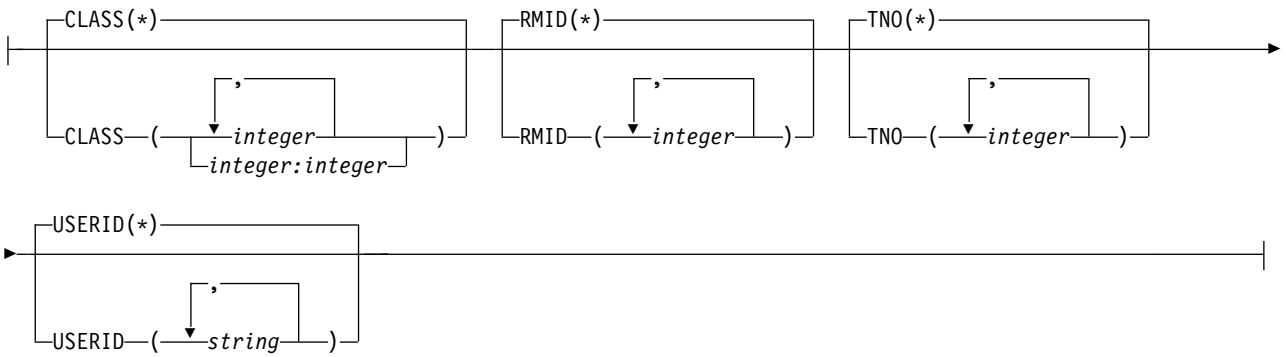
STOP TRACE





Destination block:

Constraint block:



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

Parameter descriptions for STOP TRACE

Each option that you use limits the effect of the command to active traces that were started using the same option, either explicitly or by default, with exactly the same parameter values.

You must specify a trace type or an asterisk. STOP TRACE(*) stops all active traces.

The trace types are:

ACCTG
Accounting data (the synonym is A)

Note: Accounting data can be lost if the accounting trace is started or stopped while applications are running. For information about the conditions that must be satisfied for successful collection of accounting data, see Using IBM MQ trace.

CHINIT
Service data from the channel initiator. The synonym is CHI or DQM.

If the only trace running on the CHINIT is the one started automatically when the CHINIT was started, that tracing can be stopped only by explicitly stating the TNO for the default CHINIT trace (0). For example: `STOP TRACE(CHINIT) TNO(0)`

GLOBAL
Service data from the entire queue manager except for the channel initiator. The synonym is G.

STAT Statistical data (the synonym is S)

***** All active traces

CMDSCOPE

This parameter specifies how the command runs when the queue manager is a member of a queue-sharing group.

CMDSCOPE cannot be used for commands issued from the first initialization input data set CSQINP1.

' ' The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name

The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

COMMENT(*string*)

Specifies a comment that is reproduced in the trace output record (except in the resident trace tables), and can be used to record why the command was issued.

string is any character string. It must be enclosed in single quotation marks if it includes a blank, comma, or special character.

Destination block

DEST Limits the action to traces started for particular destinations. More than one value can be specified, but do not use the same value twice. If no value is specified, the list is not limited.

Possible values and their meanings are:

GTF The Generalized Trace Facility

RES A wrap-around table residing in the ECSA

SMF The System Management Facility

SRV A serviceability routine designed for problem diagnosis

Constraint block

CLASS(*integer*)

Limits the command to traces started for particular classes. See the START TRACE command for a list of allowed classes. A range of classes can be specified as *m:n* (for example, CLASS(01:03)). You cannot specify a class if you did not specify a trace type.

The default is CLASS(*), which does not limit the command.

RMID(*integer*)

Limits the command to traces started for particular resource managers. See the START TRACE command for a list of allowed resource manager identifiers.

Do not use this option with the STAT, ACCTG, or CHINIT trace type.

The default is RMID(*), which does not limit the command.

TNO(*integer*)

Limits the command to particular traces, identified by their trace number (0 to 32). Up to 8 trace numbers can be used. If more than one number is used, only one value for USERID can be used.

0 is the trace that the channel initiator can start automatically. Traces 1 to 32 are those for queue manager or the channel initiator that can be started automatically by the queue manager, or manually, using the START TRACE command.

The default is TNO(*), which applies the command to all active traces with numbers 1 to 32, but **not** to the 0 trace. You can stop trace number 0 only be specifying it explicitly.

USERID(*string* **)**

Limits the action of the STOP TRACE to traces started for particular user ID. Up to 8 user IDs can be used. If more than one user ID is used, only one value can be used for TNO. Do not use this option with the STAT, ACCTG, or CHINIT trace type.

The default is USERID(*), which does not limit the command.

SUSPEND QMGR:

Use the MQSC command SUSPEND QMGR to advise other queue managers in a cluster to avoid sending messages to the local queue manager if possible.

Using MQSC commands

For information on how you use MQSC commands, see Performing local administration tasks using MQSC commands.

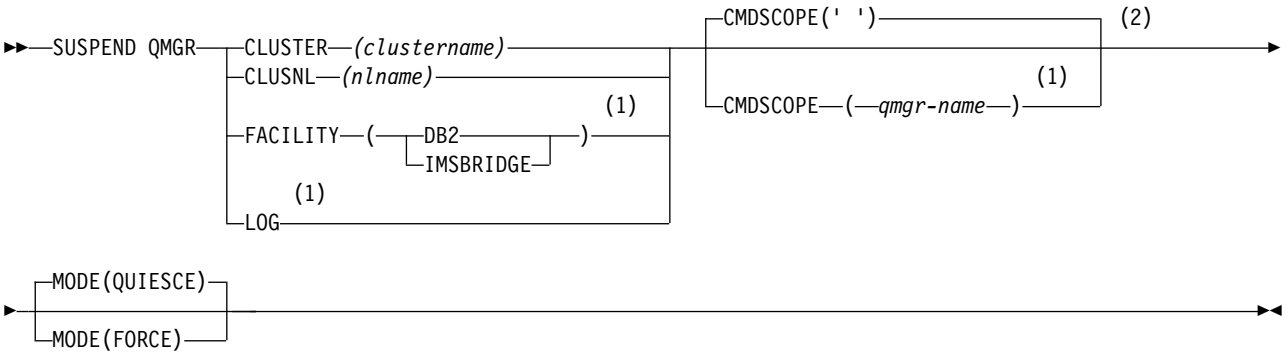
For further details about using the SUSPEND QMGR and RESUME QMGR commands to remove a queue manager from a cluster temporarily, see SUSPEND QMGR, RESUME QMGR and clusters.

z/OS On z/OS this command can also be used to suspend logging and update activity for the queue manager until a subsequent RESUME QMGR command is issued. Its action can be reversed by the RESUME QMGR command. This command does not mean that the queue manager is disabled.

- Syntax diagram
- z/OS See "Using SUSPEND QMGR on z/OS" on page 1063
- z/OS "Usage notes" on page 1063
- "Parameter descriptions for SUSPEND QMGR" on page 1063

Synonym: None

SUSPEND QMGR



Notes:

1 Valid only on z/OS.

2 Valid only on IBM MQ for z/OS when the queue manager is a member of a queue-sharing group.

z/OS

Using SUSPEND QMGR on z/OS

SUSPEND QMGR can be used on z/OS. Depending on the parameters used on the command, it may be issued from various sources. For an explanation of the symbols in this table, see "Using commands on z/OS" on page 365.

Command	Command Sources	Notes
SUSPEND QMGR CLUSTER/CLUSNL	CR	Ensure the channel initiator is running
SUSPEND QMGR FACILITY	CR	
SUSPEND QMGR LOG	C	

z/OS

Usage notes

On z/OS:

- If you define CLUSTER or CLUSNL, be aware of the following behavior:
  - The command fails if the channel initiator has not been started.
  - Any errors are reported to the system console where the channel initiator is running; they are not reported to the system that issued the command.
- The SUSPEND QMGR and RESUME QMGR commands are supported through the console only. However, all the other SUSPEND and RESUME commands are supported through the console and command server.

Parameter descriptions for SUSPEND QMGR

The SUSPEND QMGR with the CLUSTER or CLUSNL parameters to specify the cluster or clusters for which availability is suspended, how the suspension takes effect .

z/OS On z/OS, controls logging and update activity and how the command runs when the queue manager is a member of a queue-sharing group.

You can use the SUSPEND QMGR FACILITY(Db2) to terminate the queue manager connection to Db2. This command might be useful if you want to apply service to Db2. Be aware, if you use this option then there is no access to Db2 resources, for example, large messages which might be offloaded to Db2 from a coupling facility.

z/OS You can use the SUSPEND QMGR FACILITY(IMSBRIDGE) to stop sending messages from the IBM MQ IMS bridge to IMS OTMA. z/OS See Controlling the IMS bridge for more information about controlling message delivery to shared and non-shared queues.

CLUSTER ***(clustername)***
The name of the cluster for which availability is to be suspended.

CLUSNL ***(nlname)***
The name of the namelist that specifies a list of clusters for which availability is to be suspended.

z/OS **FACILITY**
Specifies the facility to which connection is to be terminated. The parameter must have one of the following values:

Db2 Causes the existing connection to Db2 to be terminated. The connection is re-established when the "RESUME QMGR" on page 1000 command is issued. When the Db2 connection

is SUSPENDED, any API requests which must access Db2 to complete will be suspended until the RESUME QMGR FACILITY(Db2) command is issued. API requests include:

- The first MQOPEN of a shared queue since the queue manager started
- MQPUT, MQPUT1 and MQGET to or from a shared queue where the message payload has been offloaded to Db2

z/OS **IMSBRIDGE**

Stops the sending of messages from IMS bridge queues to OTMA. The IMS connection is not affected. When the tasks that transmit messages to IMS have been terminated, no further messages are sent to IMS until one of the following actions happens:

- OTMA or IMS is stopped and restarted
- IBM MQ is stopped and restarted
- A "RESUME QMGR" on page 1000 command is processed

Return messages from IMS OTMA to the queue manager are unaffected.

To monitor progress of the command, issue the following command and ensure that none of the queues are open:

```
DIS Q(*) CMDSCOPE(qmgr) STGCLASS(bridge_stgclass) IPPROCS
```

If any queue is open, use DISPLAY QSTATUS to verify that the MQ-IMS bridge does not have it open.

This parameter is valid only on z/OS.

z/OS **LOG**

Suspends logging and update activity for the queue manager until a subsequent RESUME request is issued. Any unwritten log buffers are externalized, a system checkpoint is taken (non-data sharing environment only), and the BSDS is updated with the high-written RBA before the update activity is suspended. A highlighted message (CSQJ372I) is issued and remains on the system console until update activity has been resumed. Valid on z/OS only. If LOG is specified, the command can be issued only from the z/OS system console.

This option is not permitted when a system quiesce is active by either the ARCHIVE LOG or STOP QMGR command.

Update activity remains suspended until a RESUME QMGR LOG or STOP QMGR command is issued.

This command must not be used during periods of high activity, or for long periods of time. Suspending update activity can cause timing-related events such as lock timeouts or IBM MQ diagnostic memory dumps when delays are detected.

z/OS **CMDSCOPE**

This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

' '
The command runs on the queue manager on which it was entered. This is the default value.

qmgr-name
The command runs on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

MODE

Specifies how the suspension of availability is to take effect:

QUIESCE
Other queue managers in the cluster are advised to avoid sending messages to the local queue manager if possible. It does not mean that the queue manager is disabled.

FORCE
All inbound cluster channels from other queue managers in the cluster are stopped forcibly. This occurs only if the queue manager has also been forcibly suspended from all other clusters to which the cluster receiver channel for this cluster belongs.

The MODE keyword is permitted only with CLUSTER or CLUSNL. It is not permitted with the LOG or FACILITY parameter.

Related reference:

"RESUME QMGR" on page 1000
Use the MQSC command RESUME QMGR to inform other queue managers in a cluster that the local queue manager is available again for processing and can be sent messages. It reverses the action of the SUSPEND QMGR command.

Related information:

SUSPEND QMGR, RESUME QMGR and clusters
Use the `SUSPEND QMGR` and `RESUME QMGR` command to temporarily reduce the inbound cluster activity to this queue manager, for example, before you perform maintenance on this queue manager, and then reinstate it.

CL commands reference for IBM i

A list of CL commands for IBM i, grouped according to command type.

- Authentication Information Commands
  - CHGMQMAUTI, Change IBM MQ Authentication Information
  - CPYMQMAUTI, Copy IBM MQ Authentication Information
  - CRTMQMAUTI, Create IBM MQ Authentication Information
  - DLTMQMAUTI, Delete IBM MQ Authentication Information
  - DSPMQMAUTI, Display IBM MQ Authentication Information
  - WRKMQMAUTI, Work with IBM MQ Authentication Information
- Authority Commands
  - DSPMQMAUT, Display IBM MQ Object Authority
  - GRTMQMAUT, Grant IBM MQ Object Authority
  - RFRMQMAUT, Refresh IBM MQ Object Authority
  - RVKMQMAUT, Revoke IBM MQ Object Authority
  - WRKMQMAUT, Work with IBM MQ Authority
  - WRKMQMAUTD, Work with IBM MQ Authority Data
- Broker Commands

  The following commands do not perform any function and are only provided for compatibility with previous releases of IBM MQ.
  - CLRMQMBRK, Clear IBM MQ Broker
  - DLTMQMBRK, Delete IBM MQ Broker
  - DSPMQMBRK, Display IBM MQ Pub/Sub Broker
  - DSPMQMBRK, Display IBM MQ Broker
  - ENDMQMBRK, End IBM MQ Broker
  - STRMQMBRK, Start IBM MQ Broker

- Channel Commands
  - CHGMQMCHL, Change IBM MQ Channel
  - CPYMQMCHL, Copy IBM MQ Channel
  - CRTMQMCHL, Create IBM MQ Channel
  - DLTMQMCHL, Delete IBM MQ Channel
  - DSPMQMCHL, Display IBM MQ Channel
  - ENDMQMCHL, End IBM MQ Channel
  - PNGMQMCHL, Ping IBM MQ Channel
  - RSTMQMCHL, Reset IBM MQ Channel
  - RSVMQMCHL, Resolve IBM MQ Channel
  - STRMQMCHL, Start IBM MQ Channel
  - STRMQMCHLI, Start IBM MQ Channel Initiator
  - WRKMQMCHL, Work with IBM MQ Channels
  - WRKMQMCHST, Work with IBM MQ Channel Status
- Cluster Commands
  - RFRMQMCL, Refresh IBM MQ Cluster
  - RSMMQMCLQM, Resume IBM MQ Cluster Queue Manager
  - RSTMQMCL, Reset IBM MQ Cluster
  - SPDMQMCLQM, Suspend IBM MQ Cluster Queue Manager
  - WRKMQMCL, Work with IBM MQ Clusters
  - WRKMQMCLQ, Work with IBM MQ Cluster Queues
- Command Server Commands
  - DSPMQMCSVR, Display IBM MQ Command Server
  - ENDMQMCSVR, End IBM MQ Command Server
  - STRMQMCSVR, Start IBM MQ Command Server
- Connection Commands
  - ENDMQMCONN, End IBM MQ Connection
  - WRKMQMCONN, Work with IBM MQ Connections
- Data Conversion Exit Command
  - CVTMQMDTA, Convert IBM MQ Data Type
- Listener Commands
  - CHGMQMLSR, Change IBM MQ Listener Object
  - CPYMQMLSR, Copy IBM MQ Listener Object
  - CRTMQMLSR, Create IBM MQ Listener Object
  - DLTMQMLSR, Delete IBM MQ Listener Object
  - DSPMQMLSR, Display IBM MQ Listener Object
  - ENDMQMLSR, End IBM MQ Listener
  - STRMQMLSR, Start IBM MQ Listener
  - WRKMQMLSR, Work with IBM MQ Listeners
- Media Recovery Commands
  - RCDMQMIMG, Record IBM MQ Object Image
  - RCRMQMOBJ, Re-create IBM MQ Object
  - WRKMQMTRN, Work with IBM MQ Transactions
- Name Command
  - DSPMQMOBJN, Display IBM MQ Object Names

- Namelist Commands
  - CHGMQMNL, Change IBM MQ Namelist
  - CPYMQMNL, Copy IBM MQ Namelist
  - CRTMQMNL, Create IBM MQ Namelist
  - DLTMQMNL, Delete IBM MQ Namelist
  - DSPMQMNL, Display IBM MQ Namelist
  - WRKMQMNL, Work with IBM MQ Namelists
- Process Commands
  - CHGMQMPRC, Change IBM MQ Process
  - CPYMQMPRC, Copy IBM MQ Process
  - CRTMQMPRC, Create IBM MQ Process
  - DLTMQMPRC, Delete IBM MQ Process
  - DSPMQMPRC, Display IBM MQ Process
  - WRKMQMPRC, Work with IBM MQ Processes
- Queue Commands
  - CHGMQMQ, Change IBM MQ Queue
  - CLRMQMQ, Clear IBM MQ Queue
  - CPYMQMQ, Copy IBM MQ Queue
  - CRTMQMQ, Create IBM MQ Queue
  - DLTMQMQ, Delete IBM MQ Queue
  - DSPMQMQ, Display IBM MQ Queue
  - WRKMQMMSG, Work with IBM MQ Messages
  - WRKMQMQ, Work with IBM MQ Queues
  - WRKMQMQSTS, Work with IBM MQ Queue Status
- Queue Manager Commands
  - CCTMQM, Connect to Message Queue Manager
  - CHGMQM, Change Message Queue Manager
  - CRTMQM, Create Message Queue Manager
  - DLTMQM, Delete Message Queue Manager
  - DSCMQM, Disconnect from Message Queue Manager
  - DSPMQM, Display Message Queue Manager
  - DSPMQMSTS, Display Message Queue Manager Status
  - ENDMQM, End Message Queue Manager
  - RFRMQM, Refresh Message Queue Manager
  - STRMQM, Start Message Queue Manager
  - STRMQMTRM, Start IBM MQ Trigger Monitor
  - WRKMQM, Work with Message Queue Manager
- Service Commands
  - CHGMQMSVC, Change IBM MQ Service
  - CPYMQMSVC, Copy IBM MQ Service
  - CRTMQMSVC, Create IBM MQ Service
  - DLTMQMSVC, Delete IBM MQ Service
  - DSPMQMSVC, Display IBM MQ Service
  - ENDMQMSVC, End IBM MQ Service
  - STRMQMSVC, Start IBM MQ Service

  - WRKMQMSVC, Work with IBM MQ Services
- Subscription Commands
  - CHGMQMSUB, Change IBM MQ Subscription
  - CPYMQMSUB, Copy IBM MQ Subscription
  - CRTMQMSUB, Create IBM MQ Subscription
  - DLTMQMSUB, Delete IBM MQ Subscription
  - DSPMQMSUB, Display IBM MQ Subscription
  - WRKMQMSUB, Work with IBM MQ Subscription
- Topic Commands
  - CHGMQMTOP, Change IBM MQ Topic
  - CLRMQMTOP, Clear IBM MQ Topic
  - CPYMQMTOP, Copy IBM MQ Topic
  - CRTMQMTOP, Create IBM MQ Topic
  - DLTMQMTOP, Delete IBM MQ Topic
  - DSPMQMTOP, Display IBM MQ Topic
  - WRKMQMTOP, Work with IBM MQ Topics
- Trace Command
  - TRCMQM, Trace IBM MQ Job
- IBM MQSC Commands
  - RUNMQSC, Run IBM MQSC Commands
  - STRMQMMQSC, Start IBM MQSC Commands
- IBM MQ Dead-Letter Queue Handler Command
  - STRMQMDLQ, Start IBM MQ Dead-Letter Queue Handler
- IBM MQ Route Information
  - DSPMQMRTE, Display IBM MQ Route Information
- IBM MQ Configuration Dump
  - Dump MQ Configuration (DMPMQMCFG)
- IBM MQ Version Details
  - DSPMQMVER, Display IBM MQ Version

Related information:

Managing IBM MQ for IBM i using CL commands

Programmable command formats reference

Programmable Command Formats (PCFs) define command and reply messages that can be exchanged between a program and any queue manager (that supports PCFs) in a network. PCFs simplify queue manager administration and other network administration.

For an introduction to PCFs, see Introduction to Programmable Command Formats.

For the full list of PCFs, see “Definitions of the Programmable Command Formats” on page 1069.

PCF commands and responses have a consistent structure including of a header and any number of parameter structures of defined types. For information about these structures, see “Structures for commands and responses” on page 1591.

For an example PCF, see “PCF example” on page 1618.

Related concepts:

"IBM MQ Control commands reference" on page 195
Reference information about the IBM MQ control commands.

"MQSC reference" on page 363
Use MQSC commands to manage queue manager objects, including the queue manager itself, queues, process definitions, channels, client connection channels, listeners, services, namelists, clusters, and authentication information objects.

Related reference:

IBM i "CL commands reference for IBM i" on page 1065
A list of CL commands for IBM i, grouped according to command type.

Definitions of the Programmable Command Formats

All the available Programmable Command Formats (PCFs) are listed including their parameters (required and optional), response data and error codes.

Following is the reference information for the Programmable Command Formats (PCFs) of commands and responses sent between an IBM MQ systems management application program and an IBM MQ queue manager.

z/OS "Backup CF Structure on z/OS" on page 1083

"Change, Copy, and Create Authentication Information Object" on page 1084

z/OS "Change, Copy, and Create CF Structure on z/OS" on page 1093

"Change, Copy, and Create Channel" on page 1098

"Change, Copy, and Create Channel (MQTT)" on page 1131

"Change, Copy, and Create Channel Listener on Multiplatforms" on page 1137

"Change, Copy, and Create Namelist" on page 1143

"Change, Copy, and Create Process" on page 1146

"Change, Copy, and Create Queue" on page 1150

"Change Queue Manager" on page 1168

"Change Security on z/OS" on page 1196

z/OS "Change SMDS on z/OS" on page 1197

"Change, Copy, and Create Service on Multiplatforms" on page 1198

z/OS "Change, Copy, and Create Storage Class on z/OS" on page 1200

"Change, Copy, and Create Subscription" on page 1203

"Change, Copy, and Create Topic" on page 1207

"Clear Queue" on page 1216

"Clear Topic String" on page 1217

"Delete Authentication Information Object" on page 1218

"Delete Authority Record on Multiplatforms" on page 1219

z/OS "Delete CF Structure on z/OS" on page 1221

"Delete Channel" on page 1221

"Delete Channel (MQTT)" on page 1223

"Delete Channel Listener on Multiplatforms" on page 1223

"Delete Namelist" on page 1224

"Delete Process" on page 1225

"Delete Queue" on page 1227

"Delete Service on Multiplatforms" on page 1229

How the definitions are shown:

The definitions of the Programmable Command Formats (PCFs) including their commands, responses, parameters, constants, and error codes are shown in a consistent format.

For each PCF command or response, there is a description of what the command or response does, giving the command identifier in parentheses. See Constants for all values of the command identifier. Each command description starts with a table that identifies the platforms on which the command is valid. For additional, more detailed, usage notes for each command, see the corresponding command description in the Definitions of the PCFs.

IBM MQ products, other than IBM MQ for z/OS, can use the IBM MQ Administration Interface (MQAI), which provides a simplified way for applications written in the C and Visual Basic programming language to build and send PCF commands. For information about the MQAI see the second section of this topic.

Commands

The *required parameters* and the *optional parameters* are listed.

Multi On Multiplatforms, the parameters must occur in this order:

1. All required parameters, in the order stated, followed by
2. Optional parameters as required, in any order, unless noted in the PCF definition.

z/OS On z/OS, the parameters can be in any order.

Responses

The response data attribute is *always returned* whether it is requested or not. This parameter is required to identify, uniquely, the object when there is a possibility of multiple reply messages being returned.

The other attributes shown are *returned if requested* as optional parameters on the command. The response data attributes are not returned in a defined order.

Parameters and response data

Each parameter name is followed by its structure name in parentheses (details are given in "Structures for commands and responses" on page 1591). The parameter identifier is given at the beginning of the description.

Constants

For the values of constants used by PCF commands and responses see Constants.

z/OS

Informational messages

On z/OS, a number of command responses return a structure, MQIACF_COMMAND_INFO, with values that provide information about the command.

Table 125. MQIACF_COMMAND_INFO values

MQIACF_COMMAND_INFO value	Meaning
MQCMDI_CMDSCOPE_ACCEPTED	A command that specified *CommandScope* was entered. It has been passed to the one or more requested queue managers for processing
MQCMDI_CMDSCOPE_GENERATED	A command that specified *CommandScope* was generated in response to the command originally entered
MQCMDI_CMDSCOPE_COMPLETED	Processing for the command that specified *CommandScope* - either entered or generated by another command - has completed successfully on all requested queue managers
MQCMDI_QSG_DISP_COMPLETED	Processing for the command that refers to an object with the indicated disposition has completed successfully
MQCMDI_COMMAND_ACCEPTED	Initial processing for the command has completed successfully. The command requires further action by the channel initiator, for which a request has been queued. Messages reporting the success or otherwise of the action are be sent to the command issuer later
MQCMDI_CLUSTER_REQUEST_QUEUED	Initial processing for the command has completed successfully. The command requires further action by the cluster repository manager, for which a request has been queued
MQCMDI_CHANNEL_INIT_STARTED	A Start Channel Initiator command has been issued and the channel initiator address space has been started successfully
MQCMDI_RECOVER_STARTED	The queue manager has successfully started a task to process the Recover CF Structure command for the named structure
MQCMDI_BACKUP_STARTED	The queue manager has successfully started a task to process the Backup CF Structure command for the named structure
MQCMDI_RECOVER_COMPLETED	The named CF structure has been recovered successfully. The structure is available for use again
MQCMDI_SEC_TIMER_ZERO	The Change Security command was entered with the *SecurityInterval* attribute set to 0. This means that no user timeouts occur
MQCMDI_REFRESH_CONFIGURATION	A Change Queue Manager command has been issued that enables configuration events. Event messages need to be generated to ensure that the configuration information is complete and up to date
MQCMDI_IMS_BRIDGE_SUSPENDED	The MQ-IMS bridge facility is suspended.
MQCMDI_DB2_SUSPENDED	The connection to Db2 is suspended
MQCMDI_DB2_OBSOLETE_MSGS	Obsolete Db2 messages exist in the queue-sharing group

Error codes

z/OS In z/OS, PCF commands can return MQRC reason codes instead of MQRCCF codes

MQRCCF codes are used in UNIX, Linux or Windows. At the end of most command format definitions, there is a list of error codes that might be returned by that command.

Error codes applicable to all commands

In addition to those error codes listed under each command format, any command might return the following error codes in the response format header (descriptions of the MQRC_* error codes are given in the Reason codes z/OS and IBM MQ for z/OS messages, completion, and reason codes documentation):

Reason (MQLONG)
The value can be any of the following values:

MQRC_NONE
(0, X'000') No reason to report.

MQRC_MSG_TOO_BIG_FOR_Q
(2030, X'7EE') Message length greater than maximum for queue.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_ATTR_VALUE_ERROR
Attribute value not valid.

MQRCCF_CFBF_FILTER_VAL_LEN_ERROR
Filter value length not valid.

MQRCCF_CFBF_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFBF_OPERATOR_ERROR
Operator error.

MQRCCF_CFBF_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFBS_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFBS_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFBS_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFBS_STRING_LENGTH_ERROR
String length not valid.

MQRCCF_CFGR_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFGR_PARM_COUNT_ERROR
Parameter count not valid.

MQRCCF_CFGR_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFH_COMMAND_ERROR
Command identifier not valid.

MQRCCF_CFH_CONTROL_ERROR
Control option not valid.

MQRCCF_CFH_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFH_MSG_SEQ_NUMBER_ERR
Message sequence number not valid.

MQRCCF_CFH_PARM_COUNT_ERROR
Parameter count not valid.

MQRCCF_CFH_TYPE_ERROR
Type not valid.

MQRCCF_CFH_VERSION_ERROR
Structure version number is not valid.

MQRCCF_CFIF_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIF_OPERATOR_ERROR
Operator error.

MQRCCF_CFIF_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFIL_COUNT_ERROR
Count of parameter values not valid.

MQRCCF_CFIL_DUPLICATE_VALUE
Duplicate parameter.

MQRCCF_CFIL_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIL_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFSF_FILTER_VAL_LEN_ERROR
Filter value length not valid.

MQRCCF_CFSF_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFSF_OPERATOR_ERROR
Operator error.

MQRCCF_CFSF_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFSL_COUNT_ERROR
Count of parameter values not valid.

MQRCCF_CFSL_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFSL_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFSL_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFSL_STRING_LENGTH_ERROR
String length value not valid.

MQRCCF_CFSL_TOTAL_LENGTH_ERROR
Total string length error.

MQRCCF_CFST_CONFLICTING_PARM
Conflicting parameters.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFST_STRING_LENGTH_ERROR
String length value not valid.

MQRCCF_COMMAND_FAILED
Command failed.

MQRCCF_ENCODING_ERROR
Encoding error.

MQRCCF_MD_FORMAT_ERROR
Format not valid.

MQRCCF_MSG_SEQ_NUMBER_ERROR
Message sequence number not valid.

MQRCCF_MSG_TRUNCATED
Message truncated.

MQRCCF_MSG_LENGTH_ERROR
Message length not valid.

MQRCCF_OBJECT_NAME_ERROR
Object name not valid.

MQRCCF_OBJECT_OPEN
Object is open.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too large.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_PARM_SYNTAX_ERROR
Syntax error found in parameter.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

MQRCCF_UNKNOWN_OBJECT_NAME
Unknown object name.

PCF commands and responses in groups:

In this product documentation, the commands and data responses are given in alphabetical order.

They can be usefully grouped as follows:

Authentication Information commands
- "Change, Copy, and Create Authentication Information Object" on page 1084
- "Delete Authentication Information Object" on page 1218
- "Inquire Authentication Information Object" on page 1238
- "Inquire Authentication Information Object Names" on page 1245

Authority Record commands
- "Delete Authority Record on Multiplatforms" on page 1219
- "Inquire Authority Records on Multiplatforms" on page 1247
- "Inquire Authority Service on Multiplatforms" on page 1253
- "Inquire Entity Authority on Multiplatforms" on page 1363
- "Set Authority Record on Multiplatforms" on page 1555

z/OS
CF commands
- "Backup CF Structure on z/OS" on page 1083
- "Change, Copy, and Create CF Structure on z/OS" on page 1093
- "Delete CF Structure on z/OS" on page 1221
- "Inquire CF Structure on z/OS" on page 1254
- "Inquire CF Structure Names on z/OS" on page 1259
- "Inquire CF Structure Status on z/OS" on page 1260
- "Recover CF Structure on z/OS" on page 1531

Channel commands
- "Change, Copy, and Create Channel" on page 1098
- "Delete Channel" on page 1221
- "Inquire Channel" on page 1266
- z/OS "Inquire Channel Initiator on z/OS" on page 1293
- "Inquire Channel Names" on page 1304
- "Inquire Channel Status" on page 1307
- "Ping Channel" on page 1526

- "Reset Channel" on page 1538
- "Resolve Channel" on page 1548
- "Start Channel" on page 1572
- z/OS "Start Channel Initiator" on page 1576
- "Stop Channel" on page 1580
- z/OS "Stop Channel Initiator on z/OS" on page 1585

Channel commands (MQTT)
- "Change, Copy, and Create Channel (MQTT)" on page 1131
- "Delete Channel (MQTT)" on page 1223
- "Inquire Channel (MQTT)" on page 1274
- "Inquire Channel Status (MQTT)" on page 1319
- "Purge Channel" on page 1530
- "Start Channel (MQTT)" on page 1575
- "Stop Channel (MQTT)" on page 1584

Channel Authentication commands
- "Inquire Channel Authentication Records" on page 1287
- "Set Channel Authentication Record" on page 1560

Channel Listener commands
- "Change, Copy, and Create Channel Listener on Multiplatforms" on page 1137
- "Delete Channel Listener on Multiplatforms" on page 1223
- "Inquire Channel Listener on Multiplatforms" on page 1296
- "Inquire Channel Listener Status on Multiplatforms" on page 1300
- "Start Channel Listener" on page 1577
- "Stop Channel Listener" on page 1586

Cluster commands
- "Inquire Cluster Queue Manager" on page 1335
- "Refresh Cluster" on page 1532
- "Reset Cluster" on page 1540
- "Resume Queue Manager Cluster" on page 1550
- "Suspend Queue Manager Cluster" on page 1590

Communication Information commands
- "Change, Copy, and Create Communication Information Object on Multiplatforms" on page 1140
- "Delete Communication Information Object on Multiplatforms" on page 1223
- "Inquire Communication Information Object on Multiplatforms" on page 1348

Connection commands
- "Inquire Connection" on page 1352
- "Stop Connection on Multiplatforms" on page 1587

Escape command
- "Escape on Multiplatforms" on page 1233

Namelist commands

- "Change, Copy, and Create Namelist" on page 1143
- "Delete Namelist" on page 1224
- "Inquire Namelist" on page 1375
- "Inquire Namelist Names" on page 1379

Process commands

- "Change, Copy, and Create Process" on page 1146
- "Delete Process" on page 1225
- "Inquire Process" on page 1383
- "Inquire Process Names" on page 1387

Publish/subscribe commands

- "Change, Copy, and Create Subscription" on page 1203
- "Change, Copy, and Create Topic" on page 1207
- "Clear Topic String" on page 1217
- "Delete Subscription" on page 1230
- "Delete Topic" on page 1231
- "Inquire Pub/Sub Status" on page 1389
- "Inquire Subscription" on page 1485
- "Inquire Subscription Status" on page 1492
- "Inquire Topic" on page 1500
- "Inquire Topic Names" on page 1510
- "Inquire Topic Status" on page 1512

Queue commands

- "Change, Copy, and Create Queue" on page 1150
- "Clear Queue" on page 1216
- "Delete Queue" on page 1227
- "Inquire Queue" on page 1393
- "Inquire Queue Names" on page 1453
- "Inquire Queue Status" on page 1455
- z/OS "Move Queue on z/OS" on page 1525
- "Reset Queue Statistics" on page 1545

Queue Manager commands

- "Change Queue Manager" on page 1168
- "Inquire Queue Manager" on page 1412
- "Inquire Queue Manager Status on Multiplatforms" on page 1448
- "Ping Queue Manager on Multiplatforms" on page 1530
- "Refresh Queue Manager" on page 1533
- "Reset Queue Manager" on page 1542
- z/OS "Resume Queue Manager on z/OS" on page 1550
- z/OS "Suspend Queue Manager on z/OS" on page 1589

Security commands

- "Change Security on z/OS" on page 1196
- "Inquire Security on z/OS" on page 1467
- "Refresh Security" on page 1536
- z/OS "Reverify Security on z/OS" on page 1551

Service commands

- "Change, Copy, and Create Service on Multiplatforms" on page 1198
- "Delete Service on Multiplatforms" on page 1229
- "Inquire Service on Multiplatforms" on page 1469
- "Inquire Service Status on Multiplatforms" on page 1472
- "Start Service on Multiplatforms" on page 1579
- "Stop Service on Multiplatforms" on page 1588

z/OS

SMDS commands

- "Change SMDS on z/OS" on page 1197
- "Inquire SMDS on z/OS" on page 1476
- "Inquire SMDS Connection on z/OS" on page 1477
- "Reset SMDS on z/OS" on page 1547
- "Start SMDS Connection on z/OS" on page 1579
- "Stop SMDS Connection on z/OS" on page 1588

z/OS

Storage class commands

- z/OS "Change, Copy, and Create Storage Class on z/OS" on page 1200
- "Delete Storage Class on z/OS" on page 1229
- "Inquire Storage Class on z/OS" on page 1480
- "Inquire Storage Class Names on z/OS" on page 1483

z/OS

System commands

- "Inquire Archive on z/OS" on page 1234
- "Set Archive on z/OS" on page 1552
- "Inquire Group on z/OS" on page 1368
- "Inquire Log on z/OS" on page 1371
- "Set Log on z/OS" on page 1567
- "Inquire System on z/OS" on page 1496
- "Set System on z/OS" on page 1571
- "Inquire Usage on z/OS" on page 1519

Data responses to commands

- "Escape (Response) on Multiplatforms" on page 1233
- z/OS "Inquire Archive (Response) on z/OS" on page 1234
- "Inquire Authentication Information Object (Response)" on page 1241
- "Inquire Authentication Information Object Names (Response)" on page 1246

- "Inquire Authority Records (Response) on Multiplatforms" on page 1250
- "Inquire Authority Service (Response) on Multiplatforms" on page 1254
- z/OS "Inquire CF Structure (Response) on z/OS" on page 1256
- z/OS "Inquire CF Structure Names (Response) on z/OS" on page 1260
- z/OS "Inquire CF Structure Status (Response) on z/OS" on page 1261
- "Inquire Channel (Response)" on page 1276
- "Inquire Channel Authentication Records (Response)" on page 1291
- "Inquire Channel Initiator (Response) on z/OS" on page 1294
- "Inquire Channel Listener (Response) on Multiplatforms" on page 1298
- "Inquire Channel Listener Status (Response) on Multiplatforms" on page 1302
- "Inquire Channel Names (Response)" on page 1306
- "Inquire Channel Status (Response)" on page 1322
- "Inquire Channel Status (Response) (MQTT)" on page 1333
- "Inquire Cluster Queue Manager (Response)" on page 1340
- "Inquire Communication Information Object (Response) on Multiplatforms" on page 1349
- "Inquire Connection (Response)" on page 1356
- "Inquire Entity Authority (Response) on Multiplatforms" on page 1365
- z/OS "Inquire Group (Response) on z/OS" on page 1369
- z/OS "Inquire Log (Response) on z/OS" on page 1371
- "Inquire Namelist (Response)" on page 1378
- "Inquire Namelist Names (Response)" on page 1380
- "Inquire Process (Response)" on page 1385
- "Inquire Process Names (Response)" on page 1388
- "Inquire Pub/Sub Status (Response)" on page 1390
- "Inquire Queue (Response)" on page 1401
- "Inquire Queue Manager (Response)" on page 1423
- "Inquire Queue Manager Status (Response) on Multiplatforms" on page 1450
- "Inquire Queue Names (Response)" on page 1454
- "Reset Queue Statistics (Response)" on page 1546
- "Inquire Queue Status (Response)" on page 1460
- z/OS "Inquire Security (Response) on z/OS" on page 1468
- "Inquire Service (Response) on Multiplatforms" on page 1471
- "Inquire Service Status (Response) on Multiplatforms" on page 1474
- z/OS "Inquire Storage Class (Response) on z/OS" on page 1482
- z/OS "Inquire Storage Class Names (Response) on z/OS" on page 1484
- z/OS "Inquire SMDS (Response) on z/OS" on page 1476
- z/OS "Inquire SMDS Connection (Response) on z/OS" on page 1478
- "Inquire Subscription (Response)" on page 1488
- "Inquire Subscription Status (Response)" on page 1494
- z/OS "Inquire System (Response) on z/OS" on page 1496
- "Inquire Topic (Response)" on page 1504
- "Inquire Topic Names (Response)" on page 1511
- "Inquire Topic Status (Response)" on page 1513

- z/OS "Inquire Usage (Response) on z/OS" on page 1520

Backup CF Structure on z/OS: z/OS

The Backup CF Structure (MQCMD_BACKUP_CF_STRUC) command initiates a CF application structure backup.

Note: This command is supported only on z/OS when the queue manager is a member of a queue-sharing group.

Required parameters

CFStrucName (MQCFST)
The name of the CF application structure to be backed up (parameter identifier: MQCA_CF_STRUC_NAME).

The maximum length is MQ_CF_STRUC_NAME_LENGTH.

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_QSG_NAME_LENGTH.

ExcludeInterval (MQCFIN)
Exclude interval (parameter identifier: MQIACF_EXCLUDE_INTERVAL).

Specifies a value in seconds that defines the length of time immediately before the current time where the backup starts. The backup excludes backing-up the last *n* seconds activity. For example, if 30 seconds is specified, the backup does not include the last 30 seconds worth of activity for this application-structure.

The value must be in the range 30 through 600. The default value is 30.

Change, Copy, and Create Authentication Information Object:

The Change authentication information command changes attributes of an existing authentication information object. The Create and Copy authentication information commands create new authentication information objects - the Copy command uses attribute values of an existing object.

The Change authentication information (MQCMD_CHANGE_AUTH_INFO) command changes the specified attributes in an authentication information object. For any optional parameters that are omitted, the value does not change.

The Copy authentication information (MQCMD_COPY_AUTH_INFO) command creates new authentication information object using, for attributes not specified in the command, the attribute values of an existing authentication information object.

The Create authentication information (MQCMD_CREATE_AUTH_INFO) command creates an authentication information object. Any attributes that are not defined explicitly are set to the default values on the destination queue manager. A system default authentication information object exists and default values are taken from it.

Required parameters (Change authentication information)

AuthInfoName (MQCFST)
The authentication information object name (parameter identifier: MQCA_AUTH_INFO_NAME).

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

AuthInfoType (MQCFIN)
The type of authentication information object (parameter identifier: MQIA_AUTH_INFO_TYPE).

The value can be:

MQAIT_CRL_LDAP
This defines this authentication information object as specifying an LDAP server containing Certificate Revocation Lists.

MQAIT_OCSP
This value defines this authentication information object as specifying certificate revocation checking using OCSP.

AuthInfoType MQAIT_OCSP does not apply for use on IBM i or z/OS queue managers, but it can be specified on those platforms to be copied to the client channel definition table for client use.

MQAIT_IDPW_OS
This value defines this authentication information object as specifying certificate revocation checking using user ID and password checking through the operating system.

MQAIT_IDPW_LDAP
This value defines this authentication information object as specifying certificate revocation checking using user ID and password checking through an LDAP server.

Important: This option is not valid on z/OS.

See Securing for more information.

Required parameters (Copy authentication information)

FromAuthInfoName (MQCFST)
The name of the authentication information object definition to be copied from (parameter identifier: MQCACF_FROM_AUTH_INFO_NAME).

z/OS On z/OS, the queue manager searches for an object with the name you specify and a disposition of MQQSGD_Q_MGR or MQQSGD_COPY to copy from. This parameter is ignored if a value of MQQSGD_COPY is specified for *QSGDisposition*. In this case, an object with the name specified by *ToAuthInfoName* and the disposition of MQQSGD_GROUP is searched for to copy from.

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

ToAuthInfoName (MQCFST)
The name of the authentication information object to copy to (parameter identifier: MQCACF_TO_AUTH_INFO_NAME).

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

AuthInfoType (MQCFIN)
The type of authentication information object (parameter identifier: MQIA_AUTH_INFO_TYPE). The value must match the AuthInfoType of the authentication information object from which you are copying.

The value can be:

MQAIT_CRL_LDAP
This value defines this authentication information object as specifying Certificate Revocation Lists that are held on LDAP.

MQAIT_OCSP
This value defines this authentication information object as specifying certificate revocation checking using OCSP.

MQAIT_IDPW_OS
This value defines this authentication information object as specifying certificate revocation checking using user ID and password checking through the operating system.

MQAIT_IDPW_LDAP
This value defines this authentication information object as specifying certificate revocation checking using user ID and password checking through an LDAP server.

Important: This option is not valid on z/OS.

See Securing for more information.

Required parameters (Create authentication information)

AuthInfoName (MQCFST)
Authentication information object name (parameter identifier: MQCA_AUTH_INFO_NAME).

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

AuthInfoType (MQCFIN)
The type of authentication information object (parameter identifier: MQIA_AUTH_INFO_TYPE).

The following values are accepted:

MQAIT_CRL_LDAP
This value defines this authentication information object as specifying an LDAP server containing Certificate Revocation Lists.

MQAIT_OCSP
This value defines this authentication information object as specifying certificate revocation checking using OCSP.

An authentication information object with AuthInfoType MQAIT_OCSP does not apply for use on IBM i or z/OS queue managers, but it can be specified on those platforms to be copied to the client channel definition table for client use.

MQAIT_IDPW_OS
This value defines this authentication information object as specifying certificate revocation checking using user ID and password checking through the operating system.

MQAIT_IDPW_LDAP
This value defines this authentication information object as specifying certificate revocation checking using user ID and password checking through an LDAP server.

Important: This option is not valid on z/OS.

See Securing for more information.

Optional parameters (Change, Copy, and Create Authentication Information Object)

AdoptContext (MQCFIN)
Whether to use the presented credentials as the context for this application (parameter identifier MQIA_ADOPT_CONTEXT). This means that they are used for authorization checks, shown on administrative displays, and appear in messages.

YES The user ID presented in the MQCSP structure, which has been successfully validated by password, is adopted as the context to use for this application. Therefore, this user ID will be the credentials checked for authorization to use IBM MQ resources.

If the user ID presented is an LDAP user ID, and authorization checks are done using operating system user IDs, the ShortUser associated with the user entry in LDAP will be adopted as the credentials for authorization checks to be done against.

NO Authentication will be performed on the user ID and password presented in the MQCSP structure, but then the credentials will not be adopted for further use. Authorization will be performed using the user ID the application is running under.

This attribute is only valid for **AuthInfoType** of *MQAIT_IDPW_OS* and *MQAIT_IDPW_LDAP*.

The maximum length is MQIA_ADOPT_CONTEXT_LENGTH.

AuthInfoConnName (MQCFST)
The connection name of the authentication information object (parameter identifier: MQCA_AUTH_INFO_CONN_NAME).

This parameter is relevant only when AuthInfoType is set to *MQAIT_CRL_LDAP* or *MQAIT_IDPW_LDAP*, when it is required.

When used with an AuthInfoType of *MQAIT_IDPW_LDAP*, this can be a comma separated list of connection names.

Multi On Multiplatforms, the maximum length is MQ_AUTH_INFO_CONN_NAME_LENGTH.

z/OS On z/OS, the maximum length is MQ_LOCAL_ADDRESS_LENGTH.

AuthInfoDesc (MQCFST)
The description of the authentication information object (parameter identifier: MQCA_AUTH_INFO_DESC).

The maximum length is MQ_AUTH_INFO_DESC_LENGTH.

AuthenticationMethod (MQCFIN)
Authentication methods for user passwords (parameter identifier: MQIA_AUTHENTICATION_METHOD). Possible values are:

MQAUTHENTICATE_OS
Use the traditional UNIX password verification method

This is the default value.

MQAUTHENTICATE_PAM
Use the Pluggable Authentication Method to authenticate the user passwords.

You can set the PAM value only on UNIX and Linux platforms.

This attribute is valid only for an **AuthInfoType** of *MQAIT_IDPW_OS*, and is not valid on IBM MQ for z/OS.

AuthorizationMethod (MQCFIN)
Authorization methods for the queue manager (parameter identifier: MQIA_LDAP_AUTHORMD). Possible values are:

MQLDAP_AUTHORMD_OS
Use operating system groups to determine permissions associated with a user.

This is how IBM MQ has previously worked, and is the default value.

MQLDAP_AUTHORMD_SEARCHGRP
A group entry in the LDAP repository contains an attribute listing the Distinguished Name of all the users belonging to that group. Membership is indicated by the attribute defined in FindGroup. This value is typically *member* or *uniqueMember*.

MQLDAP_AUTHORMD_SEARCHUSR
A user entry in the LDAP repository contains an attribute listing the Distinguished Name of all the groups to which the specified user belongs. The attribute to query is defined by the FindGroup value, typically *memberOf*.

V 9.0.5 **MQLDAP_AUTHORMD_SRCHGRPSN**
A group entry in the LDAP repository contains an attribute listing the short user name of all the users belonging to that group. The attribute in the user record that contains the short user name is specified by ShortUser.

Membership is indicated by the attribute defined in FindGroup. This value is typically *memberUid*.

Note: This authorization method should only be used if all user short names are distinct.

Many LDAP servers use an attribute of the group object to determine group membership and you should, therefore, set this value to *MQLDAP_AUTHORMD_SEARCHGRP*.

Microsoft Active Directory typically stores group memberships as a user attribute. The IBM Tivoli Directory Server supports both methods.

In general, retrieving memberships through a user attribute will be faster than searching for groups that list the user as a member.

BaseDNGroup (MQCFST)
In order to be able to find group names, this parameter must be set with the base DN to search for groups in the LDAP server (parameter identifier: MQCA_LDAP_BASE_DN_GROUPS).

The maximum length is MQ_LDAP_BASE_DN_LENGTH.

BaseDNUser (MQCFST)
In order to be able to find the short user name attribute (see ShortUser) this parameter must be set with the base DN to search for users within the LDAP server (parameter identifier: MQCA_LDAP_BASE_DN_USERS).

This attribute is valid only for an **AuthInfoType** of *MQAIT_IDPW_LDAP* and is mandatory.

The maximum length is MQ_LDAP_BASE_DN_LENGTH.

Checkclient (MQCFIN)
This attribute is valid only for an **AuthInfoType** of *MQAIT_IDPW_OS* or *MQAIT_IDPW_LDAP* (parameter identifier: MQIA_CHECK_CLIENT_BINDING). The possible values are:

MQCHK_NONE
Switches off checking.

MQCHK_OPTIONAL
Ensures that if a user ID and password are provided by an application, they are a valid pair, but that it is not mandatory to provide them. This option might be useful during migration, for example.

MQCHK_REQUIRED
Requires that all applications provide a valid user ID and password.

MQCHK_REQUIRED_ADMIN
Privileged users must supply a valid user ID and password, but non-privileged users are treated as with the `OPTIONAL` setting. z/OS (This setting is not allowed on z/OS systems.)

Checklocal (MQCFIN)
This attribute is valid only for an **AuthInfoType** of *MQAIT_IDPW_OS* or *MQAIT_IDPW_LDAP* (parameter identifier: MQIA_CHECK_LOCAL_BINDING). The possible values are:

MQCHK_NONE
Switches off checking.

MQCHK_OPTIONAL
Ensures that if a user ID and password are provided by an application, they are a valid pair, but that it is not mandatory to provide them. This option might be useful during migration, for example.

MQCHK_REQUIRED
Requires that all applications provide a valid user ID and password.

z/OS If your user ID has UPDATE access to the BATCH profile in the MQCONN class, you can treat **MQCHK_REQUIRED** as if it is **MQCHK_OPTIONAL**. That is, you do not have to supply a password, but if you do, the password must be the correct one.

MQCHK_REQUIRED_ADMIN
Privileged users must supply a valid user ID and password, but non-privileged users are treated as with the `OPTIONAL` setting. z/OS (This setting is not allowed on z/OS systems.)

ClassGroup (MQCFST)
The LDAP object class used for group records in the LDAP repository (parameter identifier: MQCA_LDAP_GROUP_OBJECT_CLASS).

If the value is blank, groupOfNames is used.

Other commonly used values include *groupOfUniqueNames* or *group*.

The maximum length is MQ_LDAP_CLASS_LENGTH.

Classuser (MQCFST)
The LDAP object class used for user records in the LDAP repository (parameter identifier MQCA_LDAP_USER_OBJECT_CLASS).

If blank, the value defaults to *inetOrgPerson*, which is generally the value needed.

For Microsoft Active Directory, the value you require required is often *user*.

This attribute is valid only for an **AuthInfoType** of *MQAIT_IDPW_LDAP*.

z/OS **CommandScope (MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

FailureDelay (MQCFIN)
When a user ID and password are provided for connection authentication, and the authentication fails due to the user ID or password being incorrect, this is the delay, in seconds, before the failure is returned to the application (parameter identifier: MQIA_AUTHENTICATION_FAIL_DELAY).

This can aid in avoiding busy loops from an application that simply retries, continuously, after receiving a failure.

The value must be in the range 0 - 60 seconds. The default value is 1. This parameter is valid only for an **AuthInfoType** of *MQAIT_IDPW_OS* or *MQAIT_IDPW_LDAP*.

FindGroup (MQCFST)
Name of the attribute used within an LDAP entry to determine group membership (parameter identifier: MQCA_LDAP_FIND_GROUP_FIELD).

When AuthorizationMethod = *MQLDAP_AUTHORMD_SEARCHGRP*, this attribute is typically set to *member* or *uniqueMember*.

When AuthorizationMethod = *MQLDAP_AUTHORMD_SEARCHUSR*, this attribute is typically set to *memberOf*.

V 9.0.5 When AuthorizationMethod = *MQLDAP_AUTHORMD_SRCHGRPSN*, this attribute is typically set to *memberUid*.

When left blank, if:

- AuthorizationMethod = *MQLDAP_AUTHORMD_SEARCHGRP*, this attribute defaults to *memberOf*.
- AuthorizationMethod = *MQLDAP_AUTHORMD_SEARCHUSR*, this attribute defaults to *member*.
- V 9.0.5 AuthorizationMethod = *MQLDAP_AUTHORMD_SRCHGRPSN*, this attribute defaults to *memberUid*.

The maximum length is MQ_LDAP_FIELD_LENGTH.

GroupField (MQCFST)
LDAP attribute that represents a simple name for the group (parameter identifier: MQCA_LDAP_GROUP_ATTR_FIELD).

If the value is blank, commands like setmqaut must use a qualified name for the group. The value can either be a full DN, or a single attribute.

The maximum length is MQ_LDAP_FIELD_LENGTH.

GroupNesting (MQCFIN)
Whether groups are members of other groups (parameter identifier: MQIA_LDAP_NESTGRP). The values can be:

MQLDAP_NESTGRP_NO
Only the initially discovered groups are considered for authorization.

MQLDAP_NESTGRP_YES
The group list is searched recursively to enumerate all the groups to which a user belongs.

The group's Distinguished Name is used when searching the group list recursively, regardless of the authorization method selected in AuthorizationMethod.

LDAPPassword (MQCFST)
The LDAP password (parameter identifier: MQCA_LDAP_PASSWORD).

This parameter is relevant only when **AuthInfoType** is set to *MQAIT_CRL_LDAP* or *MQAIT_IDPW_LDAP*.

The maximum length is MQ_LDAP_PASSWORD_LENGTH.

LDAPUserName (MQCFST)
The LDAP user name (parameter identifier: MQCA_LDAP_USER_NAME).

This parameter is relevant only when AuthInfoType is set to *MQAIT_CRL_LDAP* or *MQAIT_IDPW_LDAP*.

Multi On Multiplatforms, the maximum length is MQ_DISTINGUISHED_NAME_LENGTH.

z/OS On z/OS, the maximum length is MQ_SHORT_DNAME_LENGTH.

OCSPResponderURL (MQCFST)
The URL at which the OCSP responder can be contacted (parameter identifier: MQCA_AUTH_INFO_OCSP_URL).

This parameter is relevant only when AuthInfoType is set to MQAIT_OCSP, when it is required.

This field is case-sensitive. It must start with the string http:// in lowercase. The rest of the URL might be case sensitive, depending on the OCSP server implementation.

The maximum length is MQ_AUTH_INFO_OCSP_URL_LENGTH.

z/OS **QSGDisposition (MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OSonly.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be any of the following values:

QSGDisposition	Change	Copy, Create
MQQSGD_COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameter MQQSGD_Q_MGR, is not affected by this command.	The object is defined on the page set of the queue manager that executes the command using the MQQSGD_GROUP object of the same name as the *ToAuthInfoName* object (for Copy) or the *AuthInfoName* object (for Create).

QSGDisposition	Change	Copy, Create
MQQSGD_GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command. If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group so that they refresh local copies on page set zero: `DEFINE AUTHINFO(name)` `REPLACE QSGDISP(COPY)` The Change for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.	The object definition resides in the shared repository. This definition is allowed only if the queue manager is in a queue-sharing group. If the definition is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group so that they make or refresh local copies on page set zero: `DEFINE AUTHINFO(name)` `REPLACE QSGDISP(COPY)` The Copy or Create for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.
MQQSGD_PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with MQQSGD_Q_MGR, or MQQSGD_COPY. Any object residing in the shared repository is unaffected.	Not permitted.
MQQSGD_Q_MGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command. This value is the default value.	The object is defined on the page set of the queue manager that executes the command. This value is the default value.

Replace (MQCFIN)
Replace attributes (parameter identifier: MQIACF_REPLACE).

If an Authentication Information object with the same name as AuthInfoName or ToAuthInfoName exists, it specifies whether it is to be replaced. The value can be any of the following values:

MQRP_YES
Replace existing definition

MQRP_NO
Do not replace existing definition

SecureComms (MQCFIN)
Whether connectivity to the LDAP server should be done securely using TLS (parameter identifier MQIA_LDAP_SECURE_COMM).

MQSECCOMM_YES
Connectivity to the LDAP server is made securely using TLS.

The certificate used is the default certificate for the queue manager, named in CERTLABL on the queue manager object, or if that is blank, the one described in Digital certificate labels, understanding the requirements.

The certificate is located in the key repository specified in SSLKEYR on the queue manager object. A cipherspec will be negotiated that is supported by both IBM MQ and the LDAP server.

If the queue manager is configured to use SSLFIPS(YES) or SUITEB cipher specs, then this is taken account of in the connection to the LDAP server as well.

MQSECCOMM_ANON
Connectivity to the LDAP server is made securely using TLS just as for MQSECCOMM_YES with one difference.

No certificate is sent to the LDAP server; the connection will be made anonymously. To use this setting, ensure that the key repository specified in SSLKEYR, on the queue manager object, does not contain a certificate marked as the default.

MQSECCOMM_NO
Connectivity to the LDAP server does not use TLS.

This attribute is valid only for an **AuthInfoType** of *MQAIT_IDPW_LDAP*.

ShortUser (MQCFST)
A field in the user record to be used as a short user name in IBM MQ (parameter identifier MQCA_LDAP_SHORT_USER_FIELD).

This field must contain values of 12 characters or less. This short user name is used for the following purposes:

- If LDAP authentication is enabled, but LDAP authorization is not enabled, this is used as an operating system user ID for authorization checks. In this case, the attribute must represent an operating system user ID.
- If LDAP authentication and authorization are both enabled, this is used as the user ID carried with the message in order for the LDAP user name to be rediscovered when the user ID inside the message needs to be used.

  For example, on another queue manager, or when writing report messages. In this case, the attribute does not need to represent an operating system user ID, but must be a unique string. An employee serial number is an example of a good attribute for this purpose.

This attribute is valid only for an **AuthInfoType** of *MQAIT_IDPW_LDAP* and is mandatory.

The maximum length is MQ_LDAP_FIELD_LENGTH.

UserField (MQCFST)
If the user ID provided by an application for authentication does not contain a qualifier for the field in the LDAP user record, that is, it does not contain an ' = ' sign, this attribute identifies the field in the LDAP user record that is used to interpret the provided user ID (parameter identifier MQCA_LDAP_USER_ATTR_FIELD).

This field can be blank. If this is the case, any unqualified user IDs use the ShortUser field to interpret the provided user ID.

The contents of this field will be concatenated with an ' = ' sign, together with the value provided by the application, to form the full user ID to be located in an LDAP user record. For example, the application provides a user of `fred` and this field has the value `cn`, then the LDAP repository will be searched for `cn=fred`.

The maximum length is MQ_LDAP_FIELD_LENGTH.

Change, Copy, and Create CF Structure on z/OS: z/OS

The Change CF Structure command changes existing CF application structures. The Copy and Create CF Structure commands create new CF application structures - the Copy command uses attribute values of an existing CF application structure.

Note: These commands are supported only on z/OS when the queue manager is a member of a queue-sharing group.

The Change CF Structure (MQCMD_CHANGE_CF_STRUC) command changes the specified attributes in a CF application structure. For any optional parameters that are omitted, the value does not change.

The Copy CF Structure (MQCMD_COPY_CF_STRUC) command creates new CF application structure using, for attributes not specified in the command, the attribute values of an existing CF application structure.

The Create CF Structure (MQCMD_CREATE_CF_STRUC) command creates a CF application structure. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameters (Change and Create CF Structure)

CFStrucName (MQCFST)
The name of the CF application structure with backup and recovery parameters that you want to define (parameter identifier: MQCA_CF_STRUC_NAME).

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

Required parameters (Copy CF Structure)

FromCFStrucName (MQCFST)
The name of the CF application structure to be copied from (parameter identifier: MQCACF_FROM_CF_STRUC_NAME).

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

ToCFStrucName (MQCFST)
The name of the CF application structure to copy to (parameter identifier: MQCACF_TO_CF_STRUC_NAME).

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

Optional parameters (Change, Copy, and Create CF Structure)

CFConlos (MQCFIN)
CFConlos (parameter identifier: MQIA_CF_CFCONLOS).

Specifies the action to be taken when a queue manager loses connectivity to the CF structure. The following constant names are valid values for CFConlos:

MQCFCONLOS_ASQMGR
The action taken is based on the setting of the CFCONLOS queue manager attribute. This value is the default for newly created CF structure objects with CFLEVEL(5).

MQCFCONLOS_TERMINATE
The queue manager terminates when connectivity to the structure is lost. This value is the default if the CF structure object is not at CFLEVEL(5), and for existing CF structure objects that are changed to CFLEVEL(5).

MQCFCONLOS_TOLERATE
The queue manager tolerates loss of connectivity to the structure without terminating.

This parameter is only valid from CFLEVEL(5).

CFLevel (MQCFIN)
The functional capability level for this CF application structure (parameter identifier: MQIA_CF_LEVEL).

Specifies the functional capability level for the CF application structure. The value can be any of the following values:

1 A CF structure that can be "auto-created" by a queue manager at command level 520.

2 A CF structure at command level 520 that can only be created or deleted by a queue manager at command level 530 or greater.

3

A CF structure at command level 530. This *CFLevel* is required if you want to use persistent messages on shared queues, or for message grouping, or both. This level is the default *CFLevel* for queue managers at command level 600.

You can only increase the value of *CFLevel* to 3 if all the queue managers in the queue-sharing group are at command level 530 or greater - this restriction is to ensure that there are no latent command level 520 connections to queues referencing the CF structure.

You can only decrease the value of *CFLevel* from 3 if all the queues that reference the CF structure are both empty (have no messages or uncommitted activity) and closed.

4

This *CFLevel* supports all the *CFLevel* (3) functions. *CFLevel* (4) allows queues defined with CF structures at this level to have messages with a length greater than 63 KB.

Only a queue manager with a command level of 600 can connect to a CF structure at *CFLevel* (4).

You can only increase the value of *CFLevel* to 4 if all the queue managers in the queue-sharing group are at command level 600 or greater.

You can only decrease the value of *CFLevel* from 4 if all the queues that reference the CF structure are both empty (have no messages or uncommitted activity) and closed.

5

This *CFLevel* supports all the *CFLevel* (4) functions. *CFLevel* (5) allows persistent, and nonpersistent messages to be selectively stored in Db2 or shared message data sets.

Only queue managers with a command level of 710 or higher and with OPMODE set to enable IBM WebSphere MQ Version 7.1.0 new functions can connect to a CF structure at *CFLevel* (5).

Structures are required to be at CFLEVEL(5) to support toleration of loss of connectivity.

z/OS For more information, see Where are shared queue messages held?.

CFStrucDesc (MQCFST)
The description of the CF structure (parameter identifier: MQCA_CF_STRUC_DESC).

The maximum length is MQ_CF_STRUC_DESC_LENGTH.

DSBlock (MQCFIN)
The logical block size for shared message data sets (parameter identifier: MQIACF_CF_SMDS_BLOCK_SIZE).

The unit in which shared message data set space is allocated to individual queues. The value can be any of the following values:

MQDSB_8K
The logical block size is set to 8 K.

MQDSB_16K
The logical block size is set to 16K.

MQDSB_32K
The logical block size is set to 32 K.

MQDSB_64K
The logical block size is set to 64 K.

MQDSB_128K
The logical block size is set to 128 K.

MQDSB_256K
The logical block size is set to 256 K.

MQDSB_512K
The logical block size is set to 512 K.

MQDSB_1024K
The logical block size is set to 1024 K.

MQDSB_1M
The logical block size is set to 1 M.

Value can not be set unless CFLEVEL(5) is defined.

The default value is 256 K unless CFLEVEL is not 5. In this case a value of 0 is used.

DSBufs (MQCFIN)
The shared message data set buffers group (parameter identifier: MQIA_CF_SMDS_BUFFERS).

Specifies the number of buffers to be allocated in each queue manager for accessing shared message data sets. The size of each buffer is equal to the logical block size.

A value in the range 1 - 9999.

Value can not be set unless CFLEVEL(5) is defined.

DSEXPAND (MQCFIN)
The shared message data set expand option (parameter identifier: MQIACF_CF_SMDS_EXPAND).

Specifies whether or not the queue manager should expand a shared message data set when it is nearly full, and further blocks are required in the data set. The value can be any of the following values:

MQDSE_YES
The data set can be expanded.

MQDSE_NO
The data set cannot be expanded.

MQDSE_DEFAULT
Only returned on DISPLAY CFSTRUCT when not explicitly set

Value can not be set unless CFLEVEL(5) is defined.

DSGroup (MQCFST)
The shared message data set group name (parameter identifier: MQCACF_CF_SMDS_GENERIC_NAME).

Specifies a generic data set name to be used for the group of shared message data sets associated with this CF structure.

The string must contain exactly one asterisk ('*'), which will be replaced with the queue manager name of up to 4 characters.

The maximum length of this parameter is 44 characters.

Value can not be set unless CFLEVEL(5) is defined.

Offload (MQCFIN)
Offload (parameter identifier: MQIA_CF_OFFLOAD).

Specifies the OFFLOAD option for large (>63 K) shared messages on z/OS. The value can be:

MQCFOFFLD_DB2
Large shared messages can be stored in Db2.

MQCFOFFLD_SMDS
Large shared messages can be stored in z/OS shared message data sets.

MQCFOFFLD_NONE
Used when the property *Offload* has not been explicitly set.

Value can not be set unless CFLEVEL(5) is defined.

The default value is MQCFOFFLD_NONE if not at CFLEVEL(5).

For existing CF structure objects that are changed to CFLEVEL(5) the default is MQCFOFFLD_DB2.

For newly created CF structure objects with CFLEVEL(5) the default is MQCFOFFLD_SMDS.

z/OS For more information about the group of parameters (*OFFLDxSZ* and *OFFLDxTH*), see Specifying offload options for shared message data sets

OFFLD1SZ (MQCFST)
The offload size property 1 (Parameter identifier: MQCACF_CF_OFFLOAD_SIZE1)

Specifies the first offload rule, based on upon message size and the coupling facility structure percentage use threshold. This property indicates the size of the messages to be offloaded. The property is specified as a string with values in the range 0K - 64K.

The default value is 32K. This property is used with *OFFLD1TH*.

Value can not be set unless CFLEVEL(5) is defined.

The value 64K indicates that the rule is not being used.

The maximum length is 3.

OFFLD2SZ (MQCFST)
The offload size property 2 (Parameter identifier: MQCACF_CF_OFFLOAD_SIZE2)

Specifies the second offload rule, based on upon message size and the coupling facility structure percentage use threshold. This property indicates the size of the messages to be offloaded. The property is specified as a string with values in the range 0K - 64K.

The default value is 4K. This property is used with *OFFLD2TH*.

Value can not be set unless CFLEVEL(5) is defined.

The value 64K indicates that the rule is not being used.

The maximum length is 3.

OFFLD3SZ (MQCFST)
The offload size property 3(Parameter identifier: MQCACF_CF_OFFLOAD_SIZE3)

Specifies the third offload rule, based on upon message size and the coupling facility structure percentage use threshold. This property indicates the size of the messages to be offloaded. The property is specified as a string with values in the range 0K - 64K.

The default value is 0K. This property is used with *OFFLD3TH*.

Value can not be set unless CFLEVEL(5) is defined.

The value 64K indicates that the rule is not being used.

The maximum length is 3.

OFFLD1TH (MQCFIN)
The offload threshold property 1 (Parameter identifier: MQIA_CF_OFFLOAD_THRESHOLD1)

Specifies the first offload rule, based on upon message size and the coupling facility structure percentage use threshold. This property indicates the coupling facility structure percentage full.

The default value is 70. This property is used with *OFFLD1SZ*.

Value can not be set unless CFLEVEL(5) is defined.

OFFLD2TH (MQCFIN)
The offload threshold property 2 (Parameter identifier: MQIA_CF_OFFLOAD_THRESHOLD2)

Specifies the second offload rule, based on upon message size and the coupling facility structure percentage use threshold. This property indicates the coupling facility structure percentage full.

The default value is 80. This property is used with *OFFLD2SZ*.

Value can not be set unless CFLEVEL(5) is defined.

OFFLD3TH (MQCFIN)
The offload threshold property 3 (Parameter identifier: MQIA_CF_OFFLOAD_THRESHOLD3)

Specifies the third offload rule, based on upon message size and the coupling facility structure percentage use threshold. This property indicates the coupling facility structure percentage full.

The default value is 90. This property is used with *OFFLD3SZ*.

Value can not be set unless CFLEVEL(5) is defined.

Recauto (MQCFIN)
Recauto (parameter identifier: MQIA_CF_RECAUTO).

Specifies the automatic recovery action to be taken, when a queue manager detects that the structure is failed or when a queue manager loses connectivity to the structure, and no systems in the SysPlex have connectivity to the Coupling Facility that the structure is allocated in. The value can be:

MQRECAUTO_YES
The structure and associated shared message data sets which also need recovery are automatically recovered. This value is the default for newly created CF structure objects with CFLEVEL(5).

MQRECAUTO_NO
The structure is not automatically recovered. This value is the default if the CF structure object is not at CFLEVEL(5), and for existing CF structure objects that are changed to CFLEVEL(5).

Recovery (MQCFIN)
Recovery (parameter identifier: MQIA_CF_RECOVER).

Specifies whether CF recovery is supported for the application structure. The value can be:

MQCFR_YES
Recovery is supported.

MQCFR_NO
Recovery is not supported.

Replace (MQCFIN)
Replace attributes (parameter identifier: MQIACF_REPLACE).

If a CF structure definition with the same name as *ToCFStrucName* exists, this value specifies whether it is to be replaced. The value can be any of the following values:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

Change, Copy, and Create Channel:

The Change Channel command changes existing channel definitions. The Copy and Create Channel commands create new channel definitions - the Copy command uses attribute values of an existing channel definition.

The Change Channel (MQCMD_CHANGE_CHANNEL) command changes the specified attributes in a channel definition. For any optional parameters that are omitted, the value does not change.

The Copy Channel (MQCMD_COPY_CHANNEL) command creates new channel definition using, for attributes not specified in the command, the attribute values of an existing channel definition.

The Create Channel (MQCMD_CREATE_CHANNEL) command creates an IBM MQ channel definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager. If a system default channel exists for the type of channel being created, the default values are taken from there.

The following table shows the parameters that are applicable to each type of channel.

Table 126. Change, Copy, Create Channel parameters

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver	AMQP
V 9.0.0 *AMQPKeepAlive*									✓ 9.0.0
BatchHeartBeat	✓	✓					✓	✓	
BatchInterval	✓	✓					✓	✓	
BatchDataLimit	✓	✓					✓	✓	
BatchSize	✓	✓	✓	✓			✓	✓	
CertificateLabel	✓	✓	✓	✓			✓	✓	✓ 9.0.0
ChannelDesc	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
ChannelMonitoring	✓	✓	✓	✓		✓	✓	✓	
ChannelStatistics	✓	✓	✓	✓			✓	✓	
ChannelName (see footnote 1)	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
ChannelType (see footnote 3)	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
ClientChannelWeight					✓				

Table 126. Change, Copy, Create Channel parameters (continued)

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver	AMQP V 8.0.0
ClusterName							✓	✓	
ClusterNameList							✓	✓	
CLWLChannelPriority							✓	✓	
CLWLChannelRank							✓	✓	
CLWLChannelWeight							✓	✓	
z/OS *Command*	✓	✓	✓	✓	✓	✓	✓	✓	
ConnectionAffinity					✓				
ConnectionName	✓	✓		✓	✓		✓	✓	
DataConversion	✓	✓		✓	✓		✓	✓	
DefaultChannelDisposition	✓	✓	✓	✓		✓	✓	✓	
DefReconnect					✓				
DiscInterval	✓	✓				✓	✓	✓	
FromChannelName (see footnote 2)	✓	✓	✓	✓	✓	✓	✓	✓	
HeaderCompression	✓	✓	✓	✓	✓	✓	✓	✓	
HeartBeatInterval	✓	✓	✓	✓	✓	✓	✓	✓	
KeepAliveInterval	✓	✓	✓	✓	✓	✓	✓	✓	
LocalAddress	✓	✓		✓	✓		✓	✓	✓ 9.0.0
LongRetryCount	✓	✓					✓	✓	
LongRetryInterval	✓	✓					✓	✓	
MaxInstances						✓			✓ 9.0.0
MaxInstancesPerClient						✓			
MaxMsgLength	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
MCAName	✓	✓		✓			✓		
MCAType	✓	✓		✓			✓	✓	

Table 126. Change, Copy, Create Channel parameters (continued)

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver	V 9.0.0 AMQP
MCAUserIdentifier			✓	✓		✓		✓	✓ 9.0.0
MessageCompression	✓	✓	✓	✓	✓	✓	✓	✓	
ModeName	✓	✓		✓	✓		✓	✓	
MsgExit	✓	✓	✓	✓			✓	✓	
MsgRetryCount			✓	✓				✓	
MsgRetryExit			✓	✓				✓	
MsgRetryInterval			✓	✓				✓	
MsgRetryUserData			✓	✓				✓	
MsgUserData	✓	✓	✓	✓			✓	✓	
NetworkPriority								✓	
NonPersistentMsgSp	✓	✓	✓	✓			✓	✓	
Password	✓	✓		✓	✓		✓		
V 9.0.0 *Port*									✓ 9.0.0
PropertyControl	✓	✓					✓	✓	
PutAuthority			✓	✓		✓ [4]		✓	
QMgrName					✓				
z/OS *QSGDisp*	✓	✓	✓	✓	✓	✓	✓	✓	
ReceiveExit	✓	✓	✓	✓	✓	✓	✓	✓	
ReceiveUserData	✓	✓	✓	✓	✓	✓	✓	✓	
Replace	✓	✓	✓	✓	✓	✓	✓	✓	
SecurityExit	✓	✓	✓	✓	✓	✓	✓	✓	
SecurityUserData	✓	✓	✓	✓	✓	✓	✓	✓	
SendExit	✓	✓	✓	✓	✓	✓	✓	✓	
SendUserData	✓	✓	✓	✓	✓	✓	✓	✓	

Table 126. Change, Copy, Create Channel parameters (continued)

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver	V 9.0.0 AMQP
SeqNumberWrap	✓	✓	✓	✓			✓	✓	
SharingConversations					✓	✓			
ShortRetryCount	✓	✓					✓	✓	
ShortRetryInterval	✓	✓					✓	✓	
SSLCipherSpec	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
SSLClientAuth		✓	✓	✓		✓		✓	✓ 9.0.0
SSLPeerName	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
ToChannelName (see footnote 2)	✓	✓	✓	✓	✓	✓	✓	✓	
TpName	✓	✓		✓	✓	✓	✓	✓	
TpRoot									✓ 9.0.0
TransportType	✓	✓	✓	✓	✓	✓	✓	✓	
UseCltId									✓ 9.0.0
UseDLQ	✓	✓	✓	✓			✓	✓	
UserIdentifier	✓	✓		✓	✓		✓		
XmitQName	✓	✓							

Note:

1. Required parameter on Change and Create Channel commands.
2. Required parameter on Copy Channel command.
3. Required parameter on Change, Create, and Copy Channel commands.
4. PUTAUT is valid for a channel type of SVRCONN on z/OS only.
5. Required parameter on Create Channel command if `TrpType` is TCP.
6. Required parameter on Create Channel command for a channel type of MQTT.

Required parameters (Change, Create Channel)

`ChannelName (MQCFST)`

Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Specifies the name of the channel definition to be changed, or created

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

This parameter is required on all types of channel; on a CLUSSDR it can be different from on the other channel types. If your convention for naming channels includes the name of the queue

manager, you can make a CLUSSDR definition using the +QMNAME+ construction, and IBM MQ substitutes the correct repository queue manager name in place of +QMNAME+. This facility applies only to IBM i, UNIX, Linux, and Windows only. See Configuring a queue manager cluster for more details.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

Specifies the type of the channel being changed, copied, or created. The value can be any of the following values:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLNTCONN
Client connection.

MQCHT_CLUSRCVR
Cluster-receiver.

MQCHT_CLUSSDR
Cluster-sender.

V 9.0.0 **MQCHT_AMQP**
AMQP.

Required parameters (Copy Channel)

FromChannelName (MQCFST)
From channel name (parameter identifier: MQCACF_FROM_CHANNEL_NAME).

The name of the existing channel definition that contains values for the attributes that are not specified in this command.

z/OS On z/OS, the queue manager searches for an object with the name you specify and a disposition of MQQSGD_Q_MGR or MQQSGD_COPY to copy from. This parameter is ignored if a value of MQQSGD_COPY is specified for *QSGDisposition*. In this case, an object with the name specified by *ToChannelName* and the disposition MQQSGD_GROUP is searched for to copy from.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

Specifies the type of the channel being changed, copied, or created. The value can be any of the following values:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLNTCONN
Client connection.

MQCHT_CLUSRCVR
Cluster-receiver.

MQCHT_CLUSSDR
Cluster-sender.

V 9.0.0 **MQCHT_AMQP**
AMQP.

ToChannelName (MQCFST)
To channel name (parameter identifier: MQCACF_TO_CHANNEL_NAME).

The name of the new channel definition.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Channel names must be unique; if a channel definition with this name exists, the value of *Replace* must be MQRP_YES. The channel type of the existing channel definition must be the same as the channel type of the new channel definition otherwise it cannot be replaced.

Optional parameters (Change, Copy, and Create Channel)

V 9.0.0 **AMQPKeepAlive (MQCFIN)**
The AMQP channel keep alive interval (parameter identifier: MQIACH_AMQP_KEEP_ALIVE).

The keep alive time for an AMQP channel in seconds. If the AMQP client has not sent any frames within the keep alive interval, then the connection is closed with a `amqp:resource-limit-exceeded` AMQP error condition.

This parameter is valid only for *ChannelType* values of MQCHT_AMQP.

BatchHeartbeat (MQCFIN)
The batch heartbeat interval (parameter identifier: MQIACH_BATCH_HB).

Batch heartbeating allows sender-type channels to determine whether the remote channel instance is still active, before going in-doubt. The value can be in the range 0 - 999999. A value of 0 indicates that batch heart-beating is not to be used. Batch heartbeat is measured in milliseconds.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

BatchInterval (MQCFIN)
Batch interval (parameter identifier: MQIACH_BATCH_INTERVAL). The approximate time in milliseconds that a channel keeps a batch open, if fewer than `BatchSize` messages or `BatchDataLimit` bytes have been transmitted in the current batch.

The batch is terminated when one of the following conditions is met:

- `BatchSize` messages have been sent.
- `BatchDataLimit` bytes have been sent.
- The transmission queue is empty and `BatchInterval` milliseconds have elapsed since the start of the batch.

BatchInterval must be in the range 0 - 999999999. A value of zero means that the batch is terminated as soon as the transmission queue becomes empty, or the BatchSize or BatchDataLimit is reached. This parameter applies only to channels with a *ChannelType* of: MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

BatchDataLimit (MQCFIN)

Batch data limit (parameter identifier: MQIACH_BATCH_DATA_LIMIT).

The limit, in kilobytes, of the amount of data that can be sent through a channel before taking a sync point. A sync point is taken after the message that caused the limit to be reached has flowed across the channel. A value of zero in this attribute means that no data limit is applied to batches over this channel.

The value must be in the range 0 - 999999. The default value is 5000.

The **BATCHLIM** parameter is supported on all platforms.

This parameter only applies to channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSRCVR, or MQCHT_CLUSSDR.

BatchSize (MQCFIN)

Batch size (parameter identifier: MQIACH_BATCH_SIZE).

The maximum number of messages that must be sent through a channel before a checkpoint is taken.

The batch size which is used is the lowest of the following:

- The *BatchSize* of the sending channel
- The *BatchSize* of the receiving channel
- The maximum number of uncommitted messages at the sending queue manager
- The maximum number of uncommitted messages at the receiving queue manager

The maximum number of uncommitted messages is specified by the **MaxUncommittedMsgs** parameter of the Change Queue Manager command.

Specify a value in the range 1 - 9999.

This parameter is not valid for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN.

CertificateLabel (MQCFST)

Certificate label (parameter identifier: MQCA_CERT_LABEL).

Certificate label for this channel to use.

The label identifies which personal certificate in the key repository is sent to the remote peer. If this attribute is blank, the certificate is determined by the queue manager **CertificateLabel** parameter.

Note that inbound channels (including receiver, cluster-receiver, unqualified server, and server-connection channels) only send the configured certificate if the IBM MQ version of the remote peer fully supports certificate label configuration, and the channel is using a TLS CipherSpec.

In all other cases, the queue manager **CertificateLabel** parameter determines the certificate sent. In particular, the following only ever receive the certificate configured by the **CertificateLabel** parameter of the queue manager, regardless of the channel-specific label setting:

- All current Java and JMS clients.
- Versions of IBM MQ prior to Version 8.0.

ChannelDesc (MQCFST)

Channel description (parameter identifier: MQCACH_DESC).

The maximum length of the string is MQ_CHANNEL_DESC_LENGTH.

Use characters from the character set, identified by the coded character set identifier (CCSID) for the message queue manager on which the command is executing, to ensure that the text is translated correctly.

ChannelMonitoring (MQCFIN)
Online monitoring data collection (parameter identifier: MQIA_MONITORING_CHANNEL).

Specifies whether online monitoring data is to be collected and, if so, the rate at which the data is collected. The value can be any of the following values:

MQMON_OFF
Online monitoring data collection is turned off for this channel.

MQMON_Q_MGR
The value of the queue manager's **ChannelMonitoring** parameter is inherited by the channel.

MQMON_LOW
If the value of the queue manager's *ChannelMonitoring* parameter is not MQMON_NONE, online monitoring data collection is turned on, with a low rate of data collection, for this channel.

MQMON_MEDIUM
If the value of the queue manager's *ChannelMonitoring* parameter is not MQMON_NONE, online monitoring data collection is turned on, with a moderate rate of data collection, for this channel.

MQMON_HIGH
If the value of the queue manager's *ChannelMonitoring* parameter is not MQMON_NONE, online monitoring data collection is turned on, with a high rate of data collection, for this channel.

ChannelStatistics (MQCFIN)
Statistics data collection (parameter identifier: MQIA_STATISTICS_CHANNEL).

Specifies whether statistics data is to be collected and, if so, the rate at which the data is collected. The value can be:

MQMON_OFF
Statistics data collection is turned off for this channel.

MQMON_Q_MGR
The value of the queue manager's **ChannelStatistics** parameter is inherited by the channel.

MQMON_LOW
If the value of the queue manager's *ChannelStatistics* parameter is not MQMON_NONE, online monitoring data collection is turned on, with a low rate of data collection, for this channel.

MQMON_MEDIUM
If the value of the queue manager's *ChannelStatistics* parameter is not MQMON_NONE, online monitoring data collection is turned on, with a moderate rate of data collection, for this channel.

MQMON_HIGH
If the value of the queue manager's *ChannelStatistics* parameter is not MQMON_NONE, online monitoring data collection is turned on, with a high rate of data collection, for this channel.

z/OS On z/OS systems, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying `LOW`, `MEDIUM`, or `HIGH` makes no difference to your results.

ClientChannelWeight (MQCFIN)
Client Channel Weight (parameter identifier: MQIACH_CLIENT_CHANNEL_WEIGHT).

The client channel weighting attribute is used so client channel definitions can be selected at random, with the larger weightings having a higher probability of selection, when more than one suitable definition is available.

Specify a value in the range 0 - 99. The default is 0.

This parameter is only valid for channels with a ChannelType of MQCHT_CLNTCONN

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster to which the channel belongs.

This parameter applies only to channels with a *ChannelType* of:

- MQCHT_CLUSSDR
- MQCHT_CLUSRCVR

Only one of the values of *ClusterName* and *ClusterNamelist* can be nonblank; the other must be blank.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

ClusterNamelist (MQCFST)
Cluster namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

The name, of the namelist, that specifies a list of clusters to which the channel belongs.

This parameter applies only to channels with a *ChannelType* of:

- MQCHT_CLUSSDR
- MQCHT_CLUSRCVR

Only one of the values of *ClusterName* and *ClusterNamelist* can be nonblank; the other must be blank.

CLWLChannelPriority (MQCFIN)
Channel priority for the purposes of cluster workload distribution (parameter identifier: MQIACH_CLWL_CHANNEL_PRIORITY).

Specify a value in the range 0 - 9 where 0 is the lowest priority and 9 is the highest.

This parameter applies only to channels with a *ChannelType* of:

- MQCHT_CLUSSDR
- MQCHT_CLUSRCVR

CLWLChannelRank (MQCFIN)
Channel rank for the purposes of cluster workload distribution (parameter identifier: MQIACH_CLWL_CHANNEL_RANK).

Specify a value in the range 0 - 9 where 0 is the lowest priority and 9 is the highest.

This parameter applies only to channels with a *ChannelType* of:

- MQCHT_CLUSSDR
- MQCHT_CLUSRCVR

CLWLChannelWeight (MQCFIN)
Channel weighting for the purposes of cluster workload distribution (parameter identifier: MQIACH_CLWL_CHANNEL_WEIGHT).

Specify a weighting for the channel for use in workload management. Specify a value in the range 1 - 99 where 1 is the lowest priority and 99 is the highest.

This parameter applies only to channels with a *ChannelType* of:

- MQCHT_CLUSSDR

- MQCHT_CLUSRCVR

z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

ConnectionAffinity (MQCFIN)
Channel Affinity (parameter identifier: MQIACH_CONNECTION_AFFINITY)

The channel affinity attribute specifies whether client applications that connect multiple times using the same queue manager name, use the same client channel. The value can be any of the following values:

MQCAFTY_PREFERRED
The first connection in a process reading a client channel definition table (CCDT) creates a list of applicable definitions based on the weighting with any zero ClientChannelWeight definitions first in alphabetical order. Each connection in the process attempts to connect using the first definition in the list. If a connection is unsuccessful the next definition is used. Unsuccessful nonzero ClientChannelWeight definitions are moved to the end of the list. Zero ClientChannelWeight definitions remain at the start of the list and are selected first for each connection. For C, C++ and .NET (including fully managed .NET) clients the list is updated if the CCDT has been modified since the list was created. Each client process with the same host name creates the same list.

This value is the default value.

MQCAFTY_NONE
The first connection in a process reading a CCDT creates a list of applicable definitions. All connections in a process independently select an applicable definition based on the weighting with any applicable zero ClientChannelWeight definitions selected first in alphabetical order. For C, C++ and .NET (including fully managed .NET) clients the list is updated if the CCDT has been modified since the list was created.

This parameter is only valid for channels with a ChannelType of MQCHT_CLNTCONN.

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

Multi On Multiplatforms, the maximum length of the string is 264.

z/OS On z/OS, the maximum length of the string is 48.

Specify *ConnectionName* as a comma-separated list of names of machines for the stated *TransportType*. Typically, only one machine name is required. You can provide multiple machine names to configure multiple connections with the same properties. The connections are tried in the order they are specified in the connection list until a connection is successfully established. If no connection is

successful, the channel starts to try processing again. Connection lists are an alternative to queue manager groups to configure connections for reconnectable clients, and also to configure channel connections to multi-instance queue managers.

Specify the name of the machine as required for the stated *TransportType*:

- For MQXPT_LU62 on IBM i, and UNIX, specify the name of the CPI-C communications side object. On Windows specify the CPI-C symbolic destination name.

  z/OS On z/OS, there are two forms in which to specify the value:

  Logical unit name
  The logical unit information for the queue manager, comprising the logical unit name, TP name, and optional mode name. This name can be specified in one of three forms:

Form	Example
luname	IGY12355
luname/TPname	IGY12345/APING
luname/TPname/modename	IGY12345/APINGD/#INTER

  For the first form, the TP name and mode name must be specified for the *TpName* and *ModeName* parameters; otherwise these parameters must be blank.

  Note: For client-connection channels, only the first form is allowed.

  Symbolic name
  The symbolic destination name for the logical unit information for the queue manager, as defined in the side information data set. The **TpName** and **ModeName** parameters must be blank.

  Note: For cluster-receiver channels, the side information is on the other queue managers in the cluster. Alternatively, in this case it can be a name that a channel auto-definition exit can resolve into the appropriate logical unit information for the local queue manager.

  The specified or implied LU name can be that of a VTAM generic resources group.

- For MQXPT_TCP, you can specify a connection name, or a connection list, containing the host name or the network address of the remote machine. Separate connection names in a connection list with commas.

  z/OS On z/OS, the connection name can include the IP_name of a z/OS dynamic DNS group or a network dispatcher input port. Do not include this parameter for channels with a *ChannelType* value of MQCHT_CLUSSDR.

  Multi On Multiplatforms, the TCP/IP connection name parameter of a cluster-receiver channel is optional. If you leave the connection name blank, IBM MQ generates a connection name for you, assuming the default port and using the current IP address of the system. You can override the default port number, but still use the current IP address of the system. For each connection name leave the IP name blank, and provide the port number in parentheses; for example:

  ```
  (1415)
  ```

  The generated **CONNAME** is always in the dotted decimal (IPv4) or hexadecimal (IPv6) form, rather than in the form of an alphanumeric DNS host name.

- For MQXPT_NETBIOS specify the NetBIOS station name.
- For MQXPT_SPX specify the 4 byte network address, the 6 byte node address, and the 2 byte socket number. These values must be entered in hexadecimal, with a period separating the network and node addresses. The socket number must be enclosed in brackets, for example:

  ```
  0a0b0c0d.804abcde23a1(5e86)
  ```

If the socket number is omitted, the IBM MQ default value (5e86 hex) is assumed.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, MQCHT_CLNTCONN, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

Note: If you are using clustering between IPv6 -only and IPv4 -only queue managers, do not specify an IPv6 network address as the *ConnectionName* for cluster-receiver channels. A queue manager that is capable only of IPv4 communication is unable to start a cluster sender channel definition that specifies the *ConnectionName* in IPv6 hexadecimal form. Consider, instead, using host names in a heterogeneous IP environment.

DataConversion (MQCFIN)
Whether sender must convert application data (parameter identifier: MQIACH_DATA_CONVERSION).

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

The value can be any of the following values:

MQCDC_NO_SENDER_CONVERSION
No conversion by sender.

MQCDC_SENDER_CONVERSION
Conversion by sender.

DefaultChannelDisposition (MQCFIN)
Intended disposition of the channel when activated or started (parameter identifier: MQIACH_DEF_CHANNEL_DISP).

This parameter applies to z/OS only.

The value can be any of the following values:

MQCHLD_PRIVATE
The intended use of the object is as a private channel.

This value is the default value.

MQCHLD_FIXSHARED
The intended use of the object is as a fixshared channel.

MQCHLD_SHARED
The intended use of the object is as a shared channel.

DefReconnect (MQCFIN)
Client channel default reconnection option (parameter identifier: MQIACH_DEF_RECONNECT).

The default automatic client reconnection option. You can configure a IBM MQ MQI client to automatically reconnect a client application. The IBM MQ MQI client tries to reconnect to a queue manager after a connection failure. It tries to reconnect without the application client issuing an MQCONN or MQCONNX MQI call.

MQRCN_NO
MQRCN_NO is the default value.

Unless overridden by **MQCONNX**, the client is not reconnected automatically.

MQRCN_YES
Unless overridden by **MQCONNX**, the client reconnects automatically.

MQRCN_Q_MGR
Unless overridden by **MQCONNX**, the client reconnects automatically, but only to the same queue manager. The QMGR option has the same effect as MQCNO_RECONNECT_Q_MGR.

MQRCN_DISABLED
Reconnection is disabled, even if requested by the client program using the **MQCONNX** MQI call.

Table 127. Automatic reconnection depends on the values set in the application and in the channel definition

DefReconnect	Reconnection options set in the application			
	MQCNO_RECONNECT	MQCNO_RECONNECT_Q_MGR	MQCNO_RECONNECT_AS_DEF	MQCNO_RECONNECT_DISABLED
MQRCN_NO	YES	QMGR	NO	NO
MQRCN_YES	YES	QMGR	YES	NO
MQRCN_Q_MGR	YES	QMGR	QMGR	NO
MQRCN_DISABLED	NO	NO	NO	NO

This parameter is valid only for a *ChannelType* value of MQCHT_CLNTCONN.

DiscInterval (MQCFIN)
Disconnection interval (parameter identifier: MQIACH_DISC_INTERVAL).

This interval defines the maximum number of seconds that the channel waits for messages to be put on a transmission queue before terminating the channel. A value of zero causes the message channel agent to wait indefinitely.

Specify a value in the range 0 - 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER MQCHT_SERVER, MQCHT_SVRCONN, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

For server-connection channels using the TCP protocol, this interval is the minimum time in seconds for which the server-connection channel instance remains active without any communication from its partner client. A value of zero disables this disconnect processing. The server-connection inactivity interval only applies between MQ API calls from a client, so no client is disconnected during an extended MQGET with wait call. This attribute is ignored for server-connection channels using protocols other than TCP.

HeaderCompression (MQCFIL)
Header data compression techniques supported by the channel (parameter identifier: MQIACH_HDR_COMPRESSION).

The list of header data compression techniques supported by the channel. For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference with the first compression technique supported by the remote end of the channel being used.

The mutually supported compression techniques of the channel are passed to the message exit of the sending channel where the compression technique used can be altered on a per message basis. Compression alters the data passed to send and receive exits.

Specify one or more of:

MQCOMPRESS_NONE
No header data compression is performed. This value is the default value.

MQCOMPRESS_SYSTEM
Header data compression is performed.

HeartbeatInterval (MQCFIN)
Heartbeat interval (parameter identifier: MQIACH_HB_INTERVAL).

The interpretation of this parameter depends on the channel type, as follows:

- For a channel type of MQCHT_SENDER, MQCHT_SERVER, MQCHT_RECEIVER, MQCHT_REQUESTER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR, this interval is the time in seconds between heartbeat flows passed from the sending MCA when there are no messages on

the transmission queue. This interval gives the receiving MCA the opportunity to quiesce the channel. To be useful, *HeartbeatInterval* must be less than *DiscInterval*. However, the only check is that the value is within the permitted range.

This type of heartbeat is supported on the following platforms: IBM i, UNIX, Windows, and z/OS.

- For a channel type of MQCHT_CLNTCONN or MQCHT_SVRCONN, this interval is the time in seconds between heartbeat flows passed from the server MCA when that MCA has issued an MQGET call with the MQGMO_WAIT option on behalf of a client application. This interval allows the server MCA to handle situations where the client connection fails during an MQGET with MQGMO_WAIT.

  This type of heartbeat is supported on all platforms.

The value must be in the range 0 - 999 999. A value of 0 means that no heartbeat exchange occurs. The value that is used is the larger of the values specified at the sending side and receiving side.

KeepAliveInterval (MQCFIN)

KeepAlive interval (parameter identifier: MQIACH_KEEP_ALIVE_INTERVAL).

Specifies the value passed to the communications stack for KeepAlive timing for the channel.

For this attribute to be effective, TCP/IP keepalive must be enabled. On z/OS, you enable TCP/IP keepalive by issuing the Change Queue Manager command with a value of MQTCPKEEP in the *TCPKeepAlive* parameter; if the *TCPKeepAlive* queue manager parameter has a value of MQTCPKEEP_NO, the value is ignored, and the KeepAlive facility is not used. On other platforms, TCP/IP keepalive is enabled when the KEEPALIVE=YES parameter is specified in the TCP stanza in the distributed queuing configuration file, qm.ini, or through the IBM MQ Explorer. Keepalive must also be enabled within TCP/IP itself, using the TCP profile configuration data set.

Although this parameter is available on all platforms, its setting is implemented only on z/OS. On platforms other than z/OS, you can access and modify the parameter, but it is only stored and forwarded; there is no functional implementation of the parameter. This parameter is useful in a clustered environment where a value set in a cluster-receiver channel definition on AIX, for example, flows to (and is implemented by) z/OS queue managers that are in, or join, the cluster.

Specify either:

integer
The KeepAlive interval to be used, in seconds, in the range 0 - 99 999. If you specify a value of 0, the value used is that specified by the INTERVAL statement in the TCP profile configuration data set.

MQKAI_AUTO
The KeepAlive interval is calculated based upon the negotiated heartbeat value as follows:

- If the negotiated *HeartbeatInterval* is greater than zero, KeepAlive interval is set to that value plus 60 seconds.
- If the negotiated *HeartbeatInterval* is zero, the value used is that specified by the INTERVAL statement in the TCP profile configuration data set.

Multi On Multiplatforms, if you need the functionality provided by the **KeepAliveInterval** parameter, use the **HeartBeatInterval** parameter.

LocalAddress (MQCFST)

Local communications address for the channel (parameter identifier: MQCACH_LOCAL_ADDRESS).

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

The value that you specify depends on the transport type *(TransportType)* to be used:

TCP/IP
The value is the optional IP address and optional port or port range to be used for outbound TCP/IP communications. The format for this information is as follows:

```
LOCLADDR([ip-addr][(low-port[,high-port])][,[ip-addr][(low-port[,high-port])]])
```

where `ip-addr` is specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric form, and `low-port` and `high-port` are port numbers enclosed in parentheses. All are optional.

Specify *[,[ip-addr][(low-port[,high-port])]]* multiple times for each additional local address. Use multiple local addresses if you want to specify a specific subset of local network adapters. You can also use *[,[ip-addr][(low-port[,high-port])]]* to represent a particular local network address on different servers that are part of a multi-instance queue manager configuration.

All Others

The value is ignored; no error is diagnosed.

Use this parameter if you want a channel to use a particular IP address, port, or port range for outbound communications. This parameter is useful when a machine is connected to multiple networks with different IP addresses.

Examples of use

Value	Meaning
9.20.4.98	Channel binds to this address locally
9.20.4.98 (1000)	Channel binds to this address and port 1000 locally
9.20.4.98 (1000,2000)	Channel binds to this address and uses a port in the range 1000 - 2000 locally
(1000)	Channel binds to port 1000 locally
(1000,2000)	Channel binds to a port in the range 1000 - 2000 locally

This parameter is valid for the following channel types:

- MQCHT_SENDER
- MQCHT_SERVER
- MQCHT_REQUESTER
- MQCHT_CLNTCONN
- MQCHT_CLUSRCVR
- MQCHT_CLUSSDR

Note:

- Do not confuse this parameter with *ConnectionName*. The *LocalAddress* parameter specifies the characteristics of the local communications; the *ConnectionName* parameter specifies how to reach a remote queue manager.

`LongRetryCount (MQCFIN)`

Long retry count (parameter identifier: MQIACH_LONG_RETRY).

When a sender or server channel is attempting to connect to the remote machine, and the count specified by *`ShortRetryCount`* has been exhausted, this count specifies the maximum number of further attempts that are made to connect to the remote machine, at intervals specified by *`LongRetryInterval`*.

If this count is also exhausted without success, an error is logged to the operator, and the channel is stopped. The channel must later be restarted with a command (it is not started automatically by the channel initiator), and it then makes only one attempt to connect, as it is assumed that the problem has now been cleared by the administrator. The retry sequence is not carried out again until after the channel has successfully connected.

Specify a value in the range 0 - 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

LongRetryInterval (MQCFIN)
Long timer (parameter identifier: MQIACH_LONG_TIMER).

Specifies the long retry wait interval for a sender or server channel that is started automatically by the channel initiator. It defines the interval in seconds between attempts to establish a connection to the remote machine, after the count specified by *ShortRetryCount* has been exhausted.

The time is approximate; zero means that another connection attempt is made as soon as possible.

Specify a value in the range 0 - 999 999. Values exceeding this value are treated as 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

MaxInstances (MQCFIN)
Maximum number of simultaneous instances of a server-connection channel or an AMQP channel (parameter identifier: MQIACH_MAX_INSTANCES).

Specify a value in the range 0 - 999 999 999.

The default value is 999 999 999.

A value of zero indicates that no client connections are allowed on the channel.

If the value is reduced below the number of instances of the server-connection channel that are currently running, the running channels are not affected. This parameter applies even if the value is zero. However, if the value is reduced below the number of instances of the server-connection channel that are currently running, then new instances cannot be started until sufficient existing instances have ceased to run.

V 9.0.0 If an AMQP client attempts to connect to an AMQP channel, and the number of connected clients has reached MaxInstances, the channel closes the connection with a close frame. The close frame contains the following message: `amqp:resource-limit-exceeded`. If a client connects with an ID that is already connected (that is, it performs a client-takeover), and the client is permitted to take over the connection, the takeover will succeed regardless of whether the number of connected clients has reached MaxInstances.

This parameter is valid only for channels with a *ChannelType* value of MQCHT_SVRCONN or MQCHT_AMQP.

MaxInstancesPerClient (MQCFIN)
Maximum number of simultaneous instances of a server-connection channel that can be started from a single client (parameter identifier: MQIACH_MAX_INSTS_PER_CLIENT). In this context, connections that originate from the same remote network address are regarded as coming from the same client.

Specify a value in the range 0 - 999 999 999.

The default value is 999 999 999.

A value of zero indicates that no client connections are allowed on the channel.

If the value is reduced below the number of instances of the server-connection channel that are currently running from individual clients, the running channels are not affected. This parameter applies even if the value is zero. However, if the value is reduced below the number of instances of the server-connection channel that are currently running from individual clients, new instances from those clients cannot start until sufficient existing instances have ceased to run.

This parameter is valid only for channels with a *ChannelType* value of MQCHT_SVRCONN.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIACH_MAX_MSG_LENGTH).

Specifies the maximum message length that can be transmitted on the channel. This value is compared with the value for the remote channel and the actual maximum is the lower of the two values.

The value zero means the maximum message length for the queue manager.

The lower limit for this parameter is 0. The maximum message length is 100 MB (104 857 600 bytes).

MCAName (MQCFST)
Message channel agent name (parameter identifier: MQCACH_MCA_NAME).

Note: An alternative way of providing a user ID for a channel to run under is to use channel authentication records. With channel authentication records, different connections can use the same channel while using different credentials. If both MCAUSER on the channel is set and channel authentication records are used to apply to the same channel, the channel authentication records take precedence. The MCAUSER on the channel definition is only used if the channel authentication record uses USERSRC(CHANNEL). For more details, see Channel authentication records

This parameter is reserved, and if specified can be set only to blanks.

The maximum length of the string is MQ_MCA_NAME_LENGTH.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

MCAType (MQCFIN)
Message channel agent type (parameter identifier: MQIACH_MCA_TYPE).

Specifies the type of the message channel agent program.

Multi On Multiplatforms, this parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, or MQCHT_CLUSSDR.

z/OS On z/OS, this parameter is valid only for a *ChannelType* value of MQCHT_CLURCVR.

The value can be any of the following values:

MQMCAT_PROCESS
Process.

MQMCAT_THREAD
Thread.

MCAUserIdentifier (MQCFST)
Message channel agent user identifier (parameter identifier: MQCACH_MCA_USER_ID).

If this parameter is nonblank, it is the user identifier which is to be used by the message channel agent for authorization to access IBM MQ resources, including (if *PutAuthority* is MQPA_DEFAULT) authorization to put the message to the destination queue for receiver or requester channels.

If it is blank, the message channel agent uses its default user identifier.

This user identifier can be overridden by one supplied by a channel security exit.

This parameter is not valid for channels with a *ChannelType* of MQCHT_SDR, MQCHT_SVR, MQCHT_CLNTCONN, MQCHT_CLUSSDR.

The maximum length of the MCA user identifier depends on the environment in which the MCA is running. MQ_MCA_USER_ID_LENGTH gives the maximum length for the environment for which your application is running. MQ_MAX_MCA_USER_ID_LENGTH gives the maximum for all supported environments.

On Windows, you can optionally qualify a user identifier with the domain name in the following format:

```
user@domain
```

MessageCompression (MQCFIL)
The list of message data compression techniques supported by the channel (parameter identifier: MQIACH_MSG_COMPRESSION). For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference with the first compression technique supported by the remote end of the channel being used.

The mutually supported compression techniques of the channel are passed to the message exit of the sending channel where the compression technique used can be altered on a per message basis. Compression alters the data passed to send and receive exits.

Specify one or more of:

MQCOMPRESS_NONE
No message data compression is performed. This value is the default value.

MQCOMPRESS_RLE
Message data compression is performed using run-length encoding.

MQCOMPRESS_ZLIBFAST
Message data compression is performed using ZLIB encoding with speed prioritized.

MQCOMPRESS_ZLIBHIGH
Message data compression is performed using ZLIB encoding with compression prioritized.

MQCOMPRESS_ANY
Any compression technique supported by the queue manager can be used. This value is only valid for receiver, requester, and server-connection channels.

ModeName (MQCFST)
Mode name (parameter identifier: MQCACH_MODE_NAME).

This parameter is the LU 6.2 mode name.

The maximum length of the string is MQ_MODE_NAME_LENGTH.

- On IBM i, NSS Client HP Integrity NonStop Server, UNIX, and Windows, this parameter can be set only to blanks. The actual name is taken instead from the CPI-C Communications Side Object or (on Windows) from the CPI-C symbolic destination name properties.

This parameter is valid only for channels with a *TransportType* of MQXPT_LU62. It is not valid for receiver or server-connection channels.

MsgExit (MQCFSL)
Message exit name (parameter identifier: MQCACH_MSG_EXIT_NAME).

If a nonblank name is defined, the exit is invoked immediately after a message has been retrieved from the transmission queue. The exit is given the entire application message and message descriptor for modification.

For channels with a channel type (*ChannelType*) of MQCHT_SVRCONN or MQCHT_CLNTCONN, this parameter is accepted but ignored, since message exits are not invoked for such channels.

The format of the string is the same as for *SecurityExit*.

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

You can specify a list of exit names by using an MQCFSL structure instead of an MQCFST structure.

- The exits are invoked in the order specified in the list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.

- The total length of all the exit names in the list (excluding trailing blanks in each name) must not exceed MQ_TOTAL_EXIT_NAME_LENGTH. An individual string must not exceed MQ_EXIT_NAME_LENGTH.
- On z/OS, you can specify the names of up to eight exit programs.

MsgRetryCount (MQCFIN)
Message retry count (parameter identifier: MQIACH_MR_COUNT).

Specifies the number of times that a failing message must be retried.

Specify a value in the range 0 - 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER, MQCHT_REQUESTER, or MQCHT_CLUSRCVR.

MsgRetryExit (MQCFST)
Message retry exit name (parameter identifier: MQCACH_MR_EXIT_NAME).

If a nonblank name is defined, the exit is invoked before performing a wait before retrying a failing message.

The format of the string is the same as for *SecurityExit*.

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER, MQCHT_REQUESTER, or MQCHT_CLUSRCVR.

MsgRetryInterval (MQCFIN)
Message retry interval (parameter identifier: MQIACH_MR_INTERVAL).

Specifies the minimum time interval in milliseconds between retries of failing messages.

Specify a value in the range 0 - 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER, MQCHT_REQUESTER, or MQCHT_CLUSRCVR.

MsgRetryUserData (MQCFST)
Message retry exit user data (parameter identifier: MQCACH_MR_EXIT_USER_DATA).

Specifies user data that is passed to the message retry exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER, MQCHT_REQUESTER, or MQCHT_CLUSRCVR.

MsgUserData (MQCFSL)
Message exit user data (parameter identifier: MQCACH_MSG_EXIT_USER_DATA).

Specifies user data that is passed to the message exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

For channels with a channel type (*ChannelType*) of MQCHT_SVRCONN or MQCHT_CLNTCONN, this parameter is accepted but ignored, since message exits are not invoked for such channels.

You can specify a list of exit user data strings by using an MQCFSL structure instead of an MQCFST structure.

- Each exit user data string is passed to the exit at the same ordinal position in the *MsgExit* list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.

- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all the exit user data in the list (excluding trailing blanks in each string) must not exceed MQ_TOTAL_EXIT_DATA_LENGTH. An individual string must not exceed MQ_EXIT_DATA_LENGTH.
- On z/OS, you can specify up to eight strings.

NetworkPriority (MQCFIN)
Network priority (parameter identifier: MQIACH_NETWORK_PRIORITY).

The priority for the network connection. If there are multiple paths available, distributed queuing selects the path with the highest priority.

The value must be in the range 0 (lowest) - 9 (highest).

This parameter applies only to channels with a *ChannelType* of MQCHT_CLUSRCVR

NonPersistentMsgSpeed (MQCFIN)
Speed at which nonpersistent messages are to be sent (parameter identifier: MQIACH_NPM_SPEED).

This parameter is supported in the following environments: IBM i, UNIX, Linux, and Windows.

Specifying MQNPMS_FAST means that nonpersistent messages on a channel need not wait for a syncpoint before being made available for retrieval. The advantage of this is that nonpersistent messages become available for retrieval far more quickly. The disadvantage is that because they do not wait for a syncpoint, they might be lost if there is a transmission failure.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_RECEIVER, MQCHT_REQUESTER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR. The value can be any of the following values:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

Password (MQCFST)
Password (parameter identifier: MQCACH_PASSWORD).

This parameter is used by the message channel agent when attempting to initiate a secure SNA session with a remote message channel agent. On IBM i, HP Integrity NonStop Server, and UNIX, it is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, MQCHT_CLNTCONN, or MQCHT_CLUSSDR. On z/OS, it is valid only for a *ChannelType* value of MQCHT_CLNTCONN.

The maximum length of the string is MQ_PASSWORD_LENGTH. However, only the first 10 characters are used.

V 9.0.0 **Port (MQCFIN)**
Port number (parameter identifier MQIACH_PORT).

The port number used to connect an AMQP channel. The default port for AMQP 1.0 connections is 5672. If you are already using port 5672, you can specify a different port.

This attribute is applicable to AMQP channels.

PropertyControl (MQCFIN)
Property control attribute (parameter identifier MQIA_PROPERTY_CONTROL).

Specifies what happens to properties of messages when the message is about to be sent to a V6 or prior queue manager (a queue manager that does not understand the concept of a property descriptor). The value can be any of the following values:

MQPROP_COMPATIBILITY
If the message contains a property with a prefix of **mcd.**, **jms.**, **usr.** or **mqext.**, all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except those properties contained in the message descriptor (or extension), are discarded and are no longer accessible to the application.

This value is the default value; it allows applications which expect JMS-related properties to be in an MQRFH2 header in the message data to continue to work unmodified.

MQPROP_NONE
All properties of the message, except those properties in the message descriptor (or extension), are removed from the message before the message is sent to the remote queue manager.

MQPROP_ALL
All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except those properties in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

This attribute is applicable to Sender, Server, Cluster Sender, and Cluster Receiver channels.

PutAuthority (MQCFIN)
Put authority (parameter identifier: MQIACH_PUT_AUTHORITY).

Specifies which user identifiers are used to establish authority to put messages to the destination queue (for message channels) or to execute an MQI call (for MQI channels).

This parameter is valid only for channels with a *ChannelType* value of MQCHT_RECEIVER, MQCHT_REQUESTER, MQCHT_CLUSRCVR, or MQCHT_SVRCONN. The value can be any of the following values:

MQPA_DEFAULT
Default user identifier is used.

z/OS On z/OS, MQPA_DEFAULT might involve using both the user ID received from the network and that derived from MCAUSER.

MQPA_CONTEXT
The user ID from the *UserIdentifier* field of the message descriptor is used.

z/OS On z/OS, MQPA_CONTEXT might involve also using the user ID received from the network or that derived from MCAUSER, or both.

MQPA_ALTERNATE_OR_MCA
The user ID from the *UserIdentifier* field of the message descriptor is used. Any user ID received from the network is not used. This value is supported only on z/OS.

MQPA_ONLY_MCA
The user ID derived from MCAUSER is used. Any user ID received from the network is not used. This value is supported only on z/OS.

QMgrName (MQCFST)
Queue manager name (parameter identifier: MQCA_Q_MGR_NAME).

For channels with a *ChannelType* of MQCHT_CLNTCONN, this name is the name of a queue manager to which a client application can request connection.

For channels of other types, this parameter is not valid. The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

z/OS

QSGDisposition (MQCFIN)

Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be any of the following values:

QSGDisposition	Change	Copy, Create
MQQSGD_COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.	The object is defined on the page set of the queue manager that executes the command using the MQQSGD_GROUP object of the same name as the *ToChannelName* object (for Copy) or *ChannelName* object (for Create).
MQQSGD_GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command. If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero: `DEFINE CHANNEL(channel-name) CHLTYPE(type) REPLACE QSGDISP(COPY)` The Change for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.	The object definition resides in the shared repository. This definition is allowed only if the queue manager is in a queue-sharing group. If the definition is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to attempt to make or refresh local copies on page set zero: `DEFINE CHANNEL(channe-name) CHLTYPE(type) REPLACE QSGDISP(COPY)` The Copy or Create for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.
MQQSGD_PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with MQQSGD_Q_MGR or MQQSGD_COPY. Any object residing in the shared repository is unaffected.	Not permitted.
MQQSGD_Q_MGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command. This value is the default value.	The object is defined on the page set of the queue manager that executes the command. This value is the default value.

ReceiveExit (MQCFSL)

Receive exit name (parameter identifier: MQCACH_RCV_EXIT_NAME).

If a nonblank name is defined, the exit is invoked before data received from the network is processed. The complete transmission buffer is passed to the exit and the contents of the buffer can be modified as required.

The format of the string is the same as for *SecurityExit*.

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

You can specify a list of exit names by using an MQCFSL structure instead of an MQCFST structure.

- The exits are invoked in the order specified in the list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all the exit names in the list (excluding trailing blanks in each name) must not exceed MQ_TOTAL_EXIT_NAME_LENGTH. An individual string must not exceed MQ_EXIT_NAME_LENGTH.
- On z/OS, you can specify the names of up to eight exit programs.

ReceiveUserData (MQCFSL)
Receive exit user data (parameter identifier: MQCACH_RCV_EXIT_USER_DATA).

Specifies user data that is passed to the receive exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

You can specify a list of exit user data strings by using an MQCFSL structure instead of an MQCFST structure.

- Each exit user data string is passed to the exit at the same ordinal position in the *ReceiveExit* list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all the exit user data in the list (excluding trailing blanks in each string) must not exceed MQ_TOTAL_EXIT_DATA_LENGTH. An individual string must not exceed MQ_EXIT_DATA_LENGTH.
- On z/OS, you can specify up to eight strings.

Replace (MQCFIN)
Replace channel definition (parameter identifier: MQIACF_REPLACE).

The value can be any of the following values:

MQRP_YES
Replace existing definition.

If *ChannelType* is MQCHT_CLUSSDR, MQRP_YES can be specified only if the channel was created manually.

MQRP_NO
Do not replace existing definition.

SecurityExit (MQCFST)
Security exit name (parameter identifier: MQCACH_SEC_EXIT_NAME).

If a nonblank name is defined, the security exit is invoked at the following times:

- Immediately after establishing a channel.

  Before any messages are transferred, the exit is enabled to instigate security flows to validate connection authorization.
- Upon receipt of a response to a security message flow.

  Any security message flows received from the remote processor on the remote machine are passed to the exit.

The exit is given the entire application message and message descriptor for modification.

The format of the string depends on the platform, as follows:

- On IBM i and UNIX, it is of the form
  `libraryname(functionname)`

  Note: On IBM i systems, the following form is also supported for compatibility with older releases:
  `progname libname`

  where *progname* occupies the first 10 characters, and *libname* the second 10 characters (both blank-padded to the right if necessary).
- On Windows, it is of the form
  `dllname(functionname)`

  where *dllname* is specified without the suffix `.DLL`.
- On z/OS, it is a load module name, maximum length 8 characters (128 characters are allowed for exit names for client-connection channels, subject to a maximum total length of 999).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

SecurityUserData (MQCFST)
Security exit user data (parameter identifier: MQCACH_SEC_EXIT_USER_DATA).

Specifies user data that is passed to the security exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

SendExit (MQCFSL)
Send exit name (parameter identifier: MQCACH_SEND_EXIT_NAME).

If a nonblank name is defined, the exit is invoked immediately before data is sent out on the network. The exit is given the complete transmission buffer before it is transmitted; the contents of the buffer can be modified as required.

The format of the string is the same as for *SecurityExit*.

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

You can specify a list of exit names by using an MQCFSL structure instead of an MQCFST structure.

- The exits are invoked in the order specified in the list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all the exit names in the list (excluding trailing blanks in each name) must not exceed MQ_TOTAL_EXIT_NAME_LENGTH. An individual string must not exceed MQ_EXIT_NAME_LENGTH.
- On z/OS, you can specify the names of up to eight exit programs.

SendUserData (MQCFSL)
Send exit user data (parameter identifier: MQCACH_SEND_EXIT_USER_DATA).

Specifies user data that is passed to the send exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

You can specify a list of exit user data strings by using an MQCFSL structure instead of an MQCFST structure.

- Each exit user data string is passed to the exit at the same ordinal position in the *SendExit* list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all the exit user data in the list (excluding trailing blanks in each string) must not exceed MQ_TOTAL_EXIT_DATA_LENGTH. An individual string must not exceed MQ_EXIT_DATA_LENGTH.
- On z/OS, you can specify up to eight strings.

SeqNumberWrap (MQCFIN)
Sequence wrap number (parameter identifier: MQIACH_SEQUENCE_NUMBER_WRAP).

Specifies the maximum message sequence number. When the maximum is reached, sequence numbers wrap to start again at 1.

The maximum message sequence number is not negotiable; the local and remote channels must wrap at the same number.

Specify a value in the range 100 - 999 999 999.

This parameter is not valid for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN.

SharingConversations (MQCFIN)
Maximum number of sharing conversations (parameter identifier: MQIACH_SHARING_CONVERSATIONS).

Specifies the maximum number of conversations that can share a particular TCP/IP MQI channel instance (socket).

Specify a value in the range 0 - 999 999 999. The default value is 10 and the migrated value is 10.

This parameter is valid only for channels with a *ChannelType* of MQCHT_CLNTCONN or MQCHT_SVRCONN. It is ignored for channels with a *TransportType* other than MQXPT_TCP.

The number of shared conversations does not contribute to the *MaxInstances* or *MaxInstancesPerClient* totals.

A value of:

1 Means that there is no sharing of conversations over a TCP/IP channel instance, but client heartbeating is available whether in an MQGET call or not, read ahead and client asynchronous consumption are available, and channel quiescing is more controllable.

0 Specifies no sharing of conversations over a TCP/IP channel instance. The channel instance runs in a mode before that of IBM WebSphere MQ Version 7.0, regarding:

- Administrator stop-quiesce
- Heartbeating
- Read ahead
- Client asynchronous consumption

ShortRetryCount (MQCFIN)
Short retry count (parameter identifier: MQIACH_SHORT_RETRY).

The maximum number of attempts that are made by a sender or server channel to establish a connection to the remote machine, at intervals specified by *ShortRetryInterval* before the (normally longer) *LongRetryCount* and *LongRetryInterval* are used.

Retry attempts are made if the channel fails to connect initially (whether it is started automatically by the channel initiator or by an explicit command), and also if the connection fails after the channel has successfully connected. However, if the cause of the failure is such that retry is unlikely to be successful, retries are not attempted.

Specify a value in the range 0 - 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

ShortRetryInterval (MQCFIN)

Short timer (parameter identifier: MQIACH_SHORT_TIMER).

Specifies the short retry wait interval for a sender or server channel that is started automatically by the channel initiator. It defines the interval in seconds between attempts to establish a connection to the remote machine.

The time is approximate. From IBM MQ Version 8.0, zero means that another connection attempt is made as soon as possible.

Specify a value in the range 0 - 999 999. Values exceeding this value are treated as 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

SSLCipherSpec (MQCFST)

CipherSpec (parameter identifier: MQCACH_SSL_CIPHER_SPEC).

The length of the string is MQ_SSL_CIPHER_SPEC_LENGTH.

It is valid only for channels with a transport type (TRPTYPE) of TCP. If the TRPTYPE is not TCP, the data is ignored and no error message is issued.

The SSLCIPH values must specify the same CipherSpec on both ends of the channel.

Specify the name of the CipherSpec that you are using. Alternatively, on IBM i, and z/OS, you can specify the two-digit hexadecimal code.

The following table shows the CipherSpecs that can be used with IBM MQ TLS.

On IBM i, installation of AC3 is a prerequisite of the use of TLS.

Platform support[1]	CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[2]	Suite B
ULW z/OS	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	SHA-1	AES	128	Yes	No
ULW z/OS	TLS_RSA_WITH_AES_256_CBC_SHA [3]	TLS 1.0	SHA-1	AES	256	Yes	No
All	ECDHE_ECDSA_AES_128_CBC_SHA256	TLS 1.2	SHA-256	AES	128	Yes	No
All	ECDHE_ECDSA_AES_256_CBC_SHA384 [3]	TLS 1.2	SHA-384	AES	256	Yes	No
Multi	ECDHE_ECDSA_AES_128_GCM_SHA256 [4]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	128 bit
Multi	ECDHE_ECDSA_AES_256_GCM_SHA384[3] [4]	TLS 1.2	AEAD AES-128 GCM	AES	256	Yes	192 bit
All	ECDHE_RSA_AES_128_CBC_SHA256	TLS 1.2	SHA-256	AES	128	Yes	No
All	ECDHE_RSA_AES_256_CBC_SHA384 [3]	TLS 1.2	SHA-384	AES	256	Yes	No

Platform support[1]	CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[2]	Suite B
Multi	ECDHE_RSA_AES_128_GCM_SHA256 [4]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	No
Multi	ECDHE_RSA_AES_256_GCM_SHA384 [3] [4]	TLS 1.2	AEAD AES-128 GCM	AES	SHA384	Yes	No
[5] IBM i	ECDHE_ECDSA_RC4_128_SHA256	TLS 1.2	AEAD AES-128 GCM	AES	SHA256	Yes	No
IBM i	ECDHE_ECDSA_3DES_EDE_CBC_SHA256	TLS 1.2	AEAD AES-128 GCM	3DES	SHA256	Yes	No
IBM i	ECDHE_ECDSA_NULL_SHA256	TLS 1.2	AEAD AES-128 GCM	ECDSA	SHA256	Yes	No
IBM i	ECDHE_ECDSA_AES_256_GCM_SHA384 [3] [4]	TLS 1.2	AEAD AES-128 GCM	AES	SHA384	Yes	No
ULW z/OS	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	SHA-256	AES	128	Yes	No
ULW z/OS	TLS_RSA_WITH_AES_256_CBC_SHA256 [3]	TLS 1.2	SHA-256	AES	256	Yes	No
Multi	TLS_RSA_WITH_AES_128_GCM_SHA256 [4]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	No
Multi	TLS_RSA_WITH_AES_256_GCM_SHA384 [3] [4]	TLS 1.2	AEAD AES-128 GCM	AES	256	Yes	No

Notes:

1. If no specific platform is noted, the CipherSpec is available on all platforms. For a list of platforms covered by each platform icon, see Release and platform icons in the product documentation.
2. Specifies whether the CipherSpec is FIPS-certified on a FIPS-certified platform. See Federal Information Processing Standards (FIPS) for an explanation of FIPS.
3. This CipherSpec cannot be used to secure a connection from the IBM MQ Explorer to a queue manager unless the appropriate unrestricted policy files are applied to the JRE used by the Explorer.
4. Following a recommendation by NIST, GCM CipherSpecs now have a restriction which means that after 2^22 TLS records are sent, using the same session key, the connection will be terminated with message AMQ9288.

   To prevent this error from happening: avoid using GCM Ciphers, enable secret key reset, or start your IBM MQ queue manager with the environment variable GSK_ENFORCE_GCM_RESTRICTION=GSK_FALSE set.
   Important: The GCM restriction is active, regardless of the FIPS mode being used.
5. IBM i The CipherSpecs listed as supported on IBM i, apply to Versions 7.2 and 7.3 of IBM i.

When you request a personal certificate, you specify a key size for the public and private key pair. The key size that is used during the TLS handshake can depend on the size stored in the certificate and on the CipherSpec:

- On UNIX, Windows platforms, and z/OS, when a CipherSpec name includes _EXPORT, the maximum handshake key size is 512 bits. If either of the certificates exchanged during the TLS handshake has a key size greater than 512 bits, a temporary 512-bit key is generated for use during the handshake.
- On UNIX and Windows platforms, when a CipherSpec name includes _EXPORT1024, the handshake key size is 1024 bits.
- Otherwise the handshake key size is the size stored in the certificate.

If the SSLCIPH parameter is blank, no attempt is made to use TLS on the channel.

SSLClientAuth (MQCFIN)
Client authentication (parameter identifier: MQIACH_SSL_CLIENT_AUTH).

The value can be any of the following values:

MQSCA_REQUIRED
Client authentication required.

MQSCA_OPTIONAL
Client authentication optional.

Defines whether IBM MQ requires a certificate from the TLS client.

The TLS client is the end of the message channel that initiates the connection. The TLS Server is the end of the message channel that receives the initiation flow.

The parameter is used only for channels with SSLCIPH specified. If SSLCIPH is blank, the data is ignored and no error message is issued.

SSLPeerName (MQCFST)
Peer name (parameter identifier: MQCACH_SSL_PEER_NAME).

Note: An alternative way of restricting connections into channels by matching against the TLS Subject Distinguished Name, is to use channel authentication records. With channel authentication records, different TLS Subject Distinguished Name patterns can be applied to the same channel. If both SSLPEER on the channel and a channel authentication record are used to apply to the same channel, the inbound certificate must match both patterns in order to connect. For more information, see Channel authentication records.

Multi On Multiplatforms, the length of the string is MQ_SSL_PEER_NAME_LENGTH.

z/OS On z/OS, the length of the string is MQ_SSL_SHORT_PEER_NAME_LENGTH.

Specifies the filter to use to compare with the Distinguished Name of the certificate from the peer queue manager or client at the other end of the channel. (A Distinguished Name is the identifier of the TLS certificate.) If the Distinguished Name in the certificate received from the peer does not match the SSLPEER filter, the channel does not start.

This parameter is optional; if it is not specified, the Distinguished Name of the peer is not checked when the channel is started. (The Distinguished Name from the certificate is still written into the SSLPEER definition held in memory, and passed to the security exit). If SSLCIPH is blank, the data is ignored and no error message is issued.

This parameter is valid for all channel types.

The SSLPEER value is specified in the standard form used to specify a Distinguished Name. For example: `SSLPEER('SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN="H1_C_FR1",O=IBM,C=GB')`

You can use a semi-colon as a separator instead of a comma.

The possible attribute types supported are:

Attribute	Description
SERIALNUMBER	Certificate serial number
MAIL	Email address
E	Email address (Deprecated in preference to MAIL)
UID or USERID	User identifier
CN	Common Name
T	Title
OU	Organizational Unit name
DC	Domain component
O	Organization name
STREET	Street / First line of address
L	Locality name
ST (or SP or S)	State or Province name
PC	Postal code / zip code
C	Country
UNSTRUCTUREDNAME	Host name
UNSTRUCTUREDADDRESS	IP address
DNQ	Distinguished name qualifier

IBM MQ only accepts uppercase letters for the attribute types.

If any of the unsupported attribute types are specified in the SSLPEER string, an error is output either when the attribute is defined or at run time (depending on which platform you are running on), and the string is deemed not to have matched the Distinguished Name of the flowed certificate.

If the Distinguished Name of the flowed certificate contains multiple OU (organizational unit) attributes, and SSLPEER specifies these attributes to be compared, they must be defined in descending hierarchical order. For example, if the Distinguished Name of the flowed certificate contains the OUs `OU=Large Unit,OU=Medium Unit,OU=Small Unit`, specifying the following SSLPEER values work:

```
('OU=Large Unit,OU=Medium Unit') ('OU=*,OU=Medium Unit,OU=Small Unit') ('OU=*,OU=Medium Unit')
```

but specifying the following SSLPEER values fail:

```
('OU=Medium Unit,OU=Small Unit') ('OU=Large Unit,OU=Small Unit') ('OU=Medium Unit')
```

Any or all the attribute values can be generic, either an asterisk (*) on its own, or a stem with initiating or trailing asterisks. This value allows the SSLPEER to match any Distinguished Name value, or any value starting with the stem for that attribute.

If an asterisk is specified at the beginning or end of any attribute value in the Distinguished Name on the certificate, you can specify * to check for an exact match in SSLPEER. For example, if you have an attribute of CN=Test* in the Distinguished Name of the certificate, you can use the following command:

```
SSLPEER('CN=Test\*')
```

TpName (MQCFST)

Transaction program name (parameter identifier: MQCACH_TP_NAME).

This name is the LU 6.2 transaction program name.

The maximum length of the string is MQ_TP_NAME_LENGTH.

- On IBM i, HP Integrity NonStop Server, UNIX, and Windows platforms, this parameter can be set only to blanks. The actual name is taken instead from the CPI-C Communications Side Object or (on Windows) from the CPI-C symbolic destination name properties.

This parameter is valid only for channels with a *TransportType* of MQXPT_LU62. It is not valid for receiver channels.

V 9.0.0 **TPRoot (MQCFST)**

Topic root for an AMQP channel. (parameter identifier: MQCACH_TOPIC_ROOT).

The default value for TPRoot is SYSTEM.BASE.TOPIC. With this value, the topic string an AMQP client uses to publish or subscribe has no prefix, and the client can exchange messages with other MQ pub/sub applications. To have AMQP clients publish and subscribe under a topic prefix, first create an MQ topic object with a topic string set to the prefix you want, then set TPRoot to the name of the MQ topic object you created.

This parameter is valid only for AMQP channels.

TransportType (MQCFIN)

Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

No check is made that the correct transport type has been specified if the channel is initiated from the other end. The value can be any of the following values:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

MQXPT_NETBIOS
NetBIOS.

This value is supported in Windows. It also applies to z/OS for defining client-connection channels that connect to servers on the platforms supporting NetBIOS.

MQXPT_SPX
SPX.

This value is supported in Windows. It also applies to z/OS for defining client-connection channels that connect to servers on the platforms supporting SPX.

V 9.0.0 **UseCltId (MQCFIN)**

Determines how authorization checks are done for AMQP channels. (parameter identifier: MQIACH_USE_CLIENT_ID).

The value can be any of the following values:

MQUCI_NO
The MCA user ID should be used for authorization checks.

MQUCI_YES
The client ID should be used for authorization checks.

This parameter is valid only for AMQP channels.

UseDLQ (MQCFIN)

Determines whether the dead-letter queue is used when messages cannot be delivered by channels. (parameter identifier: MQIA_USE_DEAD_LETTER_Q).

The value can be any of the following values:

MQUSEDLQ_NO
Messages that cannot be delivered by a channel are treated as a failure. The channel either discards the message, or the channel ends, in accordance with the NonPersistentMsgSpeed setting.

MQUSEDLQ_YES
When the DEADQ queue manager attribute provides the name of a dead-letter queue, then it is used, else the behavior is as for MQUSEDLQ_NO.

UserIdentifier (MQCFST)
Task user identifier (parameter identifier: MQCACH_USER_ID).

This parameter is used by the message channel agent when attempting to initiate a secure SNA session with a remote message channel agent. On IBM i and UNIX, it is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, MQCHT_CLNTCONN, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR. On z/OS, it is valid only for a *ChannelType* value of MQCHT_CLNTCONN.

The maximum length of the string is MQ_USER_ID_LENGTH. However, only the first 10 characters are used.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

A transmission queue name is required (either previously defined or specified here) if *ChannelType* is MQCHT_SENDER or MQCHT_SERVER. It is not valid for other channel types.

Error codes (Change, Copy, and Create Channel)

This command might return the following error codes in the response format header, in addition to those codes listed in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_BATCH_INT_ERROR
Batch interval not valid.

MQRCCF_BATCH_INT_WRONG_TYPE
Batch interval parameter not allowed for this channel type.

MQRCCF_BATCH_SIZE_ERROR
Batch size not valid.

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_CLUSTER_NAME_CONFLICT
Cluster name conflict.

MQRCCF_DISC_INT_ERROR
Disconnection interval not valid.

MQRCCF_DISC_INT_WRONG_TYPE
Disconnection interval not allowed for this channel type.

MQRCCF_HB_INTERVAL_ERROR
Heartbeat interval not valid.

MQRCCF_HB_INTERVAL_WRONG_TYPE
Heartbeat interval parameter not allowed for this channel type.

MQRCCF_LONG_RETRY_ERROR
Long retry count not valid.

MQRCCF_LONG_RETRY_WRONG_TYPE
Long retry parameter not allowed for this channel type.

MQRCCF_LONG_TIMER_ERROR
Long timer not valid.

MQRCCF_LONG_TIMER_WRONG_TYPE
Long timer parameter not allowed for this channel type.

MQRCCF_MAX_INSTANCES_ERROR
Maximum instances value not valid.

MQRCCF_MAX_INSTS_PER_CLNT_ERR
Maximum instances per client value not valid.

MQRCCF_MAX_MSG_LENGTH_ERROR
Maximum message length not valid.

MQRCCF_MCA_NAME_ERROR
Message channel agent name error.

MQRCCF_MCA_NAME_WRONG_TYPE
Message channel agent name not allowed for this channel type.

MQRCCF_MCA_TYPE_ERROR
Message channel agent type not valid.

MQRCCF_MISSING_CONN_NAME
Connection name parameter required but missing.

MQRCCF_MR_COUNT_ERROR
Message retry count not valid.

MQRCCF_MR_COUNT_WRONG_TYPE
Message-retry count parameter not allowed for this channel type.

MQRCCF_MR_EXIT_NAME_ERROR
Channel message-retry exit name error.

MQRCCF_MR_EXIT_NAME_WRONG_TYPE
Message-retry exit parameter not allowed for this channel type.

MQRCCF_MR_INTERVAL_ERROR
Message retry interval not valid.

MQRCCF_MR_INTERVAL_WRONG_TYPE
Message-retry interval parameter not allowed for this channel type.

MQRCCF_MSG_EXIT_NAME_ERROR
Channel message exit name error.

MQRCCF_NET_PRIORITY_ERROR
Network priority value error.

MQRCCF_NET_PRIORITY_WRONG_TYPE
Network priority attribute not allowed for this channel type.

MQRCCF_NPM_SPEED_ERROR
Nonpersistent message speed not valid.

MQRCCF_NPM_SPEED_WRONG_TYPE
Nonpersistent message speed parameter not allowed for this channel type.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_PUT_AUTH_ERROR
Put authority value not valid.

MQRCCF_PUT_AUTH_WRONG_TYPE
Put authority parameter not allowed for this channel type.

MQRCCF_RCV_EXIT_NAME_ERROR
Channel receive exit name error.

MQRCCF_SEC_EXIT_NAME_ERROR
Channel security exit name error.

MQRCCF_SEND_EXIT_NAME_ERROR
Channel send exit name error.

MQRCCF_SEQ_NUMBER_WRAP_ERROR
Sequence wrap number not valid.

MQRCCF_SHARING_CONVS_ERROR
Value given for Sharing Conversations not valid.

MQRCCF_SHARING_CONVS_TYPE
Sharing Conversations parameter not valid for this channel type.

MQRCCF_SHORT_RETRY_ERROR
Short retry count not valid.

MQRCCF_SHORT_RETRY_WRONG_TYPE
Short retry parameter not allowed for this channel type.

MQRCCF_SHORT_TIMER_ERROR
Short timer value not valid.

MQRCCF_SHORT_TIMER_WRONG_TYPE
Short timer parameter not allowed for this channel type.

MQRCCF_SSL_CIPHER_SPEC_ERROR
TLS CipherSpec not valid.

MQRCCF_SSL_CLIENT_AUTH_ERROR
TLS client authentication not valid.

MQRCCF_SSL_PEER_NAME_ERROR
TLS peer name not valid.

MQRCCF_WRONG_CHANNEL_TYPE
Parameter not allowed for this channel type.

MQRCCF_XMIT_PROTOCOL_TYPE_ERR
Transmission protocol type not valid.

MQRCCF_XMIT_Q_NAME_ERROR
Transmission queue name error.

MQRCCF_XMIT_Q_NAME_WRONG_TYPE
Transmission queue name not allowed for this channel type.

Change, Copy, and Create Channel (MQTT): AIX Linux Windows

The Change Channel command changes existing Telemetry channel definitions. The Copy and Create Channel commands create new Telemetry channel definitions - the Copy command uses attribute values of an existing channel definition.

The Change Channel (MQCMD_CHANGE_CHANNEL) command changes the specified attributes in a channel definition. For any optional parameters that are omitted, the value does not change.

The Copy Channel (MQCMD_COPY_CHANNEL) command creates new channel definition using, for attributes not specified in the command, the attribute values of an existing channel definition.

The Create Channel (MQCMD_CREATE_CHANNEL) command creates an IBM MQ channel definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager. If a system default channel exists for the type of channel being created, the default values are taken from there.

Required parameters (Change, Create Channel)

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Specifies the name of the channel definition to be changed, or created

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

Specifies the type of the channel being changed, copied, or created. The value can be any of the following values:

MQCHT_MQTT
Telemetry.

TrpType (MQCFIN)
Transmission protocol type of the channel (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE). This parameter is required for a create command in telemetry.

No check is made that the correct transport type has been specified if the channel is initiated from the other end. The value is:

MQXPT_TCP
TCP.

Port (MQCFIN)
The port number to use if *TrpType* is set to MQXPT_TCP. This parameter is required for a create command in telemetry, if *TrpType* is set to MQXPT_TCP.

The value is in the range 1 - 65335.

Required parameters (Copy Channel)

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

Specifies the type of the channel being changed, copied, or created. The value can be any of the following values:

MQCHT_MQTT
Telemetry.

Optional parameters (Change, Copy, and Create Channel)

Backlog (MQCFIN)
The number of concurrent connection requests that the telemetry channel supports at any one time (parameter identifier: MQIACH_BACKLOG).

The value is in the range 0 - 999999999.

JAASConfig (MQCFST)
The file path of the JAAS configuration (parameter identifier: MQCACH_JAAS_CONFIG).

The maximum length of this value is MQ_JAAS_CONFIG_LENGTH.

Only one of JAASCONFIG, MCAUSER, and USECLIENTID can be specified for a telemetry channel; if none is specified, no authentication is performed. If JAASConfig is specified, the client flows a user name and password. In all other cases, the flowed user name is ignored.

LocalAddress (MQCFST)
Local communications address for the channel (parameter identifier: MQCACH_LOCAL_ADDRESS).

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

The value that you specify depends on the transport type *(TransportType)* to be used:

TCP/IP
The value is the optional IP address and optional port or port range to be used for outbound TCP/IP communications. The format for this information is as follows:

```
[ip-addr][(low-port[,high-port])]
```

where `ip-addr` is specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric form, and `low-port` and `high-port` are port numbers enclosed in parentheses. All are optional.

All Others
The value is ignored; no error is diagnosed.

Use this parameter if you want a channel to use a particular IP address, port, or port range for outbound communications. This parameter is useful when a machine is connected to multiple networks with different IP addresses.

Examples of use

Value	Meaning
9.20.4.98	Channel binds to this address locally
9.20.4.98 (1000)	Channel binds to this address and port 1000 locally
9.20.4.98 (1000,2000)	Channel binds to this address and uses a port in the range 1000 - 2000 locally
(1000)	Channel binds to port 1000 locally
(1000,2000)	Channel binds to a port in the range 1000 - 2000 locally

Note:

- Do not confuse this parameter with *ConnectionName*. The *LocalAddress* parameter specifies the characteristics of the local communications; the *ConnectionName* parameter specifies how to reach a remote queue manager.

MqiachProtocol (MQCFIL)
Client protocols supported by the MQTT channel (parameter identifier: MQIACH_PROTOCOL).

The value can be one or more of the following values:

MQPROTO_MQTTV311
The channel accepts connections from clients using the protocol defined by the MQTT Version 3.1.1 Oasis standard. The functionality provided by this protocol is almost identical to that provided by the pre-existing MQTTV3 protocol.

MQPROTO_MQTTV3
The channel accepts connections from clients using the MQTT V3.1 Protocol Specification from mqtt.org.

MQPROTO_HTTP
The channel accepts HTTP requests for pages, or WebSockets connections to MQ Telemetry.

If you specify no client protocols, the channel accepts connections from clients using any of the supported protocols.

If you are using IBM MQ Version 8.0.0, Fix Pack 3 or later, and your configuration includes an MQTT channel that was last modified in an earlier version of the product, you must explicitly change the protocol setting to prompt the channel to use the `MQTTV311` option. This is so even if the channel does not specify any client protocols, because the specific protocols to use with the channel are stored at the time the channel is configured, and previous versions of the product have no awareness of the `MQTTV311` option. To prompt a channel in this state to use the `MQTTV311` option, explicitly add the option then save your changes. The channel definition is now aware of the option. If you subsequently change the settings again, and specify no client protocols, the `MQTTV311` option is still included in the stored list of supported protocols.

SSLCipherSuite (MQCFST)
CipherSuite (parameter identifier: MQCACH_SSL_CIPHER_SUITE).

The length of the string is MQ_SSL_CIPHER_SUITE_LENGTH.

SSL CIPHER SUITE character channel parameter type.

SSLClientAuth (MQCFIN)
Client authentication (parameter identifier: MQIACH_SSL_CLIENT_AUTH).

The value can be any of the following values:

MQSCA_REQUIRED
Client authentication required

MQSCA_OPTIONAL
Client authentication is optional.

MQSCA_NEVER_REQUIRED
Client authentication is never required, and must not be provided.

Defines whether IBM MQ requires a certificate from the TLS client.

The TLS client is the end of the message channel that initiates the connection. The TLS Server is the end of the message channel that receives the initiation flow.

The parameter is used only for channels with SSLCIPH specified. If SSLCIPH is blank, the data is ignored and no error message is issued.

SSLKeyFile (MQCFST)
The store for digital certificates and their associated private keys (parameter identifier: MQCA_SSL_KEY_REPOSITORY).

If you do not specify a key file, TLS is not used.

The maximum length of this parameter is MQ_SSL_KEY_REPOSITORY_LENGTH.

SSLPassPhrase (MQCFST)
The password for the key repository (parameter identifier: MQCACH_SSL_KEY_PASSPHRASE).

If no pass phrase is entered, then unencrypted connections must be used.

The maximum length of this parameter is MQ_SSL_KEY_PASSPHRASE_LENGTH.

TransportType (MQCFIN)
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

No check is made that the correct transport type has been specified if the channel is initiated from the other end. The value can be any of the following values:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

MQXPT_NETBIOS
NetBIOS.

This value is supported in Windows.

MQXPT_SPX
SPX.

This value is supported in Windows.

This parameter is required for a create command in telemetry; see TransportType for more information.

UseClientIdentifier (MQCFIN)
Determines whether to use the client ID of a new connection as the user ID for that connection (parameter identifier: MQIACH_USE_CLIENT_ID).

The value is either:

MQUCI_YES
Yes.

MQUCI_NO
No.

Only one of JAASCONFIG, MCAUSER, and USECLIENTID can be specified for a telemetry channel; if none is specified, no authentication is performed. If USECLIENTID is specified, the flowed user name of the client is ignored.

Error codes (Change, Copy, and Create Channel)

This command might return the following error codes in the response format header, in addition to those codes listed in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_BATCH_INT_ERROR
Batch interval not valid.

MQRCCF_BATCH_INT_WRONG_TYPE
Batch interval parameter not allowed for this channel type.

MQRCCF_BATCH_SIZE_ERROR
Batch size not valid.

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_CLUSTER_NAME_CONFLICT
Cluster name conflict.

MQRCCF_DISC_INT_ERROR
Disconnection interval not valid.

MQRCCF_DISC_INT_WRONG_TYPE
Disconnection interval not allowed for this channel type.

MQRCCF_HB_INTERVAL_ERROR
Heartbeat interval not valid.

MQRCCF_HB_INTERVAL_WRONG_TYPE
Heartbeat interval parameter not allowed for this channel type.

MQRCCF_LONG_RETRY_ERROR
Long retry count not valid.

MQRCCF_LONG_RETRY_WRONG_TYPE
Long retry parameter not allowed for this channel type.

MQRCCF_LONG_TIMER_ERROR
Long timer not valid.

MQRCCF_LONG_TIMER_WRONG_TYPE
Long timer parameter not allowed for this channel type.

MQRCCF_MAX_INSTANCES_ERROR
Maximum instances value not valid.

MQRCCF_MAX_INSTS_PER_CLNT_ERR
Maximum instances per client value not valid.

MQRCCF_MAX_MSG_LENGTH_ERROR
Maximum message length not valid.

MQRCCF_MCA_NAME_ERROR
Message channel agent name error.

MQRCCF_MCA_NAME_WRONG_TYPE
Message channel agent name not allowed for this channel type.

MQRCCF_MCA_TYPE_ERROR
Message channel agent type not valid.

MQRCCF_MISSING_CONN_NAME
Connection name parameter required but missing.

MQRCCF_MR_COUNT_ERROR
Message retry count not valid.

MQRCCF_MR_COUNT_WRONG_TYPE
Message-retry count parameter not allowed for this channel type.

MQRCCF_MR_EXIT_NAME_ERROR
Channel message-retry exit name error.

MQRCCF_MR_EXIT_NAME_WRONG_TYPE
Message-retry exit parameter not allowed for this channel type.

MQRCCF_MR_INTERVAL_ERROR
Message retry interval not valid.

MQRCCF_MR_INTERVAL_WRONG_TYPE
Message-retry interval parameter not allowed for this channel type.

MQRCCF_MSG_EXIT_NAME_ERROR
Channel message exit name error.

MQRCCF_NET_PRIORITY_ERROR
Network priority value error.

MQRCCF_NET_PRIORITY_WRONG_TYPE
Network priority attribute not allowed for this channel type.

MQRCCF_NPM_SPEED_ERROR
Nonpersistent message speed not valid.

MQRCCF_NPM_SPEED_WRONG_TYPE
Nonpersistent message speed parameter not allowed for this channel type.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_PUT_AUTH_ERROR
Put authority value not valid.

MQRCCF_PUT_AUTH_WRONG_TYPE
Put authority parameter not allowed for this channel type.

MQRCCF_RCV_EXIT_NAME_ERROR
Channel receive exit name error.

MQRCCF_SEC_EXIT_NAME_ERROR
Channel security exit name error.

MQRCCF_SEND_EXIT_NAME_ERROR
Channel send exit name error.

MQRCCF_SEQ_NUMBER_WRAP_ERROR
Sequence wrap number not valid.

MQRCCF_SHARING_CONVS_ERROR
Value given for Sharing Conversations not valid.

MQRCCF_SHARING_CONVS_TYPE
Sharing Conversations parameter not valid for this channel type.

MQRCCF_SHORT_RETRY_ERROR
Short retry count not valid.

MQRCCF_SHORT_RETRY_WRONG_TYPE
Short retry parameter not allowed for this channel type.

MQRCCF_SHORT_TIMER_ERROR
Short timer value not valid.

MQRCCF_SHORT_TIMER_WRONG_TYPE
Short timer parameter not allowed for this channel type.

MQRCCF_SSL_CIPHER_SPEC_ERROR
TLS CipherSpec not valid.

MQRCCF_SSL_CLIENT_AUTH_ERROR
TLS client authentication not valid.

MQRCCF_SSL_PEER_NAME_ERROR
TLS peer name not valid.

MQRCCF_WRONG_CHANNEL_TYPE
Parameter not allowed for this channel type.

MQRCCF_XMIT_PROTOCOL_TYPE_ERR
Transmission protocol type not valid.

MQRCCF_XMIT_Q_NAME_ERROR
Transmission queue name error.

MQRCCF_XMIT_Q_NAME_WRONG_TYPE
Transmission queue name not allowed for this channel type.

Change, Copy, and Create Channel Listener on Multiplatforms: 

The Change Channel Listener command changes existing channel listener definitions. The Copy and Create Channel Listener commands create new channel listener definitions - the Copy command uses attribute values of an existing channel listener definition.

The Change Channel Listener (MQCMD_CHANGE_LISTENER) command changes the specified attributes of an existing IBM MQ listener definition. For any optional parameters that are omitted, the value does not change.

The Copy Channel Listener (MQCMD_ COPY_LISTENER) command creates an IBM MQ listener definition, using, for attributes not specified in the command, the attribute values of an existing listener definition.

The Create Channel Listener (MQCMD_CREATE_LISTENER) command creates an IBM MQ listener definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameters (Change and Create Channel Listener)

`ListenerName (MQCFST)`
The name of the listener definition to be changed or created (parameter identifier: MQCACH_LISTENER_NAME).

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

`TransportType (MQCFIN)`
Transmission protocol (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value can be:

MQXPT_TCP
TCP.

MQXPT_LU62
LU 6.2. This value is valid only on Windows.

MQXPT_NETBIOS
NetBIOS. This value is valid only on Windows.

MQXPT_SPX
SPX. This value is valid only on Windows.

Required parameters (Copy Channel Listener)

`FromListenerName (MQCFST)`
The name of the listener definition to be copied from (parameter identifier: MQCACF_FROM_LISTENER_NAME).

This parameter specifies the name of the existing listener definition that contains values for the attributes not specified in this command.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

ToListenerName (MQCFST)
To listener name (parameter identifier: MQCACF_TO_LISTENER_NAME).

This parameter specifies the name of the new listener definition. If a listener definition with this name exists, *Replace* must be specified as MQRP_YES.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

Optional parameters (Change, Copy, and Create Channel Listener)

Adapter (MQCFIN)
Adapter number (parameter identifier: MQIACH_ADAPTER).

The adapter number on which NetBIOS listens. This parameter is valid only on Windows.

Backlog (MQCFIN)
Backlog (parameter identifier: MQIACH_BACKLOG).

The number of concurrent connection requests that the listener supports.

Commands (MQCFIN)
Adapter number (parameter identifier: MQIACH_COMMAND_COUNT).

The number of commands that the listener can use. This parameter is valid only on Windows.

IPAddress (MQCFST)
IP address (parameter identifier: MQCACH_IP_ADDRESS).

IP address for the listener specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric host name form. If you do not specify a value for this parameter, the listener listens on all configured IPv4 and IPv6 stacks.

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH

ListenerDesc (MQCFST)
Description of listener definition (parameter identifier: MQCACH_LISTENER_DESC).

This parameter is a plain-text comment that provides descriptive information about the listener definition. It must contain only displayable characters.

If characters are used that are not in the coded character set identifier (CCSID) for the queue manager on which the command is executing, they might be translated incorrectly.

The maximum length of the string is MQ_LISTENER_DESC_LENGTH.

LocalName (MQCFST)
NetBIOS local name (parameter identifier: MQCACH_LOCAL_NAME).

The NetBIOS local name that the listener uses. This parameter is valid only on Windows.

The maximum length of the string is MQ_CONN_NAME_LENGTH

NetbiosNames (MQCFIN)
NetBIOS names (parameter identifier: MQIACH_NAME_COUNT).

The number of names that the listener supports. This parameter is valid only on Windows.

Port (MQCFIN)
Port number (parameter identifier: MQIACH_PORT).

The port number for TCP/IP. This parameter is valid only if the value of *TransportType* is MQXPT_TCP.

Replace (MQCFIN)
Replace attributes (parameter identifier: MQIACF_REPLACE).

If a namelist definition with the same name as *ToListenerName* exists, this definition specifies whether it is to be replaced. The value can be:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

Sessions (MQCFIN)
NetBIOS sessions (parameter identifier: MQIACH_SESSION_COUNT).

The number of sessions that the listener can use. This parameter is valid only on Windows.

Socket (MQCFIN)
SPX socket number (parameter identifier: MQIACH_SOCKET).

The SPX socket on which to listen. This parameter is valid only if the value of *TransportType* is MQXPT_SPX.

StartMode (MQCFIN)
Service mode (parameter identifier: MQIACH_LISTENER_CONTROL).

Specifies how the listener is to be started and stopped. The value can be any of the following values:

MQSVC_CONTROL_MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by user command. This value is the default value.

MQSVC_CONTROL_Q_MGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

MQSVC_CONTROL_Q_MGR_START
The listener is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

TPName (MQCFST)
Transaction program name (parameter identifier: MQCACH_TP_NAME).

The LU 6.2 transaction program name. This parameter is valid only on Windows.

The maximum length of the string is MQ_TP_NAME_LENGTH

Change, Copy, and Create Communication Information Object on Multiplatforms: 

The Change Communication Information Object command changes existing communication information object definitions. The Copy and Create Communication Information Object commands create new communication information object definitions - the Copy command uses attribute values of an existing communication information object definition.

The Change communication information (MQCMD_CHANGE_COMM_INFO) command changes the specified attributes of an existing IBM MQ communication information object definition. For any optional parameters that are omitted, the value does not change.

The Copy communication information (MQCMD_COPY_COMM_INFO) command creates an IBM MQ communication information object definition, using, for attributes not specified in the command, the attribute values of an existing communication information definition.

The Create communication information (MQCMD_CREATE_COMM_INFO) command creates an IBM MQ communication information object definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameter (Change communication information)

ComminfoName (MQCFST)
The name of the communication information definition to be changed (parameter identifier: MQCA_COMM_INFO_NAME).

The maximum length of the string is MQ_COMM_INFO_NAME_LENGTH.

Required parameters (Copy communication information)

FromComminfoName (MQCFST)
The name of the communication information object definition to be copied from (parameter identifier: MQCACF_FROM_COMM_INFO_NAME).

The maximum length of the string is MQ_COMM_INFO_NAME_LENGTH.

ToComminfoName (MQCFST)
The name of the communication information definition to copy to (parameter identifier: MQCACF_TO_COMM_INFO_NAME).

The maximum length of the string is MQ_COMM_INFO_NAME_LENGTH.

Required parameters (Create communication information)

ComminfoName (MQCFST)
The name of the communication information definition to be created (parameter identifier: MQCA_COMM_INFO_NAME).

The maximum length of the string is MQ_COMM_INFO_NAME_LENGTH.

Optional parameters (Change, Copy, and Create communication information)

Bridge (MQCFIN)
Controls whether publications from applications not using Multicast are bridged to applications using multicast (parameter identifier: MQIA_MCAST_BRIDGE).

Bridging does not apply to topics that are marked as **MCAST(ONLY)**. As these topics can only have multicast traffic, it is not applicable to bridge to the non-multicast publish/subscribe domain.

MQMCB_DISABLED
Publications from applications not using multicast are not bridged to applications that do use Multicast. This is the default for IBM i.

MQMCB_ENABLED
Publications from applications not using multicast are bridged to applications that do use Multicast. This is the default for platforms other than IBM i. This value is not valid on IBM i.

CCSID (MQCFIN)
The coded character set identifier that messages are transmitted on (parameter identifier: MQIA_CODED_CHAR_SET_ID).

Specify a value in the range 1 to 65535.

The CCSID must specify a value that is defined for use on your platform, and use a character set that is appropriate to the platform. If you use this parameter to change the CCSID, applications that are running when the change is applied continue to use the original CCSID. Because of this, you must stop and restart all running applications before you continue.

This includes the command server and channel programs. To do this, stop and restart the queue manager after making the change. The default value is ASPUB which means that the coded character set is taken from the one that is supplied in the published message.

CommEvent (MQCFIN)
Controls whether event messages are generated for multicast handles that are created using this COMMINFO object (parameter identifier: MQIA_COMM_EVENT).

Events are only generated if monitoring is also enabled using the **MonitorInterval** parameter.

MQEVR_DISABLED
Publications from applications not using multicast are not bridged to applications that do use multicast. This is the default value.

MQEVR_ENABLED
Publications from applications not using multicast are bridged to applications that do use multicast.

MQEVR_EXCEPTION
Event messages are written if the message reliability is below the reliability threshold The reliability threshold is set to 90 by default.

Description (MQCFST)
Plain-text comment that provides descriptive information about the communication information object (parameter identifier: MQCA_COMM_INFO_DESC).

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

The maximum length is MQ_COMM_INFO_DESC_LENGTH.

Encoding (MQCFIN)
The encoding that the messages are transmitted in (parameter identifier: MQIACF_ENCODING).

MQENC_AS_PUBLISHED
The encoding of the message is taken from the one that is supplied in the published message. This is the default value.

MQENC_NORMAL

MQENC_REVERSED

MQENC_S390

MQENC_TNS

GrpAddress (MQCFST)
The group IP address or DNS name (parameter identifier: MQCACH_GROUP_ADDRESS).

It is the administrator's responsibility to manage the group addresses. It is possible for all multicast clients to use the same group address for every topic; only the messages that match outstanding subscriptions on the client are delivered. Using the same group address can be inefficient because every client must examine and process every multicast packet in the network. It is more efficient to allocate different IP group addresses to different topics or sets of topics, but this requires careful management, especially if other non-MQ multicast applications are in use on the network. The default value is `239.0.0.0`.

The maximum length is MQ_GROUP_ADDRESS_LENGTH.

MonitorInterval (MQCFIN)
How frequently monitoring information is updated and event messages are generated (parameter identifier: MQIA_MONITOR_INTERVAL).

The value is specified as a number of seconds in the range 0 to 999 999. A value of `0` indicates that no monitoring is required.

If a non-zero value is specified, monitoring is enabled. Monitoring information is updated and event messages (if enabled using *`CommEvent`*, are generated about the status of the multicast handles created using this communication information object.

MsgHistory (MQCFIN)
This value is the amount of message history in kilobytes that is kept by the system to handle retransmissions in the case of NACKs (parameter identifier: MQIACH_MSG_HISTORY).

The value is in the range 0 to 999 999 999. A value of `0` gives the least level of reliability. The default value is `100`.

MulticastHeartbeat (MQCFIN)
The heartbeat interval is measured in milliseconds, and specifies the frequency at which the transmitter notifies any receivers that there is no further data available (parameter identifier: MQIACH_MC_HB_INTERVAL).

The value is in the range 0 to 999 999. The default value is `2000` milliseconds.

MulticastPropControl (MQCFIN)
The multicast properties control how many of the MQMD properties and user properties flow with the message (parameter identifier: MQIACH_MULTICAST_PROPERTIES).

MQMCP_ALL
All user properties and all the fields of the MQMD are transported. This is the default value.

MQMCP_REPLY
Only user properties, and MQMD fields that deal with replying to the messages, are transmitted. These properties are:

- MsgType
- MessageId
- CorrelId
- ReplyToQ
- ReplyToQmgr

MQMCP_USER
Only the user properties are transmitted.

MQMCP_NONE
No user properties or MQMD fields are transmitted.

MQMCP_COMPAT
Properties are transmitted in a format compatible with previous MQ multicast clients.

NewSubHistory (MQCFIN)
The new subscriber history controls whether a subscriber joining a publication stream receives as

much data as is currently available, or receives only publications made from the time of the subscription (parameter identifier: MQIACH_NEW_SUBSCRIBER_HISTORY).

MQNSH_NONE
A value of NONE causes the transmitter to transmit only publication made from the time of the subscription. This is the default value.

MQNSH_ALL
A value of ALL causes the transmitter to retransmit as much history of the topic as is known. In some circumstances, this can give a similar behavior to retained publications.

Using the value of MQNSH_ALL might have a detrimental effect on performance if there is a large topic history because all the topic history is retransmitted.

PortNumber (MQCFIN)
The port number to transmit on (parameter identifier: MQIACH_PORT).

The default port number is 1414.

Type (MQCFIN)
The type of the communications information object (parameter identifier: MQIA_COMM_INFO_TYPE).

The only type supported is MQCIT_MULTICAST.

Change, Copy, and Create Namelist:

The Change Namelist command changes existing namelist definitions. The Copy and Create Namelist commands create new namelist definitions - the Copy command uses attribute values of an existing namelist definition.

The Change Namelist (MQCMD_CHANGE_NAMELIST) command changes the specified attributes of an existing IBM MQ namelist definition. For any optional parameters that are omitted, the value does not change.

The Copy Namelist (MQCMD_COPY_NAMELIST) command creates an IBM MQ namelist definition, using, for attributes not specified in the command, the attribute values of an existing namelist definition.

The Create Namelist (MQCMD_CREATE_NAMELIST) command creates an IBM MQ namelist definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameter (Change and Create Namelist)

NamelistName (MQCFST)
The name of the namelist definition to be changed (parameter identifier: MQCA_NAMELIST_NAME).

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

Required parameters (Copy Namelist)

FromNamelistName (MQCFST)
The name of the namelist definition to be copied from (parameter identifier: MQCACF_FROM_NAMELIST_NAME).

This parameter specifies the name of the existing namelist definition that contains values for the attributes not specified in this command.

z/OS On z/OS, the queue manager searches for an object with the name you specify and a disposition of MQQSGD_Q_MGR or MQQSGD_COPY to copy from. This parameter is ignored if a

value of MQQSGD_COPY is specified for *QSGDisposition*. In this case, an object with the name specified by *ToNamelistName* and the disposition MQQSGD_GROUP is searched for to copy from.

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

ToNamelistName (MQCFST)
To namelist name (parameter identifier: MQCACF_TO_NAMELIST_NAME).

This parameter specifies the name of the new namelist definition. If a namelist definition with this name exists, *Replace* must be specified as MQRP_YES.

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

Optional parameters (Change, Copy, and Create Namelist) z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

NamelistDesc (MQCFST)
Description of namelist definition (parameter identifier: MQCA_NAMELIST_DESC).

This parameter is a plain-text comment that provides descriptive information about the namelist definition. It must contain only displayable characters.

If characters are used that are not in the coded character set identifier (CCSID) for the queue manager on which the command is executing, they might be translated incorrectly.

The maximum length of the string is MQ_NAMELIST_DESC_LENGTH.

z/OS

NamelistType (MQCFIN)
Type of names in the namelist (parameter identifier: MQIA_NAMELIST_TYPE). This parameter applies to z/OS only.

Specifies the type of names in the namelist. The value can be any of the following values:

MQNT_NONE
The names are of no particular type.

MQNT_Q
A namelist that holds a list of queue names.

MQNT_CLUSTER
A namelist that is associated with clustering, containing a list of the cluster names.

MQNT_AUTH_INFO
The namelist is associated with TLS, and contains a list of authentication information object names.

Names (MQCFSL)

The names to be placed in the namelist (parameter identifier: MQCA_NAMES).

The number of names in the list is given by the *Count* field in the MQCFSL structure. The length of each name is given by the *StringLength* field in that structure. The maximum length of a name is MQ_OBJECT_NAME_LENGTH.

z/OS

QSGDisposition (MQCFIN)

Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be any of the following values:

QSGDisposition	Change	Copy, Create
MQQSGD_COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.	The object is defined on the page set of the queue manager that executes the command using the MQQSGD_GROUP object of the same name as the *ToNameListName* object (for Copy) or *NameListName* object (for Create).
MQQSGD_GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command. If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group so that they refresh local copies on page set zero: `DEFINE NAMELIST(name)` `REPLACE QSGDISP(COPY)` The Change for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.	The object definition resides in the shared repository. This is allowed only if the queue manager is in a queue-sharing group. If the definition is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group so that they make or refresh local copies on page set zero: `DEFINE NAMELIST(name)` `REPLACE QSGDISP(COPY)` The Copy or Create for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.
MQQSGD_PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with MQQSGD_Q_MGR or MQQSGD_COPY. Any object residing in the shared repository is unaffected.	Not permitted.
MQQSGD_Q_MGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command. This value is the default value.	The object is defined on the page set of the queue manager that executes the command. This value is the default value.

Replace (MQCFIN)
Replace attributes (parameter identifier: MQIACF_REPLACE).

If a namelist definition with the same name as *ToNamelistName* exists, this definition specifies whether it is to be replaced. The value can be:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

Change, Copy, and Create Process:

The Change Process command changes existing process definitions. The Copy and Create Process commands create new process definitions - the Copy command uses attribute values of an existing process definition.

The Change Process (MQCMD_CHANGE_PROCESS) command changes the specified attributes of an existing IBM MQ process definition. For any optional parameters that are omitted, the value does not change.

The Copy Process (MQCMD_COPY_PROCESS) command creates an IBM MQ process definition, using, for attributes not specified in the command, the attribute values of an existing process definition.

The Create Process (MQCMD_CREATE_PROCESS) command creates an IBM MQ process definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameters (Change and Create Process)

ProcessName (MQCFST)
The name of the process definition to be changed or created (parameter identifier: MQCA_PROCESS_NAME).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

Required parameters (Copy Process)

FromProcessName (MQCFST)
The name of the process definition to be copied from (parameter identifier: MQCACF_FROM_PROCESS_NAME).

Specifies the name of the existing process definition that contains values for the attributes not specified in this command.

z/OS On z/OS, the queue manager searches for an object with the name you specify and a disposition of MQQSGD_Q_MGR or MQQSGD_COPY to copy from. This parameter is ignored if a value of MQQSGD_COPY is specified for *QSGDisposition*. In this case, an object with the name specified by *ToProcessName* and the disposition MQQSGD_GROUP is searched for to copy from.

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

ToProcessName (MQCFST)
To process name (parameter identifier: MQCACF_TO_PROCESS_NAME).

The name of the new process definition. If a process definition with this name exists, *Replace* must be specified as MQRP_YES.

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

Optional parameters (Change, Copy, and Create Process)

ApplId (MQCFST)
Application identifier (parameter identifier: MQCA_APPL_ID).

ApplId is the name of the application to be started. The application must be on the platform for which the command is executing. The name might typically be a fully qualified file name of an executable object. Qualifying the file name is particularly important if you have multiple IBM MQ installations, to ensure the correct version of the application is run.

The maximum length of the string is MQ_PROCESS_APPL_ID_LENGTH.

ApplType (MQCFIN)
Application type (parameter identifier: MQIA_APPL_TYPE).

Valid application types are:

MQAT_OS400
IBM i application.

MQAT_DOS
DOS client application.

MQAT_WINDOWS
Windows client application.

MQAT_AIX
AIX application (same value as MQAT_UNIX).

MQAT_CICS
CICS transaction.

MQAT_NSK
HP Integrity NonStop Server application.

z/OS **MQAT_ZOS**
z/OS application.

MQAT_DEFAULT
Default application type.

integer: System-defined application type in the range zero through 65 535 or a user-defined application type in the range 65 536 through 999 999 999 (not checked).

Only specify application types (other than user-defined types) that are supported on the platform at which the command is executed:

- IBM i On IBM i: MQAT_OS400, MQAT_CICS, and MQAT_DEFAULT are supported.
- On HP Integrity NonStop Server: MQAT_NSK, MQAT_DOS, MQAT_WINDOWS, and MQAT_DEFAULT are supported.
- UNIX On UNIX: MQAT_UNIX, MQAT_OS2, MQAT_DOS, MQAT_WINDOWS, MQAT_CICS, and MQAT_DEFAULT are supported.
- Windows On Windows: MQAT_WINDOWS_NT, MQAT_OS2, MQAT_DOS, MQAT_WINDOWS, MQAT_CICS, and MQAT_DEFAULT are supported.
- z/OS On z/OS: MQAT_DOS, MQAT_IMS MQAT_MVS, MQAT_UNIX, MQAT_CICS, and MQAT_DEFAULT are supported.

z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- Blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- A queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. In a shared queue environment, you can provide a different queue manager name from the one you are using to enter the command. The command server must be enabled.
- An asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

EnvData (MQCFST)

Environment data (parameter identifier: MQCA_ENV_DATA).

A character string that contains environment information pertaining to the application to be started.

The maximum length of the string is MQ_PROCESS_ENV_DATA_LENGTH.

ProcessDesc (MQCFST)

Description of process definition (parameter identifier: MQCA_PROCESS_DESC).

A plain-text comment that provides descriptive information about the process definition. It must contain only displayable characters.

The maximum length of the string is MQ_PROCESS_DESC_LENGTH.

Use characters from the coded character set identifier (CCSID) for this queue manager. Other characters might be translated incorrectly if the information is sent to another queue manager.

z/OS

QSGDisposition (MQCFIN)

Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be any of the following values:

QSGDisposition	Change	Copy, Create
MQQSGD_COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.	The object is defined on the page set of the queue manager that executes the command using the MQQSGD_GROUP object of the same name as the *ToProcessName* object (for Copy) or *ProcessName* object (for Create).

QSGDisposition	Change	Copy, Create
MQQSGD_GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). On the page set of the queue manager that executes the command, only a local copy of the object is altered by this command. If the command is successful, the following command is generated. `DEFINE PROCESS(process-name)` `REPLACE QSGDISP(COPY)` The command is sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero. The Change for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.	The object definition resides in the shared repository. `GROUP` is allowed only if the queue manager is in a queue-sharing group. If the definition is successful, the following command is generated. `DEFINE PROCESS(process-name)` `REPLACE QSGDISP(COPY)` The command is sent to all active queue managers in the queue-sharing group to attempt to make or refresh local copies on page set zero. The Copy or Create for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.
MQQSGD_PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with MQQSGD_Q_MGR or MQQSGD_COPY. Any object residing in the shared repository is unaffected.	Not permitted.
MQQSGD_Q_MGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command. MQQSGD_Q_MGR is the default value.	The object is defined on the page set of the queue manager that executes the command. MQQSGD_Q_MGR is the default value.

Replace (MQCFIN)
Replace attributes (parameter identifier: MQIACF_REPLACE).

If a process definition with the same name as *ToProcessName* exists, specify whether to replace it.

The value can be any of the following values:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

UserData (MQCFST)
User data (parameter identifier: MQCA_USER_DATA).

A character string that contains user information pertaining to the application (defined by *ApplId*) that is to be started.

For Microsoft Windows, the character string must not contain double quotation marks if the process definition is going to be passed to **runmqtrm**.

The maximum length of the string is MQ_PROCESS_USER_DATA_LENGTH.

Change, Copy, and Create Queue:

The Change Queue command changes existing queue definitions. The Copy and Create Queue commands create new queue definitions - the Copy command uses attribute values of an existing queue definition.

The Change Queue command MQCMD_CHANGE_Q changes the specified attributes of an existing IBM MQ queue. For any optional parameters that are omitted, the value does not change.

The Copy Queue command MQCMD_COPY_Q creates a queue definition of the same type. For attributes not specified in the command, it uses the attribute values of an existing queue definition.

The Create Queue command MQCMD_CREATE_Q creates a queue definition with the specified attributes. All attributes that are not specified are set to the default value for the type of queue that is created.

Required parameters (Change and Create Queue)

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

The name of the queue to be changed. The maximum length of the string is MQ_Q_NAME_LENGTH.

Required parameters (Copy Queue)

FromQName (MQCFST)
From queue name (parameter identifier: MQCACF_FROM_Q_NAME).

Specifies the name of the existing queue definition.

z/OS On z/OS, the queue manager searches for an object with the name you specify and a disposition of MQQSGD_Q_MGR, MQQSGD_COPY, or MQQSGD_SHARED to copy from. This parameter is ignored if a value of MQQSGD_COPY is specified for *QSGDisposition*. In this case, an object with the name specified by *ToQName* and the disposition MQQSGD_GROUP is searched for to copy from.

The maximum length of the string is MQ_Q_NAME_LENGTH.

ToQName (MQCFST)
To queue name (parameter identifier: MQCACF_TO_Q_NAME).

Specifies the name of the new queue definition.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Queue names must be unique; if a queue definition exists with the name and type of the new queue, *Replace* must be specified as MQRP_YES. If a queue definition exists with the same name as and a different type from the new queue, the command fails.

Required parameters (all commands)

QType (MQCFIN)
Queue type (parameter identifier: MQIA_Q_TYPE).

The value specified must match the type of the queue being changed.

The value can be any of the following values:

MQQT_ALIAS
Alias queue definition.

MQQT_LOCAL
Local queue.

MQQT_REMOTE
Local definition of a remote queue.

MQQT_MODEL
Model queue definition.

Optional parameters (Change, Copy, and Create Queue)

BackoutRequeueName (MQCFST) - see MQSC BOQNAME
Excessive backout requeue name (parameter identifier: MQCA_BACKOUT_REQ_Q_NAME).

Specifies the name of the queue to which a message is transferred if it is backed out more times than the value of *BackoutThreshold*. The queue does not have to be a local queue.

The backout queue does not need to exist at this time but it must exist when the *BackoutThreshold* value is exceeded.

The maximum length of the string is MQ_Q_NAME_LENGTH.

BackoutThreshold (MQCFIN)
Backout threshold (parameter identifier: MQIA_BACKOUT_THRESHOLD).

The number of times a message can be backed out before it is transferred to the backout queue specified by *BackoutRequeueName*.

If the value is later reduced, messages that are already on the queue that were backed out at least as many times as the new value remain on the queue. Those messages are transferred if they are backed out again.

Specify a value in the range 0 - 999,999,999.

BaseObjectName (MQCFST)
Name of the object to which the alias resolves (parameter identifier: MQCA_BASE_OBJECT_NAME).

This parameter is the name of a queue or topic that is defined to the local queue manager.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

BaseQName (MQCFST)
Queue name to which the alias resolves (parameter identifier: MQCA_BASE_Q_NAME).

This parameter is the name of a local or remote queue that is defined to the local queue manager.

The maximum length of the string is MQ_Q_NAME_LENGTH.

z/OS **CFStructure (MQCFST)**
Coupling facility structure name (parameter identifier: MQCA_CF_STRUC_NAME). This parameter applies to z/OS only.

Specifies the name of the coupling facility structure where you want to store messages when you use shared queues. The name:

- Cannot have more than 12 characters
- Must start with an uppercase letter (A - Z)
- Can include only the characters A - Z and 0 - 9

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

The name of the queue-sharing group to which the queue manager is connected is prefixed to the name you supply. The name of the queue-sharing group is always four characters, padded with @ symbols if necessary. For example, if you use a queue-sharing group named NY03 and you supply the name PRODUCT7, the resultant coupling facility structure name is NY03PRODUCT7. Note the administrative structure for the queue-sharing group (in this case NY03CSQ_ADMIN) cannot be used for storing messages.

For local and model queues, the following rules apply. The rules apply if you use the Create Queue command with a value of MQRP_YES in the **Replace** parameter. The rules also apply if you use the Change Queue command.

- On a local queue with a value of MQQSGD_SHARED in the **QSGDisposition** parameter, *CFStructure* cannot change.

  If you need to change either the *CFStructure* or *QSGDisposition* value, you must delete and redefine the queue. To preserve any of the messages on the queue you must offload the messages before you delete the queue. Reload the messages after you redefine the queue, or move the messages to another queue.
- On a model queue with a value of MQQDT_SHARED_DYNAMIC in the **DefinitionType** parameter, *CFStructure* cannot be blank.
- On a local queue with a value other than MQQSGD_SHARED in the **QSGDisposition** parameter, the value of *CFStructure* does not matter. The value *CFStructure* also does not matter for a model queue with a value other than MQQDT_SHARED_DYNAMIC in the **DefinitionType** parameter.

For local and model queues, when you use the Create Queue command with a value of MQRP_NO in the **Replace** parameter, the coupling facility structure:

- On a local queue with a value of MQQSGD_SHARED in the **QSGDisposition** parameter, or a model queue with a value of MQQDT_SHARED_DYNAMIC in the **DefinitionType** parameter, *CFStructure* cannot be blank.
- On a local queue with a value other than MQQSGD_SHARED in the **QSGDisposition** parameter, the value of *CFStructure* does not matter. The value *CFStructure* also does not matter for a model queue with a value other than MQQDT_SHARED_DYNAMIC in the **DefinitionType** parameter.

Note: Before you can use the queue, the structure must be defined in the coupling facility Resource Management (CFRM) policy data set.

ClusterChannelName (MQCFST)

This parameter is supported only on transmission queues.

ClusterChannelName is the generic name of the cluster-sender channels that use this queue as a transmission queue. The attribute specifies which cluster-sender channels send messages to a cluster-receiver channel from this cluster transmission queue. (Parameter identifier: MQCA_CLUS_CHL_NAME.)

You can also set the transmission queue attribute ClusterChannelName attribute to a cluster-sender channel manually. Messages that are destined for the queue manager connected by the cluster-sender channel are stored in the transmission queue that identifies the cluster-sender channel. They are not stored in the default cluster transmission queue. If you set the ClusterChannelName attribute to blanks, the channel switches to the default cluster transmission queue when the channel restarts. The default queue is either SYSTEM.CLUSTER.TRANSMIT.*ChannelName* or SYSTEM.CLUSTER.TRANSMIT.QUEUE, depending on the value of the queue manager DefClusterXmitQueueType attribute.

By specifying asterisks, "*", in ClusterChannelName, you can associate a transmission queue with a set of cluster-sender channels. The asterisks can be at the beginning, end, or any number of places in the middle of the channel name string. ClusterChannelName is limited to a length of 20 characters: MQ_CHANNEL_NAME_LENGTH.

The default queue manager configuration is for all cluster-sender channels to send messages from a single transmission queue, SYSTEM.CLUSTER.TRANSMIT.QUEUE. The default configuration can be changed by modified by changing the queue manager attribute, DefClusterXmitQueueType. The default value of the attribute is SCTQ. You can change the value to CHANNEL. If you set the DefClusterXmitQueueType attribute to CHANNEL, each cluster-sender channel defaults to using a specific cluster transmission queue, SYSTEM.CLUSTER.TRANSMIT.*ChannelName*.

ClusterName (MQCFST)

Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster to which the queue belongs.

Changes to this parameter do not affect instances of the queue that are open.

Only one of the resultant values of **ClusterName** and **ClusterNamelist** can be nonblank; you cannot specify a value for both.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

ClusterNamelist (MQCFST)
Cluster namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

The name of the namelist, that specifies a list of clusters to which the queue belongs.

Changes to this parameter do not affect instances of the queue that are open.

Only one of the resultant values of **ClusterName** and **ClusterNamelist** can be nonblank; you cannot specify a value for both.

CLWLQueuePriority (MQCFIN)
Cluster workload queue priority (parameter identifier: MQIA_CLWL_Q_PRIORITY).

Specifies the priority of the queue in cluster workload management; see Configuring a queue manager cluster. The value must be in the range 0 - 9, where 0 is the lowest priority and 9 is the highest.

CLWLQueueRank (MQCFIN)
Cluster workload queue rank (parameter identifier: MQIA_CLWL_Q_RANK).

Specifies the rank of the queue in cluster workload management. The value must be in the range 0 - 9, where 0 is the lowest priority and 9 is the highest.

CLWLUseQ (MQCFIN)
Cluster workload use remote queue (parameter identifier: MQIA_CLWL_USEQ).

Specifies whether remote and local queues are to be used in cluster workload distribution. The value can be any of the following values:

MQCLWL_USEQ_AS_Q_MGR
Use the value of the **CLWLUseQ** parameter on the definition of the queue manager.

MQCLWL_USEQ_ANY
Use remote and local queues.

MQCLWL_USEQ_LOCAL
Do not use remote queues.

z/OS **CommandScope (MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is run when the queue manager is a member of a queue-sharing group. You can specify one of the following values:

- Blank, or omit the parameter altogether. The command is run on the queue manager on which it was entered.
- A queue manager name. The command is run on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment. The command server must be enabled.
- An asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Custom (MQCFST)
Custom attribute for new features (parameter identifier: MQCA_CUSTOM).

This attribute contains the values of attributes, as pairs of attribute name and value, separated by at least one space. The attribute name-value pairs have the form NAME(VALUE). Single quotation marks must be escaped with another single quotation mark.

CAPEXPRY (*integer*)
The maximum time, expressed in tenths of a second, until a message put using an object handle, opened using this object on the resolution path, remains in the system until it becomes eligible for expiry processing.

For more information on message expiry processing, see Enforcing lower expiration times.

The value can be one of the following:

integer
The value must be in the range one through to 999 999 999.

NOLIMIT
There is no limit on the expiry time of messages put using this object. This is the default value.

Specifying a value for CAPEXPRY that is not valid, does not cause the command to fail. Instead, the default value is used.

DefaultPutResponse (MQCFIN)
Default put response type definition (parameter identifier: MQIA_DEF_PUT_RESPONSE_TYPE).

The parameter specifies the type of response to be used for put operations to the queue when an application specifies MQPMO_RESPONSE_AS_Q_DEF. The value can be any of the following values:

MQPRT_SYNC_RESPONSE
The put operation is issued synchronously, returning a response.

MQPRT_ASYNC_RESPONSE
The put operation is issued asynchronously, returning a subset of MQMD fields.

DefBind (MQCFIN)
Bind definition (parameter identifier: MQIA_DEF_BIND).

The parameter specifies the binding to be used when MQOO_BIND_AS_Q_DEF is specified on the MQOPEN call. The value can be any of the following values:

MQBND_BIND_ON_OPEN
The binding is fixed by the MQOPEN call.

MQBND_BIND_NOT_FIXED
The binding is not fixed.

MQBND_BIND_ON_GROUP
Allows an application to request that a group of messages are all allocated to the same destination instance.

Changes to this parameter do not affect instances of the queue that are open.

DefinitionType (MQCFIN)
Queue definition type (parameter identifier: MQIA_DEFINITION_TYPE).

The value can be any of the following values:

MQQDT_PERMANENT_DYNAMIC
Dynamically defined permanent queue.

MQQDT_SHARED_DYNAMIC
Dynamically defined shared queue. This option is available on z/OS only.

MQQDT_TEMPORARY_DYNAMIC
Dynamically defined temporary queue.

DefInputOpenOption (MQCFIN)
Default input open option (parameter identifier: MQIA_DEF_INPUT_OPEN_OPTION).

Specifies the default share option for applications opening this queue for input.

The value can be any of the following values:

MQOO_INPUT_EXCLUSIVE
Open queue to get messages with exclusive access.

MQOO_INPUT_SHARED
Open queue to get messages with shared access.

DefPersistence (MQCFIN)
Default persistence (parameter identifier: MQIA_DEF_PERSISTENCE).

Specifies the default for message-persistence on the queue. Message persistence determines whether messages are preserved across restarts of the queue manager.

The value can be any of the following values:

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

DefPriority (MQCFIN)
Default priority (parameter identifier: MQIA_DEF_PRIORITY).

Specifies the default priority of messages put on the queue. The value must be in the range zero through to the maximum priority value that is supported (9).

DefReadAhead (MQCFIN)
Default read ahead (parameter identifier: MQIA_DEF_READ_AHEAD).

Specifies the default read ahead behavior for non-persistent messages delivered to the client.

The value can be any of the following values:

MQREADA_NO
Non-persistent messages are not read ahead unless the client application is configured to request read ahead.

MQREADA_YES
Non-persistent messages are sent ahead to the client before an application requests them. Non-persistent messages can be lost if the client ends abnormally or if the client does not consume all the messages it is sent.

MQREADA_DISABLED
Read ahead of non-persistent messages is not enabled for this queue. Messages are not sent ahead to the client regardless of whether read ahead is requested by the client application.

Multi **DistLists (MQCFIN)**
Distribution list support (parameter identifier: MQIA_DIST_LISTS).

Specifies whether distribution-list messages can be placed on the queue.

Note: This attribute is set by the sending message channel agent (MCA). The sending MCA removes messages from the queue each time it establishes a connection to a receiving MCA on a partner queue manager. The attribute is not normally set by administrators, although it can be set if the need arises.

This parameter is supported on Multiplatforms.

The value can be any of the following values:

MQDL_SUPPORTED
Distribution lists supported.

MQDL_NOT_SUPPORTED
Distribution lists not supported.

Force (MQCFIN)
Force changes (parameter identifier: MQIACF_FORCE).

Specifies whether the command must be forced to complete when conditions are such that completing the command would affect an open queue. The conditions depend upon the type of the queue that is being changed:

QALIAS *BaseQName* is specified with a queue name and an application has the alias queue open.

QLOCAL Either of the following conditions indicates that a local queue would be affected:

- *Shareability* is specified as MQQA_NOT_SHAREABLE and more than one application has the local queue open for input.
- The *Usage* value is changed and one or more applications has the local queue open, or there are one or more messages on the queue. (The *Usage* value must not normally be changed while there are messages on the queue. The format of messages changes when they are put on a transmission queue.)

QREMOTE
Either of the following conditions indicates that a remote queue would be affected:

- If *XmitQName* is specified with a transmission-queue name, or blank, and an application has a remote queue open that would be affected by this change.
- If any of the following parameters are specified with a queue or queue manager name, and one or more applications has a queue open that resolved through this definition as a queue manager alias. The parameters are:
  1. *RemoteQName*
  2. *RemoteQMgrName*
  3. *XmitQName*

QMODEL This parameter is not valid for model queues.

Note: A value of MQFC_YES is not required if this definition is in use as a reply-to queue definition only.
The value can be any of the following values:

MQFC_YES
Force the change.

MQFC_NO
Do not force the change.

HardenGetBackout (MQCFIN)
Harden the backout count, or not (parameter identifier: MQIA_HARDEN_GET_BACKOUT).

Specifies whether the count of backed out messages is saved (hardened) across restarts of the message queue manager.

Note: IBM MQ for IBM i always hardens the count, regardless of the setting of this attribute.

The value can be any of the following values:

MQQA_BACKOUT_HARDENED
Backout count remembered.

MQQA_BACKOUT_NOT_HARDENED
Backout count might not be remembered.

V 9.0.2 **ImageRecoverQueue (MQCFST)**

Specifies whether a local or permanent dynamic queue object is recoverable from a media image, if linear logging is being used (parameter identifier: MQIA_MEDIA_IMAGE_RECOVER_Q).

This parameter is not valid on z/OS. Possible values are:

MQIMGRCOV_YES
These queue objects are recoverable.

MQIMGRCOV_NO
The "rcdmqimg (record media image)" on page 283 and"rcrmqobj (re-create object)" on page 289 commands are not permitted for these objects, and automatic media images, if enabled, are not written for these objects.

MQIMGRCOV_AS_Q_MGR
If you specify MQIMGRCOV_AS_Q_MGR , and the **ImageRecoverQueue** attribute for the queue manager specifies MQIMGRCOV_YES , these queue objects are recoverable.

If you specify MQIMGRCOV_AS_Q_MGR and the **ImageRecoverQueue** attribute for the queue manager specifies MQIMGRCOV_NO, the "rcdmqimg (record media image)" on page 283 and"rcrmqobj (re-create object)" on page 289 commands are not permitted for these objects, and automatic media images, if enabled, are not written for these objects.

MQIMGRCOV_AS_Q_MGR is the default value.

IndexType (MQCFIN)

Index type (parameter identifier: MQIA_INDEX_TYPE). This parameter applies to z/OS only.

Specifies the type of index maintained by the queue manager to expedite MQGET operations on the queue. For shared queues, the type of index determines what type of MQGET calls can be used. The value can be any of the following values:

MQIT_NONE
No index.

MQIT_MSG_ID
The queue is indexed using message identifiers.

MQIT_CORREL_ID
The queue is indexed using correlation identifiers.

MQIT_MSG_TOKEN
The queue is indexed using message tokens.

MQIT_GROUP_ID
The queue is indexed using group identifiers.

Messages can be retrieved using a selection criterion only if an appropriate index type is maintained, as the following table shows:

Retrieval selection criterion	*IndexType* **required**	
	Shared queue	**Other queue**
None (sequential retrieval)	Any	Any
Message identifier	MQIT_MSG_ID or MQIT_NONE	Any
Correlation identifier	MQIT_CORREL_ID	Any
Message and correlation identifiers	MQIT_MSG_ID or MQIT_CORREL_ID	Any
Group identifier	MQIT_GROUP_ID	Any
Grouping	MQIT_GROUP_ID	MQIT_GROUP_ID

Retrieval selection criterion	*IndexType* required	
Message token	Not allowed	MQIT_MSG_TOKEN

InhibitGet (MQCFIN)
Get operations are allowed or inhibited (parameter identifier: MQIA_INHIBIT_GET).

The value can be:

MQQA_GET_ALLOWED
Get operations are allowed.

MQQA_GET_INHIBITED
Get operations are inhibited.

InhibitPut (MQCFIN)
Put operations are allowed or inhibited (parameter identifier: MQIA_INHIBIT_PUT).

Specifies whether messages can be put on the queue.

The value can be any of the following values:

MQQA_PUT_ALLOWED
Put operations are allowed.

MQQA_PUT_INHIBITED
Put operations are inhibited.

InitiationQName (MQCFST)
Initiation queue name (parameter identifier: MQCA_INITIATION_Q_NAME).

The local queue for trigger messages relating to this queue. The initiation queue must be on the same queue manager.

The maximum length of the string is MQ_Q_NAME_LENGTH.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

The maximum length for messages on the queue. Applications might use the value of this attribute to determine the size of buffer they need to retrieve messages from the queue. If you change this value it might cause an application to operate incorrectly.

Do not set a value that is greater than the *MaxMsgLength* attribute of a queue manager.

The lower limit for this parameter is 0. The upper limit depends on the environment:

- On z/OS NSS Client HP Integrity NonStop Server, IBM i, UNIX, Linux, and Windows, the maximum message length is 100 MB (104,857,600 bytes).
- On other UNIX systems, the maximum message length is 4 MB (4,194,304 bytes).

MaxQDepth (MQCFIN)
Maximum queue depth (parameter identifier: MQIA_MAX_Q_DEPTH).

The maximum number of messages allowed on the queue.

Note: Other factors might cause the queue to be treated as full. For example, it appears to be full if there is no storage available for a message.

Specify a value greater than or equal to 0, and less than or equal to 999,999,999.

MsgDeliverySequence (MQCFIN)
Messages are delivered in priority order or sequence (parameter identifier: MQIA_MSG_DELIVERY_SEQUENCE).

The value can be any of the following values:

MQMDS_PRIORITY
Messages are returned in priority order.

MQMDS_FIFO
Messages are returned in FIFO order (first in, first out).

NonPersistentMessageClass (MQCFIN)
The level of reliability to be assigned to non-persistent messages that are put to the queue (parameter identifier: MQIA_NPM_CLASS).

The value can be:

MQNPM_CLASS_NORMAL
Non-persistent messages persist as long as the lifetime of the queue manager session. They are discarded in the event of a queue manager restart. This value is the default value.

MQNPM_CLASS_HIGH
The queue manager attempts to retain non-persistent messages for the lifetime of the queue. Non-persistent messages might still be lost in the event of a failure.

This parameter is valid only on local and model queues. It is not valid on z/OS.

ProcessName (MQCFST)
Name of process definition for the queue (parameter identifier: MQCA_PROCESS_NAME).

Specifies the local name of the IBM MQ process that identifies the application to be started when a trigger event occurs.

- If the queue is a transmission queue, the process definition contains the name of the channel to be started. This parameter is optional for transmission queues. If you do not specify it, the channel name is taken from the value specified for the **TriggerData** parameter.
- In other environments, the process name must be nonblank for a trigger event to occur, although it can be set after creating the queue.

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

PropertyControl (MQCFIN)
Property control attribute (parameter identifier: MQIA_PROPERTY_CONTROL).

Specifies how message properties are handled when messages are retrieved from queues using the MQGET call with the MQGMO_PROPERTIES_AS_Q_DEF option. The value can be any of the following values:

MQPROP_COMPATIBILITY
If the message contains a property with a prefix of **mcd.**, **jms.**, **usr.** or **mqext.**, all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except those properties contained in the message descriptor (or extension), are discarded and are no longer accessible to the application.

This value is the default value. It allows applications which expect JMS-related properties to be in an MQRFH2 header in the message data to continue to work unmodified.

MQPROP_NONE
All properties of the message are removed from the message before the message is sent to the remote queue manager. Properties in the message descriptor, or extension, are not removed.

MQPROP_ALL
All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except those properties in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

MQPROP_FORCE_MQRFH2
Properties are always returned in the message data in an MQRFH2 header regardless of whether the application specifies a message handle.

A valid message handle supplied in the MsgHandle field of the MQGMO structure on the MQGET call is ignored. Properties of the message are not accessible using the message handle.

MQPROP_V6COMPAT
Any application MQRFH2 header is received as it was sent. Any properties set using MQSETMP must be retrieved using MQINQMP. They are not added to the MQRFH2 created by the application. Properties that were set in the MQRFH2 header by the sending application cannot be retrieved using MQINQMP.

This parameter is applicable to Local, Alias, and Model queues.

QDepthHighEvent (MQCFIN)
Controls whether Queue Depth High events are generated (parameter identifier: MQIA_Q_DEPTH_HIGH_EVENT).

A Queue Depth High event indicates that an application put a message on a queue. This event caused the number of messages on the queue to become greater than or equal to the queue depth high threshold. See the **QDepthHighLimit** parameter.

Note: The value of this attribute can change implicitly; see "Definitions of the Programmable Command Formats" on page 1069.

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthHighLimit (MQCFIN)
High limit for queue depth (parameter identifier: MQIA_Q_DEPTH_HIGH_LIMIT).

The threshold against which the queue depth is compared to generate a Queue Depth High event.

This event indicates that an application put a message to a queue. This event caused the number of messages on the queue to become greater than or equal to the queue depth high threshold. See the **QDepthHighEvent** parameter.

The value is expressed as a percentage of the maximum queue depth, *MaxQDepth*. It must be greater than or equal to 0 and less than or equal to 100.

QDepthLowEvent (MQCFIN)
Controls whether Queue Depth Low events are generated (parameter identifier: MQIA_Q_DEPTH_LOW_EVENT).

A Queue Depth Low event indicates that an application retrieved a message from a queue. This event caused the number of messages on the queue to become less than or equal to the queue depth low threshold. See the **QDepthLowLimit** parameter.

Note: The value of this attribute can change implicitly. See "Definitions of the Programmable Command Formats" on page 1069.

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthLowLimit (MQCFIN)
Low limit for queue depth (parameter identifier: MQIA_Q_DEPTH_LOW_LIMIT).

The threshold against which the queue depth is compared to generate a Queue Depth Low event.

This event indicates that an application retrieved a message from a queue. This event caused the number of messages on the queue to become less than or equal to the queue depth low threshold. See the **QDepthLowEvent** parameter.

Specify the value as a percentage of the maximum queue depth (**MaxQDepth** attribute), in the range 0 through 100.

QDepthMaxEvent (MQCFIN)
Controls whether Queue Full events are generated (parameter identifier: MQIA_Q_DEPTH_MAX_EVENT).

A Queue Full event indicates that an MQPUT call to a queue was rejected because the queue is full. That is, the queue depth reached its maximum value.

Note: The value of this attribute can change implicitly; see "Definitions of the Programmable Command Formats" on page 1069.

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDesc (MQCFST)
Queue description (parameter identifier: MQCA_Q_DESC).

Text that briefly describes the object.

The maximum length of the string is MQ_Q_DESC_LENGTH.

Use characters from the character set identified by the coded character set identifier (CCSID) for the message queue manager on which the command is executing. This choice ensures that the text is translated correctly if it is sent to another queue manager.

QServiceInterval (MQCFIN)
Target for queue service interval (parameter identifier: MQIA_Q_SERVICE_INTERVAL).

The service interval used for comparison to generate Queue Service Interval High and Queue Service Interval OK events. See the *QServiceIntervalEvent* parameter.

Specify a value in the range 0 through 999 999 999 milliseconds.

QServiceIntervalEvent (MQCFIN)
Controls whether Service Interval High or Service Interval OK events are generated (parameter identifier: MQIA_Q_SERVICE_INTERVAL_EVENT).

A Queue Service Interval High event is generated when a check indicates that no messages were retrieved from, or put to, the queue for at least the time indicated by the **QServiceInterval** attribute.

A Queue Service Interval OK event is generated when a check indicates that a message was retrieved from the queue within the time indicated by the **QServiceInterval** attribute.

Note: The value of this attribute can change implicitly; see "Definitions of the Programmable Command Formats" on page 1069.

The value can be any of the following values:

MQQSIE_HIGH
Queue Service Interval High events enabled.

- Queue Service Interval High events are enabled and
- Queue Service Interval OK events are disabled.

MQQSIE_OK
Queue Service Interval OK events enabled.

- Queue Service Interval High events are disabled and
- Queue Service Interval OK events are enabled.

MQQSIE_NONE
No queue service interval events enabled.

- Queue Service Interval High events are disabled and
- Queue Service Interval OK events are also disabled.

z/OS **QSGDisposition (MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be any of the following values:

QSGDisposition	Change	Copy, Create
MQQSGD_COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.	The object is defined on the page set of the queue manager that executes the command using the MQQSGD_GROUP object of the same name as the *ToQName* object (for Copy) or the *QName* object (for Create). For local queues, messages are stored on the page sets of each queue manager and are available only through that queue manager.
MQQSGD_GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command. If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero: `DEFINE QUEUE(q-name)` `REPLACE QSGDISP(COPY)` The Change for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.	The object definition resides in the shared repository. This value is allowed only in a shared queue manager environment. If the definition is successful, the following MQSC command is generated and sent to all active queue managers to attempt to make or refresh local copies on page set zero: `DEFINE QUEUE(q-name)` `REPLACE QSGDISP(COPY)` The Copy or Create for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.
MQQSGD_PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with MQQSGD_Q_MGR or MQQSGD_COPY. Any object residing in the shared repository is unaffected.	Not permitted.

QSGDisposition	Change	Copy, Create
MQQSGD_Q_MGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command. This value is the default value.	The object is defined on the page set of the queue manager that executes the command. This value is the default value. For local queues, messages are stored on the page sets of each queue manager and are available only through that queue manager.
MQQSGD_SHARED	This value applies only to local queues. The object definition resides in the shared repository. The object was defined by a command using the parameter MQQSGD_SHARED. Any object residing on the page set of the queue manager that executes the command, or any object defined by a command using the parameter MQQSGD_GROUP, is not affected by this command.	This option applies only to local queues. The object is defined in the shared repository. Messages are stored in the coupling facility and are available to any queue manager in the queue-sharing group. You can specify MQQSGD_SHARED only if: • *CFStructure* is nonblank • *IndexType* is not MQIT_MSG_TOKEN • The queue is not one of the following: – SYSTEM.CHANNEL.INITQ – SYSTEM.COMMAND.INPUT

QueueAccounting (MQCFIN)
Controls the collection of accounting data (parameter identifier: MQIA_ACCOUNTING_Q).

The value can be:

MQMON_Q_MGR
The collection of accounting data for the queue is performed based upon the setting of the **QueueAccounting** parameter on the queue manager.

MQMON_OFF
Accounting data collection is disabled for the queue.

MQMON_ON
If the value of the queue manager's *QueueAccounting* parameter is not MQMON_NONE, accounting data collection is enabled for the queue.

QueueMonitoring (MQCFIN)
Online monitoring data collection (parameter identifier: MQIA_MONITORING_Q).

Specifies whether online monitoring data is to be collected and, if so, the rate at which the data is collected. The value can be any of the following values:

MQMON_OFF
Online monitoring data collection is turned off for this queue.

MQMON_Q_MGR
The value of the queue manager's **QueueMonitoring** parameter is inherited by the queue.

MQMON_LOW
If the value of the queue manager **QueueMonitoring** parameter is not MQMON_NONE, online monitoring data collection is turned on. The rate of data collection is low for this queue.

MQMON_MEDIUM
If the value of the queue manager **QueueMonitoring** parameter is not MQMON_NONE, online monitoring data collection is turned on. The rate of data collection is moderate for this queue.

MQMON_HIGH
If the value of the queue manager **QueueMonitoring** parameter is not MQMON_NONE, online monitoring data collection is turned on. The rate of data collection is high for this queue.

QueueStatistics (MQCFIN)
Statistics data collection (parameter identifier: MQIA_STATISTICS_Q).

Specifies whether statistics data collection is enabled. The value can be any of the following values:

MQMON_Q_MGR
The value of the queue manager's **QueueStatistics** parameter is inherited by the queue.

MQMON_OFF
Statistics data collection is disabled

MQMON_ON
If the value of the queue manager's *QueueStatistics* parameter is not MQMON_NONE, statistics data collection is enabled

This parameter is valid only on IBM i, UNIX, and Windows.

RemoteQMgrName (MQCFST)
Name of remote queue manager (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

If an application opens the local definition of a remote queue, *RemoteQMgrName* must not be blank or the name of the queue manager the application is connected to. If *XmitQName* is blank there must be a local queue called *RemoteQMgrName*. That queue is used as the transmission queue.

If this definition is used for a queue manager alias, *RemoteQMgrName* is the name of the queue manager. The queue manager name can be the name of the connected queue manager. If *XmitQName* is blank, when the queue is opened there must be a local queue called *RemoteQMgrName*. That queue is used as the transmission queue.

If this definition is used for a reply-to queue alias, *RemoteQMgrName* is the name of the queue manager that is to be the reply-to queue manager.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

RemoteQName (MQCFST)
Name of remote queue as known locally on the remote queue manager (parameter identifier: MQCA_REMOTE_Q_NAME).

If this definition is used for a local definition of a remote queue, *RemoteQName* must not be blank when the open occurs.

If this definition is used for a queue manager alias definition, *RemoteQName* must be blank when the open occurs.

If this definition is used for a reply-to queue alias, this name is the name of the queue that is to be the reply-to queue.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Replace (MQCFIN)
Replace attributes (parameter identifier: MQIACF_REPLACE). This parameter is not valid on a Change Queue command.

If the object exists, the effect is like issuing the Change Queue command. It is like a Change Queue command without the MQFC_YES option on the **Force** parameter, and with all of the other attributes specified. In particular, note that any messages which are on the existing queue are retained.

The Change Queue command without MQFC_YES on the **Force** parameter, and the Create Queue command with MQRP_YES on the **Replace** parameter, are different. The difference is that the Change Queue command does not change unspecified attributes. Create Queue with MQRP_YES sets all the attributes. If you use MQRP_YES, unspecified attributes are taken from the default definition, and the attributes of the object being replaced, if one exists, are ignored.)

The command fails if both of the following statements are true:

- The command sets attributes that would require the use of MQFC_YES on the **Force** parameter if you were using the Change Queue command.
- The object is open.

The Change Queue command with MQFC_YES on the **Force** parameter succeeds in this situation.

If MQSCO_CELL is specified on the **Scope** parameter on UNIX, and there is already a queue with the same name in the cell directory, the command fails. The command fails even if MQRP_YES is specified.

The value can be any of the following values:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

RetentionInterval (MQCFIN)
Retention interval (parameter identifier: MQIA_RETENTION_INTERVAL).

The number of hours for which the queue might be needed, based on the date and time when the queue was created.

This information is available to a housekeeping application or an operator and can be used to determine when a queue is no longer required. The queue manager does not delete queues nor does it prevent queues from being deleted if their retention interval is not expired. It is the responsibility of the user to take any required action.

Specify a value in the range 0 - 999,999,999.

Scope (MQCFIN)
Scope of the queue definition (parameter identifier: MQIA_SCOPE).

Specifies whether the scope of the queue definition extends beyond the queue manager which owns the queue. It does so if the queue name is contained in a cell directory, so that it is known to all the queue managers within the cell.

If this attribute is changed from MQSCO_CELL to MQSCO_Q_MGR, the entry for the queue is deleted from the cell directory.

Model and dynamic queues cannot be changed to have cell scope.

If it is changed from MQSCO_Q_MGR to MQSCO_CELL, an entry for the queue is created in the cell directory. If there is already a queue with the same name in the cell directory, the command fails. The command also fails if no name service supporting a cell directory is configured.

The value can be:

MQSCO_Q_MGR
Queue manager scope.

MQSCO_CELL
Cell scope.

This value is not supported on IBM i.

This parameter is not available on z/OS.

Shareability (MQCFIN)
The queue can be shared, or not (parameter identifier: MQIA_SHAREABILITY).

Specifies whether multiple instances of applications can open this queue for input.

The value can be any of the following values:

MQQA_SHAREABLE
Queue is shareable.

MQQA_NOT_SHAREABLE
Queue is not shareable.

z/OS **StorageClass (MQCFST)**
Storage class (parameter identifier: MQCA_STORAGE_CLASS). This parameter applies to z/OS only.

Specifies the name of the storage class.

The maximum length of the string is MQ_STORAGE_CLASS_LENGTH.

TargetType (MQCFIN)
Target type (parameter identifier: MQIA_BASE_TYPE).

Specifies the type of object to which the alias resolves.

The value can be any of the following values:

MQOT_Q The object is a queue.

MQOT_TOPIC
The object is a topic.

TriggerControl (MQCFIN)
Trigger control (parameter identifier: MQIA_TRIGGER_CONTROL).

Specifies whether trigger messages are written to the initiation queue.

The value can be:

MQTC_OFF
Trigger messages not required.

MQTC_ON
Trigger messages required.

TriggerData (MQCFST)
Trigger data (parameter identifier: MQCA_TRIGGER_DATA).

Specifies user data that the queue manager includes in the trigger message. This data is made available to the monitoring application that processes the initiation queue and to the application that is started by the monitor.

The maximum length of the string is MQ_TRIGGER_DATA_LENGTH.

TriggerDepth (MQCFIN)
Trigger depth (parameter identifier: MQIA_TRIGGER_DEPTH).

Specifies (when *TriggerType* is MQTT_DEPTH) the number of messages that initiates a trigger message to the initiation queue. The value must be in the range 1 through 999 999 999.

TriggerMsgPriority (MQCFIN)
Threshold message priority for triggers (parameter identifier: MQIA_TRIGGER_MSG_PRIORITY).

Specifies the minimum priority that a message must have before it can cause, or be counted for, a trigger event. The value must be in the range of priority values that is supported (0 through 9).

TriggerType (MQCFIN)
Trigger type (parameter identifier: MQIA_TRIGGER_TYPE).

Specifies the condition that initiates a trigger event. When the condition is true, a trigger message is sent to the initiation queue.

The value can be any of the following values:

MQTT_NONE
No trigger messages.

MQTT_EVERY
Trigger message for every message.

MQTT_FIRST
Trigger message when queue depth goes from 0 to 1.

MQTT_DEPTH
Trigger message when depth threshold exceeded.

Usage (MQCFIN)
Usage (parameter identifier: MQIA_USAGE).

Specifies whether the queue is for normal usage or for transmitting messages to a remote message queue manager.

The value can be any of the following values:

MQUS_NORMAL
Normal usage.

MQUS_TRANSMISSION
Transmission queue.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME).

Specifies the local name of the transmission queue to be used for messages destined for either a remote queue or for a queue manager alias definition.

If *XmitQName* is blank, a queue with the same name as *RemoteQMgrName* is used as the transmission queue.

This attribute is ignored if the definition is being used as a queue manager alias and *RemoteQMgrName* is the name of the connected queue manager.

It is also ignored if the definition is used as a reply-to queue alias definition.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Error codes (Change, Copy, and Create Queue)

This command might return the following errors in the response format header, in addition to the values shown on in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CELL_DIR_NOT_AVAILABLE
Cell directory is not available.

MQRCCF_CLUSTER_NAME_CONFLICT
Cluster name conflict.

MQRCCF_CLUSTER_Q_USAGE_ERROR
Cluster usage conflict.

MQRCCF_DYNAMIC_Q_SCOPE_ERROR
Dynamic queue scope error.

MQRCCF_FORCE_VALUE_ERROR
Force value not valid.

MQRCCF_Q_ALREADY_IN_CELL
Queue exists in cell.

MQRCCF_Q_TYPE_ERROR
Queue type not valid.

Change Queue Manager:

The Change Queue Manager (MQCMD_CHANGE_Q_MGR) command changes the specified attributes of the queue manager.

For any optional parameters that are omitted, the value does not change.

Required parameters:
None

Optional parameters (Change Queue Manager)

Multi **AccountingConnOverride (MQCFIN)**
Specifies whether applications can override the settings of the *QueueAccounting* and *MQIAccounting* queue manager parameters (parameter identifier: MQIA_ACCOUNTING_CONN_OVERRIDE).

The value can be any of the following values:

MQMON_DISABLED
Applications cannot override the settings of the **QueueAccounting** and **MQIAccounting** parameters.

This value is the initial default value for the queue manager.

MQMON_ENABLED
Applications can override the settings of the **QueueAccounting** and **MQIAccounting** parameters by using the options field of the MQCNO structure of the MQCONNX API call.

This parameter is valid only on Multiplatforms.

Multi **AccountingInterval (MQCFIN)**
The time interval, in seconds, at which intermediate accounting records are written (parameter identifier: MQIA_ACCOUNTING_INTERVAL).

Specify a value in the range 1 - 604,000.

This parameter is valid only on Multiplatforms.

ActivityRecording (MQCFIN)
Specifies whether activity reports can be generated (parameter identifier: MQIA_ACTIVITY_RECORDING).

The value can be:

MQRECORDING_DISABLED
Activity reports cannot be generated.

MQRECORDING_MSG
Activity reports can be generated and sent to the reply queue specified by the originator in the message causing the report.

MQRECORDING_Q
Activity reports can be generated and sent to SYSTEM.ADMIN.ACTIVITY.QUEUE.

z/OS **AdoptNewMCACheck (MQCFIN)**
The elements checked to determine whether an MCA must be adopted (restarted) when a new inbound channel is detected. It must be adopted (restarted) if it that has the same name as a currently active MCA (parameter identifier: MQIA_ADOPTNEWMCA_CHECK).

The value can be:

MQADOPT_CHECK_Q_MGR_NAME
Check the queue manager name.

MQADOPT_CHECK_NET_ADDR
Check the network address.

MQADOPT_CHECK_ALL
Check the queue manager name and network address. Perform this check to prevent your channels from being inadvertently shut down. This value is the initial default value of the queue manager.

MQADOPT_CHECK_NONE
Do not check any elements.

This parameter applies to z/OS only.

z/OS **AdoptNewMCAType (MQCFIN)**
Adoption of orphaned channel instances (parameter identifier: MQIA_ADOPTNEWMCA_TYPE).

Specify whether an orphaned MCA instance is to be adopted when a new inbound channel request is detected matching the **AdoptNewMCACheck** parameters.

The value can be:

MQADOPT_TYPE_NO
Do not adopt orphaned channel instances.

MQADOPT_TYPE_ALL
Adopt all channel types. This value is the initial default value of the queue manager.

This parameter applies to z/OS only.

AuthorityEvent (MQCFIN)
Controls whether authorization (Not Authorized) events are generated (parameter identifier: MQIA_AUTHORITY_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled. This value is not permitted on z/OS.

BridgeEvent (MQCFIN)
Controls whether IMS bridge events are generated (parameter identifier: MQIA_BRIDGE_EVENT). This parameter applies to z/OS only.

The value can be:

MQEVR_DISABLED
Event reporting disabled. This value is the default value.

MQEVR_ENABLED
Event reporting enabled.

CertificateLabel (MQCFST)
Specifies the certificate label for this queue manager to use. The label identifies which personal certificate in the key repository has been selected (parameter identifier: MQCA_CERT_LABEL).

The default and migrated queue manager values are:

- ULW On UNIX, Linux, and Windows: *ibmwebspheremqxxxx* where *xxxx* is the queue manager name folded to lower case.
- IBM i On IBM i:

- If you specified SSLKEYR(*SYSTEM), the value is blank.

  Note that it is forbidden to use a nonblank queue manager CERTLABL with SSLKEYR(*SYSTEM). Attempting to do so results in an MQRCCF_Q_MGR_ATTR_CONFLICT error.
- Otherwise, *ibmwebspheremqxxxx* where *xxxx* is the queue manager name folded to lower case.

- z/OS On z/OS: *ibmWebSphereMQXXXX* where *XXXX* is the queue manager name.

CertificateValPolicy (MQCFIN)
Specifies which TLS certificate validation policy is used to validate digital certificates received from remote partner systems (parameter identifier: MQIA_CERT_VAL_POLICY).

This attribute can be used to control how strictly the certificate chain validation conforms to industry security standards. For more information, see Certificate validation policies in IBM MQ.

The value can be any of the following values:

MQ_CERT_VAL_POLICY_ANY
Apply each of the certificate validation policies supported by the secure sockets library and accept the certificate chain if any of the policies considers the certificate chain valid. This setting can be used for maximum backwards compatibility with older digital certificates which do not comply with the modern certificate standards.

MQ_CERT_VAL_POLICY_RFC5280
Apply only the RFC 5280 compliant certificate validation policy. This setting provides stricter validation than the ANY setting, but rejects some older digital certificates.

This parameter is only valid on UNIX, Linux, and Windows and can be used only on a queue manager with a command level of 711, or higher.

Changes to **CertificateValPolicy** become effective either:

- When a new channel process is started.
- For channels that run as threads of the channel initiator, when the channel initiator is restarted.
- For channels that run as threads of the listener, when the listener is restarted.
- For channels that run as threads of a process pooling process, when the process pooling process is started or restarted and first runs a TLS channel. If the process pooling process has already run a TLS channel, and you want the change to become effective immediately, run the MQSC command **REFRESH SECURITY TYPE(SSL)**. The process pooling process is amqrmppa on UNIX, Linux, and Windows.
- When a **REFRESH SECURITY TYPE(SSL)** command is issued.

z/OS **CFConlos (MQCFIN)**
Specifies the action to be taken when the queue manager loses connectivity to the administration structure, or any CF structure with CFConlos set to ASQMGR (parameter identifier: MQIA_QMGR_CFCONLOS).

The value can be:

MQCFCONLOS_TERMINATE
The queue manager terminates when connectivity to CF structures is lost.

MQCFCONLOS_TOLERATE
The queue manager tolerates loss of connectivity to CF structures without terminating.

This parameter applies to z/OS only.

You can select MQCFCONLOS_TOLERATE only if all the queue managers in the queue-sharing group are at command level 710 or greater and have OPMODE set to NEWFUNC.

ChannelAutoDef (MQCFIN)
Controls whether receiver and server-connection channels can be auto-defined (parameter identifier: MQIA_CHANNEL_AUTO_DEF).

Auto-definition for cluster-sender channels is always enabled.

This parameter is supported in the following environments: IBM i, UNIX, Linux, and Windows systems.

The value can be:

MQCHAD_DISABLED
Channel auto-definition disabled.

MQCHAD_ENABLED
Channel auto-definition enabled.

ChannelAutoDefEvent (MQCFIN)
Controls whether channel auto-definition events are generated (parameter identifier: MQIA_CHANNEL_AUTO_DEF_EVENT), when a receiver, server-connection, or cluster-sender channel is auto-defined.

This parameter is supported in the following environments: IBM i, UNIX, Linux, and Windows systems.

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

ChannelAutoDefExit (MQCFIN)
Channel auto-definition exit name (parameter identifier: MQCA_CHANNEL_AUTO_DEF_EXIT).

This exit is invoked when an inbound request for an undefined channel is received, if:

1. The channel is a cluster-sender, or
2. Channel auto-definition is enabled (see *ChannelAutoDef*).

This exit is also invoked when a cluster-receiver channel is started.

The format of the name is the same as for the *SecurityExit* parameter described in "Change, Copy, and Create Channel" on page 1098.

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

This parameter is supported in the following environments: z/OS, IBM i, UNIX, Linux, and Windows. On z/OS, it applies only to cluster-sender and cluster-receiver channels.

ChannelAuthenticationRecords (MQCFIN)
Controls whether channel authentication records are used. Channel authentication records can still be set and displayed regardless of the value of this attribute. (parameter identifier: MQIA_CHLAUTH_RECORDS).

The value can be:

MQCHLA_DISABLED
Channel authentication records are not checked.

MQCHLA_ENABLED
Channel authentication records are checked.

ChannelEvent (MQCFIN)
Controls whether channel events are generated (parameter identifier: MQIA_CHANNEL_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

MQEVR_EXCEPTION
Reporting of exception channel events enabled.

Multi **ChannelInitiatorControl (MQCFIN)**
Specifies whether the channel initiator is to be started when the queue manager starts (parameter identifier: MQIA_CHINIT_CONTROL).

The value can be:

MQSVC_CONTROL_MANUAL
The channel initiator is not to be started automatically.

MQSVC_CONTROL_Q_MGR
The channel initiator is to be started automatically when the queue manager starts.

This parameter is valid only on Multiplatforms.

ChannelMonitoring (MQCFIN)
Default setting for online monitoring for channels (parameter identifier: MQIA_MONITORING_CHANNEL).

The value can be:

MQMON_NONE
Online monitoring data collection is turned off for channels regardless of the setting of their **ChannelMonitoring** parameter.

MQMON_OFF
Online monitoring data collection is turned off for channels specifying a value of MQMON_Q_MGR in their **ChannelMonitoring** parameter. This value is the initial default value of the queue manager.

MQMON_LOW
Online monitoring data collection is turned on, with a low ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their **ChannelMonitoring** parameter.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their **ChannelMonitoring** parameter.

MQMON_HIGH
Online monitoring data collection is turned on, with a high ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their **ChannelMonitoring** parameter.

ChannelStatistics (MQCFIN)
Controls whether statistics data is to be collected for channels (parameter identifier: MQIA_STATISTICS_CHANNEL).

The value can be:

MQMON_NONE
Statistics data collection is turned off for channels regardless of the setting of their **ChannelStatistics** parameter. This value is the initial default value of the queue manager.

MQMON_OFF
Statistics data collection is turned off for channels specifying a value of MQMON_Q_MGR in their *ChannelStatistics* parameter.

MQMON_LOW
Statistics data collection is turned on, with a low ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their **ChannelStatistics** parameter.

MQMON_MEDIUM
Statistics data collection is turned on, with a moderate ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their **ChannelStatistics** parameter.

MQMON_HIGH
Statistics data collection is turned on, with a high ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their **ChannelStatistics** parameter.

z/OS On z/OS systems, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying LOW, MEDIUM, or HIGH makes no difference to your results.

z/OS **ChinitAdapters (MQCFIN)**
Number of adapter subtasks (parameter identifier: MQIA_CHINIT_ADAPTERS).

The number of adapter subtasks to use for processing IBM MQ calls. This parameter applies to z/OS only.

Specify a value in the range 1 - 9999. The initial default value of the queue manager is 8.

z/OS **ChinitDispatchers (MQCFIN)**
Number of dispatchers (parameter identifier: MQIA_CHINIT_DISPATCHERS).

The number of dispatchers to use for the channel initiator. This parameter applies to z/OS only.

Specify a value in the range 1 - 9999. The initial default value of the queue manager is 5.

z/OS **ChinitServiceParm (MQCFIN)**
Reserved for use by IBM (parameter identifier: MQCA_CHINIT_SERVICE_PARM).

This parameter applies to z/OS only.

z/OS **ChinitTraceAutoStart (MQCFIN)**
Specifies whether the channel initiator trace must start automatically (parameter identifier: MQIA_CHINIT_TRACE_AUTO_START).

The value can be:

MQTRAXSTR_YES
Channel initiator trace is to start automatically.

MQTRAXSTR_NO
Channel initiator trace is not to start automatically. This value is the initial default value of the queue manager.

This parameter applies to z/OS only.

z/OS **ChinitTraceTableSize (MQCFIN)**
The size, in megabytes, of the trace data space of the channel initiator (parameter identifier: MQIA_CHINIT_TRACE_TABLE_SIZE).

Specify a value in the range 2 - 2048. The initial default value of the queue manager is 2.

This parameter applies to z/OS only.

ClusterSenderMonitoringDefault (MQCFIN)
Default setting for online monitoring for automatically defined cluster-sender channels (parameter identifier: MQIA_MONITORING_AUTO_CLUSSDR).

Specifies the value to be used for the *ChannelMonitoring* attribute of automatically defined cluster-sender channels. The value can be any of the following values:

MQMON_Q_MGR
Collection of online monitoring data is inherited from the setting of the queue manager's **ChannelMonitoring** parameter. This value is the initial default value of the queue manager.

MQMON_OFF
Monitoring for the channel is disabled.

MQMON_LOW
Unless *ChannelMonitoring* is MQMON_NONE, this value specifies a low rate of data collection with a minimal effect on system performance. The data collected is not likely to be the most current.

MQMON_MEDIUM
Unless *ChannelMonitoring* is MQMON_NONE, this value specifies a moderate rate of data collection with limited effect on system performance.

MQMON_HIGH
Unless *ChannelMonitoring* is MQMON_NONE, this value specifies a high rate of data collection with a likely effect on system performance. The data collected is the most current available.

z/OS On z/OS systems, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying LOW, MEDIUM, or HIGH makes no difference to your results.

ClusterSenderStatistics (MQCFIN)
Controls whether statistics data is to be collected for auto-defined cluster-sender channels (parameter identifier: MQIA_STATISTICS_AUTO_CLUSSDR).

The value can be:

MQMON_Q_MGR
Collection of statistics data is inherited from the setting of the queue manager's **ChannelStatistics** parameter. This value is the initial default value of the queue manager.

MQMON_OFF
Statistics data collection for the channel is disabled.

MQMON_LOW
Unless *ChannelStatistics* is MQMON_NONE, this value specifies a low rate of data collection with a minimal effect on system performance.

MQMON_MEDIUM
Unless *ChannelStatistics* is MQMON_NONE, this value specifies a moderate rate of data collection.

MQMON_HIGH
Unless *ChannelStatistics* is MQMON_NONE, this value specifies a high rate of data collection.

z/OS On z/OS systems, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying LOW, MEDIUM, or HIGH makes no difference to your results.

ClusterWorkLoadData (MQCFST)
Cluster workload exit data (parameter identifier: MQCA_CLUSTER_WORKLOAD_DATA).

This parameter is passed to the cluster workload exit when it is called.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

ClusterWorkLoadExit (MQCFST)
Cluster workload exit name (parameter identifier: MQCA_CLUSTER_WORKLOAD_EXIT).

If a nonblank name is defined this exit is invoked when a message is put to a cluster queue.

The format of the name is the same as for the *SecurityExit* parameter described in "Change, Copy, and Create Channel" on page 1098.

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

ClusterWorkLoadLength (MQCFIN)
Cluster workload length (parameter identifier: MQIA_CLUSTER_WORKLOAD_LENGTH).

The maximum length of the message passed to the cluster workload exit.

The value of this attribute must be in the range 0 - 999,999 999.

CLWLMRUChannels (MQCFIN)
Cluster workload most recently used (MRU) channels (parameter identifier: MQIA_CLWL_MRU_CHANNELS).

The maximum number of active most recently used outbound channels.

Specify a value in the range 1 - 999,999 999.

CLWLUseQ (MQCFIN)
Use of remote queue (parameter identifier: MQIA_CLWL_USEQ).

Specifies whether a cluster queue manager is to use remote puts to other queues defined in other queue managers within the cluster during workload management.

Specify either:

MQCLWL_USEQ_ANY
Use remote queues.

MQCLWL_USEQ_LOCAL
Do not use remote queues.

CodedCharSetId (MQCFIN)
Queue manager coded character set identifier (parameter identifier: MQIA_CODED_CHAR_SET_ID).

The coded character set identifier (CCSID) for the queue manager. The CCSID is the identifier used with all character string fields defined by the application programming interface (API). If the CCSID in a message descriptor is set to the value MQCCSI_Q_MGR, it applies to the character data written into the body of a message. Data is written using MQPUT or MQPUT1. Character data is identified by the format specified for the message.

Specify a value in the range 1 - 65,535.

The CCSID must specify a value that is defined for use on the platform and use an appropriate character set. The character set must be:

- EBCDIC on IBM i
- ASCII or ASCII-related on other platforms

Stop and restart the queue manager after execution of this command so that all processes reflect the changed CCSID of the queue manager.

This parameter is not supported on z/OS.

CommandEvent (MQCFIN)
Controls whether command events are generated (parameter identifier: MQIA_COMMAND_EVENT).

The value can be any of the following values:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

MQEVR_NO_DISPLAY
Event reporting enabled for all successful commands except Inquire commands.

z/OS **CommandScope (MQCFIN)**

Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following values:

- Blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- A queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment. The command server must be enabled.
- An asterisk "*". The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Multi **CommandServerControl (MQCFIN)**

Specifies whether the command server is to be started when the queue manager starts (parameter identifier: MQIA_CMD_SERVER_CONTROL).

The value can be:

MQSVC_CONTROL_MANUAL
The command server is not to be started automatically.

MQSVC_CONTROL_Q_MGR
The command server is to be started automatically when the queue manager starts.

This parameter is valid only on Multiplatforms.

ConfigurationEvent (MQCFIN)

Controls whether configuration events are generated (parameter identifier: MQIA_CONFIGURATION_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

ConnAuth (MQCFST)

The name of an authentication information object that is used to provide the location of user ID and password authentication (parameter identifier: MQCA_CONN_AUTH).

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH. Only authentication information objects with type IDPWOS or IDPWLDAP can be specified; other types result in an error message when the OAM (on UNIX, Linux, and Windows) or the security component (on z/OS) reads the configuration.

Custom (MQCFST)

Custom attribute for new features (parameter identifier: MQCA_CUSTOM).

This attribute is reserved for the configuration of new features before separate attributes are introduced. It can contain the values of zero or more attributes as pairs of attribute name and value, separated by at least one space. The attribute name-value pairs have the form NAME(VALUE). Single quotation marks must be escaped with another single quotation mark.

This description is updated when features using this attribute are introduced. Currently there are no possible values for *Custom*.

The maximum length of the string is MQ_CUSTOM_LENGTH.

DeadLetterQName (MQCFIN)
Dead letter (undelivered message) queue name (parameter identifier: MQCA_DEAD_LETTER_Q_NAME).

Specifies the name of the local queue that is to be used for undelivered messages. Messages are put on this queue if they cannot be routed to their correct destination. The maximum length of the string is MQ_Q_NAME_LENGTH.

DefClusterXmitQueueType (MQCFIN)
The DefClusterXmitQueueType attribute controls which transmission queue is selected by default by cluster-sender channels to get messages from, to send the messages to cluster-receiver channels. (Parameter identifier: MQIA_DEF_CLUSTER_XMIT_Q_TYPE.)

The values of DefClusterXmitQueueType are MQCLXQ_SCTQ or MQCLXQ_CHANNEL.

MQCLXQ_SCTQ

All cluster-sender channels send messages from SYSTEM.CLUSTER.TRANSMIT.QUEUE. The correlID of messages placed on the transmission queue identifies which cluster-sender channel the message is destined for.

SCTQ is set when a queue manager is defined. This behavior is implicit in versions of IBM WebSphere MQ, earlier than Version 7.5. In earlier versions, the queue manager attribute DefClusterXmitQueueType was not present.

MQCLXQ_CHANNEL
Each cluster-sender channel sends messages from a different transmission queue. Each transmission queue is created as a permanent dynamic queue from the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE.

DefXmitQName (MQCFST)
Default transmission queue name (parameter identifier: MQCA_DEF_XMIT_Q_NAME).

This parameter is the name of the default transmission queue that is used for the transmission of messages to remote queue managers. It is selected if there is no other indication of which transmission queue to use.

The maximum length of the string is MQ_Q_NAME_LENGTH.

DNSGroup (MQCFST)
DNS group name (parameter identifier: MQCA_DNS_GROUP).

This parameter is no longer used. Refer to WLM/DNS no longer supported. This parameter applies to z/OS only.

The maximum length of the string is MQ_DNS_GROUP_NAME_LENGTH.

z/OS **DNSWLM (MQCFIN)**
WLM/DNS Control: (parameter identifier: MQIA_DNS_WLM).

This parameter is no longer used. Refer to WLM/DNS no longer supported.

The value can be any of the following values:

MQDNSWLM_NO
This is the only value supported by the queue manager.

This parameter applies to z/OS only.

z/OS **ExpiryInterval (MQCFIN)**
Interval between scans for expired messages (parameter identifier: MQIA_EXPIRY_INTERVAL). This parameter applies to z/OS only.

Specifies the frequency with which the queue manager scans the queues looking for expired messages. Specify a time interval in seconds in the range 1 - 99,999,999, or the following special value:

MQEXPI_OFF
No scans for expired messages.

The minimum scan interval used is 5 seconds, even if you specify a lower value.

EncryptionPolicySuiteB (MQCFIL)
Specifies whether Suite B-compliant cryptography is used and what level of strength is employed (parameter identifier MQIA_SUITE_B_STRENGTH).

The value can be one or more of:

MQ_SUITE_B_NONE
Suite B-compliant cryptography is not used.

MQ_SUITE_B_128_BIT
Suite B 128-bit strength security is used.

MQ_SUITE_B_192_BIT
Suite B 192-bit strength security is used.

If invalid lists are specified, such as MQ_SUITE_B_NONE with MQ_SUITE_B_128_BIT, the error MQRCCF_SUITE_B_ERROR is issued.

Force (MQCFIN)
Force changes (parameter identifier: MQIACF_FORCE).

Specifies whether the command is forced to complete if both of the following are true:

- *DefXmitQName* is specified, and
- An application has a remote queue open, the resolution for which is affected by this change.

z/OS **GroupUR (MQCFIN)**
Controls whether CICS and XA client applications can establish transactions with a GROUP unit of recovery disposition.

This attribute is only valid on z/OS and can be enabled only when the queue manager is a member of a queue-sharing group.

The value can be:

MQGUR_DISABLED
CICS and XA client applications must connect using a queue manager name.

MQGUR_ENABLED
CICS and XA client applications can establish transactions with a group unit of recovery disposition by specifying a queue-sharing group name when they connect.

z/OS See Unit of recovery disposition in a queue-sharing group.

z/OS **IGQPutAuthority (MQCFIN)**
Command scope (parameter identifier: MQIA_IGQ_PUT_AUTHORITY). This parameter is valid only on z/OS when the queue manager is a member of a queue-sharing group.

Specifies the type of authority checking and, therefore, the user IDs to be used by the IGQ agent (IGQA). This parameter establishes the authority to put messages to a destination queue. The value can be any of the following values:

MQIGQPA_DEFAULT
Default user identifier is used.

The user identifier used for authorization is the value of the *UserIdentifier* field. The *UserIdentifier* field is in the separate MQMD that is associated with the message when the message is on the shared transmission queue. This value is the user identifier of the program

that placed the message on the shared transmission queue. It is typically the same as the user identifier under which the remote queue manager is running.

If the RESLEVEL profile indicates that more than one user identifier is to be checked, the user identifier of the local IGQ agent (*IGQUserId*) is checked.

MQIGQPA_CONTEXT
Context user identifier is used.

The user identifier used for authorization is the value of the *UserIdentifier* field. The *UserIdentifier* field is in the separate MQMD that is associated with the message when the message is on the shared transmission queue. This value is the user identifier of the program that placed the message on the shared transmission queue. It is typically the same as the user identifier under which the remote queue manager is running.

If the RESLEVEL profile indicates that more than one user identifier is to be checked, the user identifier of the local IGQ agent (*IGQUserId*) is checked.. The value of the *UserIdentifier* field in the embedded MQMD is also checked. The latter user identifier is typically the user identifier of the application that originated the message.

MQIGQPA_ONLY_IGQ
Only the IGQ user identifier is used.

The user identifier used for authorization is the user identifier of the local IGQ agent (*IGQUserId*).

If the RESLEVEL profile indicates that more than one user identifier is to be checked, this user identifier is used for all checks.

MQIGQPA_ALTERNATE_OR_IGQ
Alternate user identifier or IGQ-agent user identifier is used.

The user identifier used for authorization is the user identifier of the local IGQ agent (*IGQUserId*).

If the RESLEVEL profile indicates that more than one user identifier is to be checked, the value of the *UserIdentifier* field in the embedded MQMD is also checked. The latter user identifier is typically the user identifier of the application that originated the message.

z/OS **IGQUserId (MQCFST)**
Intra-group queuing agent user identifier (parameter identifier: MQCA_IGQ_USER_ID). This parameter is valid only on z/OS when the queue manager is a member of a queue-sharing group.

Specifies the user identifier that is associated with the local intra-group queuing agent. This identifier is one of the user identifiers that might be checked for authorization when the IGQ agent puts messages on local queues. The actual user identifiers checked depend on the setting of the *IGQPutAuthority* attribute, and on external security options.

The maximum length is MQ_USER_ID_LENGTH.

V 9.0.2 **ImageInterval (MQCFIN)**
The target frequency with which the queue manager automatically writes media images, in minutes since the previous media image for an object (parameter identifier: MQIA_MEDIA_IMAGE_INTERVAL). This parameter is not valid on z/OS.

The value can be:

The time in minutes from 1 - 999 999 999, at which the queue manager automatically writes media images.

The default value is *60* minutes.

MQMEDIMGINTVL_OFF
Automatic media images are not written on a time interval basis.

V 9.0.2 **ImageLogLength (MQCFIN)**

The target size of the recovery log, written before the queue manager automatically writes media images, in number of megabytes since the previous media image for an object. This limits the amount of log to be read when recovering an object (parameter identifier: MQIA_MEDIA_IMAGE_LOG_LENGTH). This parameter is not valid on z/OS.

The value can be:

The target size of the recovery log in megabytes from 1 - 999 999 999.

MQMEDIMGLOGLN_OFF
Automatic media images are not written based on the size of log written.

MQMEDIMGLOGLN_OFF is the default value.

V 9.0.2 **ImageRecoverObject (MQCFST)**

Specifies whether authentication information, channel, client connection, listener, namelist, process, alias queue, remote queue, and service objects are recoverable from a media image, if linear logging is being used (parameter identifier: MQIA_MEDIA_IMAGE_RECOVER_OBJ). This parameter is not valid on z/OS.

The value can be:

MQIMGRCOV_NO
The "rcdmqimg (record media image)" on page 283 and"rcrmqobj (re-create object)" on page 289 commands are not permitted for these objects, and automatic media images, if enabled, are not written for these objects.

MQIMGRCOV_YES
These objects are recoverable.

MQIMGRCOV_YES is the default value.

V 9.0.2 **ImageRecoverQueue (MQCFST)**

Specifies the default **ImageRecoverQueue** attribute for local and permanent dynamic queue objects, when used with this parameter (parameter identifier: MQIA_MEDIA_IMAGE_RECOVER_Q). This parameter is not valid on z/OS.

The value can be:

MQIMGRCOV_NO
The **ImageRecoverQueue** attribute for local and permanent dynamic queue objects is set to MQIMGRCOV_NO .

MQIMGRCOV_YES
The **ImageRecoverQueue** attribute for local and permanent dynamic queue objects is set to MQIMGRCOV_YES .

MQIMGRCOV_YES is the default value.

V 9.0.2 **ImageSchedule (MQCFST)**

Whether the queue manager automatically writes media images (parameter identifier: MQIA_MEDIA_IMAGE_SCHEDUING). This parameter is not valid on z/OS.

The value can be:

MQMEDIMGSCHED_AUTO
The queue manager attempts to automatically write a media image for an object, before **ImageInterval** minutes have elapsed, or **ImageLogLength** megabytes of recovery log have been written, since the previous media image for the object was taken.

The previous media image might have been taken manually or automatically, depending on the settings of **ImageInterval** or **ImageLogLength**.

MQMEDIMGSCHED_MANUAL
Automatic media images are not written.

MQMEDIMGSCHED_MANUAL is the default value.

InhibitEvent (MQCFIN)
Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated (parameter identifier: MQIA_INHIBIT_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

z/OS **IntraGroupqueuing (MQCFIN)**
Command scope (parameter identifier: MQIA_INTRA_GROUP_QUEUING). This parameter is valid only on z/OS when the queue manager is a member of a queue-sharing group.

Specifies whether intra-group queuing is used. The value can be any of the following values:

MQIGQ_DISABLED
Intra-group queuing disabled.

MQIGQ_ENABLED
Intra-group queuing enabled.

IPAddressVersion (MQCFIN)
IP address version selector (parameter identifier: MQIA_IP_ADDRESS_VERSION).

Specifies which IP address version, either IPv4 or IPv6, is used. The value can be:

MQIPADDR_IPv4
IPv4 is used.

MQIPADDR_IPv6
IPv6 is used.

This parameter is only relevant for systems that run both IPv4 and IPv6. It affects only channels defined as having a *TransportType* of MQXPY_TCP when one of the following conditions is true:

- The channel attribute *ConnectionName* is a host name that resolves to both an IPv4 and IPv6 address and its **LocalAddress** parameter is not specified.
- The channel attributes *ConnectionName* and *LocalAddress* are both host names that resolve to both IPv4 and IPv6 addresses.

z/OS **ListenerTimer (MQCFIN)**
Listener restart interval (parameter identifier: MQIA_LISTENER_TIMER).

The time interval, in seconds, between attempts by IBM MQ to restart the listener after an APPC or TCP/IP failure. This parameter applies to z/OS only.

Specify a value in the range 5 - 9,999. The initial default value of the queue manager is 60.

LocalEvent (MQCFIN)
Controls whether local error events are generated (parameter identifier: MQIA_LOCAL_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

Multi **LoggerEvent (MQCFIN)**
Controls whether recovery log events are generated (parameter identifier: MQIA_LOGGER_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled. This value is valid only on queue managers that use linear logging.

This parameter is valid only on Multiplatforms.

z/OS **LUGroupName (MQCFST)**
Generic LU name for the LU 6.2 listener (parameter identifier: MQCA_LU_GROUP_NAME).

The generic LU name to be used by the LU 6.2 listener that handles inbound transmissions for the queue-sharing group.

This parameter applies to z/OS only.

The maximum length of the string is MQ_LU_NAME_LENGTH.

z/OS **LUName (MQCFST)**
LU name to use for outbound LU 6.2 transmissions (parameter identifier: MQCA_LU_NAME).

The name of the LU to use for outbound LU 6.2 transmissions. Set this parameter to be the same as the name of the LU to be used by the listener for inbound transmissions.

This parameter applies to z/OS only.

The maximum length of the string is MQ_LU_NAME_LENGTH.

z/OS **LU62ARMSuffix (MQCFST)**
APPCPM suffix (parameter identifier: MQCA_LU62_ARM_SUFFIX).

The suffix of the APPCPM member of SYS1.PARMLIB. This suffix nominates the LUADD for this channel initiator.

This parameter applies to z/OS only.

The maximum length of the string is MQ_ARM_SUFFIX_LENGTH.

z/OS **LU62Channels (MQCFIN)**
Maximum number of LU 6.2 channels (parameter identifier: MQIA_LU62_CHANNELS).

The maximum number of channels that can be current, or clients that can be connected, that use the LU 6.2 transmission protocol.

This parameter applies to z/OS only.

Specify a value in the range 0 - 9999. The initial default value of the queue manager is 200.

z/OS **MaxActiveChannels (MQCFIN)**
Maximum number of active channels (parameter identifier: MQIA_ACTIVE_CHANNELS).

The maximum number of channels that can be *active* at any time.

This parameter applies to z/OS only.

Sharing conversations do not contribute to the total for this parameter.

Specify a value in the range 1 - 9999. The initial default value of the queue manager is 200.

z/OS **MaxChannels (MQCFIN)**
Maximum number of current channels (parameter identifier: MQIA_MAX_CHANNELS).

The maximum number of channels that can be *current* (including server-connection channels with connected clients).

This parameter applies to z/OS only.

Sharing conversations do not contribute to the total for this parameter.

Specify a value in the range 1 - 9999.

MaxHandles (MQCFIN)
Maximum number of handles (parameter identifier: MQIA_MAX_HANDLES).

The maximum number of handles that any one connection can have open at the same time.

Specify a value in the range 0 - 999,999,999.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

Specifies the maximum length of messages allowed on queues on the queue manager. No message that is larger than either the queue attribute *MaxMsgLength* or the queue manager attribute *MaxMsgLength* can be put on a queue.

If you reduce the maximum message length for the queue manager, you must also reduce the maximum message length of the SYSTEM.DEFAULT.LOCAL.QUEUE definition, and your other queues. Reduce the definitions on the queues to less than or equal to the limit of the queue manager. If you do not reduce the message lengths appropriately, and applications inquire only the value of the queue attribute *MaxMsgLength*, they might not work correctly.

The lower limit for this parameter is 32 KB (32,768 bytes). The upper limit is 100 MB (104,857,600 bytes). This parameter is not valid on z/OS.

MaxPropertiesLength (MQCFIN)
Maximum property length (parameter identifier: MQIA_MAX_PROPERTIES_LENGTH).

Specifies the maximum length of the properties, including both the property name in bytes and the size of the property value in bytes.

Specify a value in the range 0 - 100 MB (104,857,600 bytes), or the special value:

MQPROP_UNRESTRICTED_LENGTH
The size of the properties is restricted only by the upper limit.

MaxUncommittedMsgs (MQCFIN)
Maximum uncommitted messages (parameter identifier: MQIA_MAX_UNCOMMITTED_MSGS).

Specifies the maximum number of uncommitted messages. The maximum number of uncommitted messages under any sync point is the sum of the following messages:

The number of messages that can be retrieved.

The number of messages that can be put.

The number of trigger messages generated within this unit of work.

The limit does not apply to messages that are retrieved or put outside sync point.

Specify a value in the range 1 - 10,000.

Multi **MQIAccounting (MQCFIN)**
Controls whether accounting information for MQI data is to be collected (parameter identifier: MQIA_ACCOUNTING_MQI).

The value can be:

MQMON_OFF
MQI accounting data collection is disabled. This value is the initial default value of the queue manager.

MQMON_ON
MQI accounting data collection is enabled.

This parameter is valid only on Multiplatforms.

Multi **MQIStatistics (MQCFIN)**
Controls whether statistics monitoring data is to be collected for the queue manager (parameter identifier: MQIA_STATISTICS_MQI).

The value can be:

MQMON_OFF
Data collection for MQI statistics is disabled. This value is the initial default value of the queue manager.

MQMON_ON
Data collection for MQI statistics is enabled.

This parameter is valid only on Multiplatforms.

MsgMarkBrowseInterval (MQCFIN)
Mark-browse interval (parameter identifier: MQIA_MSG_MARK_BROWSE_INTERVAL).

Specifies the time interval in milliseconds after which the queue manager can automatically unmark messages.

Specify a value in the range 0 - 999,999,999, or the special value MQMMBI_UNLIMITED.

A value of 0 causes the queue manager to unmark messages immediately.

MQMMBI_UNLIMITED indicates that the queue manager does not automatically unmark messages.

z/OS **OutboundPortMax (MQCFIN)**
The maximum value in the range for the binding of outgoing channels (parameter identifier: MQIA_OUTBOUND_PORT_MAX).

The maximum value in the range of port numbers to be used when binding outgoing channels. This parameter applies to z/OS only.

Specify a value in the range 0 - 65,535. The initial default value of the queue manager is zero.

Specify a corresponding value for *OutboundPortMin* and ensure that the value of *OutboundPortMax* is greater than or equal to the value of *OutboundPortMin*.

z/OS **OutboundPortMin (MQCFIN)**
The minimum value in the range for the binding of outgoing channels (parameter identifier: MQIA_OUTBOUND_PORT_MIN).

The minimum value in the range of port numbers to be used when binding outgoing channels. This parameter applies to z/OS only.

Specify a value in the range 0 - 65,535. The initial default value of the queue manager is zero.

Specify a corresponding value for *OutboundPortMax* and ensure that the value of *OutboundPortMin* is less than or equal to the value of *OutboundPortMax*.

Parent (MQCFST)
The name of the queue manager to which this queue manager is to connect hierarchically as its child (parameter identifier: MQCA_PARENT).

A blank value indicates that this queue manager has no parent queue manager. If there is an existing parent queue manager it is disconnected. This value is the initial default value of the queue manager.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Note:

- The use of IBM MQ hierarchical connections requires that the queue manager attribute PSMode is set to MQPSM_ENABLED.
- The value of *Parent* can be set to a blank value if PSMode is set to MQPSM_DISABLED.
- Before connecting to a queue manager hierarchically as its child, channels in both directions must exist between the parent queue manager and child queue manager.
- If a parent is defined, the **Change Queue Manager** command disconnects from the original parent and sends a connection flow to the new parent queue manager.
- Successful completion of the command does not mean that the action completed or that it is going to complete successfully. Use the **Inquire Pub/Sub Status** command to track the status of the requested parent relationship.

PerformanceEvent (MQCFIN)
Controls whether performance-related events are generated (parameter identifier: MQIA_PERFORMANCE_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

PubSubClus (MQCFIN)
Controls whether the queue manager participates in publish/subscribe clustering (parameter identifier: MQIA_PUBSUB_CLUSTER).

The value can be:

MQPSCLUS_ENABLED
The creating or receipt of clustered topic definitions and cluster subscriptions is permitted.

Note: The introduction of a clustered topic into a large IBM MQ cluster can cause a degradation in performance. This degradation occurs because all partial repositories are notified of all the other members of the cluster. Unexpected subscriptions might be created at all other nodes; for example; where proxysub(FORCE) is specified. Large numbers of channels might be started from a queue manager; for example, on resync after a queue manager failure.

MQPSCLUS_DISABLED
The creating or receipt of clustered topic definitions and cluster subscriptions is inhibited. The creations or receipts are recorded as warnings in the queue manager error logs.

PubSubMaxMsgRetryCount (MQCFIN)
The number of attempts to reprocess a message when processing a failed command message under sync point (parameter identifier: MQIA_PUBSUB_MAXMSG_RETRY_COUNT).

The value can be:

0 to 999 999 999
The initial value is 5.

PubSubMode (MQCFIN)
Specifies whether the publish/subscribe engine and the queued publish/subscribe interface are running. The publish/subscribe engine enables applications to publish or subscribe by using the application programming interface. The publish/subscribe interface monitors the queues used the queued publish/subscribe interface (parameter identifier: MQIA_PUBSUB_MODE).

The value can be:

MQPSM_COMPAT
The publish/subscribe engine is running. It is therefore possible to publish or subscribe by using the application programming interface. The queued publish/subscribe interface is not running. Therefore any message that is put to the queues that are monitored by the queued publish/subscribe interface is not acted on. MQPSM_COMPAT is used for compatibility with versions of IBM Integration Bus (formerly known as WebSphere Message Broker) prior to version 7 that use this queue manager.

MQPSM_DISABLED
The publish/subscribe engine and the queued publish/subscribe interface are not running. It is therefore not possible to publish or subscribe using the application programming interface. Any publish/subscribe messages that are put to the queues that are monitored by the queued publish/subscribe interface are not acted on.

MQPSM_ENABLED
The publish/subscribe engine and the queued publish/subscribe interface are running. It is therefore possible to publish or subscribe by using the application programming interface and the queues that are monitored by the queued publish/subscribe interface. This value is the initial default value of the queue manager.

PubSubNPInputMsg (MQCFIN)
Whether to discard (or keep) an undelivered input message (parameter identifier: MQIA_PUBSUB_NP_MSG).

The value can be:

MQUNDELIVERED_DISCARD
Non-persistent input messages are discarded if they cannot be processed.

MQUNDELIVERED_KEEP
Non-persistent input messages are not discarded if they cannot be processed. In this situation, the queued publish/subscribe interface continues to try the process again at appropriate intervals and does not continue processing subsequent messages.

PubSubNPResponse (MQCFIN)
Controls the behavior of undelivered response messages (parameter identifier: MQIA_PUBSUB_NP_RESP).

The value can be:

MQUNDELIVERED_NORMAL
Non-persistent responses that cannot be placed on the reply queue are put on the dead letter queue. If they cannot be placed on the dead letter queue they are discarded.

MQUNDELIVERED_SAFE
Non-persistent responses that cannot be placed on the reply queue are put on the dead letter queue. If the response cannot be sent and cannot be placed on the dead letter queue the queued publish/subscribe interface rolls back the current operation. The operation is tried again at appropriate intervals and does not continue processing subsequent messages.

MQUNDELIVERED_DISCARD
Non-persistent responses that are not placed on the reply queue are discarded.

MQUNDELIVERED_KEEP
Non-persistent responses are not placed on the dead letter queue or discarded. Instead, the queued publish/subscribe interface backs out the current operation and then try it again at appropriate intervals.

PubSubSyncPoint (MQCFIN)
Whether only persistent (or all) messages must be processed under sync point (parameter identifier: MQIA_PUBSUB_SYNC_PT).

The value can be:

MQSYNCPOINT_IFPER
This value makes the queued publish/subscribe interface receive non-persistent messages outside sync point. If the interface receives a publication outside sync point, the interface forwards the publication to subscribers known to it outside sync point.

MQSYNCPOINT_YES
This value makes the queued publish/subscribe interface receive all messages under sync point.

QMgrDesc (MQCFST)
Queue manager description (parameter identifier: MQCA_Q_MGR_DESC).

This parameter is text that briefly describes the object.

The maximum length of the string is MQ_Q_MGR_DESC_LENGTH.

Use characters from the character set identified by the coded character set identifier (CCSID) for the queue manager on which the command is executing. Using this character set ensures that the text is translated correctly.

z/OS **QSGCertificateLabel (MQCFST)**
Specifies the certificate label for the queue-sharing group to use (parameter identifier: MQCA_QSG_CERT_LABEL).

This parameter takes precedence over **CERTLABL** in the event that the queue manager is a member of a QSG.

QueueAccounting (MQCFIN)
Controls the collection of accounting (thread-level and queue-level accounting) data for queues (parameter identifier: MQIA_ACCOUNTING_Q).

The value can be:

MQMON_NONE
Accounting data collection for queues is disabled. This value must not be overridden by the value of the **QueueAccounting** parameter on the queue.

MQMON_OFF
Accounting data collection is disabled for queues specifying a value of MQMON_Q_MGR in the **QueueAccounting** parameter.

MQMON_ON
Accounting data collection is enabled for queues specifying a value of MQMON_Q_MGR in the **QueueAccounting** parameter.

QueueMonitoring (MQCFIN)
Default setting for online monitoring for queues (parameter identifier: MQIA_MONITORING_Q).

If the **QueueMonitoring** queue attribute is set to MQMON_Q_MGR, this attribute specifies the value which is assumed by the channel. The value can be any of the following values:

MQMON_OFF
Online monitoring data collection is turned off. This value is the initial default value of the queue manager.

MQMON_NONE
Online monitoring data collection is turned off for queues regardless of the setting of their **QueueMonitoring** attribute.

MQMON_LOW
Online monitoring data collection is turned on, with a low ratio of data collection.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate ratio of data collection.

MQMON_HIGH
Online monitoring data collection is turned on, with a high ratio of data collection.

Multi **QueueStatistics (MQCFIN)**
Controls whether statistics data is to be collected for queues (parameter identifier: MQIA_STATISTICS_Q).

The value can be:

MQMON_NONE
Statistics data collection is turned off for queues regardless of the setting of their **QueueStatistics** parameter. This value is the initial default value of the queue manager.

MQMON_OFF
Statistics data collection is turned off for queues specifying a value of MQMON_Q_MGR in their **QueueStatistics** parameter.

MQMON_ON
Statistics data collection is turned on for queues specifying a value of MQMON_Q_MGR in their **QueueStatistics** parameter.

This parameter is valid only on Multiplatforms.

z/OS **ReceiveTimeout (MQCFIN)**
How long a TCP/IP channel waits to receive data from its partner (parameter identifier: MQIA_RECEIVE_TIMEOUT).

The approximate length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to the inactive state.

This parameter applies to z/OS only. It applies to message channels, and not to MQI channels. This number can be qualified as follows:

- This number is a multiplier to be applied to the negotiated *HeartBeatInterval* value to determine how long a channel is to wait. Set *ReceiveTimeoutType* to MQRCVTIME_MULTIPLY. Specify a value of zero or in the range 2 - 99. If you specify zero, the channel waits indefinitely to receive data from its partner.
- This number is a value, in seconds, to be added to the negotiated *HeartBeatInterval* value to determine how long a channel is to wait. Set *ReceiveTimeoutType* to MQRCVTIME_ADD. Specify a value in the range 1 - 999,999.
- Tthis number is a value, in seconds, that the channel is to wait, set *ReceiveTimeoutType* to MQRCVTIME_EQUAL. Specify a value in the range 0 - 999,999. If you specify 0, the channel waits indefinitely to receive data from its partner.

The initial default value of the queue manager is zero.

z/OS **ReceiveTimeoutMin (MQCFIN)**
The minimum length of time that a TCP/IP channel waits to receive data from its partner (parameter identifier: MQIA_RECEIVE_TIMEOUT_MIN).

The minimum length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to the inactive state. This parameter applies to z/OS only.

Specify a value in the range 0 - 999,999.

z/OS **ReceiveTimeoutType (MQCFIN)**
The qualifier to apply to *ReceiveTimeout* (parameter identifier: MQIA_RECEIVE_TIMEOUT_TYPE).

The qualifier to apply to *ReceiveTimeoutType* to calculate how long a TCP/IP channel waits to receive data, including heartbeats, from its partner. It waits to receive data before returning to the inactive state. This parameter applies to z/OS only.

The value can be any of the following values:

MQRCVTIME_MULTIPLY
The *ReceiveTimeout* value is a multiplier to be applied to the negotiated value of *HeartbeatInterval* to determine how long a channel waits. This value is the initial default value of the queue manager.

MQRCVTIME_ADD
ReceiveTimeout is a value, in seconds, to be added to the negotiated value of *HeartbeatInterval* to determine how long a channel waits.

MQRCVTIME_EQUAL
ReceiveTimeout is a value, in seconds, representing how long a channel waits.

RemoteEvent (MQCFIN)
Controls whether remote error events are generated (parameter identifier: MQIA_REMOTE_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

RepositoryName (MQCFST)
Cluster name (parameter identifier: MQCA_REPOSITORY_NAME).

The name of a cluster for which this queue manager provides a repository manager service.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

No more than one of the resultant values of *RepositoryName* can be nonblank.

RepositoryNamelist (MQCFST)
Repository namelist (parameter identifier: MQCA_REPOSITORY_NAMELIST).

The name, of a namelist of clusters, for which this queue manager provides a repository manager service.

This queue manager does not have a full repository, but can be a client of other repository services that are defined in the cluster, if

- Both *RepositoryName* and *RepositoryNamelist* are blank, or
- *RepositoryName* is blank and the namelist specified by *RepositoryNamelist* is empty.

No more than one of the resultant values of *RepositoryNameList* can be nonblank.

RevDns (MQCFIN)
Whether reverse lookup of the host name from a Domain Name Server is carried out. (parameter identifier: MQIA_REVERSE_DNS_LOOKUP).

This attribute has an effect only on channels using a transport type (TRPTYPE) of TCP.

The value can be:

MQRDNS_DISABLED
DNS host names are not reverse looked-up for the IP addresses of inbound channels. With this setting any CHLAUTH rules using host names are not matched.

MQRDNS_ENABLED
DNS host names are reverse looked-up for the IP addresses of inbound channels when this information is required. This setting is required for matching against CHLAUTH rules that contain host names, and for writing out error messages.

z/OS **SecurityCase (MQCFIN)**
Security case supported (parameter identifier: MQIA_SECURITY_CASE).

Specifies whether the queue manager supports security profile names in mixed case, or in uppercase only. The value is activated when a Refresh Security command is run with *SecurityType(MQSECTYPE_CLASSES)* specified. This parameter is valid only on z/OS.

The value can be:

MQSCYC_UPPER
Security profile names must be in uppercase.

MQSCYC_MIXED
Security profile names can be in uppercase or in mixed case.

z/OS **SharedQQmgrName (MQCFIN)**
Shared-queue queue manager name (parameter identifier: MQIA_SHARED_Q_Q_MGR_NAME).

A queue manager makes an MQOPEN call for a shared queue. The queue manager that is specified in the **ObjectQmgrName** parameter of the MQOPEN call is in the same queue-sharing group as the processing queue manager. The SQQMNAME attribute specifies whether the **ObjectQmgrName** is used or whether the processing queue manager opens the shared queue directly. This parameter is valid only on z/OS.

The value can be any of the following values:

MQSQQM_USE
ObjectQmgrName is used and the appropriate transmission queue is opened.

MQSQQM_IGNORE
The processing queue manager opens the shared queue directly. This value can reduce the traffic in your queue manager network.

SSLCRLNamelist (MQCFST)
The TLS namelist (parameter identifier: MQCA_SSL_CRL_NAMELIST).

The length of the string is MQ_NAMELIST_NAME_LENGTH.

Indicates the name of a namelist of authentication information objects which are used to provide certificate revocation locations to allow enhanced TLS certificate checking.

If *SSLCRLNamelist* is blank, certificate revocation checking is not invoked.

Changes to *SSLCRLNamelist*, or to the names in a previously specified namelist, or to previously referenced authentication information objects become effective:

- Multi On Multiplatforms, when a new channel process is started.
- Multi For channels that run as threads of the channel initiator on Multiplatforms, when the channel initiator is restarted.
- Multi For channels that run as threads of the listener on Multiplatforms, when the listener is restarted.
- z/OS On z/OS, when the channel initiator is restarted.
- When a **REFRESH SECURITY TYPE(SSL)** command is issued.
- IBM i On IBM i queue managers, this parameter is ignored. However, it is used to determine which authentication information objects are written to the AMQCLCHL.TAB file.

Only authentication information objects with types of LDAPCRL or OCSP are allowed in the namelist referred to by *SSLCRLNamelist* (MQCFST). Any other type results in an error message when the list is processed and is subsequently ignored.

SSLCryptoHardware (MQCFST)
The TLS cryptographic hardware (parameter identifier: MQCA_SSL_CRYPTO_HARDWARE).

The length of the string is MQ_SSL_CRYPTO_HARDWARE_LENGTH.

Sets the name of the parameter string required to configure the cryptographic hardware present on the system.

This parameter is valid only on UNIX, Linux, and Windows.

All supported cryptographic hardware supports the PKCS #11 interface. Specify a string of the following format:

GSK_PKCS11=*PKCS_#11_driver_path_and_file_name*;*PKCS_#11_token_label*;*PKCS_#11_token_password*;*symmetric_cipher_setting*;

The PKCS #11 driver path is an absolute path to the shared library providing support for the PKCS #11 card. The PKCS #11 driver file name is the name of the shared library. An example of the value required for the PKCS #11 driver path and file name is /usr/lib/pkcs11/PKCS11_API.so

To access symmetric cipher operations through GSKit, specify the symmetric cipher setting parameter. The value of this parameter is either:

SYMMETRIC_CIPHER_OFF
Do not access symmetric cipher operations.

SYMMETRIC_CIPHER_ON
Access symmetric cipher operations.

If the symmetric cipher setting is not specified, this value has the same effect as specifying SYMMETRIC_CIPHER_OFF.

The maximum length of the string is 256 characters. The default value is blank.

If you specify a string in the wrong format, you get an error.

When the *SSLCryptoHardware* (MQCFST) value is changed, the cryptographic hardware parameters specified become the ones used for new TLS connection environments. The new information becomes effective:

- When a new channel process is started.
- For channels that run as threads of the channel initiator, when the channel initiator is restarted.
- For channels that run as threads of the listener, when the listener is restarted.
- When a Refresh Security command is issued to refresh the contents of the TLS key repository.

SSLEvent (MQCFIN)
Controls whether TLS events are generated (parameter identifier: MQIA_SSL_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

SSLFipsRequired (MQCFIN)
SSLFIPS specifies whether only FIPS-certified algorithms are to be used if cryptography is carried out in IBM MQ, rather than in cryptographic hardware (parameter identifier: MQIA_SSL_FIPS_REQUIRED).

If cryptographic hardware is configured, the cryptographic modules used are those modules provided by the hardware product. These modules might, or might not, be FIPS-certified to a particular level depending on the hardware product in use. This parameter applies to z/OS, UNIX, Linux, and Windows platforms only.

The value can be any of the following values:

MQSSL_FIPS_NO
IBM MQ provides an implementation of TLS cryptography which supplies some

FIPS-certified modules on some platforms. If you set *SSLFIPSRequired* to MQSSL_FIPS_NO, any CipherSpec supported on a particular platform can be used. This value is the initial default value of the queue manager.

If the queue manager runs without using cryptographic hardware, refer to the CipherSpecs listed in Specifying CipherSpecs employing FIPS 140-2 certified cryptography:

MQSSL_FIPS_YES
Specifies that only FIPS-certified algorithms are to be used in the CipherSpecs allowed on all TLS connections from and to this queue manager.

For a listing of appropriate FIPS 140-2 certified CipherSpecs; see Specifying CipherSpecs.

Changes to SSLFIPS become effective either:

- On UNIX, Linux, and Windows, when a new channel process is started.
- For channels that run as threads of the channel initiator on UNIX, Linux, and Windows, when the channel initiator is restarted.
- For channels that run as threads of the listener on UNIX, Linux, and Windows, when the listener is restarted.
- For channels that run as threads of a process pooling process, when the process pooling process is started or restarted and first runs a TLS channel. If the process pooling process has already run a TLS channel, and you want the change to become effective immediately, run the MQSC command **REFRESH SECURITY TYPE(SSL)**. The process pooling process is **amqrmppa** on UNIX, Linux, and Windows.
- On z/OS, when the channel initiator is restarted.
- When a **REFRESH SECURITY TYPE(SSL)** command is issued, except on z/OS.

SSLKeyRepository (MQCFST)
The TLS key repository (parameter identifier: MQCA_SSL_KEY_REPOSITORY).

The length of the string is MQ_SSL_KEY_REPOSITORY_LENGTH.

Indicates the name of the Secure Sockets Layer key repository.

The format of the name depends on the environment:

- On z/OS, it is the name of a key ring.
- On IBM i, it is of the form *pathname/keyfile*, where *keyfile* is specified without the suffix (.kdb), and identifies a GSKit key database file. The default value is /QIBM/UserData/ICSS/Cert/Server/Default.

  If you specify *SYSTEM, IBM MQ uses the system certificate store as the key repository for the queue manager. As a result, the queue manager is registered as a server application in Digital Certificate Manager (DCM). You can assign any server/client certificate in the system store to this application.

  If you change the SSLKEYR parameter to a value other than *SYSTEM, IBM MQ unregisters the queue manager as an application with DCM.
- On UNIX, it is of the form *pathname/keyfile* and on Windows *pathname\keyfile*, where *keyfile* is specified without the suffix (.kdb), and identifies a GSKit key database file. The default value for UNIX is /var/mqm/qmgrs/QMGR/ssl/key, and on Windows it is C:\Program Files\IBM\MQ\qmgrs\QMGR\ssl\key, where QMGR is replaced by the queue manager name (on UNIX, Linux, and Windows).

Multi On Multiplatforms, the syntax of this parameter is validated to ensure that it contains a valid, absolute, directory path.

If SSLKEYR is blank, or is a value that does not correspond to a key ring or key database file, channels using TLS fail to start.

Changes to SSLKeyRepository become effective as follows:

- Multi On Multiplatforms:
  - when a new channel process is started
  - for channels that run as threads of the channel initiator, when the channel initiator is restarted.
  - for channels that run as threads of the listener, when the listener is restarted.
- z/OS On z/OS, when the channel initiator is restarted.

SSLKeyResetCount (MQCFIN)
SSL key reset count (parameter identifier: MQIA_SSL_RESET_COUNT).

Specifies when TLS channel MCAs that initiate communication reset the secret key used for encryption on the channel. The value of this parameter represents the total number of unencrypted bytes that are sent and received on the channel before the secret key is renegotiated. This number of bytes includes control information sent by the MCA.

The secret key is renegotiated when (whichever occurs first):

- The total number of unencrypted bytes sent and received by the initiating channel MCA exceeds the specified value, or,
- If channel heartbeats are enabled, before data is sent or received following a channel heartbeat.

Specify a value in the range 0 - 999,999,999. A value of zero, the initial default value of the queue manager, signifies that secret keys are never renegotiated. If you specify a TLS secret key reset count between 1 byte through 32 KB, TLS channels use a secret key reset count of 32Kb. This count is to avoid the performance effect of excessive key resets which would occur for small TLS secret key reset values.

SSLTasks (MQCFIN)
Number of server subtasks to use for processing TLS calls (parameter identifier: MQIA_SSL_TASKS). This parameter applies to z/OS only.

The number of server subtasks to use for processing TLS calls. To use TLS channels, you must have at least two of these tasks running.

Specify a value in the range 0 - 9999. However, to avoid problems with storage allocation, do not set this parameter to a value greater than 50.

StartStopEvent (MQCFIN)
Controls whether start and stop events are generated (parameter identifier: MQIA_START_STOP_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

Multi **StatisticsInterval (MQCFIN)**
The time interval, in seconds, at which statistics monitoring data is written to the monitoring queue (parameter identifier: MQIA_STATISTICS_INTERVAL).

Specify a value in the range 1 - 604,000.

This parameter is valid only on Multiplatforms.

z/OS **TCPChannels (MQCFIN)**
The maximum number of channels that can be current, or clients that can be connected, that use the TCP/IP transmission protocol (parameter identifier: MQIA_TCP_CHANNELS).

Specify a value in the range 0 - 9999. The initial default value of the queue manager is 200.

Sharing conversations do not contribute to the total for this parameter.

This parameter applies to z/OS only.

z/OS **TCPKeepAlive (MQCFIN)**
Specifies whether the TCP KEEPALIVE facility is to be used to check whether the other end of a connection is still available (parameter identifier: MQIA_TCP_KEEP_ALIVE).

The value can be:

MQTCPKEEP_YES
The TCP KEEPALIVE facility is to be used as specified in the TCP profile configuration data set. The interval is specified in the *KeepAliveInterval* channel attribute.

MQTCPKEEP_NO
The TCP KEEPALIVE facility is not to be used. This value is the initial default value of the queue manager.

This parameter applies only to z/OS.

z/OS **TCPName (MQCFST)**
The name of the TCP/IP system that you are using (parameter identifier: MQIA_TCP_NAME).

The maximum length of the string is MQ_TCP_NAME_LENGTH.

This parameter applies only to z/OS.

z/OS **TCPStackType (MQCFIN)**
Specifies whether the channel initiator can use only the TCP/IP address space specified in *TCPName*, or can optionally bind to any selected TCP/IP address (parameter identifier: MQIA_TCP_STACK_TYPE).

The value can be:

MQTCPSTACK_SINGLE
The channel initiator uses the TCP/IP address space that is specified in *TCPName*. This value is the initial default value of the queue manager.

MQTCPSTACK_MULTIPLE
The channel initiator can use any TCP/IP address space available to it. It defaults to the one specified in *TCPName* if no other is specified for a channel or listener.

This parameter applies only to z/OS.

TraceRouteRecording (MQCFIN)
Specifies whether trace-route information can be recorded and a reply message generated (parameter identifier: MQIA_TRACE_ROUTE_RECORDING).

The value can be:

MQRECORDING_DISABLED
Trace-route information cannot be recorded.

MQRECORDING_MSG
Trace-route information can be recorded and replies sent to the destination specified by the originator of the message causing the trace-route record.

MQRECORDING_Q
Trace-route information can be recorded and replies sent to SYSTEM.ADMIN.TRACE.ROUTE.QUEUE.

If participation in route tracing is enabled using this queue manager attribute, the value of the attribute is only important if a reply is generated. Route tracing is enabled by not setting *TraceRouteRecording* to MQRECORDING_DISABLED. The reply must go either to SYSTEM.ADMIN.TRACE.ROUTE.QUEUE, or to the destination specified by the message itself. Provided the attribute is not disabled then messages not yet at the final destination might have information added to them. For more information about trace-route records, see Controlling trace-route messaging.

TreeLifeTime (MQCFIN)
The lifetime, in seconds, of non-administrative topics (parameter identifier: MQIA_TREE_LIFE_TIME).

Non-administrative topics are those topics created when an application publishes to, or subscribes as, a topic string that does not exist as an administrative node. When this non-administrative node no longer has any active subscriptions, this parameter determines how long the queue manager waits before removing that node. Only non-administrative topics that are in use by a durable subscription remain after the queue manager is recycled.

Specify a value in the range 0 - 604,000. A value of 0 means that non-administrative topics are not removed by the queue manager. The initial default value of the queue manager is 1800.

TriggerInterval (MQCFIN)
Trigger interval (parameter identifier: MQIA_TRIGGER_INTERVAL).

Specifies the trigger time interval, expressed in milliseconds, for use only with queues where *TriggerType* has a value of MQTT_FIRST.

In this case, trigger messages are normally generated only when a suitable message arrives on the queue, and the queue was previously empty. Under certain circumstances, however, an additional trigger message can be generated with MQTT_FIRST triggering, even if the queue was not empty. These additional trigger messages are not generated more often than every *TriggerInterval* milliseconds.

Specify a value in the range 0 - 999,999,999.

Error codes (Change Queue Manager)

This command might return the following errors in the response format header, in addition to the values shown on page "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CERT_LABEL_NOT_ALLOWED
Certificate label error.

MQRCCF_CHAD_ERROR
Channel automatic definition error.

MQRCCF_CHAD_EVENT_ERROR
Channel automatic definition event error.

MQRCCF_CHAD_EVENT_WRONG_TYPE
Channel automatic definition event parameter not allowed for this channel type.

MQRCCF_CHAD_EXIT_ERROR
Channel automatic definition exit name error.

MQRCCF_CHAD_EXIT_WRONG_TYPE
Channel automatic definition exit parameter not allowed for this channel type.

MQRCCF_CHAD_WRONG_TYPE
Channel automatic definition parameter not allowed for this channel type.

MQRCCF_FORCE_VALUE_ERROR
Force value not valid.

MQRCCF_PATH_NOT_VALID
Path not valid.

MQRCCF_PWD_LENGTH_ERROR
Password length error.

MQRCCF_PSCLUS_DISABLED_TOPDEF
Administrator or application attempted to define a cluster topic when **PubSubClub** is set to MQPSCLUS_DISABLED.

MQRCCF_PSCLUS_TOPIC_EXSITS
Administrator tried to set **PubSubClub** to MQPSCLUS_DISABLED when a cluster topic definition exists.

IBM i **MQRCCF_Q_MGR_ATTR_CONFLICT**
Queue manager attribute error. A possible cause is that you attempted to specify SSLKEYR(*SYSTEM) with a nonblank queue manager CERTLABL.

MQRCCF_Q_MGR_CCSID_ERROR
Coded character set value not valid.

MQRCCF_REPOS_NAME_CONFLICT
Repository names not valid.

MQRCCF_UNKNOWN_Q_MGR
Queue manager not known.

MQRCCF_WRONG_CHANNEL_TYPE
Channel type error.

Related information:

Channel states

Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client

Federal Information Processing Standards (FIPS) for UNIX, Linux and Windows

Change Security on z/OS: z/OS

The Change Security command changes specified attributes of an existing security definition.

The Change Security (MQCMD_CHANGE_SECURITY) command defines system-wide security options.

Required parameters

None

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

SecurityInterval (MQCFIN)
Timeout check interval (parameter identifier: MQIACF_SECURITY_INTERVAL).

Specifies the interval between checks for user IDs and associated resources to determine whether the *SecurityTimeout* has occurred. The value specifies a number of minutes in the range zero through

10080 (one week). If *SecurityInterval* is specified as zero, no user timeouts occur. If *SecurityInterval* is specified as nonzero, the user ID times out at a time between *SecurityTimeout* and *SecurityTimeout* plus *SecurityInterval*.

SecurityTimeout (MQCFIN)
Security information timeout (parameter identifier: MQIACF_SECURITY_TIMEOUT).

Specifies how long security information about an unused user ID and associated resources is retained by IBM MQ. The value specifies a number of minutes in the range zero through 10080 (one week). If *SecurityTimeout* is specified as zero, and *SecurityInterval* is nonzero, all such information is discarded by the queue manager every *SecurityInterval* number of minutes.

Change SMDS on z/OS: z/OS

The Change SMDS (MQCMD_CHANGE_SMDS) command changes the attributes of shared message data set.

The Change SMDS (MQCMD_CHANGE_SMDS) command changes the current shared message data set options for the specified queue manager and CF structure.

SMDS (MQCFST)
Specifies the queue manager for which the shared message data set properties are to be changed, or an asterisk to change the properties for all shared message data sets associated with the specified CFSTRUCT.

CFStrucName (MQCFST)
The name of the CF application structure with SMDS parameters that you want to change (parameter identifier: MQCA_CF_STRUC_NAME).

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

Optional parameters

DSBufs (MQCFIN)
The shared message data set buffers group (parameter identifier: MQIA_CF_SMDS_BUFFERS).

Specifies the number of buffers to be allocated in each queue manager for accessing shared message data sets. The size of each buffer is equal to the logical block size.

A value in the range 1 - 9999 or MQDSB_DEFAULT.

When DEFAULT is used any previous value is overridden and the DSBUFS value from the CFSTRUCT definition is used. The size of each buffer is equal to the logical block size.

Value can not be set unless CFLEVEL(5) is defined.

DSEXPAND (MQCFIN)
The shared message data set expand option (parameter identifier: MQIACF_CF_SMDS_EXPAND).

Specifies whether or not the queue manager should expand a shared message data set when it is nearly full, and further blocks are required in the data set. The value can be any of the following values:

MQDSE_YES
The data set can be expanded.

MQDSE_NO
The data set cannot be expanded.

MQDSE_DEFAULT
Only returned on DISPLAY CFSTRUCT when not explicitly set

Value can not be set unless CFLEVEL(5) is defined.

Change, Copy, and Create Service on Multiplatforms: Multi

The Change Service command changes existing service definitions. The Copy and Create service commands create new service definitions - the Copy command uses attribute values of an existing service definition.

The Change Service (MQCMD_CHANGE_SERVICE) command changes the specified attributes of an existing IBM MQ service definition. For any optional parameters that are omitted, the value does not change.

The Copy Service (MQCMD_COPY_SERVICE) command creates an IBM MQ service definition, using, for attributes not specified in the command, the attribute values of an existing service definition.

The Create Service (MQCMD_CREATE_SERVICE) command creates an IBM MQ service definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameter (Change and Create Service)

ServiceName (MQCFST)
The name of the service definition to be changed or created (parameter identifier: MQCA_SERVICE_NAME).

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Required parameters (Copy Service)

FromServiceName (MQCFST)
The name of the service definition to be copied from (parameter identifier: MQCACF_FROM_SERVICE_NAME).

This parameter specifies the name of the existing service definition that contains values for the attributes not specified in this command.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

ToServiceName (MQCFST)
To service name (parameter identifier: MQCACF_TO_SERVICE_NAME).

This parameter specifies the name of the new service definition. If a service definition with this name exists, *Replace* must be specified as MQRP_YES.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Optional parameters (Change, Copy, and Create Service)

Replace (MQCFIN)
Replace attributes (parameter identifier: MQIACF_REPLACE).

If a namelist definition with the same name as *ToServiceName* exists, this specifies parameter whether it is to be replaced. The value can be:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

ServiceDesc (MQCFST)
Description of service definition (parameter identifier: MQCA_SERVICE_DESC).

This parameter is a plain-text comment that provides descriptive information about the service definition. It must contain only displayable characters.

If characters are used that are not in the coded character set identifier (CCSID) for the queue manager on which the command is executing, they might be translated incorrectly.

The maximum length of the string is MQ_SERVICE_DESC_LENGTH.

ServiceType (MQCFIN)
The mode in which the service is to run (parameter identifier: MQIA_SERVICE_TYPE).

Specify either:

MQSVC_TYPE_SERVER
Only one instance of the service can be executed at a time, with the status of the service made available by the Inquire Service Status command.

MQSVC_TYPE_COMMAND
Multiple instances of the service can be started.

StartArguments (MQCFST)
Arguments to be passed to the program on startup (parameter identifier: MQCA_SERVICE_START_ARGS).

Specify each argument within the string as you would on a command line, with a space to separate each argument to the program.

The maximum length of the string is MQ_SERVICE_ARGS_LENGTH.

StartCommand (MQCFST)
Service program name (parameter identifier: MQCA_SERVICE_START_COMMAND).

Specifies the name of the program which is to run. You must specify a fully qualified path name to the executable program.

The maximum length of the string is MQ_SERVICE_COMMAND_LENGTH.

StartMode (MQCFIN)
Service mode (parameter identifier: MQIA_SERVICE_CONTROL).

Specifies how the service is to be started and stopped. The value can be any of the following values:

MQSVC_CONTROL_MANUAL
The service is not to be started automatically or stopped automatically. It is to be controlled by user command. This value is the default value.

MQSVC_CONTROL_Q_MGR
The service being defined is to be started and stopped at the same time as the queue manager is started and stopped.

MQSVC_CONTROL_Q_MGR_START
The service is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

StderrDestination (MQCFST)
Specifies the path to a file to which the standard error (stderr) of the service program must be redirected (parameter identifier: MQCA_STDERR_DESTINATION).

If the file does not exist when the service program is started, the file is created.

The maximum length of the string is MQ_SERVICE_PATH_LENGTH.

StdoutDestination (MQCFST)
Specifies the path to a file to which the standard output (stdout) of the service program must be redirected (parameter identifier: MQCA_STDOUT_DESTINATION).

If the file does not exist when the service program is started, the file is created.

The maximum length of the string is MQ_SERVICE_PATH_LENGTH.

StopArguments (MQCFST)
Specifies the arguments to be passed to the stop program when instructed to stop the service (parameter identifier: MQCA_SERVICE_STOP_ARGS).

Specify each argument within the string as you would on a command line, with a space to separate each argument to the program.

The maximum length of the string is MQ_SERVICE_ARGS_LENGTH.

StopCommand (MQCFST)
Service program stop command (parameter identifier: MQCA_SERVICE_STOP_COMMAND).

This parameter is the name of the program that is to run when the service is requested to stop. You must specify a fully qualified path name to the executable program.

The maximum length of the string is MQ_SERVICE_COMMAND_LENGTH.

Change, Copy, and Create Storage Class on z/OS: z/OS

The Change Storage Class command changes existing storage class definitions. The Copy and Create Storage Class commands create new storage class definitions - the Copy command uses attribute values of an existing storage class definition.

The Change Storage Class (MQCMD_CHANGE_STG_CLASS) command changes the characteristics of a storage class. For any optional parameters that are omitted, the value does not change.

The Copy Storage Class (MQCMD_COPY_STG_CLASS) command creates a storage class to page set mapping using, for attributes not specified in the command, the attribute values of an existing storage class.

The Create Storage Class (MQCMD_CREATE_STG_CLASS) command creates a storage class to page set mapping. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameter (Change and Create Storage Class)

StorageClassName (MQCFST)
The name of the storage class to be changed or created (parameter identifier: MQCA_STORAGE_CLASS).

The maximum length of the string is MQ_STORAGE_CLASS_LENGTH.

Required parameters (Copy Storage Class)

FromStorageClassName (MQCFST)
The name of the storage class to be copied from (parameter identifier: MQCACF_FROM_STORAGE_CLASS).

On z/OS, the queue manager searches for an object with the name you specify and a disposition of MQQSGD_Q_MGR or MQQSGD_COPY to copy from. This parameter is ignored if a value of MQQSGD_COPY is specified for *QSGDisposition*. In this case, an object with the name specified by *ToStorageClassName* and the disposition MQQSGD_GROUP is searched for to copy from.

The maximum length of the string is MQ_STORAGE_CLASS_LENGTH.

ToStorageClassName (MQCFST)
The name of the storage class to copy to (parameter identifier: MQCACF_TO_STORAGE_CLASS).

The maximum length of the string is MQ_STORAGE_CLASS_LENGTH.

Optional parameters (Change, Copy, and Create Storage Class)

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

PageSetId (MQCFIN)
Page set identifier that the storage class is to be associated with (parameter identifier: MQIA_PAGESET_ID).

Specify a string of two numeric characters in the range 00 through 99.

If you do not specify this parameter, the default is taken from the default storage class SYSTEMST.

No check is made that the page set has been defined; an error is raised only if you try to put a message to a queue that specifies this storage class (MQRC_PAGESET_ERROR).

PassTicketApplication (MQCFST)
Pass ticket application (parameter identifier: MQCA_PASS_TICKET_APPL).

The application name that is passed to RACF when authenticating the passticket specified in the MQIIH header.

The maximum length is MQ_PASS_TICKET_APPL_LENGTH.

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be any of the following values:

QSGDisposition	Change	Copy, Create
MQQSGD_COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.	The object is defined on the page set of the queue manager that executes the command using the MQQSGD_GROUP object of the same name as the *ToStorageClassName* object (for Copy) or the *StorageClassName* object (for Create).

QSGDisposition	Change	Copy, Create
MQQSGD_GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command. If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero: `DEFINE STGCLASS(storage-class) REPLACE QSGDISP(COPY)` The Change for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.	The object definition resides in the shared repository. This parameter is allowed only if the queue manager is in a queue-sharing group. If the definition is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to attempt to make or refresh local copies on page set zero: `DEFINE STGCLASS(storage-class) REPLACE QSGDISP(COPY)` The Copy or Create for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.
MQQSGD_PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with MQQSGD_Q_MGR or MQQSGD_COPY. Any object residing in the shared repository is unaffected.	Not permitted.
MQQSGD_Q_MGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command. This value is the default value.	The object is defined on the page set of the queue manager that executes the command. This value is the default value.

Replace (MQCFIN)
Replace attributes (parameter identifier: MQIACF_REPLACE).

If a storage class definition with the same name as *ToStorageClassName* exists, this parameter specifies whether it is to be replaced. The value can be:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

StorageClassDesc (MQCFST)
The description of the storage class (parameter identifier: MQCA_STORAGE_CLASS_DESC).

The maximum length is MQ_STORAGE_CLASS_DESC_LENGTH.

XCFGroupName (MQCFST)
XCF group name (parameter identifier: MQCA_XCF_GROUP_NAME).

If you are using the IMS bridge, this parameter is the name of the XCF group to which the IMS system belongs.

The maximum length is MQ_XCF_GROUP_NAME_LENGTH.

XCFMemberName (MQCFST)
XCF member name (parameter identifier: MQCA_XCF_MEMBER_NAME).

If you are using the IMS bridge, this parameter is the XCF member name of the IMS system within the XCF group specified in *XCFGroupName*.

The maximum length is MQ_XCF_MEMBER_NAME_LENGTH.

Change, Copy, and Create Subscription:

The Change Subscription command changes existing subscription definitions. The Copy and Create Subscription commands create new subscription definitions - the Copy command uses attribute values of an existing subscription definition.

The Change Subscription (MQCMD_CHANGE_SUBSCRIPTION) command changes the specified attributes of an existing IBM MQ subscription. For any optional parameters that are omitted, the value does not change.

The Copy Subscription (MQCMD_COPY_SUBSCRIPTION) command creates an IBM MQ subscription, using, for attributes not specified in the command, the attribute values of an existing subscription.

The Create Subscription (MQCMD_CREATE_SUBSCRIPTION) command creates an IBM MQ administrative subscription so that existing applications can participate in publish/subscribe application.

Required parameters (Change Subscription)

SubName (MQCFST)
The name of the subscription definition to be changed (parameter identifier: MQCACF_SUB_NAME).

The maximum length of the string is MQ_SUB_NAME_LENGTH.

or

SubId (MQCFBS)
The unique identifier of the subscription definition to be changed (parameter identifier: MQBACF_SUB_ID).

The maximum length of the string is MQ_CORREL_ID_LENGTH.

Required parameters (Copy Subscription)

ToSubscriptionName (MQCFBS)
The name of the subscription to copy to (parameter identifier: MQCACF_TO_SUB_NAME).

The maximum length of the string is MQ_SUBSCRIPTION_NAME_LENGTH.

You require at least one of *FromSubscriptionName* or *SubId*.

FromSubscriptionName (MQCFST)
The name of the subscription definition to be copied from (parameter identifier: MQCACF_FROM_SUB_NAME).

z/OS On z/OS, the queue manager searches for an object with the name you specify and a disposition of MQQSGD_Q_MGR or MQQSGD_COPY to copy from. This parameter is ignored if a value of MQQSGD_COPY is specified for *QSGDisposition*. In this case, an object with the name specified by *ToSubscriptionName* and the disposition MQQSGD_GROUP is used.

The maximum length of the string is MQ_SUBSCRIPTION_NAME_LENGTH.

SubId (MQCFBS)
The unique identifier of the subscription definition to be changed (parameter identifier: MQBACF_SUB_ID).

The maximum length of the string is MQ_CORREL_ID_LENGTH.

Required parameters (Create Subscription)

You must provide the *SubName*.

SubName (MQCFST)
The name of the subscription definition to be changed (parameter identifier: MQCACF_SUB_NAME).

The maximum length of the string is MQ_SUB_NAME_LENGTH.

You require at least one of *TopicObject* or *TopicString*.

TopicObject (MQCFST)
The name of a previously defined topic object from which is obtained the topic name for the subscription (parameter identifier: MQCA_TOPIC_NAME). Although the parameter is accepted, the value specified cannot be different from the original value for Change Subscription.

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

TopicString (MQCFST)
The resolved topic string (parameter identifier: MQCA_TOPIC_STRING). .

The maximum length of the string is MQ_TOPIC_STR_LENGTH.

Optional parameters (Change, Copy, and Create Subscription) z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is processed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- a queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Destination (MQCFST)
Destination (parameter identifier: MQCACF_DESTINATION).

Specifies the name of the alias, local, remote, or cluster queue to which messages for this subscription are put.

This parameter is mandatory if *DestinationClass* is set to MQDC_PROVIDED, but is not applicable if *DestinationClass* is set to MQDC_MANAGED.

DestinationClass (MQCFIN)
Destination class (parameter identifier: MQIACF_DESTINATION_CLASS).

Specifies whether the destination is managed.

Specify either:

MQDC_MANAGED
The destination is managed.

MQDC_PROVIDED
The destination queue is as specified in the *Destination* field.

Although the parameter is accepted, the value specified cannot be different from the original value for Change Subscription.

DestinationCorrelId (MQCFBS)
Destination correlation identifier (parameter identifier: MQBACF_DESTINATION_CORREL_ID).

Provides a correlation identifier that is placed in the *CorrelId* field of the message descriptor for all the messages sent to this subscription.

The maximum length is MQ_CORREL_ID_LENGTH.

DestinationQueueManager (MQCFST)
Destination queue manager (parameter identifier: MQCACF_DESTINATION_Q_MGR).

Specifies the name of the destination queue manager, either local or remote, to which messages for the subscription are forwarded.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Expiry (MQCFIN)
The time, in tenths of a second, at which a subscription expires after its creation date and time (parameter identifier: MQIACF_EXPIRY).

The default value of unlimited means that the subscription never expires.

After a subscription has expired it becomes eligible to be discarded by the queue manager and receives no further publications.

PublishedAccountingToken (MQCFBS)
Value of the accounting token used in the *AccountingToken* field of the message descriptor (parameter identifier: MQBACF_ACCOUNTING_TOKEN).

The maximum length of the string is MQ_ACCOUNTING_TOKEN_LENGTH.

PublishedApplicationIdentifier (MQCFST)
Value of the application identity data used in the *ApplIdentityData* field of the message descriptor (parameter identifier: MQCACF_APPL_IDENTITY_DATA.

The maximum length of the string is MQ_APPL_IDENTITY_DATA_LENGTH.

PublishPriority (MQCFIN)
The priority of the message sent to this subscription (parameter identifier: MQIACF_PUB_PRIORITY).

The value can be:

MQPRI_PRIORITY_AS_PUBLISHED
Priority of messages sent to this subscription is taken from the priority supplied to the published message. This value is the supplied default value.

MQPRI_PRIORITY_AS_QDEF
Priority of messages sent to this subscription is determined by the default priority of the queue defined as a destination.

0-9 An integer value providing an explicit priority for messages sent to this subscription.

PublishSubscribeProperties (MQCFIN)
Specifies how publish/subscribe related message properties are added to messages sent to this subscription (parameter identifier: MQIACF_PUBSUB_PROPERTIES).

The value can be:

MQPSPROP_COMPAT
If the original publication is a PCF message, then the publish/subscribe properties are added

as PCF attributes. Otherwise, publish/subscribe properties are added within an MQRFH version 1 header. This method is compatible with applications coded for use with previous versions of IBM MQ.

MQPSPROP_NONE
Do not add publish/subscribe properties to the messages. This value is the supplied default value.

MQPSPROP_RFH2
Publish/subscribe properties are added within an MQRFH version 2 header. This method is compatible with applications coded for use with IBM Integration Bus, formerly known as WebSphere Message Broker.

Selector (MQCFST)
Specifies the selector applied to messages published to the topic (parameter identifier: MQCACF_SUB_SELECTOR). Although the parameter is accepted, the value specified cannot be different from the original value for Change Subscription.

Only those messages that satisfy the selection criteria are put to the destination specified by this subscription.

The maximum length of the string is MQ_SELECTOR_LENGTH.

SubscriptionLevel (MQCFIN)
The level within the subscription interception hierarchy at which this subscription is made (parameter identifier: MQIACF_SUB_LEVEL). To ensure that an intercepting application receives messages before any other subscribers, make sure that it has the highest subscription level of all subscribers.

The value can be:

0 - 9 An integer in the range 0-9. The default value is 1. Subscribers with a subscription level of 9 intercept publications before they reach subscribers with lower subscription levels.

SubscriptionScope (MQCFIN)
Determines whether this subscription is passed to other queue managers in the network (parameter identifier: MQIACF_SUBSCRIPTION_SCOPE). Although the parameter is accepted, the value specified cannot be different from the original value for Change Subscription.

The value can be:

MQTSCOPE_ALL
The subscription is forwarded to all queue managers directly connected through a publish/subscribe collective or hierarchy. This value is the supplied default value.

MQTSCOPE_QMGR
The subscription only forwards messages published on the topic within this queue manager.

SubscriptionUser (MQCFST)
The userid that 'owns' this subscription. This parameter is either the userid associated with the creator of the subscription, or, if subscription takeover is permitted, the userid which last took over the subscription. (parameter identifier: MQCACF_SUB_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH.

TopicString (MQCFST)
The resolved topic string (parameter identifier: MQCA_TOPIC_STRING). Although the parameter is accepted, the value specified cannot be different from the original value for Change Subscription.

The maximum length of the string is MQ_TOPIC_STR_LENGTH.

Userdata (MQCFST)
User data (parameter identifier: MQCACF_SUB_USER_DATA).

Specifies the user data associated with the subscription

The maximum length of the string is MQ_USER_DATA_LENGTH.

VariableUser (MQCFST)
Specifies whether a user other than the one who created the subscription, that is, the user shown in *SubscriptionUser* can take over the ownership of the subscription (parameter identifier: MQIACF_VARIABLE_USER_ID).

The value can be:

MQVU_ANY_USER
Any user can take over the ownership. This value is the supplied default value.

MQVU_FIXED_USER
No other user can take over the ownership.

WildcardSchema (MQCFIN)
Specifies the schema to be used when interpreting any wildcard characters contained in the *TopicString* (parameter identifier: MQIACF_WILDCARD_SCHEMA). Although the parameter is accepted, the value specified cannot be different from the original value for Change Subscription.

The value can be:

MQWS_CHAR
Wildcard characters represent portions of strings for compatibility with IBM MQ V6.0 broker.

MQWS_TOPIC
Wildcard characters represent portions of the topic hierarchy for compatibility with IBM Integration Bus. This value is the supplied default value.

Change, Copy, and Create Topic:

The Change Topic command changes existing topic definitions. The Copy and Create Topic commands create new topic definitions - the Copy command uses attribute values of an existing topic definition.

The Change Topic (MQCMD_CHANGE_TOPIC) command changes the specified attributes of an existing IBM MQ administrative topic definition. For any optional parameters that are omitted, the value does not change.

The Copy Topic (MQCMD_COPY_TOPIC) command creates an IBM MQ administrative topic definition by using, for attributes not specified in the command, the attribute values of an existing topic definition.

The Create Topic (MQCMD_CREATE_TOPIC) command creates an IBM MQ administrative topic definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameter (Change Topic)

TopicName (MQCFST)
The name of the administrative topic definition to be changed (parameter identifier: MQCA_TOPIC_NAME).

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

Required parameters (Copy Topic)

FromTopicName (MQCFST)
The name of the administrative topic object definition to be copied from (parameter identifier: MQCACF_FROM_TOPIC_NAME).

z/OS On z/OS, the queue manager searches for an object with the name you specify and a disposition of MQQSGD_Q_MGR or MQQSGD_COPY to copy from. This parameter is ignored if a

value of MQQSGD_COPY is specified for *QSGDisposition*. In this case, an object with the name specified by *ToTopicName* and the disposition MQQSGD_GROUP is searched for to copy from.

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

TopicString (MQCFST)
The topic string (parameter identifier: MQCA_TOPIC_STRING). This string uses the forward slash (/) character as a delimiter for elements within the topic tree.

The maximum length of the string is MQ_TOPIC_STR_LENGTH.

ToTopicName (MQCFST)
The name of the administrative topic definition to copy to (parameter identifier: MQCACF_TO_TOPIC_NAME).

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

Required parameters (Create Topic)

TopicName (MQCFST)
The name of the administrative topic definition to be created (parameter identifier: MQCA_TOPIC_NAME).

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

TopicString (MQCFST)
The topic string (parameter identifier: MQCA_TOPIC_STRING).

This parameter is required and cannot contain the empty string. The "/" character within this string has a special meaning. It delimits the elements in the topic tree. A topic string can start with the "/" character but is not required to. A string starting with the "/" character is not the same as a string that does not start with the "/" character. A topic string cannot end with the "/" character.

The maximum length of the string is MQ_TOPIC_STR_LENGTH.

Optional parameters (Change, Copy, and Create Topic)

ClusterName (MQCFST)
The name of the cluster to which this topic belongs. (parameter identifier: MQCA_CLUSTER_NAME). The maximum length of the string is MQ_CLUSTER_NAME_LENGTH. Setting this parameter to a cluster that this queue manager is a member of makes all queue managers in the cluster aware of this topic. Any publication to this topic or a topic string below it put to any queue manager in the cluster is propagated to subscriptions on any other queue manager in the cluster. For more details, see Distributed publish/subscribe networks.

The value can be any of the following values:

Blank If no topic object above this topic in the topic tree has set this parameter to a cluster name, then this topic does not belong to a cluster. Publications and subscriptions for this topic are not propagated to publish/subscribe cluster-connected queue managers. If a topic node higher in the topic tree has a cluster name set, publications and subscriptions to this topic are also propagated throughout the cluster.

This value is the default value for this parameter if no value is specified.

String The topic belongs to this cluster. It is not recommended that this is set to a different cluster from a topic object above this topic object in the topic tree. Other queue managers in the cluster will honor this object's definition unless a local definition of the same name exists on those queue managers.

Additionally, if PublicationScope or SubscriptionScope are set to MQSCOPE_ALL, this value is the cluster to be used for the propagation of publications and subscriptions, for this topic, to publish/subscribe cluster-connected queue managers.

ClusterPubRoute (MQCFIN)
The routing behavior of publications between queue managers in a cluster (parameter identifier: MQIA_CLUSTER_PUB_ROUTE).

The value can be any of the following values:

MQCLROUTE_DIRECT
When you configure a direct routed clustered topic on a queue manager, all queue managers in the cluster become aware of all other queue managers in the cluster. When performing publish and subscribe operations, each queue manager can connect direct to any other queue manager in the cluster.

MQCLROUTE_TOPIC_HOST
When you use topic host routing, all queue managers in the cluster become aware of the cluster queue managers that host the routed topic definition (that is, the queue managers on which you have defined the topic object). When performing publish and subscribe operations, queue managers in the cluster connect only to these topic host queue managers, and not directly to each other. The topic host queue managers are responsible for routing publications from queue managers on which publications are published to queue managers with matching subscriptions.

After a topic object has been clustered (through setting the **CLUSTER** property) you cannot change the value of the **CLROUTE** property. The object must be un-clustered (**CLUSTER** set to `' '`) before you can change the value. Un-clustering a topic converts the topic definition to a local topic, which results in a period during which publications are not delivered to subscriptions on remote queue managers; this should be considered when performing this change. See The effect of defining a non-cluster topic with the same name as a cluster topic from another queue manager. If you try to change the value of the **CLROUTE** property while it is clustered, the system generates an `MQRCCF_CLROUTE_NOT_ALTERABLE` exception.

See also Routing for publish/subscribe clusters: Notes on behavior and Designing publish/subscribe clusters.

z/OS **CommandScope (MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

CommunicationInformation (MQCFST)
The Multicast communication information object (parameter identifier: MQCA_COMM_INFO_NAME).

The maximum length of the string is MQ_COMM_INFO_NAME_LENGTH.

Custom (MQCFST)
Custom attribute for new features (parameter identifier: `MQCA_CUSTOM`).

This attribute contains the values of attributes, as pairs of attribute name and value, separated by at least one space. The attribute name-value pairs have the form `NAME(VALUE)`. Single quotation marks must be escaped with another single quotation mark.

CAPEXPRY (*integer*)
The maximum time, expressed in tenths of a second, until a message published to a topic which inherits properties from this object, remains in the system until it becomes eligible for expiry processing.

For more information on message expiry processing, see Enforcing lower expiration times.

The value can be one of the following:

integer
The value must be in the range one through to 999 999 999.

NOLIMIT
There is no limit on the expiry time of messages put using this object.

ASPARENT
The maximum message expiry time is based on the setting of the closest parent administrative topic object in the topic tree. This is the default value.

Specifying a value for CAPEXPRY that is not valid, does not cause the command to fail. Instead,, the default value is used.

DefPersistence (MQCFIN)
Default persistence (parameter identifier: MQIA_TOPIC_DEF_PERSISTENCE).

Specifies the default for message-persistence of messages published to the topic. Message persistence determines whether messages are preserved across restarts of the queue manager.

The value can be any of the following values:

MQPER_PERSISTENCE_AS_PARENT
The default persistence is based on the setting of the closest parent administrative topic object in the topic tree.

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

DefPriority (MQCFIN)
Default priority (parameter identifier: MQIA_DEF_PRIORITY).

Specifies the default priority of messages published to the topic.

Specify either:

integer
The default priority to be used, in the range zero through to the maximum priority value that is supported (9).

MQPRI_PRIORITY_AS_PARENT
The default priority is based on the setting of the closest parent administrative topic object in the topic tree.

DefPutResponse (MQCFIN)
Default put response (parameter identifier: MQIA_DEF_PUT_RESPONSE_TYPE).

The value can be:

MQPRT_ASYNC_RESPONSE
The put operation is issued asynchronously, returning a subset of MQMD fields.

MQPRT_RESPONSE_AS_PARENT
The default put response is based on the setting of the closest parent administrative topic object in the topic tree.

MQPRT_SYNC_RESPONSE
The put operation is issued synchronously, returning a response.

DurableModelQName (MQCFST)
Name of the model queue to be used for durable subscriptions (parameter identifier: MQCA_MODEL_DURABLE_Q).

The maximum length of the string is MQ_Q_NAME_LENGTH.

DurableSubscriptions (MQCFIN)
Whether applications are permitted to make durable subscriptions (parameter identifier: MQIA_DURABLE_SUB).

The value can be:

MQSUB_DURABLE_AS_PARENT
Whether durable subscriptions are permitted is based on the setting of the closest parent administrative topic object in the topic tree.

MQSUB_DURABLE_ALLOWED
Durable subscriptions are permitted.

MQSUB_DURABLE_INHIBITED
Durable subscriptions are not permitted.

InhibitPublications (MQCFIN)
Whether publications are allowed for this topic (parameter identifier: MQIA_INHIBIT_PUB).

The value can be:

MQTA_PUB_AS_PARENT
Whether messages can be published to this topic is based on the setting of the closest parent administrative topic object in the topic tree.

MQTA_PUB_INHIBITED
Publications are inhibited for this topic.

MQTA_PUB_ALLOWED
Publications are allowed for this topic.

InhibitSubscriptions (MQCFIN)
Whether subscriptions are allowed for this topic (parameter identifier: MQIA_INHIBIT_SUB).

The value can be:

MQTA_SUB_AS_PARENT
Whether applications can subscribe to this topic is based on the setting of the closest parent administrative topic object in the topic tree.

MQTA_SUB_INHIBITED
Subscriptions are inhibited for this topic.

MQTA_SUB_ALLOWED
Subscriptions are allowed for this topic.

Multicast (MQCFIN)
Whether multicast is allowable in the topic tree (parameter identifier: MQIA_MULTICAST).

The value can be:

MQMC_AS_PARENT
Whether multicast is allowed on this topic is based on the setting of the closest parent administrative topic object in the topic tree.

MQMC_ENABLED
Multicast is allowed on this topic.

MQMC_DISABLED
Multicast is not allowed on this topic.

MQMC_ONLY
Only subscriptions and publications made using multicast are allowed on this topic.

NonDurableModelQName (MQCFST)
Name of the model queue to be used for non-durable subscriptions (parameter identifier: MQCA_MODEL_NON_DURABLE_Q).

The maximum length of the string is MQ_Q_NAME_LENGTH.

NonPersistentMsgDelivery (MQCFIN)
The delivery mechanism for non-persistent messages published to this topic (parameter identifier: MQIA_NPM_DELIVERY).

The value can be:

MQDLV_AS_PARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

MQDLV_ALL
Non-persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_DUR
Non-persistent messages must be delivered to all durable subscribers. Failure to deliver a non-persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_AVAIL
Non-persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

PersistentMsgDelivery (MQCFIN)
The delivery mechanism for persistent messages published to this topic (parameter identifier: MQIA_PM_DELIVERY).

The value can be:

MQDLV_AS_PARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

MQDLV_ALL
Persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_DUR
Persistent messages must be delivered to all durable subscribers. Failure to deliver a

persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_AVAIL
Persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

ProxySubscriptions (MQCFIN)
Whether a proxy subscription is to be sent for this topic to directly connected queue managers, even if no local subscriptions exist (parameter identifier: MQIA_PROXY_SUB).

The value can be:

MQTA_PROXY_SUB_FORCE
A proxy subscription is sent to connected queue managers even if no local subscriptions exist.

Note: The proxy subscription is sent when this value is set on Create or Change of the topic.

MQTA_PROXY_SUB_FIRSTUSE
For each unique topic string at or below this topic object, a proxy subscription is asynchronously sent to all neighboring queue managers in the following scenarios:

- When a local subscription is created.
- When a proxy subscription is received that must be propagated to further directly connected queue managers.

This value is the default value for this parameter if no value is specified.

PublicationScope (MQCFIN)
Whether this queue manager propagates publications for this topic, to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: MQIA_PUB_SCOPE).

The value can be:

MQSCOPE_AS_PARENT
Whether this queue manager propagates publications, for this topic, to queue managers as part of a hierarchy or as part of a publish/subscribe cluster is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

This value is the default value for this parameter if no value is specified.

MQSCOPE_QMGR
Publications for this topic are not propagated to other queue managers.

MQSCOPE_ALL
Publications for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

Note: This behavior can be over-ridden on a publication-by-publication basis, by using MQPMO_SCOPE_QMGR on the Put Message Options.

z/OS **QSGDisposition (MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be any of the following values:

QSGDisposition	Change	Copy, Create
MQQSGD_COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined by using a command that had the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined by using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.	The object is defined on the page set of the queue manager that executes the command by using the MQQSGD_GROUP object of the same name as the *ToTopicName* object (for Copy) or *TopicName* object (for Create).
MQQSGD_GROUP	The object definition resides in the shared repository. The object was defined by using a command that had the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command. If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group so that they refresh local copies on page set zero: `DEFINE TOPIC(name)` `REPLACE QSGDISP(COPY)` The Change for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.	The object definition resides in the shared repository. This definition is allowed only if the queue manager is in a queue-sharing group. If the definition is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group so that they make or refresh local copies on page set zero: `DEFINE TOPIC(name)` `REPLACE QSGDISP(COPY)` The Copy or Create for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.
MQQSGD_PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with MQQSGD_Q_MGR or MQQSGD_COPY. Any object residing in the shared repository is unaffected.	Not permitted.
MQQSGD_Q_MGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command. This value is the default value.	The object is defined on the page set of the queue manager that executes the command. This value is the default value.

Replace (MQCFIN)
Replace attributes (parameter identifier: MQIACF_REPLACE).

If a topic definition with the same name as *ToTopicName* exists, this parameter specifies whether it is to be replaced. The value can be as follows:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

SubscriptionScope (MQCFIN)
Whether this queue manager propagates subscriptions for this topic, to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: MQIA_SUB_SCOPE).

The value can be:

MQSCOPE_AS_PARENT
Whether this queue manager propagates subscriptions, for this topic, to queue managers as part of a hierarchy or as part of a publish/subscribe-cluster is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

This value is the default value for this parameter if no value is specified.

MQSCOPE_QMGR
Subscriptions for this topic are not propagated to other queue managers.

MQSCOPE_ALL
Subscriptions for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

Note: This behavior can be over-ridden on a subscription-by-subcription basis, by using MQSO_SCOPE_QMGR on the Subscription Descriptor or SUBSCOPE(QMGR) on DEFINE SUB.

TopicDesc (MQCFST)
Topic description (parameter identifier: MQCA_TOPIC_DESC).

Text that briefly describes the object

The maximum length is MQ_TOPIC_DESC_LENGTH.

Use characters from the character set identified by the coded character set identifier (CCSID) for the message queue manager on which the command is executing to ensure that the text is translated correctly if it is sent to another queue manager.

TopicType (MQCFIN)
Topic type (parameter identifier: MQIA_TOPIC_TYPE).

The value specified must match the type of the topic being changed. The value can be:

MQTOPT_LOCAL
Local topic object

UseDLQ (MQCFIN)
Determines whether the dead-letter queue is used when publication messages cannot be delivered to their correct subscriber queue (parameter identifier: MQIA_USE_DEAD_LETTER_Q).

The value can be any of the following values:

MQUSEDLQ_AS_PARENT
Determines whether to use the dead-letter queue using the setting of the closest administrative topic object in the topic tree. This value is the default supplied with IBM MQ, but your installation might have changed it.

MQUSEDLQ_NO
Publication messages that cannot be delivered to their correct subscriber queue are treated as a failure to put the message. The MQPUT of an application to a topic fails in accordance with the settings of MQIA_NPM_DELIVERY and MQIA_PM_DELIVERY.

MQUSEDLQ_YES
If the DEADQ queue manager attribute provides the name of a dead-letter queue then it is used, otherwise the behavior is as for MQUSEDLQ_NO.

WildcardOperation (MQCFIN)
Behavior of subscriptions including wildcards made to this topic (parameter identifier: MQIA_WILDCARD_OPERATION).

The value can be:

MQTA_PASSTHRU
A less specific wildcard subscription is a subscription made by using wildcard topic names that are less specific than the topic string at this topic object. MQTA_PASSTHRU lets less specific wildcard subscriptions receive publications made to this topic and to topic strings more specific than this topic. This value is the default supplied with IBM MQ.

MQTA_BLOCK
A less specific wildcard subscription is a subscription made by using wildcard topic names that are less specific than the topic string at this topic object. MQTA_BLOCK stops less specific wildcard subscriptions receiving publications made to this topic or to topic strings more specific than this topic.

This value of this attribute is used when subscriptions are defined. If you alter this attribute, the set of topics covered by existing subscriptions is not affected by the modification. This value applies also, if the topology is changed when topic objects are created or deleted; the set of topics matching subscriptions created following the modification of the **WildcardOperation** attribute is created by using the modified topology. If you want to force the matching set of topics to be re-evaluated for existing subscriptions, you must restart the queue manager.

Clear Queue:

The Clear Queue (MQCMD_CLEAR_Q) command deletes all the messages from a local queue.

The command fails if the queue contains uncommitted messages.

Required parameters

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

The name of the local queue to be cleared. The maximum length of the string is MQ_Q_NAME_LENGTH.

Note: The target queue must be type local.

z/OS
Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_PRIVATE
Clear the private queue named in *QName*. The queue is private if it was created using a command with the attributes MQQSGD_PRIVATE or MQQSGD_Q_MGR. This value is the default value.

MQQSGD_SHARED
Clear the shared queue named in *QName*. The queue is shared if it was created using a command with the attribute MQQSGD_SHARED. This value applies only to local queues.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown on page "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRC_Q_NOT_EMPTY
(2055, X'807') Queue contains one or more messages or uncommitted put or get requests.

This reason occurs only if there are uncommitted updates.

MQRCCF_Q_WRONG_TYPE
Action not valid for the queue of specified type.

Clear Topic String:

The Clear Topic String (MQCMD_CLEAR_TOPIC_STRING) command clears the retained message which is stored for the specified topic.

Required parameters

TopicString (MQCFST)
Topic String (parameter identifier: MQCA_TOPIC_STRING).

The topic string to be cleared The maximum length of the string is MQ_TOPIC_STR_LENGTH.

ClearType (MQCFIN)
Clear type (parameter identifier: MQIACF_CLEAR_TYPE).

Specifies the type of clear command being issued. The value must be:

MQCLRT_RETAINED Remove the retained publication from the specified topic string.

Optional parameters

Scope (MQCFIN)
Scope of clearance (parameter identifier: MQIACF_CLEAR_SCOPE).

Whether the topic string is to be cleared locally or globally. The value can be:

MQCLRS_LOCAL
The retained message is removed from the specified topic string at the local queue manager only.

z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Delete Authentication Information Object:

The Delete authentication information (MQCMD_DELETE_AUTH_INFO) command deletes the specified authentication information object.

Required parameters

AuthInfoName (MQCFST)
Authentication information object name (parameter identifier: MQCA_AUTH_INFO_NAME).

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

z/OS

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_COPY
The object definition resides on the page set of the queue manager which executes this command. The object was defined by a command using the parameter MQQSGD_COPY. Any object in the shared repository, or any object defined by a command using the parameter MQQSGD_Q_MGR, is not affected by this command.

MQQSGD_GROUP
The object definition resides in the shared repository. The object was defined by a command using the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE AUTHINFO(name) QSGDISP(COPY)
```

The deletion of the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

MQQSGD_Q_MGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

MQQSGD_Q_MGR is the default value.

Delete Authority Record on Multiplatforms: 

The Delete Authority Record (MQCMD_DELETE_AUTH_REC) command deletes an authority record. The authorizations associated with the profile no longer apply to IBM MQ objects with names that match the profile name specified.

Required parameters

ObjectType (MQCFIN)
The type of object for which to delete authorizations (parameter identifier: MQIACF_OBJECT_TYPE).

The value can be any of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel object.

MQOT_CLNTCONN_CHANNEL
Client-connection channel object.

MQOT_COMM_INFO
Communication information object

MQOT_LISTENER
Listener object.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process.

MQOT_Q
Queue, or queues, that match the object name parameter.

MQOT_Q_MGR
Queue manager.

MQOT_REMOTE_Q_MGR_NAME
Remote queue manager.

MQOT_SERVICE
Service object.

MQOT_TOPIC
Topic object.

ProfileName (MQCFST)
Name of the profile to be deleted (parameter identifier: MQCACF_AUTH_PROFILE_NAME).

If you have defined a generic profile then you can specify it here, using wildcard characters to specify a named generic profile to be removed. If you specify an explicit profile name, the object must exist.

The maximum length of the string is MQ_AUTH_PROFILE_NAME_LENGTH.

Optional parameters

GroupNames (MQCFSL)
Group names (parameter identifier: MQCACF_GROUP_ENTITY_NAMES).

The names of groups having a profile deleted. At least one group name or principal name must be specified. An error occurs if neither are specified.

Each member in this list can be a maximum length of MQ_ENTITY_NAME_LENGTH.

PrincipalNames (MQCFSL)
Principal names (parameter identifier: MQCACF_PRINCIPAL_ENTITY_NAMES).

The names of principals having a profile deleted. At least one group name or principal name must be specified. An error occurs if neither are specified.

Each member in this list can be a maximum length of MQ_ENTITY_NAME_LENGTH.

Error codes (Delete Authority Record)

This command might return the following error codes in the response format header, in addition to the values shown on page "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRC_OBJECT_TYPE_ERROR
Invalid object type.

MQRC_UNKNOWN_ENTITY
Userid not authorized, or unknown.

MQRCCF_ENTITY_NAME_MISSING
Entity name missing.

MQRCCF_OBJECT_TYPE_MISSING
Object type missing.

MQRCCF_PROFILE_NAME_ERROR
Invalid profile name.

Delete CF Structure on z/OS: z/OS

The Delete CF Structure (MQCMD_DELETE_CF_STRUC) command deletes an existing CF application structure definition.

Note: This command is supported only on z/OS when the queue manager is a member of a queue-sharing group.

Required parameters

CFStrucName (MQCFST)
CF structure name (parameter identifier: MQCA_CF_STRUC_NAME).

The CF application structure definition to be deleted. The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

Delete Channel:

The Delete Channel (MQCMD_DELETE_CHANNEL) command deletes the specified channel definition.

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel definition to be deleted. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

None of the following attributes are applicable to MQTT channels unless specifically mentioned in the parameter description.

ChannelType (MQCFIN)
The type of channel (parameter identifier: MQIACH_CHANNEL_TYPE). This parameter is currently only used with MQTT Telemetry channels, and is required when deleting a Telemetry channel. The only value that can currently be given to the parameter is **MQCHT_MQTT**.

ChannelTable (MQCFIN)
Channel table (parameter identifier: MQIACH_CHANNEL_TABLE).

Specifies the ownership of the channel definition table that contains the specified channel definition.

The value can be any of the following values:

MQCHTAB_Q_MGR
Queue manager table.

MQCHTAB_Q_MGR is the default. This table contains channel definitions for channels of all types except MQCHT_CLNTCONN.

MQCHTAB_CLNTCONN
Client-connection table.

This table only contains channel definitions for channels of type MQCHT_CLNTCONN.

This parameter is not applicable to MQ Telemetry.

z/OS **CommandScope (MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

z/OS **QSGDisposition (MQCFIN)**

Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_COPY
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined by a command using the parameter MQQSGD_Q_MGR, is not affected by this command.

MQQSGD_GROUP
The object definition resides in the shared repository. The object was defined by a command using the parameters MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE CHANNEL(name) QSGDISP(COPY)
```

The deletion of the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

MQQSGD_Q_MGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

MQQSGD_Q_MGR is the default value.

This command might return the following error codes in the response format header, in addition to the values shown on page “Error codes applicable to all commands” on page 1075.

Error codes

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TABLE_ERROR
Channel table value not valid.

Delete Channel (MQTT): AIX Linux Windows

The Delete Telemetry Channel (MQCMD_DELETE_CHANNEL) command deletes the specified channel definition.

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel definition to be deleted. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType (MQCFIN)
The type of channel (parameter identifier: MQIACH_CHANNEL_TYPE). Required when deleting a Telemetry channel. The only value that can currently be given to the parameter is **MQCHT_MQTT**.

This command might return the following error codes in the response format header, in addition to the values shown on page "Error codes applicable to all commands" on page 1075.

Error codes

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

Delete Channel Listener on Multiplatforms: Multi

The Delete Channel Listener (MQCMD_DELETE_LISTENER) command deletes an existing channel listener definition.

Required parameters

ListenerName (MQCFST)
Listener name (parameter identifier: MQCACH_LISTENER_NAME).

This parameter is the name of the listener definition to be deleted. The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

Delete Communication Information Object on Multiplatforms: Multi

The Delete Communication Information Object (MQCMD_DELETE_COMM_INFO) command deletes the specified communication information object.

Required parameter

CominfoName (MQCFST)
The name of the communication information definition to be deleted (parameter identifier: MQCA_COMM_INFO_NAME).

Delete Namelist:

The Delete Namelist (MQCMD_DELETE_NAMELIST) command deletes an existing namelist definition.

Required parameters

NamelistName (MQCFST)
Namelist name (parameter identifier: MQCA_NAMELIST_NAME).

This parameter is the name of the namelist definition to be deleted. The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

z/OS

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_COPY
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.

MQQSGD_GROUP
The object definition resides in the shared repository. The object was defined by a command using the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE NAMELIST(name) QSGDISP(COPY)
```

The deletion of the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

MQQSGD_Q_MGR
The object definition resides on the page set of the queue manager that executes the

command. The object was defined by a command using the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

MQQSGD_Q_MGR is the default value.

Delete Policy on Multiplatforms: Multi

The Delete Policy (MQCMD_DELETE_PROT_POLICY) command deletes a security policy.

Required parameters

Policy-name (MQCFST)
The name of the security policy to be deleted (parameter identifier: MQCA_POLICY_NAME).

The name of the policy, or policies, to delete are the same as the name of the queue, or queues, that the policies control.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Error codes (Delete Security Policy)

This command might return the following error codes in the response format header, in addition to the values shown on page "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRC_OBJECT_TYPE_ERROR
Invalid object type.

MQRCCF_POLICY_NAME_ERROR
Invalid policy name.

Delete Process:

The Delete Process (MQCMD_DELETE_PROCESS) command deletes an existing process definition.

Required parameters

ProcessName (MQCFST)
Process name (parameter identifier: MQCA_PROCESS_NAME).

The process definition to be deleted. The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

z/OS
Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.

- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_COPY
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.

MQQSGD_GROUP
The object definition resides in the shared repository. The object was defined by a command using the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE PROCESS(name) QSGDISP(COPY)
```

The deletion of the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

MQQSGD_Q_MGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

MQQSGD_Q_MGR is the default value.

Delete Queue:

The Delete Queue (MQCMD_DELETE_Q) command deletes a queue.

Required parameters

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

The name of the queue to be deleted.

If the **Scope** attribute of the queue is MQSCO_CELL, the entry for the queue is deleted from the cell directory.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Optional parameters

Authrec (MQCFIN)
Authrec (parameter identifier: MQIACF_REMOVE_AUTHREC).

Specifies whether the associated authority record is also deleted.

This parameter does not apply to z/OS.

The value can be any of the following values:

MQRAR_YES
The authority record associated with the object is deleted. This is the default.

MQRAR_NO
The authority record associated with the object is not deleted.

z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Purge (MQCFIN)
Purge queue (parameter identifier: MQIACF_PURGE).

If there are messages on the queue MQPO_YES must be specified, otherwise the command fails. If this parameter is not present the queue is not purged.

Valid only for queue of type local.

The value can be any of the following values:

MQPO_YES
Purge the queue.

MQPO_NO
Do not purge the queue.

z/OS

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_COPY
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.

MQQSGD_GROUP
The object definition resides in the shared repository. The object was defined by a command using the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the deletion is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE queue(q-name) QSGDISP(COPY)
```

or, for a local queue only:

```
DELETE QLOCAL(q-name) NOPURGE QSGDISP(COPY)
```

The deletion of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

Note: You always get the NOPURGE option even if you specify MQPO_YES for *Purge*. To delete messages on local copies of the queues, you must explicitly issue, for each copy, the Delete Queue command with a *QSGDisposition* value of MQQSGD_COPY and a *Purge* value of MQPO_YES.

MQQSGD_Q_MGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

MQQSGD_Q_MGR is the default value.

MQQSGD_SHARED
Valid only for queue of type local.

The object resides in the shared repository. The object was defined by a command using the parameter MQQSGD_SHARED. Any object residing on the page set of the queue manager that executes the command, or any object defined by a command using the parameter MQQSGD_GROUP, is not affected by this command.

QType (MQCFIN)
Queue type (parameter identifier: MQIA_Q_TYPE).

If this parameter is present, the queue must be of the specified type.

The value can be:

MQQT_ALIAS
Alias queue definition.

MQQT_LOCAL
Local queue.

MQQT_REMOTE
Local definition of a remote queue.

MQQT_MODEL
Model queue definition.

Error codes (Delete Queue)

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRC_Q_NOT_EMPTY
(2055, X'807') Queue contains one or more messages or uncommitted put or get requests.

Delete Service on Multiplatforms: Multi

The Delete Service (MQCMD_DELETE_SERVICE) command deletes an existing service definition.

Required parameters

ServiceName (MQCFST)
Service name (parameter identifier: MQCA_SERVICE_NAME).

This parameter is the name of the service definition to be deleted.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Delete Storage Class on z/OS: z/OS

The Delete Storage Class (MQCMD_DELETE_STG_CLASS) command deletes an existing storage class definition.

Required parameters

StorageClassName (MQCFST)
Storage class name (parameter identifier: MQCA_STORAGE_CLASS).

The storage class definition to be deleted. The maximum length of the string is MQ_STORAGE_CLASS_LENGTH.

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.

- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_COPY
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.

MQQSGD_GROUP
The object definition resides in the shared repository. The object was defined by a command using the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE STGCLASS(name) QSGDISP(COPY)
```

The deletion of the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

MQQSGD_Q_MGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

MQQSGD_Q_MGR is the default value.

Delete Subscription:

The Delete Subscription (MQCMD_DELETE_SUBSCRIPTION) command deletes a subscription.

Required parameters

SubName (MQCFST)
Subscription name (parameter identifier: MQCACF_SUB_NAME).

Specifies the unique subscription name. The subscription name, if provided, must be fully specified; a wildcard is not acceptable.

The subscription name must refer to a durable subscription.

If *SubName* is not provided, *SubId* must be specified to identify the subscription to be deleted.

The maximum length of the string is MQ_SUB_NAME_LENGTH.

SubId (MQCFBS)
Subscription identifier (parameter identifier: MQBACF_SUB_ID).

Specifies the unique internal subscription identifier.

You must supply a value for *SubId* if you have not supplied a value for *SubName*.

The maximum length of the string is MQ_CORREL_ID_LENGTH.

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is processed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- Blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- A queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- An asterisk (*). The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter on which to filter.

Delete Topic:

The Delete Topic (MQCMD_DELETE_TOPIC) command deletes the specified administrative topic object.

Required parameters

TopicName (MQCFST)
The name of the administrative topic definition to be deleted (parameter identifier: MQCA_TOPIC_NAME).

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

Optional parameters

Authrec (MQCFIN)
Authrec (parameter identifier: MQIACF_REMOVE_AUTHREC).

Specifies whether the associated authority record is also deleted.

This parameter does not apply to z/OS.

The value can be any of the following values:

MQRAR_YES
The authority record associated with the object is deleted. This is the default.

MQRAR_NO
The authority record associated with the object is not deleted.

z/OS **CommandScope (MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

z/OS

QSGDisposition (MQCFIN)

Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_COPY

The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.

MQQSGD_GROUP

The object definition resides in the shared repository. The object was defined by a command using the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the deletion is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to make, or delete, local copies on page set zero:

```
DELETE TOPIC(name) QSGDISP(COPY)
```

The deletion of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

MQQSGD_Q_MGR

The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

MQQSGD_Q_MGR is the default value.

Escape on Multiplatforms: Multi

The Escape (MQCMD_ESCAPE) command conveys any IBM MQ command (MQSC) to a remote queue manager.

Use the Escape command when the queue manager (or application) sending the command does not support the particular IBM MQ command, and so does not recognize it and cannot construct the required PCF command.

The Escape command can also be used to send a command for which no Programmable Command Format has been defined.

The only type of command that can be carried is one that is identified as an MQSC, that is recognized at the receiving queue manager.

Required parameters

EscapeType (MQCFIN)
Escape type (parameter identifier: MQIACF_ESCAPE_TYPE).

The only value supported is:

MQET_MQSC
IBM MQ command.

EscapeText (MQCFST)
Escape text (parameter identifier: MQCACF_ESCAPE_TEXT).

A string to hold a command. The length of the string is limited only by the size of the message.

Error codes

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_ESCAPE_TYPE_ERROR
Escape type not valid.

Escape (Response) on Multiplatforms: Multi

The response to the Escape (MQCMD_ESCAPE) command consists of the response header followed by two parameter structures, one containing the escape type, and the other containing the text response. More than one such message might be issued, depending upon the command contained in the Escape request.

The *Command* field in the response header MQCFH contains the MQCMD_* command identifier of the text command contained in the **EscapeText** parameter in the original Escape command. For example, if *EscapeText* in the original Escape command specified `PING QMGR`, *Command* in the response has the value MQCMD_PING_Q_MGR.

If it is possible to determine the outcome of the command, the *CompCode* in the response header identifies whether the command was successful. The success or otherwise can therefore be determined without the recipient of the response having to parse the text of the response.

If it is not possible to determine the outcome of the command, *CompCode* in the response header has the value MQCC_UNKNOWN, and *Reason* is MQRC_NONE.

Parameters

EscapeType (MQCFIN)
Escape type (parameter identifier: MQIACF_ESCAPE_TYPE).

The only value supported is:

MQET_MQSC
IBM MQ command.

EscapeText (MQCFST)
Escape text (parameter identifier: MQCACF_ESCAPE_TEXT).

A string holding the response to the original command.

Inquire Archive on z/OS: z/OS

The Inquire Archive (MQCMD_INQUIRE_ARCHIVE) command returns archive system parameters and information.

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Inquire Archive (Response) on z/OS: z/OS

The response to the Inquire Archive (MQCMD_INQUIRE_ARCHIVE) command consists of the response header followed by the *ParameterType* structure and the combination of attribute parameter structures determined by the value of *ParameterType*.

Always returned:
ParameterType Specifies the type of archive information being returned. The value can be any of the following values:

MQSYSP_TYPE_INITIAL
The initial settings of the archive parameters.

MQSYSP_TYPE_SET
The settings of the archive parameters if they have been altered since their initial setting.

MQSYSP_TYPE_ARCHIVE_TAPE
Parameters relating to the tape unit (if in use). There is one such message per tape unit in use for archive logging.

Returned if *ParameterType* is MQSYSP_TYPE_INITIAL (one message is returned):
AllocPrimary, *AllocSecondary*, *AllocUnits*, *ArchivePrefix1*, *ArchivePrefix2*, *ArchiveRetention*, *ArchiveUnit1*, *ArchiveUnit2*, *ArchiveWTOR*, *BlockSize*, *Catalog*, *Compact*, *Protect*, *QuiesceInterval*, *RoutingCode*, *TimeStampFormat*

Returned if *ParameterType* is MQSYSP_TYPE_SET and any value is set (one message is returned):
AllocPrimary, *AllocSecondary*, *AllocUnits*, *ArchivePrefix1*, *ArchivePrefix2*, *ArchiveRetention*, *ArchiveUnit1*, *ArchiveUnit2*, *ArchiveWTOR*, *BlockSize*, *Catalog*, *Compact*, *Protect*, *QuiesceInterval*, *RoutingCode*, *TimeStampFormat*

Returned if *ParameterType* is MQSYSP_TYPE_ARCHIVE_TAPE (one message is returned for each tape unit in use for archive logging):
DataSetName, *LogCorrelId*, *UnitAddress*, *UnitStatus*, *UnitVolser*

Response data - archive parameter information

AllocPrimary (MQCFIN)
Primary space allocation for DASD data sets (parameter identifier: MQIACF_SYSP_ALLOC_PRIMARY).

Specifies the primary space allocation for DASD data sets in the units specified in the **AllocUnits** parameter.

AllocSecondary (MQCFIN)
Secondary space allocation for DASD data sets (parameter identifier: MQIACF_SYSP_ALLOC_SECONDARY).

Specifies the secondary space allocation for DASD data sets in the units specified in the **AllocUnits** parameter.

AllocUnits (MQCFIN)
Allocation unit (parameter identifier: MQIACF_SYSP_ALLOC_UNIT).

Specifies the unit in which primary and secondary space allocations are made. The value can be any of the following values:

MQSYSP_ALLOC_BLK
Blocks.

MQSYSP_ALLOC_TRK
Tracks.

MQSYSP_ALLOC_CYL
Cylinders.

ArchivePrefix1 (MQCFST)
Prefix for the first archive log data set name (parameter identifier: MQCACF_SYSP_ARCHIVE_PFX1).

The maximum length of the string is MQ_ARCHIVE_PFX_LENGTH.

ArchivePrefix2 (MQCFST)
Prefix for the second archive log data set name (parameter identifier: MQCACF_SYSP_ARCHIVE_PFX2).

The maximum length of the string is MQ_ARCHIVE_PFX_LENGTH.

ArchiveRetention (MQCFIN)
Archive retention period (parameter identifier: MQIACF_SYSP_ARCHIVE_RETAIN).

Specifies the retention period, in days, to be used when the archive log data set is created.

ArchiveUnit1 (MQCFST)
Specifies the device type or unit name of the device that is used to store the first copy of the archive log data set (parameter identifier: MQCACF_SYSP_ARCHIVE_UNIT1).

The maximum length of the string is MQ_ARCHIVE_UNIT_LENGTH.

ArchiveUnit2 (MQCFST)
Specifies the device type or unit name of the device that is used to store the second copy of the archive log data set (parameter identifier: MQCACF_SYSP_ARCHIVE_UNIT2).

The maximum length of the string is MQ_ARCHIVE_UNIT_LENGTH.

ArchiveWTOR (MQCFIN)
Specifies whether a message is to be sent to the operator and a reply is received before attempting to mount an archive log data set (parameter identifier: MQIACF_SYSP_ARCHIVE_WTOR).

The value can be:

MQSYSP_YES
A message is to be sent and a reply received before an attempt to mount an archive log data set.

MQSYSP_NO
A message is not to be sent and a reply received before an attempt to mount an archive log data set.

BlockSize (MQCFIN)
Block size of the archive log data set (parameter identifier: MQIACF_SYSP_BLOCK_SIZE).

Catalog (MQCFIN)
Specifies whether archive log data sets are cataloged in the primary integrated catalog facility (parameter identifier: MQIACF_SYSP_CATALOG).

The value can be:

MQSYSP_YES
Archive log data sets are cataloged.

MQSYSP_NO
Archive log data sets are not cataloged.

Compact (MQCFIN)
Specifies whether data written to archive logs is to be compacted (parameter identifier: MQIACF_SYSP_COMPACT).

The value can be any of the following values:

MQSYSP_YES
Data is to be compacted.

MQSYSP_NO
Data is not to be compacted.

Protect (MQCFIN)
Protection by external security manager (ESM) (parameter identifier: MQIACF_SYSP_PROTECT).

Specifies whether archive log data sets are protected by ESM profiles when the data sets are created.

The value can be any of the following values:

MQSYSP_YES
Data set profiles are created when logs are offloaded.

MQSYSP_NO
Profiles are not created.

QuiesceInterval (MQCFIN)
Maximum time allowed for the quiesce (parameter identifier: MQIACF_SYSP_QUIESCE_INTERVAL).

Specifies the maximum time, in seconds, allowed for the quiesce.

RoutingCode (MQCFIL)
z/OS routing code list (parameter identifier: MQIACF_SYSP_ROUTING_CODE).

Specifies the list of z/OS routing codes for messages about the archive log data sets to the operator. There can be 1 - 14 entries in the list.

TimeStampFormat (MQCFIN)
Time stamp included (parameter identifier: MQIACF_SYSP_TIMESTAMP).

Specifies whether the archive log data set name has a time stamp in it.

The value can be:

MQSYSP_YES
Names include a time stamp.

MQSYSP_NO
Names do not include a time stamp.

MQSYSP_EXTENDED
Names include a time stamp.

Response data - tape unit status information

DataSetName (MQCFST)
Data set name (parameter identifier: MQCACF_DATA_SET_NAME).

Specifies the data set name on the tape volume that is being processed, or was last processed.

The maximum length of the string is MQ_DATA_SET_NAME_LENGTH.

LogCorrelId (MQCFST)
Correlation identifier (parameter identifier: MQCACF_SYSP_LOG_CORREL_ID).

Specifies the correlation ID associated with the user of the tape being processed. This parameter is blank if there is no current user.

The maximum length of the string is MQ_LOG_CORREL_ID_LENGTH.

UnitAddress (MQCFIN)
Tape unit address: MQIACF_SYSP_UNIT_ADDRESS).

Specifies the physical address of the tape unit allocated to read the archive log.

UnitStatus (MQCFIN)
Status if the tape unit: MQIACF_SYSP_UNIT_STATUS).

The value can be:

MQSYSP_STATUS_BUSY
The tape unit is busy, actively processing an archive log data set.

MQSYSP_STATUS_PREMOUNT
The tape unit is active and allocated for premounting.

MQSYSP_STATUS_AVAILABLE
The tape unit is available, inactive, and waiting for work.

MQSYSP_STATUS_UNKNOWN
The tape unit status us unknown.

UnitVolser (MQCFST)
The volume serial number of the tape that is mounted (parameter identifier: MQCACF_SYSP_UNIT_VOLSER).

The maximum length of the string is MQ_VOLSER_LENGTH.

Inquire Authentication Information Object:

The Inquire authentication information object (**MQCMD_INQUIRE_AUTH_INFO**) command inquires about the attributes of authentication information objects.

Required parameters

AuthInfoName (MQCFST)
Authentication information object name (parameter identifier: MQCA_AUTH_INFO_NAME).

Specifies the name of the authentication information object about which information is to be returned.

Generic authentication information object names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all authentication information objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

Optional parameters

AuthInfoAttrs (MQCFIL)
Authentication information object attributes (parameter identifier: MQIACF_AUTH_INFO_ATTRS).

The attribute list can specify the following value - the default value if the parameter is not specified):

MQIACF_ALL
All attributes.

or a combination of the following:

MQIA_ADOPT_CONTEXT
Adopt the presented credentials as the context for the application.

MQCA_ALTERATION_DATE
Date on which the definition was last altered.

MQCA_ALTERATION_TIME
Time at which the definition was last altered.

MQCA_AUTH_INFO_DESC
Description of the authentication information object.

MQCA_AUTH_INFO_NAME
Name of the authentication information object.

MQIA_AUTH_INFO_TYPE
Type of authentication information object.

MQCA_AUTH_INFO_CONN_NAME
Connection name of the authentication information object.

This attribute is relevant only when **AuthInfoType** is set to MQAIT_CRL_LDAP or MQAIT_IDPW_LDAP.

MQIA_AUTHENTICATION_FAIL_DELAY
Delay in seconds before an authentication failure is returned to an application.

MQIA_AUTHENTICATION_METHOD
Authentication method for user passwords.

MQIA_CHECK_CLIENT_BINDING
Authentication requirements for client applications.

MQIA_CHECK_LOCAL_BINDING
Authentication requirements for locally bound applications.

MQIA_LDAP_AUTHORMD
Authorization method for the queue manager.

MQCA_LDAP_BASE_DN_GROUPS
The base Distinguished Name for groups in the LDAP server.

MQCA_LDAP_BASE_DN_USERS
The base Distinguished Name for users in the LDAP server.

MQCA_LDAP_FIND_GROUP_FIELD
Name of the attribute used within an LDAP entry to determine group membership.

MQCA_LDAP_GROUP_ATTR_FIELD
LDAP attribute that represents a simple name for the group.

MQCA_LDAP_GROUP_OBJECT_CLASS
The LDAP object class used for group records in the LDAP repository.

MQIA_LDAP_NESTGRP
Whether LDAP groups are checked for membership of other groups.

MQCA_LDAP_PASSWORD
LDAP password in the authentication information object.

This attribute is relevant only when **AuthInfoType** is set to `MQAIT_CRL_LDAP` or `MQAIT_IDPW_LDAP`.

MQIA_LDAP_SECURE_COMM
Whether connectivity to the LDAP server should be done securely using TLS.

MQCA_LDAP_SHORT_USER_FIELD
The field in the LDAP user record to be used as a short user name in IBM MQ.

MQCA_LDAP_USER_ATTR_FIELD
The field in the LDAP user record to be used to interpret the user ID provided by an application, if the user ID does not contain a qualifier.

MQCA_LDAP_USER_NAME
LDAP user name in the authentication information object.

This attribute is relevant only when **AuthInfoType** is set to `MQAIT_CRL_LDAP` or `MQAIT_IDPW_LDAP`.

MQCA_LDAP_USER_OBJECT_CLASS
The LDAP object class used for user records in the LDAP repository.

MQCA_AUTH_INFO_OCSP_URL
The URL of the OCSP responder used to check for certificate revocation.

AuthInfoType (MQCFIN)
Type of authentication information object. The following values are accepted:

MQAIT_CRL_LDAP
Authentication information objects specifying Certificate Revocation Lists held on LDAP servers.

MQAIT_OCSP
Authentication information objects specifying certificate revocation checking using OCSP.

MQAIT_IDPW_OS
Authentication information objects specifying certificate revocation checking using user ID and password checking through the operating system.

MQAIT_IDPW_LDAP
Authentication information objects specifying certificate revocation checking using user ID and password checking through an LDAP server.

MQAIT_ALL
Authentication information objects of any type.

z/OS **CommandScope (MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- Blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- Aqueue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- An asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use **CommandScope** as a parameter to filter on.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in **AuthInfoAttrs**, except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1601 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the **StringFilterCommand** parameter.

z/OS **QSGDisposition (MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. This value is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. This value is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined as either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

You cannot use **QSGDisposition** as a parameter to filter on.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in **AuthInfoAttrs**, except MQCA_AUTH_INFO_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. For information about using this filter condition, see "MQCFSF - PCF string filter parameter" on page 1608.

If you specify a string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter.

Inquire Authentication Information Object (Response):

The response of the Inquire authentication information (MQCMD_INQUIRE_AUTH_INFO) command consists of the response header followed by the *AuthInfoName* structure (and on z/OS only, the *QSGDisposition* structure), and the requested combination of attribute parameter structures (where applicable).

Always returned:
AuthInfoName z/OS , *QSGDisposition*

Returned if requested:
AdoptContext, *AlterationDate*, *AlterationTime*, *AuthInfoConnName*,*BaseDNGroup*, *BaseDNUser*, *AuthInfoType*, *CheckClient*, *CheckLocal*, *ClassUser*, *FailureDelay*, *LDAPPassword*, *LDAPUserName*, *OCSPResponderURL*, *SecureComms*, *ShortUser*, *UserField*

Response data

AdoptContext
Whether to use the presented credentials as the context for this application.

AlterationDate (MQCFST)
Alteration date of the authentication information object, in the form yyyy-mm-dd (parameter identifier: MQCA_ALTERATION_DATE).

AlterationTime (MQCFST)
Alteration time of the authentication information object, in the form hh.mm.ss (parameter identifier: MQCA_ALTERATION_TIME).

AuthInfoConnName (MQCFST)
The connection name of the authentication information object (parameter identifier: MQCA_AUTH_INFO_CONN_NAME).

The maximum length of the string is MQ_AUTH_INFO_CONN_NAME_LENGTH. On z/OS, it is MQ_LOCAL_ADDRESS_LENGTH.

This parameter is relevant only when AuthInfoType is set to *MQAIT_CRL_LDAP* or *MQAIT_IDPW_LDAP*.

AuthInfoDesc (MQCFST)
The description of the authentication information object (parameter identifier: MQCA_AUTH_INFO_DESC).

The maximum length is MQ_AUTH_INFO_DESC_LENGTH.

AuthInfoName (MQCFST)
Authentication information object name (parameter identifier: MQCA_AUTH_INFO_NAME).

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

AuthInfoType (MQCFIN)
The type of authentication information object (parameter identifier: MQIA_AUTH_INFO_TYPE).

The value can be:

MQAIT_CRL_LDAP
This authentication information object specifies Certificate Revocation Lists that are held on LDAP servers.

MQAIT_OCSP
This authentication information object specifies certificate revocation checking using OCSP.

MQAIT_IDPW_OS
This authentication information object specifies certificate revocation checking using user ID and password checking through the operating system.

MQAIT_IDPW_LDAP
This authentication information object specifies certificate revocation checking using user ID and password checking through an LDAP server.

See Securing for more information.

AuthenticationMethod (MQCFIN)
Authentication methods for user passwords (parameter identifier: MQIA_AUTHENTICATION_METHOD). Possible values are:

MQAUTHENTICATE_OS
Use the traditional UNIX password verification method.

MQAUTHENTICATE_PAM
Use the Pluggable Authentication Method to authenticate the user passwords.

You can set the PAM value only on UNIX and Linux.

This attribute is valid only for an **AuthInfoType** of *MQAIT_IDPW_OS*, and is not valid on IBM MQ for z/OS.

AuthorizationMethod (MQCFIN)
Authorization methods for the queue manager (parameter identifier MQIA_LDAP_AUTHORMD). Possible values are:

MQLDAP_AUTHORMD_OS
Use operating system groups to determine permissions associated with a user.

MQLDAP_AUTHORMD_SEARCHGRP
A group entry in the LDAP repository contains an attribute listing the Distinguished Name of all the users belonging to that group.

MQLDAP_AUTHORMD_SEARCHUSER
A user entry in the LDAP repository contains an attribute listing the Distinguished Name of all the groups to which the specified user belongs.

V 9.0.5 **MQLDAP_AUTHORMD_SRCHGRPSN**
A group entry in the LDAP repository contains an attribute listing the short user name of all the users belonging to that group.

BaseDNGroup (MQCFST)
In order to be able to find group names, this parameter must be set with the base DN to search for groups in the LDAP server (parameter identifier MQCA_LDAP_BASE_DN_GROUPS).

The maximum length of the string is MQ_LDAP_BASE_DN_LENGTH.

BaseDNUser (MQCFST)
In order to be able to find the short user name attribute (see ShortUser) this parameter must be set with the base DN to search for users within the LDAP server.

This attribute is valid only for an **AuthInfoType** of *MQAIT_IDPW_LDAP* and is mandatory (parameter identifier MQ_LDAP_BASE_DN_USERS).

The maximum length is MQ_LDAP_BASE_DN_LENGTH.

Checklocal or Checkclient (MQCFIN)
These attributes are valid only for an **AuthInfoType** of *MQAIT_IDPW_OS* or *MQAIT_IDPW_LDAP* (parameter identifier MQIA_CHECK_LOCAL_BINDING or MQIA_CHECK_CLIENT_BINDING). The possible values are:

MQCHK_NONE
Switches off checking.

MQCHK_OPTIONAL
Ensures that if a user ID and password are provided by an application, they are a valid pair, but that it is not mandatory to provide them. This option might be useful during migration, for example.

MQCHK_REQUIRED
Requires that all applications provide a valid user ID and password.

MQCHK_REQUIRED_ADMIN
Privileged users must supply a valid user ID and password, but non-privileged users are treated as with the `OPTIONAL` setting. See also the following note. z/OS (This setting is not allowed on z/OS systems.)

ClassGroup (MQCFST)
The LDAP object class used for group records in the LDAP repository (parameter identifier MQCA_LDAP_GROUP_OBJECT_CLASS).

Classuser (MQCFST)
The LDAP object class used for user records in the LDAP repository (parameter identifier MQCA_LDAP_USER_OBJECT_CLASS).

The maximum length is MQ_LDAP_CLASS_LENGTH.

FailureDelay (MQCFIN)
The failure delay (parameter identifier MQIA_AUTHENTICATION_FAIL_DELAY) when an authentication fails due to the user ID or password being incorrect, in seconds, before the failure is returned to the application.

FindGroup (MQCFST)
Name of the attribute used within an LDAP entry to determine group membership (parameter identifier MQCA_LDAP_FIND_GROUP_FIELD).

The maximum length of the string is MQ_LDAP_FIELD_LENGTH.

GroupField (MQCFST)
LDAP attribute that represents a simple name for the group (parameter identifier MQCA_LDAP_GROUP_ATTR_FIELD).

The maximum length of the string is MQ_LDAP_FIELD_LENGTH.

GroupNesting (MQCFIN)
Whether groups are members of other groups (parameter identifier MQIA_LDAP_NESTGRP). The values can be:

MQLDAP_NESTGRP_NO
Only the initially discovered groups are considered for authorization.

MQLDAP_NESTGRP_YES
The group list is searched recursively to enumerate all the groups to which a user belongs.

LDAPPassword (MQCFST)
The LDAP password (parameter identifier: MQCA_LDAP_PASSWORD).

The maximum length is MQ_LDAP_PASSWORD_LENGTH.

This parameter is relevant only when AuthInfoType is set to *MQAIT_CRL_LDAP* or *MQAIT_IDPW_LDAP*.

LDAPUserName (MQCFST)
The LDAP user name (parameter identifier: MQCA_LDAP_USER_NAME).

The Distinguished Name of the user who is binding to the directory.

The maximum length is MQ_DISTINGUISHED_NAME_LENGTH. On z/OS, it is MQ_SHORT_DNAME_LENGTH.

This parameter is relevant only when AuthInfoType is set to *MQAIT_CRL_LDAP* or *MQAIT_IDPW_LDAP*.

OCSPResponderURL (MQCFST)
The URL of the OCSP responder used to check for certificate revocation.

z/OS **QSGDisposition (MQCFIN)**
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object (that is, where it is defined and how it behaves). This parameter is valid on z/OS only. The value can be any of the following values:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

SecureComms (MQCFIN)
Whether connectivity to the LDAP server should be done securely using TLS (parameter identifier MQIA_LDAP_SECURE_COMM).

The maximum length is MQ_LDAP_SECURE_COMM_LENGTH.

ShortUser (MQCFST)
A field in the user record to be used as a short user name in IBM MQ (parameter identifier MQCA_LDAP_SHORT_USER_FIELD)..

The maximum length is MQ_LDAP_FIELD_LENGTH.

UserField (MQCFST)
Identifies the field in the LDAP user record that is used to interpret the provided user ID, only if the user ID does not contain a qualifier (parameter identifier MQCA_LDAP_USER_ATTR_FIELD).

The maximum length is MQ_LDAP_FIELD_LENGTH.

Inquire Authentication Information Object Names:

The Inquire authentication information names (MQCMD_INQUIRE_AUTH_INFO_NAMES) command asks for a list of authentication information names that match the generic authentication information name specified.

Required parameters

AuthInfoName (MQCFST)
Authentication information object name (parameter identifier: MQCA_AUTH_INFO_NAME).

Specifies the name of the authentication information object about which information is to be returned.

Generic authentication information object names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all authentication information objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

Optional parameters

AuthInfoType (MQCFIN)
Type of authentication information object. The following values are accepted:

MQAIT_CRL_LDAP
Authentication information objects specifying Certificate Revocation Lists held on LDAP servers.

MQAIT_OCSP
Authentication information objects specifying certificate revocation checking using OCSP.

MQAIT_ALL
Authentication information objects of any type. MQAIT_ALL is the default value

z/OS **CommandScope (MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

z/OS

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined as either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

Inquire Authentication Information Object Names (Response):

The response to the inquire authentication information names (MQCMD_INQUIRE_AUTH_INFO_NAMES) command consists of the response header followed by a parameter structure giving zero or more names that match the specified authentication information name.

z/OS Additionally, on z/OS only, a parameter structure, *QSGDispositions* (with the same number of entries as the *AuthInfoNames* structure), is returned. Each entry in this structure indicates the disposition of the object with the corresponding entry in the *AuthInfoNames* structure.

Always returned:
AuthInfoNames z/OS , *QSGDispositions*

Returned if requested:
None

Response data

AuthInfoNames (MQCFSL)
List of authentication information object names (parameter identifier: MQCACF_AUTH_INFO_NAMES).

z/OS

QSGDispositions (MQCFIL)
List of queue-sharing group dispositions (parameter identifier: MQIACF_QSG_DISPS).

Specifies the disposition of the object (that is, where it is defined and how it behaves). This parameter is valid on z/OS only. The value can be any of the following values:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

Inquire Authority Records on Multiplatforms: 

The Inquire Authority Records (MQCMD_INQUIRE_AUTH_RECS) command retrieves authority records associated with a profile name.

Required parameters

Options (MQCFIN)
Options to control the set of authority records that is returned (parameter identifier: MQIACF_AUTH_OPTIONS).

This parameter is required and you must include one of the following two values:

MQAUTHOPT_NAME_ALL_MATCHING
Return all profiles the names of which match the specified *ProfileName*. This means that a *ProfileName* of ABCD results in the profiles ABCD, ABC*, and AB* being returned (if ABC* and AB* have been defined as profiles).

MQAUTHOPT_NAME_EXPLICIT
Return only those profiles the names of which exactly match the *ProfileName*. No matching generic profiles are returned unless the *ProfileName* is, itself, a generic profile. You cannot specify this value and MQAUTHOPT_ENTITY_SET.

and one of the following two values:

MQAUTHOPT_ENTITY_EXPLICIT
Return all profiles the entity fields of which match the specified *EntityName*. No profiles are returned for any group in which *EntityName* is a member; only the profile defined for the specified *EntityName*.

MQAUTHOPT_ENTITY_SET
Return the profile the entity field of which matches the specified *EntityName* and the profiles pertaining to any groups in which *EntityName* is a member that contribute to the cumulative authority for the specified entity. You cannot specify this value and MQAUTHOPT_NAME_EXPLICIT.

You can also optionally specify:

MQAUTHOPT_NAME_AS_WILDCARD
Interpret *ProfileName* as a filter on the profile name of the authority records. If you do not specify this attribute and *ProfileName* contains wildcard characters, it is interpreted as a generic profile and only those authority records where the generic profile names match the value of *ProfileName* are returned.

You cannot specify MQAUTHOPT_NAME_AS_WILDCARD if you also specify MQAUTHOPT_ENTITY_SET.

ProfileName (MQCFST)
Profile name (parameter identifier: MQCACF_AUTH_PROFILE_NAME).

This parameter is the name of the profile for which to retrieve authorizations. Generic profile names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all profiles having names that start with the selected character string. An asterisk on its own matches all possible names.

If you have defined a generic profile, you can return information about it by not setting MQAUTHOPT_NAME_AS_WILDCARD in *Options*.

If you set *Options* to MQAUTHOPT_NAME_AS_WILDCARD, the only valid value for *ProfileName* is a single asterisk (*). This means that all authority records that satisfy the values specified in the other parameters are returned.

Do not specify *ProfileName* if the value of *ObjectType* is MQOT_Q_MGR.

The profile name is always returned regardless of the attributes requested.

The maximum length of the string is MQ_AUTH_PROFILE_NAME_LENGTH.

ObjectType (MQCFIN)
The type of object referred to by the profile (parameter identifier: MQIACF_OBJECT_TYPE).

The value can be any of the following values:

MQOT_ALL
All object types. MQOT_ALL is the default if you do not specify a value for *ObjectType*.

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel object.

MQOT_CLNTCONN_CHANNEL
Client-connection channel object.

MQOT_COMM_INFO
Communication information object

MQOT_LISTENER
Listener object.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process.

MQOT_Q
Queue, or queues, that match the object name parameter.

MQOT_Q_MGR
Queue manager.

MQOT_REMOTE_Q_MGR_NAME
Remote queue manager.

MQOT_SERVICE
Service object.

MQOT_TOPIC
Topic object.

Optional parameters

EntityName (MQCFST)
Entity name (parameter identifier: MQCACF_ENTITY_NAME).

Depending on the value of *EntityType*, this parameter is either:

- A principal name. This name is the name of a user for whom to retrieve authorizations to the specified object. On IBM MQ for Windows, the name of the principal can optionally include a domain name, specified in this format: `user@domain`.

- A group name. This name is the name of the user group on which to make the inquiry. You can specify one name only and this name must be the name of an existing user group.

  Windows For IBM MQ for Windows only, the group name can optionally include a domain name, specified in the following formats:

  ```
  GroupName@domain
  domain\GroupName
  ```

  The maximum length of the string is MQ_ENTITY_NAME_LENGTH.

EntityType (MQCFIN)
Entity type (parameter identifier: MQIACF_ENTITY_TYPE).

The value can be:

MQZAET_GROUP
The value of the **EntityName** parameter refers to a group name.

MQZAET_PRINCIPAL
The value of the **EntityName** parameter refers to a principal name.

ProfileAttrs (MQCFIL)
Profile attributes (parameter identifier: MQIACF_AUTH_PROFILE_ATTRS).

The attribute list might specify the following value on its own - the default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCACF_ENTITY_NAME
Entity name.

MQIACF_AUTHORIZATION_LIST
Authorization list.

MQIACF_ENTITY_TYPE
Entity type.

Note: If an entity is specified by using the parameters MQCACF_ENTITY_NAME and MQIACF_ENTITY_TYPE, then all the required parameters must be passed in first.

ServiceComponent (MQCFST)
Service component (parameter identifier: MQCACF_SERVICE_COMPONENT).

If installable authorization services are supported, this parameter specifies the name of the authorization service from which to retrieve authorization.

If you omit this parameter, the authorization inquiry is made to the first installable component for the service.

The maximum length of the string is MQ_SERVICE_COMPONENT_LENGTH.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRC_OBJECT_TYPE_ERROR
Invalid object type.

MQRC_UNKNOWN_ENTITY
User ID not authorized, or unknown.

MQRCCF_CFST_CONFLICTING_PARM
Conflicting parameters.

MQRCCF_PROFILE_NAME_ERROR
Invalid profile name.

MQRCCF_ENTITY_NAME_MISSING
Entity name missing.

MQRCCF_OBJECT_TYPE_MISSING
Object type missing.

MQRCCF_PROFILE_NAME_MISSING
Profile name missing.

Inquire Authority Records (Response) on Multiplatforms: 

The response to the Inquire Authority Records (MQCMD_INQUIRE_AUTH_RECS) command consists of the response header followed by the *QMgrName*, *Options*, *ProfileName*, and *ObjectType* structures and the requested combination of attribute parameter structures.

One PCF message is returned for each authority record that is found the profile name of which matches the options specified in the Inquire Authority Records request.

Always returned:
ObjectType, *Options*, *ProfileName*, *QMgrName*

Returned if requested:
AuthorizationList, *EntityName*, *EntityType*

Response data

AuthorizationList (MQCFIL)
Authorization list (parameter identifier: MQIACF_AUTHORIZATION_LIST).

This list can contain zero or more authorization values. Each returned authorization value means that any user ID in the specified group or principal has the authority to perform the operation defined by that value. The value can be any of the following values:

MQAUTH_NONE
The entity has authority set to 'none'.

MQAUTH_ALT_USER_AUTHORITY
Specify an alternate user ID on an MQI call.

MQAUTH_BROWSE
Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.

MQAUTH_CHANGE
Change the attributes of the specified object, using the appropriate command set.

MQAUTH_CLEAR
Clear a queue.

MQAUTH_CONNECT
Connect the application to the specified queue manager by issuing an MQCONN call.

MQAUTH_CREATE
Create objects of the specified type using the appropriate command set.

MQAUTH_DELETE
Delete the specified object using the appropriate command set.

MQAUTH_DISPLAY
Display the attributes of the specified object using the appropriate command set.

MQAUTH_INPUT
Retrieve a message from a queue by issuing an MQGET call.

MQAUTH_INQUIRE
Make an inquiry on a specific queue by issuing an MQINQ call.

MQAUTH_OUTPUT
Put a message on a specific queue by issuing an MQPUT call.

MQAUTH_PASS_ALL_CONTEXT
Pass all context.

MQAUTH_PASS_IDENTITY_CONTEXT
Pass the identity context.

MQAUTH_SET
Set attributes on a queue from the MQI by issuing an MQSET call.

MQAUTH_SET_ALL_CONTEXT
Set all context on a queue.

MQAUTH_SET_IDENTITY_CONTEXT
Set the identity context on a queue.

MQAUTH_CONTROL
For listeners and services, start and stop the specified channel, listener, or service.

For channels, start, stop, and ping the specified channel.

For topics, define, alter, or delete subscriptions.

MQAUTH_CONTROL_EXTENDED
Reset or resolve the specified channel.

MQAUTH_PUBLISH
Publish to the specified topic.

MQAUTH_SUBSCRIBE
Subscribe to the specified topic.

MQAUTH_RESUME
Resume a subscription to the specified topic.

MQAUTH_SYSTEM
Use queue manager for internal system operations.

MQAUTH_ALL
Use all operations applicable to the object.

MQAUTH_ALL_ADMIN
Use all operations applicable to the object.

MQAUTH_ALL_MQI
Use all MQI calls applicable to the object.

Use the *Count* field in the MQCFIL structure to determine how many values are returned.

EntityName (MQCFST)
Entity name (parameter identifier: MQCACF_ENTITY_NAME).

This parameter can either be a principal name or a group name.

The maximum length of the string is MQ_ENTITY_NAME_LENGTH.

EntityType (MQCFIN)
Entity type (parameter identifier: MQIACF_ENTITY_TYPE).

The value can be:

MQZAET_GROUP
The value of the **EntityName** parameter refers to a group name.

MQZAET_PRINCIPAL
The value of the **EntityName** parameter refers to a principal name.

MQZAET_UNKNOWN
On Windows, an authority record still exists from a previous queue manager which did not originally contain entity type information.

ObjectType (MQCFIN)
Object type (parameter identifier: MQIACF_OBJECT_TYPE).

The value can be:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel object.

MQOT_CLNTCONN_CHANNEL
Client-connection channel object.

MQOT_COMM_INFO
Communication information object

MQOT_LISTENER
Listener object.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process.

MQOT_Q
Queue, or queues, that match the object name parameter.

MQOT_Q_MGR
Queue manager.

MQOT_REMOTE_Q_MGR_NAME
Remote queue manager.

MQOT_SERVICE
Service object.

MQOT_TOPIC
Topic object.

Options (MQCFIN)
Options used to indicate the level of information that is returned (parameter identifier: MQIACF_AUTH_OPTIONS).

ProfileName (MQCFST)
Profile name (parameter identifier: MQCACF_AUTH_PROFILE_NAME).

The maximum length of the string is MQ_AUTH_PROFILE_NAME_LENGTH.

QMgrName (MQCFST)
Name of the queue manager on which the Inquire command is issued (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Inquire Authority Service on Multiplatforms: 

The Inquire Authority Service (MQCMD_INQUIRE_AUTH_SERVICE) command retrieves information about the level of function supported by installed authority managers.

Required parameters

AuthServiceAttrs (MQCFIL)
Authority service attributes (parameter identifier: MQIACF_AUTH_SERVICE_ATTRS).

The attribute list might specify the following value on its own - default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQIACF_INTERFACE_VERSION
Current interface version of the authority service.

MQIACF_USER_ID_SUPPORT
Whether the authority service supports user IDs.

Optional parameters

ServiceComponent (MQCFST)
Name of authorization service (parameter identifier: MQCACF_SERVICE_COMPONENT).

The name of the authorization service which is to handle the Inquire Authority Service command.

If this parameter is omitted, or specified as a blank or null string, the inquire function is called in each installed authorization service in reverse order to the order in which the services have been installed, until all authorization services have been called or until one returns a value of MQZCI_STOP in the Continuation field.

The maximum length of the string is MQ_SERVICE_COMPONENT_LENGTH.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRC_SELECTOR_ERROR
Attribute selector not valid.

MQRC_UNKNOWN_COMPONENT_NAME
Unknown service component name.

Inquire Authority Service (Response) on Multiplatforms: Multi

The response to the Inquire Authority Service (MQCMD_INQUIRE_AUTH_SERVICE) command consists of the response header followed by the *ServiceComponent* structure and the requested combination of attribute parameter structures.

Always returned:
ServiceComponent

Returned if requested:
InterfaceVersion, *UserIDSupport*

Response data

InterfaceVersion (MQCFIN)
Interface version (parameter identifier: MQIACF_INTERFACE_VERSION).

This parameter is the current interface version of the OAM.

ServiceComponent (MQCFSL)
Name of authorization service (parameter identifier: MQCACF_SERVICE_COMPONENT).

If you included a specific value for *ServiceComponent* on the Inquire Authority Service command, this field contains the name of the authorization service that handled the command. If you did not include a specific value for *ServiceComponent* on the Inquire Authority Service command, the list contains the names of all the installed authorization services.

If there is no OAM or if the OAM requested in the ServiceComponent does not exist this field is blank.

The maximum length of each element in the list is MQ_SERVICE_COMPONENT_LENGTH.

UserIDSupport (MQCFIN)
User ID support (parameter identifier: MQIACF_USER_ID_SUPPORT).

The value can be:

MQUIDSUPP_YES
The authority service supports user IDs.

MQUIDSUPP_NO
The authority service does not support user IDs.

Inquire CF Structure on z/OS: z/OS

The Inquire CF Structure (MQCMD_INQUIRE_CF_STRUC) command returns information about the attributes of one or more CF application structures.

Note: This command is supported only on z/OS when the queue manager is a member of a queue-sharing group.

Required parameters

CFStrucName (MQCFST)
CF Structure name (parameter identifier: MQCA_CF_STRUC_NAME).

Specifies the name of the CF application structure about which information is to be returned.

Generic CF structure names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all CF application structures having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length is MQ_CF_STRUC_NAME_LENGTH.

Optional parameters

CFStrucAttrs (MQCFIL)
CF application structure attributes (parameter identifier: MQIACF_CF_STRUC_ATTRS).

The attribute list might specify the following value on its own - default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_ALTERATION_DATE
The date on which the definition was last altered.

MQCA_ALTERATION_TIME
The time at which the definition was last altered.

MQIA_CF_CFCONLOS
The action to be taken when the queue manager loses connectivity to the CF application structure.

MQIA_CF_LEVEL
Functional capability level for the CF application structure.

MQIA_CF_OFFLOAD
The shared message data set OFFLOAD property for the CF application structure.

MQIA_CF_RECOVER
Whether CF recovery for the application structure is supported.

MQIA_CF_RECAUTO
Whether automatic recovery action is taken when a structure is failed or when a queue manager loses connectivity to the structure and no systems in the SysPlex have connectivity to the Coupling Facility the structure is located in.

MQIACF_CF_SMDS_BLOCK_SIZE
The shared message data set DSGROUP property for the CF application structure.

MQIA_CF_SMDS_BUFFERS
The shared message data set DSGROUP property for the CF application structure.

MQIACF_CF_SMDS_EXPAND
The shared message data set DSEXPAND property for the CF application structure.

MQCACF_CF_SMDS_GENERIC_NAME
The shared message data set DSBUFS property for the CF application structure.

MQCA_CF_STRUC_DESC
Description of CF application structure.

MQCA_CF_STRUC_NAME
Name of CF application structure.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *CFStrucAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1601 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the **StringFilterCommand** parameter.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed

in *CFStrucAttrs* except MQCA_CF_STRUC_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See “MQCFSF - PCF string filter parameter” on page 1608 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter.

Inquire CF Structure (Response) on z/OS: z/OS

The response to the Inquire CF Structure (MQCMD_INQUIRE_CF_STRUC) command consists of the response header followed by the *CFStrucName* structure and the requested combination of attribute parameter structures.

If a generic CF application structure name was specified, one such message is generated for each CF application structure found.

Always returned:
CFStrucName

Returned if requested:
AlterationDate, *AlterationTime*, *CFConlos*, *CFLevel*, *CFStrucDesc*, *DSBLOCK*, *DSBUFS*, *DSEXPAND*, *DSGROUP*, *OFFLD1SZ*, *OFFLD12SZ*, *OFFLD3SZ*, *OFFLD1TH*, *OFFLD2TH*, *OFFLD3TH*, *Offload*, *RCVDATE*, *RCVTIME*, *Recauto*, *Recovery*

Response data

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date on which the definition was last altered, in the form `yyyy-mm-dd`.

The maximum length of the string is MQ_DATE_LENGTH.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time at which the definition was last altered, in the form `hh.mm.ss`.

The maximum length of the string is MQ_TIME_LENGTH.

CFConlos (MQCFIN)
The CFConlos property (parameter identifier: MQIA_CF_CFCONLOS).

Specifies the action to be taken when a queue manager loses connectivity to the CF structure. The value can be any of the following values:

MQCFCONLOS_TERMINATE
The queue manager will terminate when connectivity to the structure is lost.

MQCFCONLOS_TOLERATE
The queue manager will tolerate loss of connectivity to the structure without terminating.

MQCFCONLOS_ASQMGR
The action taken is based on the setting of the CFCONLOS queue manager attribute

This parameter is only valid from CFLEVEL(5).

CFLevel (MQCFIN)
The functional capability level for this CF application structure (parameter identifier: MQIA_CF_LEVEL).

Specifies the functional capability level for the CF application structure. The value can be any of the following values:

1 A CF structure that can be "auto-created" by a queue manager at command level 520.

2 A CF structure at command level 520 that can only be created or deleted by a queue manager at command level 530 or greater. This level is the default *CFLevel* for queue managers at command level 530 or greater.

3

A CF structure at command level 530. This *CFLevel* is required if you want to use persistent messages on shared queues, or for message grouping, or both.

4

A CF structure at command level 600. This *CFLevel* can be used for persistent messages or for messages longer than 64 512 bytes.

5

A CF structure at command level 710. This *CFLevel* supports shared message data sets (SMDS) and Db2 for offloading messages.

Structures are required to be at CFLEVEL(5) to support toleration of loss of connectivity.

CFStrucDesc (MQCFST)
The description of the CF structure (parameter identifier: MQCA_CF_STRUC_DESC).

The maximum length is MQ_CF_STRUC_DESC_LENGTH.

CFStrucName (MQCFST)
CF Structure name (parameter identifier: MQCA_CF_STRUC_NAME).

The maximum length is MQ_CF_STRUC_NAME_LENGTH.

DSBLOCK (MQCFIN)
The CF DSBLOCK property (parameter identifier: MQIACF_CF_SMDS_BLOCK_SIZE).

The returned value is one of the following constants: MQDSB_8K, MQDSB_16K, MQDSB_32K, MQDSB_64K, MQDSB_128K, MQDSB_256K, MQDSB_512K, MQDSB_1024K, MQDSB_1M.

DSBUFS (MQCFIN)
The CF DSBUFS property (parameter identifier: MQIA_CF_SMDS_BUFFERS).

The returned value is in the range 0 - 9999.

The value is the number of buffers to be allocated in each queue manager for accessing shared message data sets. The size of each buffer is equal to the logical block size.

DSEXPAND (MQCFIN)
The CF DSEXPAND property (parameter identifier: MQIACF_CF_SMDS_EXPAND).

MQDSE_YES
The data set can be expanded.

MQDSE_NO
The data set cannot be expanded.

MQDSE_DEFAULT
Only returned on Inquire CF Struct when not explicitly set

DSGROUP (MQCFST)
The CF DSGROUP property (parameter identifier: MQCACF_CF_SMDS_GENERIC_NAME).

The returned value is a string containing a generic data set name used for the group of shared message data sets associated with this CF structure.

OFFLD1SZ (MQCFST)
The CF OFFLD1SZ property (parameter identifier: MQCACF_CF_OFFLOAD_SIZE1).

The returned value is a string in the range 0K - 64K.

Returned if the MQIACF_ALL or MQIA_CF_OFFLOAD parameters are specified.

The maximum length is 3.

OFFLD2SZ (MQCFST)
The CF OFFLD2SZ property (parameter identifier: MQCACF_CF_OFFLOAD_SIZE2).

The returned value is a string in the range 0K - 64K.

Returned if the MQIACF_ALL or MQIA_CF_OFFLOAD parameters are specified.

The maximum length is 3.

OFFLD3SZ (MQCFST)
The CF OFFLD3SZ property (parameter identifier: MQCACF_CF_OFFLOAD_SIZE3).

The returned value is a string in the range 0K - 64K.

Returned if the MQIACF_ALL or MQIA_CF_OFFLOAD parameters are specified.

The maximum length is 3.

OFFLD1TH (MQCFIN)
The CF OFFLD1TH property (parameter identifier: MQIA_CF_OFFLOAD_THRESHOLD1).

The returned value is in the range 0 - 100.

Returned if the MQIACF_ALL or MQIA_CF_OFFLOAD parameters are specified.

OFFLD2TH (MQCFIN)
The CF OFFLD2TH property (parameter identifier: MQIA_CF_OFFLOAD_THRESHOLD2).

The returned value is in the range 0 - 100.

Returned if the MQIACF_ALL or MQIA_CF_OFFLOAD parameters are specified.

OFFLD3TH (MQCFIN)
The CF OFFLD3TH property (parameter identifier: MQIA_CF_OFFLOAD_THRESHOLD3).

The returned value is in the range 0 - 100.

Returned if the MQIACF_ALL or MQIA_CF_OFFLOAD parameters are specified.

Offload (MQCFIN)
The CF OFFLOAD property (parameter identifier: MQIA_CF_OFFLOAD).

The returned values can be:

MQCFOFFLD_DB2
Large shared messages can be stored in Db2.

MQCFOFFLD_SMDS
Large shared messages can be stored in z/OS shared message data sets.

MQCFOFFLD_NONE
Used when the property *Offload* has not been explicitly set.

RCVDATE (MQCFST)
The recovery start date (parameter identifier: MQCACF_RECOVERY_DATE).

If recovery is currently enabled for the data set, this indicates the date when it was activated, in the form yyyy-mm-dd. If recovery is not enabled, this is displayed as RCVDATE().

RCVTIME (MQCFST)
The recovery start time (parameter identifier: MQCACF_RECOVERY_TIME).

If recovery is currently enabled for the data set, this indicates the time when it was activated, in the form hh.mm.ss. If recovery is not enabled, this is displayed as RCVTIME().

Recauto (MQCFIN)
Recauto (parameter identifier: MQIA_CF_RECAUTO).

Indicates whether automatic recovery action is taken when a queue manager detects that the structure is failed, or when a queue manager loses connectivity to the structure and no systems in the SysPlex have connectivity to the Coupling Facility that the structure is allocated in. The value can be:

MQRECAUTO_YES
The structure and associated shared message data sets which also need recovery will be automatically recovered.

MQRECAUTO_NO
The structure will not be automatically recovered.

Recovery (MQCFIN)
Recovery (parameter identifier: MQIA_CF_RECOVER).

Specifies whether CF recovery is supported for the application structure. The value can be:

MQCFR_YES
Recovery is supported.

MQCFR_NO
Recovery is not supported.

Inquire CF Structure Names on z/OS: z/OS

The Inquire CF Structure Names (MQCMD_INQUIRE_CF_STRUC_NAMES) command inquires for a list of CF application structure names that match the generic CF structure name specified.

Note: This command is supported only on z/OS when the queue manager is a member of a queue-sharing group.

Required parameters

CFStrucName (MQCFST)
CF Structure name (parameter identifier: MQCA_CF_STRUC_NAME).

Specifies the name of the CF application structure about which information is to be returned.

Generic CF structure names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all CF application structures having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length is MQ_CF_STRUC_NAME_LENGTH.

Inquire CF Structure Names (Response) on z/OS: z/OS

The response to the Inquire CF Structure Names (MQCMD_INQUIRE_CF_STRUC_NAMES) command consists of the response header followed by a single parameter structure giving zero or more names that match the specified CF application structure name.

Always returned:
CFStrucNames

Returned if requested:
None

Response data

CFStrucNames (MQCFSL)
List of CF application structure names (parameter identifier: MQCACF_CF_STRUC_NAMES).

Inquire CF Structure Status on z/OS: z/OS

The Inquire CF Structure Status (MQCMD_INQUIRE_CF_STRUC_STATUS) command inquires about the status of a CF application structure.

Note: This command is supported only on z/OS when the queue manager is a member of a queue-sharing group.

Required parameters

CFStrucName (MQCFST)
CF Structure name (parameter identifier: MQCA_CF_STRUC_NAME).

Specifies the name of the CF application structure for which status information is to be returned.

Generic CF structure names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all CF application structures having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length is MQ_CF_STRUC_NAME_LENGTH.

Optional parameters

CFStatusType (MQCFIN)
Status information type (parameter identifier: MQIACF_CF_STATUS_TYPE).

Specifies the type of status information you want to be returned. You can specify one of the following:

MQIACF_CF_STATUS_SUMMARY
Summary status information for the CF application structure. MQIACF_CF_STATUS_SUMMARY is the default.

MQIACF_CF_STATUS_CONNECT
Connection status information for each CF application structure for each active queue manager.

MQIACF_CF_STATUS_BACKUP
Backup status information for each CF application structure.

MQIACF_CF_STATUS_SMDS
Shared message data set information for each CF application structure.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter in the

response data except MQIACF_CF_STATUS_TYPE. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1601 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the **StringFilterCommand** parameter.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter in the response data except MQCA_CF_STRUC_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1608 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter.

Inquire CF Structure Status (Response) on z/OS: z/OS

The response to the Inquire CF Structure Status (MQCMD_INQUIRE_CF_STRUC_STATUS) command consists of the response header followed by the *CFStrucName* and *CFStatusType* structures and a set of attribute parameter structures determined by the value of *CFStatusType* in the Inquire command.

Always returned:
CFStrucName, *CFStatusType*.

CFStatusType specifies the type of status information being returned. The value can be any of the following values:

MQIACF_CF_STATUS_SUMMARY
Summary status information for the CF application structure. This is the default.

MQIACF_CF_STATUS_CONNECT
Connection status information for each CF application structure for each active queue manager.

MQIACF_CF_STATUS_BACKUP
Backup status information for each CF application structure.

MQIACF_CF_STATUS_SMDS
Shared message data set information for each CF application structure.

Returned if *CFStatusType* is MQIACF_CF_STATUS_SUMMARY:
CFStrucStatus, *CFStrucType*, *EntriesMax*, *EntriesUsed*, *FailDate*, *FailTime*, *OffLdUse*, *SizeMax*, *SizeUsed*

Returned if *CFStatusType* is MQIACF_CF_STATUS_CONNECT:
CFStrucStatus, *FailDate*, *FailTime*, *QMgrName*, *SysName*

Returned if *CFStatusType* is MQIACF_CF_STATUS_BACKUP:
BackupDate, *BackupEndRBA*, *BackupSize*, *BackupStartRBA*, *BackupTime*, *CFStrucStatus*, *FailDate*, *FailTime*, *LogQMgrNames*, *QmgrName*

Returned if *CFStatusType* is MQIACF_CF_STATUS_SMDS:
Access, *FailDate*, *FailTime*, *RcvDate*, *RcvTime*, *CFStrucStatus*

Response data

Access (MQCFIN)
Availability of the shared message data set (parameter identifier: MQIACF_CF_STRUC_ACCESS).

MQCFACCESS_ENABLED
The shared message data set is either available for use, or is to be enabled after previously being disabled, or access to the shared message data set is to be retried following an error.

MQCFACCESS_SUSPENDED
The shared message data set is unavailable because of an error.

MQCFACCESS_DISABLED
The shared message data set is either disabled, or is to be set as disabled.

BackupDate (MQCFST)
The date, in the form yyyy-mm-dd, on which the last successful backup was taken for this CF application structure (parameter identifier: MQCACF_BACKUP_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

BackupEndRBA (MQCFST)
The backup data set end RBA for the end of the last successful backup taken for this CF application structure (parameter identifier: MQCACF_CF_STRUC_BACKUP_END).

The maximum length of the string is MQ_RBA_LENGTH.

BackupSize (MQCFIN)
The size, in megabytes, of the last successful backup taken for this CF application structure (parameter identifier: MQIACF_CF_STRUC_BACKUP_SIZE).

BackupStartRBA (MQCFST)
The backup data set start RBA for the start of the last successful backup taken for this CF application structure (parameter identifier: MQCACF_CF_STRUC_BACKUP_START).

The maximum length of the string is MQ_RBA_LENGTH.

BackupTime (MQCFST)
The end time, in the form hh.mm.ss, of the last successful backup taken for this CF application structure (parameter identifier: MQCACF_BACKUP_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

CFStatusType (MQCFIN)
Status information type (parameter identifier: MQIACF_CF_STATUS_TYPE).

Specifies the type of status information being returned. The value can be any of the following values:

MQIACF_CF_STATUS_SUMMARY
Summary status information for the CF application structure.
MQIACF_CF_STATUS_SUMMARY is the default.

MQIACF_CF_STATUS_CONNECT
Connection status information for each CF application structure for each active queue manager.

MQIACF_CF_STATUS_BACKUP
Back up status information for each CF application structure.

MQIACF_CF_STATUS_SMDS
Shared message data set information for each CF application structure.

CFStrucName (MQCFST)
CF Structure name (parameter identifier: MQCA_CF_STRUC_NAME).

The maximum length is MQ_CF_STRUC_NAME_LENGTH.

CFStrucStatus (MQCFIN)
CF Structure status (parameter identifier: MQIACF_CF_STRUC_STATUS).

The status of the CF application structure.

If *CFStatusType* is MQIACF_CF_STATUS_SUMMARY, the value can be:

MQCFSTATUS_ACTIVE
The structure is active.

MQCFSTATUS_FAILED
The structure has failed.

MQCFSTATUS_NOT_FOUND
The structure is not allocated in the CF, but has been defined to Db2.

MQCFSTATUS_IN_BACKUP
The structure is in the process of being backed up.

MQCFSTATUS_IN_RECOVER
The structure is in the process of being recovered.

MQCFSTATUS_UNKNOWN
The status of the CF structure is unknown because, for example, Db2 might be unavailable.

If *CFStatusType* is MQIACF_CF_STATUS_CONNECT, the value can be:

MQCFSTATUS_ACTIVE
The structure is connected to this queue manager.

MQCFSTATUS_FAILED
The queue manager connection to this structure has failed.

MQCFSTATUS_NONE
The structure has never been connected to this queue manager.

If *CFStatusType* is MQIACF_CF_STATUS_BACKUP, the value can be:

MQCFSTATUS_ACTIVE
The structure is active.

MQCFSTATUS_FAILED
The structure has failed.

MQCFSTATUS_NONE
The structure has never been backed up.

MQCFSTATUS_IN_BACKUP
The structure is in the process of being backed up.

MQCFSTATUS_IN_RECOVER
The structure is in the process of being recovered.

If *CFStatusType* is MQIACF_CF_STATUS_SMDS, the value can be:

MQCFSTATUS_ACTIVE
The shared message data set is available for normal use

MQCFSTATUS_FAILED
The shared message data set is in an unusable state and probably requires recovery.

MQCFSTATUS_IN_RECOVER
The shared message data set is in the process of being recovered (by way of a RECOVER CFSTRUCT command).

MQCFSTATUS_NOT_FOUND
The data set has never been used, or the attempt to open it for the first time failed.

MQCFSTATUS_RECOVERED
The data set has been recovered or otherwise repaired, and is ready for use again, but requires some restart processing the next time it is opened. This restart processing ensures that obsolete references to any deleted messages have been removed from the coupling facility structure before the data set is made available again. The restart processing also rebuilds the data set space map.

MQCFSTATUS_EMPTY
The data set contains no messages. The data set is put into this state if it is closed normally by the owning queue manager at a time when it does not contain any messages. It can also be put into EMPTY state when the previous data set contents are to be discarded because the application structure has been emptied (using **RECOVER CFSTRUCT** with TYPE PURGE or, for a nonrecoverable structure only, by deleting the previous instance of the structure). The next time the data set is opened by its owning queue manager, the space map is reset to empty, and the status is changed to ACTIVE. As the previous data set contents are no longer required, a data set in this state can be replaced with a newly allocated data set, for example to change the space allocation or move it to another volume.

MQCFSTATUS_NEW
The data set is being opened and initialized for the first time, ready to be made active.

CFStrucType (MQCFIN)
CF Structure type (parameter identifier: MQIACF_CF_STRUC_TYPE).

The value can be:

MQCFTYPE_ADMIN
MQCFTYPE_ADMIN is the CF administration structure.

MQCFTYPE_APPL
MQCFTYPE_APPL is a CF application structure.

EntriesMax (MQCFIN)
Number of CF list entries defined for this CF application structure (parameter identifier: MQIACF_CF_STRUC_ENTRIES_MAX).

EntriesUsed (MQCFIN)
Number of CF list entries defined for this CF application structure that are in use (parameter identifier: MQIACF_CF_STRUC_ENTRIES_USED).

FailDate (MQCFST)
The date, in the form `yyyy-mm-dd`, on which this CF application structure failed (parameter identifier: MQCACF_FAIL_DATE).

If *CFStatusType* is MQIACF_CF_STATUS_CONNECT, it is the date on which the queue manager lost connectivity to this application structure. For the other values of *CFStatusType*, it is the date on which this CF application structure failed. This parameter is only applicable when *CFStrucStatus* is MQCFSTATUS_FAILED or MQCFSTATUS_IN_RECOVER.

The maximum length of the string is MQ_DATE_LENGTH.

FailTime (MQCFST)
The time, in the form `hh.mm.ss`, that this CF application structure failed (parameter identifier: MQCACF_FAIL_TIME).

If *CFStatusType* is MQIACF_CF_STATUS_CONNECT, it is the time that the queue manager lost connectivity to this application structure. For the other values of *CFStatusType*, it is the time that this CF application structure failed. This parameter is only applicable when *CFStrucStatus* is MQCFSTATUS_FAILED or MQCFSTATUS_IN_RECOVER.

The maximum length of the string is MQ_TIME_LENGTH.

LogQMgrNames (MQCFSL)
A list of queue managers, the logs of which are required to perform a recovery (parameter identifier: MQCACF_CF_STRUC_LOG_Q_MGRS).

The maximum length of each name is MQ_Q_MGR_NAME_LENGTH.

OffLdUse (MQCFIN)
Offload usage (parameter identifier: MQIA_CF_OFFLDUSE).

Indicates whether any offloaded large message data might currently exist in shared message data sets, Db2, or both. The value can be any of the following values:

MQCFOFFLD_DB2
Large shared messages are stored in Db2.

MQCFOFFLD_SMDS
Large shared messages are stored in z/OS shared message data sets.

MQCFOFFLD_NONE
Use on DISPLAY CFSTRUCT when the property has not been explicitly set.

MQCFOFFLD_BOTH
There might be large shared messages stored in both Db2, and shared message data sets.

Value cannot be set unless CFLEVEL(5) is defined.

QMgrName (MQCFST)
Queue manager name (parameter identifier: MQCA_Q_MGR_NAME).

This parameter is the name of the queue manager. If *CFStatusType* is MQIACF_CF_STATUS_BACKUP, it is the name of the queue manager that took the last successful backup.

The maximum length is MQ_Q_MGR_NAME_LENGTH.

RcvDate (MQCFST)
The recovery start date (parameter identifier: MQCACF_RECOVERY_DATE).

If recovery is currently enabled for the data set, this indicates the date when it was activated, in the form yyyy-mm-dd.

RcvTime (MQCFST)
The recovery start time (parameter identifier: MQCACF_RECOVERY_TIME).

If recovery is currently enabled for the data set, this indicates the time when it was activated, in the form hh.mm.ss.

SizeMax (MQCFIN)
Size of the CF application structure (parameter identifier: MQIACF_CF_STRUC_SIZE_MAX).

This parameter is the size, in kilobytes, of the CF application structure.

SizeUsed (MQCFIN)
Percentage of the CF application structure that is in use (parameter identifier: MQIACF_CF_STRUC_SIZE_USED).

This parameter is the percentage of the size of the CF application structure that is in use.

SysName (MQCFST)
Queue manager name (parameter identifier: MQCACF_SYSTEM_NAME).

This parameter is the name of the z/OS image of the queue manager that last connected to the CF application structure.

The maximum length is MQ_SYSTEM_NAME_LENGTH.

SizeMax (MQCFIN)
Size of the CF application structure (parameter identifier: MQIACF_CF_STRUC_SIZE_MAX).

This parameter is the size, in kilobytes, of the CF application structure.

Inquire Channel:

The Inquire Channel (MQCMD_INQUIRE_CHANNEL) command inquires about the attributes of IBM MQ channel definitions.

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Generic channel names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all channels having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

ChannelAttrs (MQCFIL)
Channel attributes (parameter identifier: MQIACF_CHANNEL_ATTRS).

The attribute list can specify the following value on its own - default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the parameters in the following table:

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver	AMQP
MQCA_ALTERATION_DATE Date on which the definition was last altered	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
MQCA_ALTERATION_TIME Time at which the definition was last altered	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
MQCA_CERT_LABEL Certificate label	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
MQCA_CLUSTER_NAME Name of local queue manager							✓	✓	
MQCA_CLUSTER_NAMELIST Name of local queue manager							✓	✓	
MQCA_Q_MGR_NAME Name of local queue manager					✓				

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver	AMQP V 9.0.0
MQCACH_CHANNEL_NAME Channel name. You cannot use this attribute as a filter keyword.	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
MQCACH_CONNECTION_NAME Connection name	✓	✓		✓	✓		✓	✓	
MQCACH_DESC Description	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
MQCACH_LOCAL_ADDRESS Local communications address for the channel	✓	✓		✓	✓		✓	✓	✓ 9.0.0
MQCACH_MCA_NAME Message channel agent name	✓	✓		✓			✓		
MQCACH_MCA_USER_ID MCA user identifier	✓	✓	✓	✓		✓	✓	✓	✓ 9.0.0
MQCACH_MODE_NAME Mode name	✓	✓		✓	✓		✓	✓	
MQCACH_MR_EXIT_NAME Message-retry exit name			✓	✓				✓	
MQCACH_MR_EXIT_USER_DATA Message-retry exit name			✓	✓				✓	
MQCACH_MSG_EXIT_NAME Message exit name	✓	✓	✓	✓			✓	✓	
MQCACH_MSG_EXIT_USER_DATA Message exit user data	✓	✓	✓	✓			✓	✓	
MQCACH_PASSWORD Password	✓	✓		✓	✓		✓		
MQCACH_RCV_EXIT_NAME Receive exit name	✓	✓	✓	✓	✓	✓	✓	✓	
MQCACH_RCV_EXIT_USER_DATA Receive exit user data	✓	✓	✓	✓	✓	✓	✓	✓	
MQCACH_SEC_EXIT_NAME Security exit name	✓	✓	✓	✓	✓	✓	✓	✓	

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver	V 9.0.0 AMQP
MQCACH_SEC_EXIT_USER_DATA Security exit user data	✓	✓	✓	✓	✓	✓	✓	✓	
MQCACH_SEND_EXIT_NAME Send exit name	✓	✓	✓	✓	✓	✓	✓	✓	
MQCACH_SEND_EXIT_USER_DATA Send exit user data	✓	✓	✓	✓	✓	✓	✓	✓	
MQCACH_SSL_CIPHER_SPEC TLS cipher spec	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
MQCACH_SSL_PEER_NAME TLS peer name	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
MQCACH_TP_NAME Transaction program name	✓	✓		✓	✓	✓	✓	✓	
V 9.0.0 **MQCACH_TP_ROOT** Topic root for AMQP channel									✓ 9.0.0
MQCACH_USER_ID User identifier	✓	✓		✓	✓		✓		
MQCACH_XMIT_Q_NAME Transmission queue name	✓	✓							
MQIA_MONITORING_CHANNEL Online monitoring data collection	✓	✓	✓	✓		✓	✓	✓	
MQIA_PROPERTY_CONTROL Property control attribute	✓	✓					✓	✓	
MQIA_STATISTICS_CHANNEL Online statistics collection	✓	✓	✓	✓			✓	✓	
MQIA_USE_DEAD_LETTER_Q Determines whether the dead-letter queue is used when messages cannot be delivered by channels.	✓	✓	✓	✓			✓	✓	
V 9.0.0 **MQIACH_AMQP_KEEP_ALIVE** AMQP channel keep alive interval									✓ 9.0.0

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver	AMQP V 8.0.0
MQIACH_BATCH_HB Value to use for batch heartbeating	✓	✓					✓	✓	
MQIACH_BATCH_INTERVAL Batch wait interval (seconds)	✓	✓					✓	✓	
MQIACH_BATCH_DATA_LIMIT Batch data limit (kilobytes)	✓	✓					✓	✓	
MQIACH_BATCH_SIZE Batch size	✓	✓	✓	✓			✓	✓	
MQIACH_CHANNEL_TYPE Channel type	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
MQIACH_CLIENT_CHANNEL_WEIGHT Client Channel Weight					✓				
MQIACH_CLWL_CHANNEL_PRIORITY Cluster workload channel priority							✓	✓	
MQIACH_CLWL_CHANNEL_RANK Cluster workload channel rank							✓	✓	
MQIACH_CLWL_CHANNEL_WEIGHT Cluster workload channel weight							✓	✓	
MQIACH_CONNECTION_AFFINITY Connection Affinity					✓				
MQIACH_DATA_CONVERSION Whether sender must convert application data	✓	✓					✓	✓	
MQIACH_DEF_RECONNECT Default reconnection option					✓				
MQIACH_DISC_INTERVAL Disconnection interval	✓	✓				✓	✓	✓	
MQIACH_HB_INTERVAL Heartbeat interval (seconds)	✓	✓	✓	✓	✓	✓	✓	✓	

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver	AMQP V 9.0.0
MQIACH_HDR_COMPRESSION List of header data compression techniques supported by the channel	✓	✓	✓	✓	✓	✓	✓	✓	
MQIACH_KEEP_ALIVE_INTERVAL KeepAlive interval	✓	✓	✓	✓	✓	✓	✓	✓	
MQIACH_LONG_RETRY Long retry count	✓	✓					✓	✓	
MQIACH_LONG_TIMER Long timer	✓	✓					✓	✓	
MQIACH_MAX_INSTANCES Maximum number of simultaneous instances of a server-connection channel that can be started.						✓			✓ 9.0.0
MQIACH_MAX_INSTS_PER_CLIENT Maximum number of simultaneous instances of a server-connection channel that can be started from a single client.						✓			
MQIACH_MAX_MSG_LENGTH Maximum message length	✓	✓	✓	✓	✓	✓	✓	✓	✓ 9.0.0
MQIACH_MCA_TYPE MCA type	✓	✓		✓			✓	✓	
MQIACH_MR_COUNT Message retry count			✓	✓				✓	
MQIACH_MSG_COMPRESSION List of message data compression techniques supported by the channel	✓	✓	✓	✓	✓	✓	✓	✓	
MQIACH_MR_INTERVAL Message retry interval (milliseconds)			✓	✓				✓	
MQIACH_NPM_SPEED Speed of nonpersistent messages	✓	✓	✓	✓			✓	✓	

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver	AMQP V 9.0.0
V 9.0.0 **MQIACH_PORT** AMQP port number									✓ 9.0.0
MQIACH_PUT_AUTHORITY Put authority			✓	✓		✓		✓	
MQIACH_RESET_REQUESTED Sequence number of outstanding request when a RESET CHANNEL command is used	✓	✓	✓	✓			✓	✓	
MQIACH_SEQUENCE_NUMBER_WRAP Sequence number wrap	✓	✓	✓	✓			✓	✓	
MQIACH_SHARING_CONVERSATIONS Value of Sharing Conversations						✓			
MQIACH_SHORT_RETRY Short retry count	✓	✓					✓	✓	
MQIACH_SHORT_TIMER Short timer	✓	✓					✓	✓	
MQIACH_SSL_CLIENT_AUTH TLS client authentication	✓	✓	✓	✓		✓		✓	✓ 9.0.0
V 9.0.0 **MQIACH_USE_CLIENT_ID** Specify that the client ID is used for authorization checks for an AMQP channel									✓ 9.0.0
MQIACH_XMIT_PROTOCOL_TYPE Transport (transmission protocol) type	✓	✓	✓	✓	✓	✓	✓	✓	

Note:

1. Only one of the following parameters can be specified:
   - MQCACH_JAAS_CONFIG
   - MQCACH_MCA_USER_ID
   - MQIACH_USE_CLIENT_ID

If none of these parameters is specified, no authentication is performed. If MQCACH_JAAS_CONFIG is specified, the client flows a user name and password, in all other cases the flowed user name is ignored.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

If this parameter is present, eligible channels are limited to the specified type. Any attribute selector specified in the *ChannelAttrs* list which is only valid for channels of a different type or types is ignored; no error is raised.

If this parameter is not present (or if MQCHT_ALL is specified), channels of all types other than MQCHT_MQTT are eligible. Each attribute specified must be a valid channel attribute selector (that is, it must be one from the following list), but it might not be applicable to all (or any) of the channels returned. Channel attribute selectors that are valid but not applicable to the channel are ignored, no error messages occur, and no attribute is returned.

The value can be:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLNTCONN
Client connection.

MQCHT_CLUSRCVR
Cluster-receiver.

MQCHT_CLUSSDR
Cluster-sender.

V 9.0.0 **MQCHT_AMQP**
AMQP channel.

MQCHT_MQTT
Telemetry channel.

MQCHT_ALL
All types other than MQCHT_MQTT.

The default value if this parameter is not specified is MQCHT_ALL.

Note: If this parameter is present, it must occur immediately after the **ChannelName** parameter on platforms other than z/OS otherwise resulting in a MQRCCF_MSG_LENGTH_ERROR error message.

z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.

- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *ChannelAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1601 for information about using this filter condition.

If you specify an integer filter for channel type, you cannot also specify the **ChannelType** parameter.

If you specify an integer filter, you cannot also specify a string filter using the **StringFilterCommand** parameter.

z/OS

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined as either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

You cannot use *QSGDisposition* as a parameter to filter on.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed

in *ChannelAttrs* except MQCACH_CHANNEL_NAME and MQCACH_MCA_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1608 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

Inquire Channel (MQTT): AIX Linux Windows

The Inquire Channel (MQCMD_INQUIRE_CHANNEL) command inquires about the attributes of IBM MQ channel definitions.

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Generic channel names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all channels having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

If this parameter is present, eligible channels are limited to the specified type. Any attribute selector specified in the *ChannelAttrs* list which is only valid for channels of a different type or types is ignored; no error is raised.

If this parameter is not present (or if MQCHT_ALL is specified), channels of all types are eligible. Each attribute specified must be a valid channel attribute selector (that is, it must be one from the following list), but it might not be applicable to all (or any) of the channels returned. Channel attribute selectors that are valid but not applicable to the channel are ignored, no error messages occur, and no attribute is returned.

The value must be:

MQCHT_MQTT
Telemetry channel.

Optional parameters

ChannelAttrs (MQCFIL)
Channel attributes (parameter identifier: MQIACF_CHANNEL_ATTRS).

The attribute list can specify the following value on its own - default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following parameters:

MQCA_SSL_KEY_REPOSITORY
TLS Key Repository

MQCACH_CHANNEL_NAME
Channel name. You cannot use this attribute as a filter keyword.

MQCACH_JAAS_CONFIG
The file path of the JAAS configuration

MQCACH_LOCAL_ADDRESS
Local communications address for the channel

MQCACH_MCA_USER_ID
MCA user identifier.

MQCACH_SSL_CIPHER_SPEC
TLS cipher spec.

MQCACH_SSL_KEY_PASSPHRASE
TLS key passphrase.

MQIACH_BACKLOG
The number of concurrent connection requests that the channel supports.

MQIACH_CHANNEL_TYPE
Channel type

MQIACH_PORT
Port number to use when *TransportType* is set to TCP.

MQIACH_SSL_CLIENT_AUTH
TLS client authentication.

MQIACH_USE_CLIENT_ID
Specify whether to use the *clientID* of a new connection as the *userID* for that connection

MQIACH_XMIT_PROTOCOL_TYPE
Transport (transmission protocol) type

Note:

1. Only one of the following parameters can be specified:
   - MQCACH_JAAS_CONFIG
   - MQCACH_MCA_USER_ID
   - MQIACH_USE_CLIENT_ID

If none of these parameters are specified, no authentication is performed. If MQCACH_JAAS_CONFIG is specified, the client flows a user name and password, in all other cases the flowed user name is ignored.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

Inquire Channel (Response):

The response to the Inquire Channel (MQCMD_INQUIRE_CHANNEL) command consists of the response header followed by the *ChannelName* and *ChannelType* structures (and on z/OS only, the *DefaultChannelDisposition*, and *QSGDisposition* structure), and the requested combination of attribute parameter structures (where applicable).

If a generic channel name was specified, one such message is generated for each channel found.

Always returned:

ChannelName, *ChannelType*, z/OS *DefaultChannelDisposition*, z/OS *QSGDisposition*

Returned if requested:

AlterationDate, *AlterationTime*, *BatchDataLimit*, *BatchHeartbeat*, *BatchInterval*, *BatchSize*, *Certificatelabel*, *ChannelDesc*, *ChannelMonitoring*, *ChannelStatistics*, *ClientChannelWeight*, *ClientIdentifier*, *ClusterName*, *ClusterNamelist*, *CLWLChannelPriority*, *CLWLChannelRank*, *CLWLChannelWeight*, *ConnectionAffinity*, *ConnectionName*, *DataConversion*, *DefReconnect*, *DiscInterval*, *HeaderCompression*, *HeartbeatInterval*, *InDoubtInbound*, *InDoubtOutbound*, *KeepAliveInterval*, *LastMsgTime*, *LocalAddress*, *LongRetryCount*, *LongRetryInterval*, *MaxMsgLength*, *MCAName*, *MCAType*, *MCAUserIdentifier*, *MessageCompression*, *ModeName*, *MsgExit*, *MsgRetryCount*, *MsgRetryExit*, *MsgRetryInterval*, *MsgRetryUserData*, *MsgsReceived*, *MsgsSent*, *MsgUserData*, *NetworkPriority*, *NonPersistentMsgSpeed*, *Password*, *PendingOutbound*, *PropertyControl*, *PutAuthority*, *QMgrName*, *ReceiveExit*, *ReceiveUserData*, *ResetSeq*, *SecurityExit*, *SecurityUserData*, *SendExit*, *SendUserData*, *SeqNumberWrap*, *SharingConversations*, *ShortRetryCount*, *ShortRetryInterval*, *SSLCipherSpec*, *SSLCipherSuite*, *SSLClientAuth*, *SSLPeerName*, *TpName*, *TransportType*, *UseDLQ*, *UserIdentifier*, *XmitQName*

Response data

AlterationDate (MQCFST)
Alteration date, in the form yyyy-mm-dd (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

AlterationTime (MQCFST)
Alteration time, in the form hh.mm.ss (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

BatchDataLimit (MQCFIN)

Batch data limit (parameter identifier: MQIACH_BATCH_DATA_LIMIT).

The limit, in kilobytes, of the amount of data that can be sent through a channel before taking a sync point. A sync point is taken after the message that caused the limit to be reached has flowed across the channel. A value of zero in this attribute means that no data limit is applied to batches over this channel.

This parameter only applies to channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSRCVR, or MQCHT_CLUSSDR.

BatchHeartbeat (MQCFIN)
The value being used for the batch heartbeating (parameter identifier: MQIACH_BATCH_HB).

The value can be 0 - 999999. A value of 0 indicates that heartbeating is not in use.

BatchInterval (MQCFIN)
Batch interval (parameter identifier: MQIACH_BATCH_INTERVAL).

BatchSize (MQCFIN)
Batch size (parameter identifier: MQIACH_BATCH_SIZE).

Certificatelabel (MQCFST)
Certificate label (parameter identifier: MQCA_CERT_LABEL).

Specifies the certificate label in use.

The maximum length is MQ_CERT_LABEL_LENGTH.

ChannelDesc (MQCFST)
Channel description (parameter identifier: MQCACH_DESC).

The maximum length of the string is MQ_CHANNEL_DESC_LENGTH.

ChannelMonitoring (MQCFIN)
Online monitoring data collection (parameter identifier: MQIA_MONITORING_CHANNEL).

The value can be any of the following values:

MQMON_OFF
Online monitoring data collection is turned off for this channel.

MQMON_Q_MGR
The value of the queue manager's **ChannelMonitoring** parameter is inherited by the channel.

MQMON_LOW
Online monitoring data collection is turned on, with a low rate of data collection, for this channel unless the queue manager's **ChannelMonitoring** parameter is MQMON_NONE.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate rate of data collection, for this channel unless the queue manager's *ChannelMonitoring* parameter is MQMON_NONE.

MQMON_HIGH
Online monitoring data collection is turned on, with a high rate of data collection, for this channel unless the queue manager's **ChannelMonitoring** parameter is MQMON_NONE.

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelStatistics (MQCFIN)
Statistics data collection (parameter identifier: MQIA_STATISTICS_CHANNEL).

The value can be any of the following values:

MQMON_OFF
Statistics data collection is turned off for this channel.

MQMON_Q_MGR
The value of the queue manager's **ChannelStatistics** parameter is inherited by the channel.

MQMON_LOW
Statistics data collection is turned on, with a low rate of data collection, for this channel unless the queue manager's **ChannelStatistics** parameter is MQMON_NONE.

MQMON_MEDIUM
Statistics data collection is turned on, with a moderate rate of data collection, for this channel unless the queue manager's **ChannelStatistics** parameter is MQMON_NONE.

MQMON_HIGH
Statistics data collection is turned on, with a high rate of data collection, for this channel unless the queue manager's **ChannelStatistics** parameter is MQMON_NONE.

z/OS On z/OS systems, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying `LOW`, `MEDIUM`, or `HIGH` makes no difference to your results.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

The value can be any of the following values:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLNTCONN
Client connection.

MQCHT_CLUSRCVR
Cluster-receiver.

MQCHT_CLUSSDR
Cluster-sender.

MQCHT_MQTT
Telemetry channel.

ClientChannelWeight (MQCFIN)
Client Channel Weight (parameter identifier: MQIACH_CLIENT_CHANNEL_WEIGHT).

The client channel weighting attribute is used so client channel definitions can be selected at random, with the larger weightings having a higher probability of selection, when more than one suitable definition is available.

The value can be 0 - 99. The default is 0.

This parameter is only valid for channels with a ChannelType of MQCHT_CLNTCONN

ClientIdentifier (MQCFST)
the clientId of the client (parameter identifier: MQCACH_CLIENT_ID).

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

ClusterNamelist (MQCFST)
Cluster namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

CLWLChannelPriority (MQCFIN)
Channel priority (parameter identifier: MQIACH_CLWL_CHANNEL_PRIORITY).

CLWLChannelRank (MQCFIN)
Channel rank (parameter identifier: MQIACH_CLWL_CHANNEL_RANK).

CLWLChannelWeight (MQCFIN)
Channel weighting (parameter identifier: MQIACH_CLWL_CHANNEL_WEIGHT).

ConnectionAffinity (MQCFIN)
Channel Affinity (parameter identifier: MQIACH_CONNECTION_AFFINITY)

The channel affinity attribute specifies whether client applications that connect multiple times using the same queue manager name, use the same client channel. The value can be any of the following values:

MQCAFTY_PREFERRED
The first connection in a process reading a client channel definition table (CCDT) creates a list of applicable definitions based on the weighting with any zero ClientChannelWeight definitions first in alphabetical order. Each connection in the process attempts to connect using the first definition in the list. If a connection is unsuccessful the next definition is used. Unsuccessful nonzero ClientChannelWeight definitions are moved to the end of the list. Zero ClientChannelWeight definitions remain at the start of the list and are selected first for each connection. For C, C++ and .NET (including fully managed .NET) clients the list is updated if the CCDT has been modified since the list was created. Each client process with the same host name creates the same list.

MQCAFTY_PREFERRED is the default value.

MQCAFTY_NONE
The first connection in a process reading a CCDT creates a list of applicable definitions. All connections in a process independently select an applicable definition based on the weighting with any applicable zero ClientChannelWeight definitions selected first in alphabetical order. For C, C++ and .NET (including fully managed .NET) clients the list is updated if the CCDT has been modified since the list was created.

This parameter is only valid for channels with a ChannelType of MQCHT_CLNTCONN.

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH. On z/OS, it is MQ_LOCAL_ADDRESS_LENGTH.

The *ConnectionName* is a comma-separated list.

DataConversion (MQCFIN)
Whether sender must convert application data (parameter identifier: MQIACH_DATA_CONVERSION).

The value can be:

MQCDC_NO_SENDER_CONVERSION
No conversion by sender.

MQCDC_SENDER_CONVERSION
Conversion by sender.

z/OS **DefaultChannelDisposition (MQCFIN)**
Default channel disposition (parameter identifier: MQIACH_DEF_CHANNEL_DISP).

This parameter applies to z/OS only.

Specifies the intended disposition of the channel when active. The value can be any of the following values:

MQCHLD_PRIVATE
The intended use of the object is as a private channel.

MQCHLD_FIXSHARED
The intended use of the object is as a shared channel linked to a specific queue manager.

MQCHLD_SHARED
The intended use of the object is as a shared channel.

DiscInterval (MQCFIN)
Disconnection interval (parameter identifier: MQIACH_DISC_INTERVAL).

DefReconnect (MQCFIN)
Client channel default reconnection option (parameter identifier: MQIACH_DEF_RECONNECT).

The returned values can be:

MQRCN_NO
MQRCN_NO is the default value.

Unless overridden by **MQCONNX**, the client is not reconnected automatically.

MQRCN_YES
Unless overridden by **MQCONNX**, the client reconnects automatically.

MQRCN_Q_MGR
Unless overridden by **MQCONNX**, the client reconnects automatically, but only to the same queue manager. The QMGR option has the same effect as MQCNO_RECONNECT_Q_MGR.

MQRCN_DISABLED
Reconnection is disabled, even if requested by the client program using the **MQCONNX** MQI call.

HeaderCompression (MQCFIL)
Header data compression techniques supported by the channel (parameter identifier: MQIACH_HDR_COMPRESSION). For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference.

The value can be one, or more, of

MQCOMPRESS_NONE
No header data compression is performed.

MQCOMPRESS_SYSTEM
Header data compression is performed.

HeartbeatInterval (MQCFIN)
Heartbeat interval (parameter identifier: MQIACH_HB_INTERVAL).

InDoubtInbound (MQCFIN)
Number of inbound messages to the client that are in doubt (Parameter identifier: MQIACH_IN_DOUBT_IN).

InDoubtOutbound (MQCFIN)
Number of outbound messages from the client that are in doubt (Parameter identifier: MQIACH_IN_DOUBT_OUT).

KeepAliveInterval (MQCFIN)
KeepAlive interval (parameter identifier: MQIACH_KEEP_ALIVE_INTERVAL).

LastMsgTime (MQCFST)
The time that the last message was sent or received (parameter identifier: MQCACH_LAST_MSG_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

LocalAddress (MQCFST)
Local communications address for the channel (parameter identifier: MQCACH_LOCAL_ADDRESS).

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

LongRetryCount (MQCFIN)
Long retry count (parameter identifier: MQIACH_LONG_RETRY).

LongRetryInterval (MQCFIN)
Long timer (parameter identifier: MQIACH_LONG_TIMER).

MaxInstances (MQCFIN)
Maximum number of simultaneous instances of a server-connection channel (parameter identifier: MQIACH_MAX_INSTANCES).

This parameter is returned only for server-connection channels in response to an Inquire Channel call with ChannelAttrs including MQIACF_ALL or MQIACH_MAX_INSTANCES.

MaxInstancesPerClient (MQCFIN)
Maximum number of simultaneous instances of a server-connection channel that can be started from a single client (parameter identifier: MQIACH_MAX_INSTS_PER_CLIENT).

This parameter is returned only for server-connection channels in response to an Inquire Channel call with ChannelAttrs including MQIACF_ALL or MQIACH_MAX_INSTS_PER_CLIENT.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIACH_MAX_MSG_LENGTH).

MCAName (MQCFST)
Message channel agent name (parameter identifier: MQCACH_MCA_NAME).

The maximum length of the string is MQ_MCA_NAME_LENGTH.

MCAType (MQCFIN)
Message channel agent type (parameter identifier: MQIACH_MCA_TYPE).

The value can be any of the following values:

MQMCAT_PROCESS
Process.

MQMCAT_THREAD
Thread (Windows only).

MCAUserIdentifier (MQCFST)
Message channel agent user identifier (parameter identifier: MQCACH_MCA_USER_ID).

Note: An alternative way of providing a user ID for a channel to run under is to use channel authentication records. With channel authentication records, different connections can use the same channel while using different credentials. If both MCAUSER on the channel is set and channel authentication records are used to apply to the same channel, the channel authentication records take precedence. The MCAUSER on the channel definition is only used if the channel authentication record uses USERSRC(CHANNEL). For more details, see Channel authentication records

The maximum length of the MCA user identifier depends on the environment in which the MCA is running. MQ_MCA_USER_ID_LENGTH gives the maximum length for the environment for which your application is running. MQ_MAX_MCA_USER_ID_LENGTH gives the maximum for all supported environments.

On Windows, the user identifier might be qualified with the domain name in the following format:

```
user@domain
```

MessageCompression (MQCFIL)
Message data compression techniques supported by the channel (parameter identifier: MQIACH_MSG_COMPRESSION). For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference.

The value can be one, or more, of:

MQCOMPRESS_NONE
No message data compression is performed.

MQCOMPRESS_RLE
Message data compression is performed using run-length encoding.

MQCOMPRESS_ZLIBFAST
Message data compression is performed using ZLIB encoding with speed prioritized.

MQCOMPRESS_ZLIBHIGH
Message data compression is performed using ZLIB encoding with compression prioritized.

MQCOMPRESS_ANY
Any compression technique supported by the queue manager can be used.
MQCOMPRESS_ANY is only valid for receiver, requester, and server-connection channels.

ModeName (MQCFST)
Mode name (parameter identifier: MQCACH_MODE_NAME).

The maximum length of the string is MQ_MODE_NAME_LENGTH.

MsgExit (MQCFST)
Message exit name (parameter identifier: MQCACH_MSG_EXIT_NAME).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

Multi On Multiplatforms, if more than one message exit has been defined for the channel, the list of names is returned in an MQCFSL structure instead of an MQCFST structure.

z/OS On z/OS, an MQCFSL structure is always used.

MsgsReceived (MQCFIN64)
The number of messages received by the client since it last connected (parameter identifier: MQIACH_MSGS_RECEIVED / MQIACH_MSGS_RCVD).

MsgRetryCount (MQCFIN)
Message retry count (parameter identifier: MQIACH_MR_COUNT).

MsgRetryExit (MQCFST)
Message retry exit name (parameter identifier: MQCACH_MR_EXIT_NAME).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

MsgRetryInterval (MQCFIN)
Message retry interval (parameter identifier: MQIACH_MR_INTERVAL).

MsgRetryUserData (MQCFST)
Message retry exit user data (parameter identifier: MQCACH_MR_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

MsgsSent (MQCFIN64)
The number of messages sent by the client since it last connected (parameter identifier: MQIACH_MSGS_SENT).

MsgUserData (MQCFST)
Message exit user data (parameter identifier: MQCACH_MSG_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

Multi On Multiplatforms, if more than one message exit has been defined for the channel, the list of names is returned in an MQCFSL structure instead of an MQCFST structure.

z/OS On z/OS, an MQCFSL structure is always used.

NetworkPriority (MQCFIN)
Network priority (parameter identifier: MQIACH_NETWORK_PRIORITY).

NonPersistentMsgSpeed (MQCFIN)
Speed at which non-persistent messages are to be sent (parameter identifier: MQIACH_NPM_SPEED).

The value can be:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

Password (MQCFST)
Password (parameter identifier: MQCACH_PASSWORD).

If a nonblank password is defined, it is returned as asterisks. Otherwise, it is returned as blanks.

The maximum length of the string is MQ_PASSWORD_LENGTH. However, only the first 10 characters are used.

PropertyControl (MQCFIN)
Property control attribute (parameter identifier MQIA_PROPERTY_CONTROL).

Specifies what happens to properties of messages when the message is about to be sent to a V6 or prior queue manager (a queue manager that does not understand the concept of a property descriptor). The value can be any of the following values:

MQPROP_COMPATIBILITY

Message properties	Result
The message contains a property with a prefix of **mcd.**, **jms.**, **usr.** or **mqext.**	All optional message properties (where the **Support** value is MQPD_SUPPORT_OPTIONAL), except those properties in the message descriptor or extension, are placed in one or more MQRFH2 headers in the message data before the message it sent to the remote queue manager.
The message does not contain a property with a prefix of **mcd.**, **jms.**, **usr.** or **mqext.**	All message properties, except those properties in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.
The message contains a property where the **Support** field of the property descriptor is not set to MQPD_SUPPORT_OPTIONAL	The message is rejected with reason MQRC_UNSUPPORTED_PROPERTY and treated in accordance with its report options.
The message contains one or more properties where the **Support** field of the property descriptor is set to MQPD_SUPPORT_OPTIONAL but other fields of the property descriptor are set to non-default values	The properties with non-default values are removed from the message before the message is sent to the remote queue manager.
The MQRFH2 folder that would contain the message property needs to be assigned with the *content='properties'* attribute	The properties are removed to prevent MQRFH2 headers with unsupported syntax flowing to a V6 or prior queue manager.

MQPROP_NONE
All properties of the message, except those properties in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.

If the message contains a property where the **Support** field of the property descriptor is not set to MQPD_SUPPORT_OPTIONAL then the message is rejected with reason MQRC_UNSUPPORTED_PROPERTY and treated in accordance with its report options.

MQPROP_ALL
All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except those properties in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

This attribute is applicable to Sender, Server, Cluster Sender, and Cluster Receiver channels.

PutAuthority (MQCFIN)
Put authority (parameter identifier: MQIACH_PUT_AUTHORITY).

The value can be any of the following values:

MQPA_DEFAULT
Default user identifier is used.

MQPA_CONTEXT
Context user identifier is used.

QMgrName (MQCFST)
Queue manager name (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

z/OS **QSGDisposition (MQCFIN)**
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object (that is, where it is defined and how it behaves). This parameter is valid only on z/OS. The value can be any of the following values:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

ReceiveExit (MQCFST)
Receive exit name (parameter identifier: MQCACH_RCV_EXIT_NAME).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

Multi On Multiplatforms, if more than one receive exit has been defined for the channel, the list of names is returned in an MQCFSL structure instead of an MQCFST structure.

z/OS On z/OS, an MQCFSL structure is always used.

ReceiveUserData (MQCFST)
Receive exit user data (parameter identifier: MQCACH_RCV_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

Multi On Multiplatforms, if more than one receive exit user data string has been defined for the channel, the list of strings is returned in an MQCFSL structure instead of an MQCFST structure.

z/OS On z/OS, an MQCFSL structure is always used.

ResetSeq (MQCFIN)

Pending reset sequence number (parameter identifier: MQIACH_RESET_REQUESTED).

This is the sequence number from an outstanding request and it indicates a user Reset Channel command request is outstanding.

A value of zero indicates that there is no outstanding Reset Channel. The value can be in the range 1 - 999999999.

Possible return values include MQCHRR_RESET_NOT_REQUESTED.

This parameter is not applicable on z/OS.

SecurityExit (MQCFST)
Security exit name (parameter identifier: MQCACH_SEC_EXIT_NAME).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

SecurityUserData (MQCFST)
Security exit user data (parameter identifier: MQCACH_SEC_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

SendExit (MQCFST)
Send exit name (parameter identifier: MQCACH_SEND_EXIT_NAME).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

Multi On Multiplatforms, if more than one send exit has been defined for the channel, the list of names is returned in an MQCFSL structure instead of an MQCFST structure.

z/OS On z/OS, an MQCFSL structure is always used.

SendUserData (MQCFST)
Send exit user data (parameter identifier: MQCACH_SEND_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

Multi On Multiplatforms, if more than one send exit user data string has been defined for the channel, the list of strings is returned in an MQCFSL structure instead of an MQCFST structure.

z/OS On z/OS, an MQCFSL structure is always used.

SeqNumberWrap (MQCFIN)
Sequence wrap number (parameter identifier: MQIACH_SEQUENCE_NUMBER_WRAP).

SharingConversations (MQCFIN)
Number of sharing conversations (parameter identifier: MQIACH_SHARING_CONVERSATIONS).

This parameter is returned only for TCP/IP client-connection and server-connection channels.

ShortRetryCount (MQCFIN)
Short retry count (parameter identifier: MQIACH_SHORT_RETRY).

ShortRetryInterval (MQCFIN)
Short timer (parameter identifier: MQIACH_SHORT_TIMER).

SSLCipherSpec (MQCFST)
CipherSpec (parameter identifier: MQCACH_SSL_CIPHER_SPEC).

The length of the string is MQ_SSL_CIPHER_SPEC_LENGTH.

SSLCipherSuite (MQCFST)
CipherSuite (parameter identifier: MQCACH_SSL_CIPHER_SUITE).

The length of the string is MQ_SSL_CIPHER_SUITE_LENGTH.

SSLClientAuth (MQCFIN)
Client authentication (parameter identifier: MQIACH_SSL_CLIENT_AUTH).

The value can be

MQSCA_REQUIRED
Client authentication required

MQSCA_OPTIONAL
Client authentication is optional.

The following value is also valid for Channels of type MQCHT_MQTT:

MQSCA_NEVER_REQUIRED
Client authentication is never required, and must not be provided.

Defines whether IBM MQ requires a certificate from the TLS client.

SSLPeerName (MQCFST)
Peer name (parameter identifier: MQCACH_SSL_PEER_NAME).

Note: An alternative way of restricting connections into channels by matching against the TLS Subject Distinguished Name, is to use channel authentication records. With channel authentication records, different TLS Subject Distinguished Name patterns can be applied to the same channel. If both SSLPEER on the channel and a channel authentication record are used to apply to the same channel, the inbound certificate must match both patterns in order to connect. For more information, see Channel authentication records.

The length of the string is MQ_SSL_PEER_NAME_LENGTH. On z/OS, it is MQ_SSL_SHORT_PEER_NAME_LENGTH.

Specifies the filter to use to compare with the Distinguished Name of the certificate from the peer queue manager or client at the other end of the channel. (A Distinguished Name is the identifier of the TLS certificate.) If the Distinguished Name in the certificate received from the peer does not match the SSLPEER filter, the channel does not start.

TpName (MQCFST)
Transaction program name (parameter identifier: MQCACH_TP_NAME).

The maximum length of the string is MQ_TP_NAME_LENGTH.

TransportType (MQCFIN)
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value might be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

MQXPT_NETBIOS
NetBIOS.

MQXPT_SPX
SPX.

MQXPT_DECNET
DECnet.

UseDLQ (MQCFIN)
Whether the dead-letter queue (or undelivered message queue) should be used when messages cannot be delivered by channels (parameter identifier: MQIA_USE_DEAD_LETTER_Q).

The value might be:

MQUSEDLQ_NO
Messages that cannot be delivered by a channel will be treated as a failure and either the channel will discard them, or the channel will end, in accordance with the setting of NPMSPEED.

MQUSEDLQ_YES
If the queue manager DEADQ attribute provides the name of a dead-letter queue then it will be used, otherwise the behavior will be as for MQUSEDLQ_NO.

UserIdentifier (MQCFST)
Task user identifier (parameter identifier: MQCACH_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH. However, only the first 10 characters are used.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

Inquire Channel Authentication Records:

The Inquire Channel Authentication Records (MQCMD_INQUIRE_CHLAUTH_RECS) command retrieves the allowed partner details and mappings to MCAUSER for a channel or set of channels.

Required parameters

generic-channel-name (MQCFST)
The name of the channel or set of channels on which you are inquiring (parameter identifier: MQCACH_CHANNEL_NAME).

You can use the asterisk (*) as a wildcard to specify a set of channels, unless you set Match to MQMATCH_RUNCHECK. If you set Type to BLOCKADDR, you must set the generic channel name to a single asterisk, which matches all channel names.

Optional parameters

Address (MQCFST)
The IP address to be mapped (parameter identifier: MQCACH_CONNECTION_NAME).

This parameter is valid only when **Match** is MQMATCH_RUNCHECK and must not be generic.

ByteStringFilterCommand (MQCFBF)
Byte string filter command descriptor. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFBF - PCF byte string filter parameter" on page 1596 for information about using this filter condition.

If you specify a byte string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter, or a string filter using the **StringFilterCommand** parameter.

ChannelAuthAttrs (MQCFIL)
Authority record attributes (parameter identifier: MQIACF_CHLAUTH_ATTRS).

You can specify the following value in the attribute list on its own. This is the default value if the parameter is not specified.

MQIACF_ALL
All attributes.

If MQIACF_ALL is not specified, specify a combination of the following values:

MQCA_ALTERATION_DATE
Alteration Date.

MQCA_ALTERATION_TIME
Alteration Time.

MQCA_CHLAUTH_DESC
Description.

MQCA_CUSTOM
Custom.

MQCACH_CONNECTION_NAME
IP address filter.

MQCACH_MCA_USER_ID
MCA User ID mapped on the record.

MQIACH_USER_SOURCE
The source of the user ID for this record.

MQIACH_WARNING
Warning mode.

CheckClient (MQCFIN)
The user ID and password requirements for the client connection to be successful. The following values are valid:

MQCHK_REQUIRED_ADMIN
A valid user ID and password are required for the connection to be allowed if you are using a privileged user ID.

Any connections using a non-privileged user ID are not required to provide a user ID and password.

The user ID and password are checked against the user repository details provided in an authentication information object, and supplied on ALTER QMGR in the CONNAUTH field.

If no user repository details are provided, so that user ID and password checking are not enabled on the queue manager, the connection is not successful.

This option is not valid on z/OS platforms.

MQCHK_REQUIRED
A valid user ID and password are required for the connection to be allowed.

The user ID and password are checked against the user repository details provided in an authentication information object and supplied on ALTER QMGR in the CONNAUTH field.

If no user repository details are provided, so that user ID and password checking are not enabled on the queue manager, the connection is not successful.

MQCHK_AS_Q_MGR
In order for the connection to be allowed, it must meet the connection authentication requirements defined on the queue manager.

If the CONNAUTH field provides an authentication information object, and the value of CHCKCLNT is REQUIRED, the connection fails unless a valid user ID and password are supplied.

If the CONNAUTH field does not provide an authentication information object, or the value of CHCKCLNT is not REQUIRED, the user ID and password are not required.

Attention: If you select REQUIRED or REQADM on Multiplatforms and you have not set the CONNAUTH field on the queue manager, or if the value of CHCKCLNT is NONE, the connection fails. On Multiplatforms, you receive message AMQ9793. On z/OS, you receive message CSQX793E.

ClntUser (MQCFST)
The client asserted user ID to be mapped to a new user ID, allowed through unchanged, or blocked (parameter identifier: MQCACH_CLIENT_USER_ID).

This can be the user ID flowed from the client indicating the user ID the client side process is running under, or the user ID presented by the client on an MQCONNX call using MQCSP.

This parameter is valid only with TYPE(USERMAP) and when **Match** is MQMATCH_RUNCHECK.

z/OS **CommandScope (MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following values:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which the command was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1601 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a byte string filter using the **ByteStringFilterCommand** parameter or a string filter using the **StringFilterCommand** parameter.

Match (MQCFIN)
Indicates the type of matching to be applied (parameter identifier MQIACH_MATCH). You can specify any one of the following values:

MQMATCH_RUNCHECK
A specific match is made against the supplied channel name and optionally supplied **Address**, **SSLPeer**, **QMName**, and **ClntUser** attributes to find the channel authentication record that will be matched by the channel at runtime if it connects into this queue manager. If the record discovered has **Warn** set to MQWARN_YES, a second record might also be displayed to show the actual record the channel will use at runtime. The channel name supplied in this case cannot be generic. This option must be combined with **Type** MQCAUT_ALL.

MQMATCH_EXACT
Return only those records which exactly match the channel profile name supplied. If there are no asterisks in the channel profile name, this option returns the same output as MQMATCH_GENERIC.

MQMATCH_GENERIC
Any asterisks in the channel profile name are treated as wildcards. If there are no asterisks in the channel profile name, this returns the same output as MQMATCH_EXACT. For example, a profile of ABC* could result in records for ABC, ABC*, and ABCD being returned.

MQMATCH_ALL
Return all possible records that match the channel profile name supplied. If the channel name is generic in this case, all records that match the channel name are returned even if more

specific matches exist. For example, a profile of SYSTEM.*.SVRCONN could result in records for SYSTEM.*, SYSTEM.DEF.*, SYSTEM.DEF.SVRCONN, and SYSTEM.ADMIN.SVRCONN being returned.

QMName (MQCFST)
The name of the remote partner queue manager to be matched (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

This parameter is valid only when **Match** is MQMATCH_RUNCHECK. The value cannot be generic.

SSLCertIssuer (MQCFST)
This parameter is additional to the **SSLPeer** parameter.

SSLCertIssuer restricts matches to being within certificates issued by a particular Certificate Authority.

SSLPeer (MQCFST)

The Distinguished Name of the certificate to be matched (parameter identifier: MQCACH_SSL_PEER_NAME).

This parameter is valid only when **Match** is MQMATCH_RUNCHECK.

The **SSLPeer** value is specified in the standard form used to specify a Distinguished Name and cannot be a generic value.

The maximum length of the parameter is MQ_SSL_PEER_NAME_LENGTH.

StringFilterCommand (MQCFSF)
String filter command descriptor. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1608 for information about using this filter condition.

If you specify a string filter, you cannot also specify a byte string filter using the **ByteStringFilterCommand** parameter or an integer filter using the **IntegerFilterCommand** parameter.

Type (MQCFIN)
The type of channel authentication record for which to set allowed partner details or mappings to MCAUSER (parameter identifier: MQIACF_CHLAUTH_TYPE). The following values are valid:

MQCAUT_BLOCKUSER
This channel authentication record prevents a specified user or users from connecting.

MQCAUT_BLOCKADDR
This channel authentication record prevents connections from a specified IP address or addresses.

MQCAUT_SSLPEERMAP
This channel authentication record maps TLS Distinguished Names (DNs) to MCAUSER values.

MQCAUT_ADDRESSMAP
This channel authentication record maps IP addresses to MCAUSER values.

MQCAUT_USERMAP
This channel authentication record maps asserted user IDs to MCAUSER values.

MQCAUT_QMGRMAP
This channel authentication record maps remote queue manager names to MCAUSER values.

MQCAUT_ALL
Inquire on all types of record. This is the default value.

Related information:
Channel authentication records

Inquire Channel Authentication Records (Response):

The response to the Inquire Channel Authentication Records (MQCMD_INQUIRE_CHLAUTH_RECS) command consists of the response header followed by the requested combination of attribute parameter structures.

Always returned:
ChlAuth, *Type*, *Warn(yes)*

Always returned if type is MQCAUT_BLOCKUSER:
UserList

Always returned if type is MQCAUT_BLOCKADDR:
AddrList

Always returned if type is MQCAUT_SSLPEERMAP:
Address (unless blanks), *MCAUser (unless blanks)*, *SSLCertIssuer*, *SSLPeer*, *UserSrc*

Always returned if type is MQCAUT_ADDRESSMAP:
Address (unless blanks), *MCAUser (unless blanks)*, *UserSrc*

Always returned if type is MQCAUT_USERMAP:
Address (unless blanks), *CintUser*, *MCAUser (unless blanks)*, *UserSrc*

Always returned if type is MQCAUT_QMGRMAP:
Address (unless blanks), *MCAUser (unless blanks)*, *QMName*, *UserSrc*

Returned if requested:
Address, *AlterationDate*, *AlterationTime*, *Custom*, *Description*, *MCAUser*, *SSLPeer*, *UserSrc*, *Warn*

Response data

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered, in the form `yyyy-mm-dd`.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered, in the form `hh.mm.ss`.

Address (MQCFST)
The filter used to compare with the IP address, or host name, of the partner queue manager or client at the other end of the channel (parameter identifier: MQCACH_CONNECTION_NAME).

AddrList (MQCFSL)
A list of up to 100 IP address patterns which are banned from accessing this queue manager on any channel (parameter identifier: MQCACH_CONNECTION_NAME_LIST).

Chlauth (MQCFST)
The name of the channel, or pattern that matches a set of channels, to which the channel authentication record applies (parameter identifier: MQCACH_CHANNEL_NAME).

CheckClient (MQCFIN)
The user ID and password requirements for the client connection to be successful (parameter identifier: MQIA_CHECK_CLIENT_BINDING).

ClntUser (MQCFST)
The client asserted user ID to be mapped to a new user ID, allowed through unchanged, or blocked (parameter identifier: MQCACH_CLIENT_USER_ID).

Description (MQCFST)
Descriptive information about the channel authentication record (parameter identifier: MQCA_CHLAUTH_DESC).

MCAUser (MQCFST)
The user identifier to be used when the inbound connection matches the TLS DN, IP address, client asserted user ID or remote queue manager name supplied (parameter identifier: MQCACH_MCA_USER_ID).

QMName (MQCFST)
The name of the remote partner queue manager to be mapped to a user ID, allowed through unchanged, or blocked (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

SSLCertIssuer (MQCFST)
This parameter is additional to the **SSLPeer** parameter.

SSLCertIssuer restricts matches to being within certificates issued by a particular Certificate Authority (parameter identifier: MQCA_SSL_CERT_ISSUER_NAME).

SSLPeer (MQCFST)
The filter to use to compare with the Distinguished Name of the certificate from the peer queue manager or client at the other end of the channel (parameter identifier: MQCACH_SSL_PEER_NAME).

Type (MQCFIN)
The type of channel authentication record for which to set allowed partner details or mappings to MCAUSER (parameter identifier: MQIACF_CHLAUTH_TYPE). The following values can be returned:

MQCAUT_BLOCKUSER
This channel authentication record prevents a specified user or users from connecting.

MQCAUT_BLOCKADDR
This channel authentication record prevents connections from a specified IP address or addresses.

MQCAUT_SSLPEERMAP
This channel authentication record maps TLS Distinguished Names (DNs) to MCAUSER values.

MQCAUT_ADDRESSMAP
This channel authentication record maps IP addresses to MCAUSER values.

MQCAUT_USERMAP
This channel authentication record maps asserted user IDs to MCAUSER values.

MQCAUT_QMGRMAP
This channel authentication record maps remote queue manager names to MCAUSER values.

UserList (MQCFSL)
A list of up to 100 user IDs which are banned from use of this channel or set of channels (parameter identifier: MQCACH_MCA_USER_ID_LIST). Use the special value `*MQADMIN` to mean privileged or administrative users. The definition of this value depends on the operating system, as follows:

- On Windows, all members of the mqm group, the Administrators group and SYSTEM.
- On UNIX and Linux, all members of the mqm group.
- On IBM i, the profiles (users) qmqm and qmqmadm and all members of the qmqmadm group, and any user defined with the `*ALLOBJ` special setting.
- On z/OS, the user ID that the channel initiator, queue manager and ⌂advanced message security address spaces are running under.

UserSrc (MQCFIN)
The source of the user ID to be used for MCAUSER at run time (parameter identifier: MQIACH_USER_SOURCE).

The following values can be returned:

MQUSRC_MAP
Inbound connections that match this mapping use the user ID specified in the **MCAUser** attribute.

MQUSRC_NOACCESS
Inbound connections that match this mapping have no access to the queue manager and the channel ends immediately.

MQUSRC_CHANNEL
Inbound connections that match this mapping use the flowed user ID or any user defined on the channel object in the MCAUSER field.

Warn (MQCFIN)
Indicates whether this record operates in warning mode (parameter identifier: MQIACH_WARNING).

MQWARN_NO
This record does not operate in warning mode. Any inbound connection that matches this record is blocked. This is the default value.

MQWARN_YES
This record operates in warning mode. Any inbound connection that matches this record and would therefore be blocked is allowed access. An error message is written and, if events are configured, an event message is created showing the details of what would have been blocked. The connection is allowed to continue.

Inquire Channel Initiator on z/OS: z/OS

The Inquire Channel Initiator (MQCMD_INQUIRE_CHANNEL_INIT) command returns information about the channel initiator.

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Inquire Channel Initiator (Response) on z/OS: z/OS

The response to the Inquire Channel Initiator (MQCMD_INQUIRE_CHANNEL_INIT) command consists of one response with a series of attribute parameter structures showing the status of the channel initiator (shown by the *ChannelInitiatorStatus* parameter), and one response for each listener (shown by the **ListenerStatus** parameter).

Always returned (one message with channel initiator information):
ActiveChannels, *ActiveChannelsMax*, *ActiveChannelsPaused*, *ActiveChannelsRetrying*, *ActiveChannelsStarted*, *ActiveChannelsStopped*, *AdaptersMax*, *AdaptersStarted*, *ChannelInitiatorStatus*, *CurrentChannels*, *CurrentChannelsLU62*, *CurrentChannelsMax*, *CurrentChannelsTCP*, *DispatchersMax*, *DispatchersStarted*, *SSLTasksStarted*, *TCPName*

Always returned (one message for each listener):
InboundDisposition, *ListenerStatus*, *TransportType*

Returned if applicable for the listener:
IPAddress, *LUName*, *Port*

Response data - channel initiator information

ActiveChannels (MQCFIN)
The number of active channel connections (parameter identifier: MQIACH_ACTIVE_CHL).

ActiveChannelsMax (MQCFIN)
The requested number of active channel connections (parameter identifier: MQIACH_ACTIVE_CHL_MAX).

ActiveChannelsPaused (MQCFIN)
The number of active channel connections that have paused, waiting to become active, because the limit for active channels has been reached (parameter identifier: MQIACH_ACTIVE_CHL_PAUSED).

ActiveChannelsRetrying (MQCFIN)
The number of active channel connections that are attempting to reconnect following a temporary error (parameter identifier: MQIACH_ACTIVE_CHL_RETRY).

ActiveChannelsStarted (MQCFIN)
The number of active channel connections that have started (parameter identifier: MQIACH_ACTIVE_CHL_STARTED).

ActiveChannelsStopped (MQCFIN)
The number of active channel connections that have stopped, requiring manual intervention (parameter identifier: MQIACH_ACTIVE_CHL_STOPPED).

AdaptersMax (MQCFIN)
The requested number of adapter subtasks (parameter identifier: MQIACH_ADAPS_MAX).

AdaptersStarted (MQCFIN)
The number of active adapter subtasks (parameter identifier: MQIACH_ADAPS_STARTED).

ChannelInitiatorStatus (MQCFIN)
Status of the channel initiator (parameter identifier: MQIACF_CHINIT_STATUS).

The value can be:

MQSVC_STATUS_STOPPED
The channel initiator is not running.

MQSVC_STATUS_RUNNING
The channel initiator is fully initialized and is running.

CurrentChannels (MQCFIN)
The number of current channel connections (parameter identifier: MQIACH_CURRENT_CHL).

CurrentChannelsLU62 (MQCFIN)
The number of current LU 6.2 channel connections (parameter identifier: MQIACH_CURRENT_CHL_LU62).

CurrentChannelsMax (MQCFIN)
The requested number of channel connections (parameter identifier: MQIACH_CURRENT_CHL_MAX).

CurrentChannelsTCP (MQCFIN)
The number of current TCP/IP channel connections (parameter identifier: MQIACH_CURRENT_CHL_TCP).

DispatchersMax (MQCFIN)
The requested number of dispatchers (parameter identifier: MQIACH_DISPS_MAX).

DispatchersStarted (MQCFIN)
The number of active dispatchers (parameter identifier: MQIACH_DISPS_STARTED).

SSLTasksMax (MQCFIN)
The requested number of TLS server subtasks (parameter identifier: MQIACH_SSLTASKS_MAX).

SSLTasksStarted (MQCFIN)
The number of active TLS server subtasks (parameter identifier: MQIACH_SSLTASKS_STARTED).

TCPName (MQCFST)
TCP system name (parameter identifier: MQCACH_TCP_NAME).

The maximum length is MQ_TCP_NAME_LENGTH.

Response data - listener information

InboundDisposition (MQCFIN)
Inbound transmission disposition (parameter identifier: MQIACH_INBOUND_DISP).

Specifies the disposition of the inbound transmissions that the listener handles. The value can be any of the following values:

MQINBD_Q_MGR
Handling for transmissions directed to the queue manager. MQINBD_Q_MGR is the default.

MQINBD_GROUP
Handling for transmissions directed to the queue-sharing group. MQINBD_GROUP is permitted only if there is a shared queue manager environment.

IPAddress (MQCFST)
IP address on which the listener listens (parameter identifier: MQCACH_IP_ADDRESS).

ListenerStatus (MQCFIN)
Listener status (parameter identifier: MQIACH_LISTENER_STATUS).

The value can be:

MQSVC_STATUS_RUNNING
The listener has started.

MQSVC_STATUS_STOPPED
The listener has stopped.

MQSVC_STATUS_RETRYING
The listener is trying again.

LUName (MQCFST)
LU name on which the listener listens (parameter identifier: MQCACH_LU_NAME).

The maximum length is MQ_LU_NAME_LENGTH.

Port (MQCFIN)
Port number on which the listener listens (parameter identifier: MQIACH_PORT_NUMBER).

TransportType (MQCFIN)
Transmission protocol type that the listener is using (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value can be:

MQXPT_LU62
LU62.

MQXPT_TCP
TCP.

Inquire Channel Listener on Multiplatforms: 

The Inquire Channel Listener (MQCMD_INQUIRE_LISTENER) command inquires about the attributes of existing IBM MQ listeners.

Required parameters

ListenerName (MQCFST)
Listener name (parameter identifier: MQCACH_LISTENER_NAME).

This parameter is the name of the listener with attributes that are required. Generic listener names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all listeners having names that start with the selected character string. An asterisk on its own matches all possible names.

The listener name is always returned regardless of the attributes requested.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

Optional parameters

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *ListenerAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See “MQCFIF - PCF integer filter parameter” on page 1601 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the **StringFilterCommand** parameter.

ListenerAttrs (MQCFIL)
Listener attributes (parameter identifier: MQIACF_LISTENER_ATTRS).

The attribute list might specify the following value on its own- default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_ALTERATION_DATE
Date on which the definition was last altered.

MQCA_ALTERATION_TIME
Time at which the definition was last altered.

MQCACH_IP_ADDRESS
IP address for the listener.

MQCACH_LISTENER_DESC
Description of listener definition.

MQCACH_LISTENER_NAME
Name of listener definition.

MQCACH_LOCAL_NAME
NetBIOS local name that the listener uses. MQCACH_LOCAL_NAME is valid only on Windows.

MQCACH_TP_NAME
The LU 6.2 transaction program name. MQCACH_TP_NAME is valid only on Windows.

MQIACH_ADAPTER
Adapter number on which NetBIOS listens. MQIACH_ADAPTER is valid only on Windows.

MQIACH_BACKLOG
Number of concurrent connection requests that the listener supports.

MQIACH_COMMAND_COUNT
Number of commands that the listener can use. MQIACH_COMMAND_COUNT is valid only on Windows.

MQIACH_LISTENER_CONTROL
Specifies when the queue manager starts and stops the listener.

MQIACH_NAME_COUNT
Number of names that the listener can use. MQIACH_NAME_COUNT is valid only on Windows.

MQIACH_PORT
Port number.

MQIACH_SESSION_COUNT
Number of sessions that the listener can use. MQIACH_SESSION_COUNT is valid only on Windows.

MQIACH_SOCKET
SPX socket on which to listen. MQIACH_SOCKET is valid only on Windows.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *ListenerAttrs* except MQCACH_LISTENER_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1608 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter.

TransportType (MQCFIN)
Transport protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

If you specify this parameter, information is returned relating only to those listeners defined with the specified transport protocol type. If you specify an attribute in the *ListenerAttrs* list which is valid only for listeners of a different transport protocol type, it is ignored and no error is raised. If you specify this parameter, it must occur immediately after the **ListenerName** parameter.

If you do not specify this parameter, or if you specify it with a value of MQXPT_ALL, information about all listeners is returned. Valid attributes in the *ListenerAttrs* list which are not applicable to the listener are ignored, and no error messages are issued. The value can be any of the following values:

MQXPT_ALL
All transport types.

MQXPT_LU62
SNA LU 6.2. MQXPT_LU62 is valid only on Windows.

MQXPT_NETBIOS
NetBIOS. MQXPT_NETBIOS is valid only on Windows.

MQXPT_SPX
SPX. MQXPT_SPX is valid only on Windows.

MQXPT_TCP
Transmission Control Protocol/Internet Protocol (TCP/IP).

Inquire Channel Listener (Response) on Multiplatforms: 

The response to the Inquire Channel Listener (MQCMD_INQUIRE_LISTENER) command consists of the response header followed by the *ListenerName* structure and the requested combination of attribute parameter structures.

If a generic listener name was specified, one such message is generated for each listener found.

Always returned:
ListenerName

Returned if requested:
Adapter, *AlterationDate*, *AlterationTime*, *Backlog*, *Commands*, *IPAddress*, *ListenerDesc*, *LocalName*, *NetbiosNames*, *Port*, *Sessions*, *Socket*, *StartMode*, *TPname*, *TransportType*

Response data

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date, in the form yyyy-mm-dd, on which the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time, in the form hh.mm.ss, at which the information was last altered.

Adapter (MQCFIN)
Adapter number (parameter identifier: MQIACH_ADAPTER).

The adapter number on which NetBIOS listens. This parameter is valid only on Windows.

Backlog (MQCFIN)
Backlog (parameter identifier: MQIACH_BACKLOG).

The number of concurrent connection requests that the listener supports.

Commands (MQCFIN)
Adapter number (parameter identifier: MQIACH_COMMAND_COUNT).

The number of commands that the listener can use. This parameter is valid only on Windows.

IPAddress (MQCFST)
IP address (parameter identifier: MQCACH_IP_ADDRESS).

IP address for the listener specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric host name form.

The maximum length of the string is MQ_CONN_NAME_LENGTH

ListenerDesc (MQCFST)
Description of listener definition (parameter identifier: MQCACH_LISTENER_DESC).

The maximum length of the string is MQ_LISTENER_DESC_LENGTH.

ListenerName (MQCFST)
Name of listener definition (parameter identifier: MQCACH_LISTENER_NAME).

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

LocalName (MQCFST)
NetBIOS local name (parameter identifier: MQCACH_LOCAL_NAME).

The NetBIOS local name that the listener uses. This parameter is valid only on Windows.

The maximum length of the string is MQ_CONN_NAME_LENGTH

NetbiosNames (MQCFIN)
NetBIOS names (parameter identifier: MQIACH_NAME_COUNT).

The number of names that the listener supports. This parameter is valid only on Windows.

Port (MQCFIN)
Port number (parameter identifier: MQIACH_PORT).

The port number for TCP/IP. This parameter is valid only if the value of *TransportType* is MQXPT_TCP.

Sessions (MQCFIN)
NetBIOS sessions (parameter identifier: MQIACH_SESSION_COUNT).

The number of sessions that the listener can use. This parameter is valid only on Windows.

Socket (MQCFIN)
SPX socket number (parameter identifier: MQIACH_SOCKET).

The SPX socket on which to listen. This parameter is valid only if the value of *TransportType* is MQXPT_SPX.

StartMode (MQCFIN)
Service mode (parameter identifier: MQIACH_LISTENER_CONTROL).

Specifies how the listener is to be started and stopped. The value can be any of the following values:

MQSVC_CONTROL_MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by user command. MQSVC_CONTROL_MANUAL is the default value.

MQSVC_CONTROL_Q_MGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

MQSVC_CONTROL_Q_MGR_START
The listener is to be started at the same time as the queue manager is started, but is not request to stop when the queue manager is stopped.

TPName (MQCFST)
Transaction program name (parameter identifier: MQCACH_TP_NAME).

The LU 6.2 transaction program name. This parameter is valid only on Windows.

The maximum length of the string is MQ_TP_NAME_LENGTH

TransportType (MQCFIN)
Transmission protocol (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value can be:

MQXPT_TCP
TCP.

MQXPT_LU62
LU 6.2. MQXPT_LU62 is valid only on Windows.

MQXPT_NETBIOS
NetBIOS. MQXPT_NETBIOS is valid only on Windows.

MQXPT_SPX
SPX. MQXPT_SPX is valid only on Windows.

Inquire Channel Listener Status on Multiplatforms: Multi

The Inquire Channel Listener Status (MQCMD_INQUIRE_LISTENER_STATUS) command inquires about the status of one or more IBM MQ listener instances.

You must specify the name of a listener for which you want to receive status information. You can specify a listener by using either a specific listener name or a generic listener name. By using a generic listener name, you can display either:

- Status information for all listener definitions, by using a single asterisk (*), or
- Status information for one or more listeners that match the specified name.

Required parameters

ListenerName (MQCFST)
Listener name (parameter identifier: MQCACH_LISTENER_NAME).

Generic listener names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all listeners having names that start with the selected character string. An asterisk on its own matches all possible names.

The listener name is always returned, regardless of the attributes requested.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

Optional parameters

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *ListenerStatusAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1601 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the **StringFilterCommand** parameter.

ListenerStatusAttrs (MQCFIL)
Listener status attributes (parameter identifier: MQIACF_LISTENER_STATUS_ATTRS).

The attribute list can specify the following value on its own - default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCACH_IP_ADDRESS
IP address of the listener.

MQCACH_LISTENER_DESC
Description of listener definition.

MQCACH_LISTENER_NAME
Name of listener definition.

MQCACH_LISTENER_START_DATE
The date on which the listener was started.

MQCACH_LISTENER_START_TIME
The time at which the listener was started.

MQCACH_LOCAL_NAME
NetBIOS local name that the listener uses. MQCACH_LOCAL_NAME is valid only on Windows.

MQCACH_TP_NAME
LU6.2 transaction program name. MQCACH_TP_NAME is valid only on Windows.

MQIACF_PROCESS_ID
Operating system process identifier associated with the listener.

MQIACH_ADAPTER
Adapter number on which NetBIOS listens. MQIACH_ADAPTER is valid only on Windows.

MQIACH_BACKLOG
Number of concurrent connection requests that the listener supports.

MQIACH_COMMAND_COUNT
Number of commands that the listener can use. MQIACH_COMMAND_COUNT is valid only on Windows.

MQIACH_LISTENER_CONTROL
How the listener is to be started and stopped.

MQIACH_LISTENER_STATUS
Status of the listener.

MQIACH_NAME_COUNT
Number of names that the listener can use. MQIACH_NAME_COUNT is valid only on Windows.

MQIACH_PORT
Port number for TCP/IP.

MQIACH_SESSION_COUNT
Number of sessions that the listener can use. MQIACH_SESSION_COUNT is valid only on Windows.

MQIACH_SOCKET
SPX socket. MQIACH_SOCKET is valid only on Windows.

MQIACH_XMIT_PROTOCOL_TYPE
Transport type.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *ListenerStatusAttrs* except MQCACH_LISTENER_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1608 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter.

Error code

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_LSTR_STATUS_NOT_FOUND
Listener status not found.

Inquire Channel Listener Status (Response) on Multiplatforms: Multi

The response to the Inquire Channel Listener Status (MQCMD_INQUIRE_LISTENER_STATUS) command consists of the response header followed by the *ListenerName* structure and the requested combination of attribute parameter structures.

If a generic listener name was specified, one such message is generated for each listener found.

Always returned:
ListenerName

Returned if requested:
Adapter, *Backlog*, *ChannelCount*, *Commands*, *IPAddress*, *ListenerDesc*, *LocalName*, *NetbiosNames*, *Port*, *ProcessId*, *Sessions*, *Socket*, *StartDate*, *StartMode*, *StartTime*, *Status*, *TPname*, *TransportType*

Response data

Adapter (MQCFIN)
Adapter number (parameter identifier: MQIACH_ADAPTER).

The adapter number on which NetBIOS listens.

Backlog (MQCFIN)
Backlog (parameter identifier: MQIACH_BACKLOG).

The number of concurrent connection requests that the listener supports.

Commands (MQCFIN)
Adapter number (parameter identifier: MQIACH_COMMAND_COUNT).

The number of commands that the listener can use.

IPAddress (MQCFST)
IP address (parameter identifier: MQCACH_IP_ADDRESS).

IP address for the listener specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric host name form.

The maximum length of the string is MQ_CONN_NAME_LENGTH

ListenerDesc (MQCFST)
Description of listener definition (parameter identifier: MQCACH_LISTENER_DESC).

The maximum length of the string is MQ_LISTENER_DESC_LENGTH.

ListenerName (MQCFST)
Name of listener definition (parameter identifier: MQCACH_LISTENER_NAME).

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

LocalName (MQCFST)
NetBIOS local name (parameter identifier: MQCACH_LOCAL_NAME).

The NetBIOS local name that the listener uses.

The maximum length of the string is MQ_CONN_NAME_LENGTH

NetbiosNames (MQCFIN)
NetBIOS names (parameter identifier: MQIACH_NAME_COUNT).

The number of names that the listener supports.

Port (MQCFIN)
Port number (parameter identifier: MQIACH_PORT).

The port number for TCP/IP.

ProcessId (MQCFIN)
Process identifier (parameter identifier: MQIACF_PROCESS_ID).

The operating system process identifier associated with the listener.

Sessions (MQCFIN)
NetBIOS sessions (parameter identifier: MQIACH_SESSION_COUNT).

The number of sessions that the listener can use.

Socket (MQCFIN)
SPX socket number (parameter identifier: MQIACH_SOCKET).

The SPX socket on which the listener is to listen.

StartDate (MQCFST)
Start date (parameter identifier: MQCACH_LISTENER_START_DATE).

The date, in the form `yyyy-mm-dd`, on which the listener was started.

The maximum length of the string is MQ_DATE_LENGTH

StartMode (MQCFIN)
Service mode (parameter identifier: MQIACH_LISTENER_CONTROL).

Specifies how the listener is to be started and stopped. The value can be any of the following values:

MQSVC_CONTROL_MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by user command. MQSVC_CONTROL_MANUAL is the default value.

MQSVC_CONTROL_Q_MGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

MQSVC_CONTROL_Q_MGR_START
The listener is to be started at the same time as the queue manager is started, but is not request to stop when the queue manager is stopped.

StartTime (MQCFST)
Start date (parameter identifier: MQCACH_LISTENER_START_TIME).

The time, in the form `hh.mm.ss`, at which the listener was started.

The maximum length of the string is MQ_TIME_LENGTH

Status (MQCFIN)
Listener status (parameter identifier: MQIACH_LISTENER_STATUS).

The status of the listener. The value can be any of the following values:

MQSVC_STATUS_STARTING
The listener is in the process of initializing.

MQSVC_STATUS_RUNNING
The listener is running.

MQSVC_STATUS_STOPPING
The listener is stopping.

TPName (MQCFST)
Transaction program name (parameter identifier: MQCACH_TP_NAME).

The LU 6.2 transaction program name.

The maximum length of the string is MQ_TP_NAME_LENGTH

TransportType (MQCFIN)
Transmission protocol (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value can be:

MQXPT_TCP
TCP.

MQXPT_LU62
LU 6.2. MQXPT_LU62 is valid only on Windows.

MQXPT_NETBIOS
NetBIOS. MQXPT_NETBIOS is valid only on Windows.

MQXPT_SPX
SPX. MQXPT_SPX is valid only on Windows.

Inquire Channel Names:

The Inquire Channel Names (MQCMD_INQUIRE_CHANNEL_NAMES) command inquires a list of IBM MQ channel names that match the generic channel name, and the optional channel type specified.

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Generic channel names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

If present, this parameter limits the channel names returned to channels of the specified type.

The value can be any of the following values:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLNTCONN
Client connection.

MQCHT_CLUSRCVR
Cluster-receiver.

MQCHT_CLUSSDR
Cluster-sender.

MQCHT_ALL
All types.

The default value if this parameter is not specified is MQCHT_ALL, which means that channels of all types except MQCHT_CLNTCONN are eligible.

z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

z/OS

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined with either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

Error code

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

Inquire Channel Names (Response):

The response to the Inquire Channel Names (MQCMD_INQUIRE_CHANNEL_NAMES) command consists of one response for each client connection channel (except for SYSTEM.DEF.CLNTCONN), and a final message with all the remaining channels.

Always returned:
ChannelNames, *ChannelTypes*

Returned if requested:
None

z/OS
On z/OS only, one additional parameter structure (with the same number of entries as the *ChannelNames* structure), is returned. Each entry in the structure, *QSGDispositions*, indicates the disposition of the object with the corresponding entry in the *ChannelNames* structure.

Response data

ChannelNames (MQCFSL)
List of channel names (parameter identifier: MQCACH_CHANNEL_NAMES).

ChannelTypes (MQCFIL)
List of channel types (parameter identifier: MQIACH_CHANNEL_TYPES). Possible values for fields in this structure are those values permitted for the **ChannelType** parameter, except MQCHT_ALL.

z/OS

QSGDispositions (MQCFIL)
List of queue-sharing group dispositions (parameter identifier: MQIACF_QSG_DISPS). This parameter is valid only on z/OS. The value can be:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

Inquire Channel Status:

The Inquire Channel Status (MQCMD_INQUIRE_CHANNEL_STATUS) command inquires about the status of one or more channel instances.

You must specify the name of the channel for which you want to inquire status information. This name can be a specific channel name or a generic channel name. By using a generic channel name, you can inquire either:

- Status information for all channels, or
- Status information for one or more channels that match the specified name.

You must also specify whether you want:

- The status data (of current channels only), or
- The saved status data of all channels, or
- On z/OS only, the short status data of the channel.

  Status for all channels that meet the selection criteria is returned, whether the channels were defined manually or automatically.

Selection

The way to make a selection, is to use one of the following four options:

- **XmitQname** (MQCACH_XMIT_Q_NAME)
- **ConnectionName** (MQCACH_CONNECTION_NAME)
- z/OS **ChannelDisposition** (MQIACH_CHANNEL_DISP)
- **ChannelInstanceType** (MQIACH_CHANNEL_INSTANCE_TYPE)

Multi This command includes a check on the current depth of the transmission queue for the channel, if the channel is a CLUSSDR channel. To issue this command, you must be authorized to inquire the queue depth, and to do this requires *+inq* authority on the transmission queue. Note that another name for this authority is MQZAO_INQUIRE.

Multi Without this authority this command runs without error, but a value of zero is output for the **MsgsAvailable** parameter of the "Inquire Channel Status (Response)" on page 1322 command. If you have the correct authority, the command provides the correct value for **MsgsAvailable**.

There are three classes of data available for channel status. These classes are **saved**, **current**, and **short**. The status fields available for saved data are a subset of the fields available for current data and are called **common** status fields. Although the common data *fields* are the same, the data *values* might be different for saved and current status. The rest of the fields available for current data are called **current-only** status fields.

- **Saved** data consists of the common status fields. This data is reset at the following times:
  - For all channels:
    - When the channel enters or leaves STOPPED or RETRY state
  - For a sending channel:
    - Before requesting confirmation that a batch of messages has been received
    - When confirmation has been received
  - For a receiving channel:
    - Just before confirming that a batch of messages has been received
  - For a server connection channel:
    - No data is saved

Therefore, a channel which has never been current does not have any saved status.

- **Current** data consists of the common status fields and current-only status fields. The data fields are continually updated as messages are sent or received.
- **Short** data consists of the queue manager name that owns the channel instance. This class of data is available only on z/OS.

This method of operation has the following consequences:

- An inactive channel might not have any saved status if it has never been current or has not yet reached a point where saved status is reset.
- The "common" data fields might have different values for saved and current status.
- A current channel always has current status and might have saved status.

Channels can be current or inactive:

Current channels
: These are channels that have been started, or on which a client has connected, and that have not finished or disconnected normally. They might not yet have reached the point of transferring messages, or data, or even of establishing contact with the partner. Current channels have **current** status and can also have **saved** or **short** status.

  The term **Active** is used to describe the set of current channels which are not stopped.

Inactive channels
: These are channels that have either not been started or on which a client has not connected, or that have finished or disconnected normally. (If a channel is stopped, it is not yet considered to have finished normally and is, therefore, still current.) Inactive channels have either **saved** status or no status at all.

There can be more than one instance of a receiver, requester, cluster-sender, cluster-receiver, or server-connection channel current at the same time (the requester is acting as a receiver). This situation occurs if several senders, at different queue managers, each initiate a session with this receiver, using the same channel name. For channels of other types, there can only be one instance current at any time.

For all channel types, however, there can be more than one set of saved status information available for a particular channel name. At most one of these sets relates to a current instance of the channel, the rest relate to previously current instances. Multiple instances arise if different transmission queue names or connection names have been used with the same channel. This situation can happen in the following cases:

- At a sender or server:
  - If the same channel has been connected to by different requesters (servers only),
  - If the transmission queue name has been changed in the definition, or
  - If the connection name has been changed in the definition.
- At a receiver or requester:
  - If the same channel has been connected to by different senders or servers, or
  - If the connection name has been changed in the definition (for requester channels initiating connection).

The number of sets returned for a particular channel can be limited by using the **XmitQName**, **ConnectionName** and **ChannelInstanceType** parameters.

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Generic channel names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The channel name is always returned, regardless of the instance attributes requested.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

z/OS **ChannelDisposition (MQCFIN)**

Channel disposition (parameter identifier: MQIACH_CHANNEL_DISP). This parameter applies to z/OS only.

Specifies the disposition of the channels for which information is to be returned. The value can be any of the following values:

MQCHLD_ALL
Returns requested status information for private channels.

In a shared queue environment where the command is being executed on the queue manager where it was issued, or if *ChannelInstanceType* has a value of MQOT_CURRENT_CHANNEL, this option also displays the requested status information for shared channels.

MQCHLD_PRIVATE
Returns requested status information for private channels.

MQCHLD_SHARED
Returns requested status information for shared channels.

The status information that is returned for various combinations of *ChannelDisposition*, *CommandScope*, and status type, is summarized in Table 128, Table 129 on page 1310, and Table 130 on page 1310.

Table 128. ChannelDisposition and CommandScope for Inquire Channel Status, Current

ChannelDisposition	*CommandScope* **blank or local queue manager**	*CommandScope(qmgr-name)*	*CommandScope(*)*
MQCHLD_PRIVATE	Common and current-only status for current private channels on the local queue manager	Common and current-only status for current private channels on the named queue manager	Common and current-only status for current private channels on all queue managers
MQCHLD_SHARED	Common and current-only status for current shared channels on the local queue manager	Common and current-only status for current shared channels on the named queue manager	Common and current-only status for current shared channels on all queue managers
MQCHLD_ALL	Common and current-only status for current private and shared channels on the local queue manager	Common and current-only status for current private and shared channels on the named queue manager	Common and current-only status for current private and shared channels on all active queue managers

Table 129. ChannelDisposition and CommandScope for Inquire Channel Status, Short

ChannelDisposition	*CommandScope* **blank or local queue manager**	*CommandScope(qmgr-name)*	*CommandScope(*)*
MQCHLD_PRIVATE	*ChannelStatus* and short status for current private channels on the local queue manager	*ChannelStatus* and short status for current private channels on the named queue manager	*ChannelStatus* and short status for current private channels on all active queue managers
MQCHLD_SHARED	*ChannelStatus* and short status for current shared channels on all active queue managers in the queue-sharing group	Not permitted	Not permitted
MQCHLD_ALL	*ChannelStatus* and short status for current private channels on the local queue manager and current shared channels in the queue-sharing group(1)	*ChannelStatus* and short status for current private channels on the named queue manager	*ChannelStatus* and short status for current private, and shared, channels on all active queue managers in the queue-sharing group(1)

Note:

1. In this case you get two separate sets of responses to the command on the queue manager where it was entered; one for MQCHLD_PRIVATE and one for MQCHLD_SHARED.

Table 130. ChannelDisposition and CommandScope for Inquire Channel Status, Saved

ChannelDisposition	*CommandScope* **blank or local queue manager**	*CommandScope(qmgr-name)*	*CommandScope(*)*
MQCHLD_PRIVATE	Common status for saved private channels on the local queue manager	Common status for saved private channels on the named queue manager	Common status for saved private channels on all active queue managers
MQCHLD_SHARED	Common status for saved shared channels on all active queue managers in the queue-sharing group	Not permitted	Not permitted
MQCHLD_ALL	Common status for saved private channels on the local queue manager and saved shared channels in the queue-sharing group	Common status for saved private channels on the named queue manager	Common status for saved private, and shared, channels on all active queue managers in the queue-sharing group

You cannot use this parameter as a filter keyword.

ChannelInstanceAttrs (MQCFIL)

Channel instance attributes (parameter identifier: MQIACH_CHANNEL_INSTANCE_ATTRS).

The **ChannelInstanceAttrs** parameter names the list of attributes to be returned. This parameter does not provide any way to select, based upon the value of the items in that list of attributes.

If status information is requested which is not relevant for the particular channel type, it is not an error. Similarly, it is not an error to request status information that is applicable only to active channels for saved channel instances. In both of these cases, no structure is returned in the response for the information concerned.

For a saved channel instance, the MQCACH_CURRENT_LUWID, MQIACH_CURRENT_MSGS, and MQIACH_CURRENT_SEQ_NUMBER attributes have meaningful information only if the channel instance is in doubt. However, the attribute values are still returned when requested, even if the channel instance is not in-doubt.

The attribute list might specify the following value on its own:

MQIACF_ALL
All attributes.

MQIACF_ALL is the default value used if the parameter is not specified or it can specify a combination of the following:

- Relevant for common status:

  The following information applies to all sets of channel status, whether the set is current.

  MQCACH_CHANNEL_NAME
  Channel name.

  MQCACH_CONNECTION_NAME
  Connection name.

  MQCACH_CURRENT_LUWID
  Logical unit of work identifier for current batch.

  MQCACH_LAST_LUWID
  Logical unit of work identifier for last committed batch.

  MQCACH_XMIT_Q_NAME
  Transmission queue name.

  MQIACH_CHANNEL_INSTANCE_TYPE
  Channel instance type.

  MQIACH_CHANNEL_TYPE
  Channel type.

  MQIACH_CURRENT_MSGS
  Number of messages sent or received in current batch.

  MQIACH_CURRENT_SEQ_NUMBER
  Sequence number of last message sent or received.

  MQIACH_INDOUBT_STATUS
  Whether the channel is currently in-doubt.

  MQIACH_LAST_SEQ_NUMBER
  Sequence number of last message in last committed batch.

  MQCACH_CURRENT_LUWID, MQCACH_LAST_LUWID, MQIACH_CURRENT_MSGS, MQIACH_CURRENT_SEQ_NUMBER, MQIACH_INDOUBT_STATUS and MQIACH_LAST_SEQ_NUMBER do not apply to server-connection channels, and no values are returned. If specified on the command, they are ignored.

- Relevant for current-only status:

  The following information applies only to current channel instances. The information applies to all channel types, except where stated.

  MQCA_Q_MGR_NAME
  Name of the queue manager that owns the channel instance. This parameter is valid only on z/OS.

  MQCA_REMOTE_Q_MGR_NAME
  Queue manager name, or queue-sharing group name of the remote system. The remote queue manager name is always returned regardless of the instance attributes requested.

MQCACH_CHANNEL_START_DATE
Date channel was started.

MQCACH_CHANNEL_START_TIME
Time channel was started.

MQCACH_LAST_MSG_DATE
Date last message was sent, or MQI call was handled.

MQCACH_LAST_MSG_TIME
Time last message was sent, or MQI call was handled.

MQCACH_LOCAL_ADDRESS
Local communications address for the channel.

MQCACH_MCA_JOB_NAME
Name of MCA job.

This parameter is not valid on z/OS.

You cannot use MQCACH_MCA_JOB_NAME as a parameter to filter on.

MQCACH_MCA_USER_ID
The user ID used by the MCA.

MQCACH_REMOTE_APPL_TAG
Remote partner application name. MQCACH_REMOTE_APPL_TAG is the name of the client application at the remote end of the channel. This parameter applies only to server-connection channels.

MQCACH_REMOTE_PRODUCT
Remote partner product identifier. This is the product identifier of the IBM MQ code running at the remote end of the channel.

MQCACH_REMOTE_VERSION
Remote partner version. This is the version of the IBM MQ code running at the remote end of the channel.

MQCACH_SSL_SHORT_PEER_NAME
TLS short peer name.

MQCACH_SSL_CERT_ISSUER_NAME
The full Distinguished Name of the issuer of the remote certificate.

z/OS **MQCACH_SSL_CERT_USER_ID**
User ID associated with the remote certificate; valid on z/OS only.

MQCACH_TOPIC_ROOT
Topic root for AMQP channel.

MQIA_MONITORING_CHANNEL
The level of monitoring data collection.

z/OS **MQIA_STATISTICS_CHANNEL**
The level of statistics data collection; valid on z/OS only.

MQIACF_MONITORING
All channel status monitoring attributes. These attributes are:

MQIA_MONITORING_CHANNEL
The level of monitoring data collection.

MQIACH_BATCH_SIZE_INDICATOR
Batch size.

MQIACH_COMPRESSION_RATE
The compression rate achieved displayed to the nearest percentage.

MQIACH_COMPRESSION_TIME
The amount of time per message, displayed in microseconds, spent during compression or decompression.

MQIACH_EXIT_TIME_INDICATOR
Exit time.

MQIACH_NETWORK_TIME_INDICATOR
Network time.

MQIACH_XMITQ_MSGS_AVAILABLE
Number of messages available to the channel on the transmission queue.

MQIACH_XMITQ_TIME_INDICATOR
Time on transmission queue.

You cannot use MQIACF_MONITORING as a parameter to filter on.

MQIACH_BATCH_SIZE_INDICATOR
Batch size.

You cannot use MQIACH_BATCH_SIZE_INDICATOR as a parameter to filter on.

MQIACH_BATCHES
Number of completed batches.

MQIACH_BUFFERS_RCVD
Number of buffers received.

MQIACH_BUFFERS_SENT
Number of buffers sent.

MQIACH_BYTES_RCVD
Number of bytes received.

MQIACH_BYTES_SENT
Number of bytes sent.

MQIACH_CHANNEL_SUBSTATE
The channel substate.

MQIACH_COMPRESSION_RATE
The compression rate achieved displayed to the nearest percentage.

You cannot use MQIACH_COMPRESSION_RATE as a parameter to filter on.

MQIACH_COMPRESSION_TIME
The amount of time per message, displayed in microseconds, spent during compression or decompression.

You cannot use MQIACH_COMPRESSION_TIME as a parameter to filter on.

MQIACH_CURRENT_SHARING_CONVS
Requests information about the current number of conversations on this channel instance.

This attribute applies only to TCP/IP server-connection channels.

MQIACH_EXIT_TIME_INDICATOR
Exit time.

You cannot use MQIACH_EXIT_TIME_INDICATOR as a parameter to filter on.

MQIACH_HDR_COMPRESSION
Technique used to compress the header data sent by the channel.

MQIACH_KEEP_ALIVE_INTERVAL
The KeepAlive interval in use for this session. This parameter is significant only for z/OS.

MQIACH_LONG_RETRIES_LEFT
Number of long retry attempts remaining.

MQIACH_MAX_MSG_LENGTH
Maximum message length. MQIACH_MAX_MSG_LENGTH is valid only on z/OS.

MQIACH_MAX_SHARING_CONVS
Requests information about the maximum number of conversations on this channel instance.

This attribute applies only to TCP/IP server-connection channels.

MQIACH_MCA_STATUS
MCA status.

You cannot use MQIACH_MCA_STATUS as a parameter to filter on.

MQIACH_MSG_COMPRESSION
Technique used to compress the message data sent by the channel.

MQIACH_MSGS
Number of messages sent or received, or number of MQI calls handled.

MQIACH_NETWORK_TIME_INDICATOR
Network time.

You cannot use MQIACH_NETWORK_TIME_INDICATOR as a parameter on which to filter.

MQIACH_SECURITY_PROTOCOL
Security protocol currently in use.

This parameter does not apply to client-connection channels.

This parameter does not apply to z/OS.

MQIACH_SHORT_RETRIES_LEFT
Number of short retry attempts remaining.

MQIACH_SSL_KEY_RESETS
Number of successful TLS key resets.

MQIACH_SSL_RESET_DATE
Date of previous successful TLS secret key reset.

MQIACH_SSL_RESET_TIME
Time of previous successful TLS secret key reset.

MQIACH_STOP_REQUESTED
Whether user stop request has been received.

MQIACH_XMITQ_MSGS_AVAILABLE
Number of messages available to the channel on the transmission queue.

MQIACH_XMITQ_TIME_INDICATOR
Time on transmission queue.

You cannot use MQIACH_XMITQ_TIME_INDICATOR as a parameter to filter on.

The following value is supported on all platforms:

MQIACH_BATCH_SIZE
Batch size.

The following value is supported on all platforms:

MQIACH_HB_INTERVAL
Heartbeat interval (seconds).

MQIACH_NPM_SPEED
Speed of nonpersistent messages.

The following attributes do not apply to server-connection channels, and no values are returned. If specified on the command they are ignored:

- MQIACH_BATCH_SIZE_INDICATOR
- MQIACH_BATCH_SIZE
- MQIACH_BATCHES
- MQIACH_LONG_RETRIES_LEFT
- MQIACH_NETWORK_TIME
- MQIACH_NPM_SPEED
- MQCA_REMOTE_Q_MGR_NAME
- MQIACH_SHORT_RETRIES_LEFT
- MQIACH_XMITQ_MSGS_AVAILABLE
- MQIACH_XMITQ_TIME_INDICATOR

The following attributes apply only to server-connection channels. If specified on the command for other types of channel the attribute is ignored and no value is returned:

- MQIACH_CURRENT_SHARING_CONVS
- MQIACH_MAX_SHARING_CONVS

- z/OS Relevant for short status:

The following parameter applies to current channels on z/OS:

MQCACH_Q_MGR_NAME
Name of the queue manager that owns the channel instance.

ChannelInstanceType (MQCFIN)
Channel instance type (parameter identifier: MQIACH_CHANNEL_INSTANCE_TYPE).

It is always returned regardless of the channel instance attributes requested.

The value can be:

MQOT_CURRENT_CHANNEL
The channel status.

MQOT_CURRENT_CHANNEL is the default, and indicates that only current status information for active channels is to be returned.

Both common status information and active-only status information can be requested for current channels.

MQOT_SAVED_CHANNEL
Saved channel status.

Specify MQOT_SAVED_CHANNEL to cause saved status information for both active and inactive channels to be returned.

Only common status information can be returned. Active-only status information is not returned for active channels if this keyword is specified.

z/OS **MQOT_SHORT_CHANNEL**
Short channel status (valid on z/OS only).

Specify MQOT_SHORT_CHANNEL to cause short status information for current channels to be returned.

Other common status and current-only status information are not returned for current channels if this keyword is specified.

You cannot use MQIACH_CHANNEL_INSTANCE_TYPE as a parameter to filter on.

z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

If this parameter is present, eligible channel instances are limited to those using this connection name. If it is not specified, eligible channel instances are not limited in this way.

The connection name is always returned, regardless of the instance attributes requested.

The value returned for *ConnectionName* might not be the same as in the channel definition, and might differ between the current channel status and the saved channel status. (Using *ConnectionName* for limiting the number of sets of status is therefore not recommended.)

For example, when using TCP, if *ConnectionName* in the channel definition:

- Is blank or is in *host name* format, the channel status value has the resolved IP address.
- Includes the port number, the current channel status value includes the port number (except on z/OS), but the saved channel status value does not.

The maximum length of the string is MQ_CONN_NAME_LENGTH.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *ChannelInstanceAttrs* except MQIACF_ALL and others as noted. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1601 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the **StringFilterCommand** parameter.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *ChannelInstanceAttrs* except MQCACH_CHANNEL_NAME and others as noted. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1608 for information about using this filter condition.

If you specify a string filter for **ConnectionName** or **XmitQName**, you cannot also specify the **ConnectionName** or **XmitQName** parameter.

If you specify a string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

If this parameter is present, eligible channel instances are limited to those using this transmission queue. If it is not specified, eligible channel instances are not limited in this way.

The transmission queue name is always returned, regardless of the instance attributes requested.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Error code

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHL_INST_TYPE_ERROR
Channel instance type not valid.

MQRCCF_CHL_STATUS_NOT_FOUND
Channel status not found.

MQRCCF_XMIT_Q_NAME_ERROR
Transmission queue name error.

Inquire Channel Status (AMQP): ULW V 9.0.0

The Inquire Channel Status (MQCMD_INQUIRE_CHANNEL_STATUS) (AMQP) command inquires about the status of one or more AMQP channel instances.

You must specify the name of the channel for which you want to inquire status information. This name can be a specific channel name or a generic channel name. By using a generic channel name, you can inquire either:

- Status information for all channels, or
- Status information for one or more channels that match the specified name.

If the **ClientIdentifier** parameter is not specified, the output of the **Inquire Channel Status** command is a summary of statuses of all clients connected to the channel. One PCF response message is returned per channel.

If the **ClientIdentifier** parameter is specified, separate PCF response messages are returned for each client connection. The **ClientIdentifier** parameter may be a wildcard, in which the status for all clients that match the **ClientIdentifier** string is returned (within the limits of **MaxResponses** and **ResponseRestartPoint** if they are set).

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Generic channel names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects which have names that start with the selected character string. An asterisk on its own matches all possible names.

This parameter is allowed for only when the **ResponseType** parameter is set to MQRESP_TOTAL.

The channel name is always returned, regardless of the instance attributes requested.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

The value must be:

MQCHT_AMQP
AMQP

Optional parameters

ClientIdentifer (MQCFST)
The ClientId of the client (parameter identifier: MQCACH_CLIENT_ID).

MaxResponses (MQCFIN)
The maximum number of clients to return status for.

ResponseRestartPoint (MQCFIN)
The first client to return status for. The combination of this parameter with **MaxResponses** enables the range of clients to be specified.

Summary mode

If you do not specify a **ClientIdentifier** parameter, the following fields are returned:

CHANNEL
The channel name (parameter identifier: MQCACH_CHANNEL_NAME).

CHLTYPE
The channel type of AMQP (parameter identifier: MQIACH_CHANNEL_TYPE).

CONNS
The number of connections described in the summary (parameter identifier: MQIACF_CONNECTION_COUNT).

STATUS
The current status of the client (parameter identifier: MQIACH_CHANNEL_STATUS).

Client details mode

If you specify a **ClientIdentifier** parameter, the following fields are returned:

STATUS
The current status of the client (parameter identifier: MQIACH_CHANNEL_STATUS).

CONNAME
The name of the remote connection (ip address) (parameter identifier: MQCACH_CONNECTION_NAME).

AMQPKA
The keep alive interval of the client (parameter identifier: MQIACH_AMQP_KEEP_ALIVE).

MCANAME
Message channel agent name (parameter identifier: MQCACH_MCA_USER_ID).

MSGSNT
Number of messages sent by the client since it last connected (parameter identifier: MQIACH_MSGS_SENT).

MSGRCVD
Number of messages received by the client since it last connected (parameter identifier: MQIACH_MSGS_RECEIVED / MQIACH_MSGS_RCVD).

LMSGDATE
Date last message was received or sent (parameter identifier: MQCACH_LAST_MSG_DATE).

LMSGTIME
Time last message was received or sent (parameter identifier: MQCACH_LAST_MSG_TIME).

CHLSDATE
Date channel started (parameter identifier: MQCACH_CHANNEL_START_DATE).

CHLSTIME
Time channel was started (parameter identifier: MQCACH_CHANNEL_START_TIME).

Error code

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHL_INST_TYPE_ERROR
Channel instance type not valid.

MQRCCF_CHL_STATUS_NOT_FOUND
Channel status not found.

MQRCCF_XMIT_Q_NAME_ERROR
Transmission queue name error.

Inquire Channel Status (MQTT): 

The Inquire Channel Status (MQCMD_INQUIRE_CHANNEL_STATUS) (MQTT) command inquires about the status of one or more Telemetry channel instances.

You must specify the name of the channel for which you want to inquire status information. This name can be a specific channel name or a generic channel name. By using a generic channel name, you can inquire either:

- Status information for all channels, or
- Status information for one or more channels that match the specified name.

Note: The **Inquire Channel Status** command for MQ Telemetry has the potential to return a far larger number of responses than if the command was run for an IBM MQ channel. For this reason, the MQ Telemetry server does not return more responses than fit on the reply-to queue. The number of responses is limited to the value of MAXDEPTH parameter of the SYSTEM.MQSC.REPLY.QUEUE queue. When an MQ Telemetry command is truncated by the MQ Telemetry server, the AMQ8492 message is displayed specifying how many responses are returned based on the size of MAXDEPTH.

If the **ClientIdentifier** parameter is not specified, the output of the **Inquire Channel Status** command is a summary of statuses of all clients connected to the channel. One PCF response message is returned per channel.

If the **ClientIdentifier** parameter is specified, separate PCF response messages are returned for each client connection. The **ClientIdentifier** parameter may be a wildcard, in which the status for all clients that match the **ClientIdentifier** string is returned (within the limits of **MaxResponses** and **ResponseRestartPoint** if they are set).

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Generic channel names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects which have names that start with the selected character string. An asterisk on its own matches all possible names.

This parameter is allowed for only when the **ResponseType** parameter is set to MQRESP_TOTAL.

The channel name is always returned, regardless of the instance attributes requested.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

The value must be:

MQCHT_MQTT
Telemetry.

Optional parameters

ClientIdentifer (MQCFST)
The ClientId of the client (parameter identifier: MQCACH_CLIENT_ID).

MaxResponses (MQCFIN)
The maximum number of clients to return status for (parameter identifier: MQIA_MAX_RESPONSES).

This parameter is only allowed when the **ClientIdentifier** parameter is specified.

ResponseRestartPoint (MQCFIN)
The first client to return status for (parameter identifier: MQIA_RESPONSE_RESTART_POINT). The combination of this parameter with **MaxResponses** enables the range of clients to be specified.

This parameter is only allowed when the **ClientIdentifier** parameter is specified.

Client details mode

STATUS
The current status of the client (parameter identifier: MQIACH_CHANNEL_STATUS).

CONNAME
The name of the remote connection (ip address) (parameter identifier: MQCACH_CONNECTION_NAME).

KAINT
The keep alive interval of the client (parameter identifier: MQIACH_KEEP_ALIVE_INTERVAL).

MCANAME
Message channel agent name (parameter identifier: MQCACH_MCA_USER_ID).

MSGSNT
Number of messages sent by the client since it last connected (parameter identifier: MQIACH_MSGS_SENT).

MSGRCVD
Number of messages received by the client since it last connected (parameter identifier: MQIACH_MSGS_RECEIVED / MQIACH_MSGS_RCVD).

INDOUBTIN
Number of in doubt, inbound messages to the client (parameter identifier: MQIACH_IN_DOUBT_IN).

INDOUBTOUT
Number of in doubt, outbound messages to the client (parameter identifier: MQIACH_IN_DOUBT_OUT).

PENDING
Number of outbound pending messages (parameter identifier: MQIACH_PENDING_OUT).

LMSGDATE
Date last message was received or sent (parameter identifier: MQCACH_LAST_MSG_DATE).

LMSGTIME
Time last message was received or sent (parameter identifier: MQCACH_LAST_MSG_TIME).

CHLSDATE
Date channel started (parameter identifier: MQCACH_CHANNEL_START_DATE).

CHLSTIME
Time channel was started (parameter identifier: MQCACH_CHANNEL_START_TIME).

Error code

This command might return the following error code in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHL_INST_TYPE_ERROR
Channel instance type not valid.

MQRCCF_CHL_STATUS_NOT_FOUND
Channel status not found.

MQRCCF_XMIT_Q_NAME_ERROR
Transmission queue name error.

Inquire Channel Status (Response):

The response to the Inquire Channel Status (MQCMD_INQUIRE_CHANNEL_STATUS) command consists of the response header followed by several structures.

These structures are

- The *ChannelName* structure,
- z/OS The *ChannelDisposition* structure (on z/OS only),
- The *ChannelInstanceType* structure
- The *ChannelStatus* structure (except on z/OS channels whose **ChannelInstanceType** parameter has a value of MQOT_SAVED_CHANNEL.
- The **ChannelType** structure
- The **ConnectionName** structure
- The **RemoteApplTag** structure
- The **RemoteQMgrName** structure
- The **StopRequested** structure
- The **XmitQName** structure

which are then followed by the requested combination of status attribute parameter structures. One such message is generated for each channel instance found that matches the criteria specified on the command.

z/OS On z/OS, if the value for any of these parameters exceeds 999999999, it is returned as 999999999:

- *Batches*
- *BuffersReceived*
- *BuffersSent*
- *BytesReceived*
- *BytesSent*
- *CompressionTime*
- *CurrentMsgs*
- *ExitTime*
- *Msgs*
- *NetTime*
- *SSLKeyResets*
- *XQTime*

Always returned:

z/OS *ChannelDisposition*, *ChannelInstanceType*, *ChannelName*, *ChannelStatus*, *ChannelType*, *ConnectionName*, *RemoteApplTag*, *RemoteQMgrName*, *StopRequested*, *SubState*, *XmitQName*

Returned if requested:

Batches, *BatchSize*, *BatchSizeIndicator*, *BuffersReceived*, *BuffersSent*, *BytesReceived*, *BytesSent*, *ChannelMonitoring*, *ChannelStartDate*, *ChannelStartTime*, *ClientIdentifier*, *CompressionRate*, *CompressionTime*, *CurrentLUWID*, *CurrentMsgs*, *CurrentSequenceNumber*, *CurrentSharingConversations*, *ExitTime*, *HeaderCompression*, *HeartbeatInterval*, *InDoubtInbound*, *InDoubtStatus*, *InDoubtOutbound*, *KeepAliveInterval*, *LastLUWID*, *LastMsgDate*, *LastMsgTime*, *LastSequenceNumber*, *LocalAddress*, *LongRetriesLeft*, *MaxMsgLength*, *MaxSharingConversations*, *MCAJobName*, *MCAStatus*, *MCAUserIdentifier*, *MessageCompression*, *Msgs*, *MsgsAvailable*, *MsgsReceived*, *MsgsSent*, *NetTime*, *NonPersistentMsgSpeed*, *PendingOutbound*, *QMgrName*,

ResponseType, *RemoteVersion*, *RemoteProduct*, *SecurityProtocol*, *ShortRetriesLeft*, *SSLCertRemoteIssuerName*, *SSLCertUserId*, *SSLKeyResetDate*, *SSLKeyResets*, *SSLKeyResetTime*, *SSLShortPeerName*, *XQTime*

Response data

Batches (MQCFIN)
Number of completed batches (parameter identifier: MQIACH_BATCHES).

BatchSize (MQCFIN)
Negotiated batch size (parameter identifier: MQIACH_BATCH_SIZE).

BatchSizeIndicator (MQCFIL)
Indicator of the number of messages in a batch (parameter identifier: MQIACH_BATCH_SIZE_INDICATOR). Two values are returned:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

BuffersReceived (MQCFIN)
Number of buffers received (parameter identifier: MQIACH_BUFFERS_RCVD).

BuffersSent (MQCFIN)
Number of buffers sent (parameter identifier: MQIACH_BUFFERS_SENT).

BytesReceived (MQCFIN)
Number of bytes received (parameter identifier: MQIACH_BYTES_RCVD).

BytesSent (MQCFIN)
Number of bytes sent (parameter identifier: MQIACH_BYTES_SENT).

z/OS

ChannelDisposition (MQCFIN)
Channel disposition (parameter identifier: MQIACH_CHANNEL_DISP). This parameter is valid only on z/OS.

The value can be any of the following values:

MQCHLD_PRIVATE
Status information for a private channel.

MQCHLD_SHARED
Status information for a shared channel.

MQCHLD_FIXSHARED
Status information for a shared channel, tied to a specific queue manager.

ChannelInstanceType (MQCFIN)
Channel instance type (parameter identifier: MQIACH_CHANNEL_INSTANCE_TYPE).

The value can be any of the following values:

MQOT_CURRENT_CHANNEL
Current channel status.

MQOT_SAVED_CHANNEL
Saved channel status.

z/OS **MQOT_SHORT_CHANNEL**
Short channel status, only on z/OS.

ChannelMonitoring (MQCFIN)
Current level of monitoring data collection for the channel (parameter identifier: MQIA_MONITORING_CHANNEL).

The value can be any of the following values:

MQMON_OFF
Monitoring for the channel is disabled.

MQMON_LOW
Low rate of data collection.

MQMON_MEDIUM
Medium rate of data collection.

MQMON_HIGH
High rate of data collection.

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelStartDate (MQCFST)
Date channel started, in the form `yyyy-mm-dd` (parameter identifier: MQCACH_CHANNEL_START_DATE).

The maximum length of the string is MQ_CHANNEL_DATE_LENGTH.

ChannelStartTime (MQCFST)
Time channel started, in the form `hh.mm.ss` (parameter identifier: MQCACH_CHANNEL_START_TIME).

The maximum length of the string is MQ_CHANNEL_TIME_LENGTH.

z/OS

z/OS **ChannelStatistics (MQCFIN)**
Specifies whether statistics data is to be collected for channels (parameter identifier: `MQIA_STATISTICS_CHANNEL`).

The value can be:

MQMON_OFF
Statistics data collection is turned off.

MQMON_LOW
Statistics data collection is turned on, with a low ratio of data collection.

MQMON_MEDIUM
Statistics data collection is turned on, with a moderate ratio of data collection.

MQMON_HIGH
Statistics data collection is turned on, with a high ratio of data collection.

This parameter is valid only on z/OS.

ChannelStatus (MQCFIN)
Channel status (parameter identifier: MQIACH_CHANNEL_STATUS).

Channel status has the following values defined:

MQCHS_BINDING
Channel is negotiating with the partner.

MQCHS_STARTING
Channel is waiting to become active.

MQCHS_RUNNING
Channel is transferring or waiting for messages.

MQCHS_PAUSED
Channel is paused.

MQCHS_STOPPING
Channel is in process of stopping.

MQCHS_RETRYING
Channel is reattempting to establish connection.

MQCHS_STOPPED
Channel is stopped.

MQCHS_REQUESTING
Requester channel is requesting connection.

MQCHS_SWITCHING
Channel is switching transmission queues.

MQCHS_INITIALIZING
Channel is initializing.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

The value can be any of the following values:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLNTCONN
Client connection.

MQCHT_CLUSRCVR
Cluster-receiver.

MQCHT_CLUSSDR
Cluster-sender.

CompressionRate (MQCFIL)
The compression rate achieved displayed to the nearest percentage (parameter identifier: MQIACH_COMPRESSION_RATE). Two values are returned:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

CompressionTime (MQCFIL)
The amount of time per message, displayed in microseconds, spent during compression or decompression (parameter identifier: MQIACH_COMPRESSION_TIME). Two values are returned:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_SHORT_CONN_NAME_LENGTH.

CurrentLUWID (MQCFST)
Logical unit of work identifier for in-doubt batch (parameter identifier: MQCACH_CURRENT_LUWID).

The logical unit of work identifier associated with the current batch, for a sending or a receiving channel.

For a sending channel, when the channel is in-doubt it is the LUWID of the in-doubt batch.

It is updated with the LUWID of the next batch when it is known.

The maximum length is MQ_LUWID_LENGTH.

CurrentMsgs (MQCFIN)
Number of messages in-doubt (parameter identifier: MQIACH_CURRENT_MSGS).

For a sending channel, this parameter is the number of messages that have been sent in the current batch. It is incremented as each message is sent, and when the channel becomes in-doubt it is the number of messages that are in-doubt.

For a receiving channel, it is the number of messages that have been received in the current batch. It is incremented as each message is received.

The value is reset to zero, for both sending and receiving channels, when the batch is committed.

CurrentSequenceNumber (MQCFIN)
Sequence number of last message in in-doubt batch (parameter identifier: MQIACH_CURRENT_SEQ_NUMBER).

For a sending channel, this parameter is the message sequence number of the last message sent. It is updated as each message is sent, and when the channel becomes in-doubt it is the message sequence number of the last message in the in-doubt batch.

For a receiving channel, it is the message sequence number of the last message that was received. It is updated as each message is received.

CurrentSharingConversations (MQCFIN)
Number of conversations currently active on this channel instance (parameter identifier: MQIACH_CURRENT_SHARING_CONVS).

This parameter is returned only for TCP/IP server-connection channels.

A value of zero indicates that the channel instance is running in a mode before IBM WebSphere MQ Version 7.0, regarding:

- Administrator stop-quiesce
- Heartbeating
- Read ahead
- Client asynchronous consumption

ExitTime (MQCFIL)
Indicator of the time taken executing user exits per message (parameter identifier:

MQIACH_EXIT_TIME_INDICATOR). Amount of time, in microseconds, spent processing user exits per message. Where more than one exit is executed per message, the value is the sum of all the user exit times for a single message. Two values are returned:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

HeaderCompression (MQCFIL)
Whether the header data sent by the channel is compressed (parameter identifier: MQIACH_HDR_COMPRESSION). Two values are returned:

- The default header data compression value negotiated for this channel.
- The header data compression value used for the last message sent. The header data compression value can be altered in a sending channels message exit. If no message has been sent, the second value is MQCOMPRESS_NOT_AVAILABLE.

The values can be:

MQCOMPRESS_NONE
No header data compression is performed. MQCOMPRESS_NONE is the default value.

MQCOMPRESS_SYSTEM
Header data compression is performed.

MQCOMPRESS_NOT_AVAILABLE
No message has been sent by the channel.

HeartbeatInterval (MQCFIN)
Heartbeat interval (parameter identifier: MQIACH_HB_INTERVAL).

InDoubtStatus (MQCFIN)
Whether the channel is currently in doubt (parameter identifier: MQIACH_INDOUBT_STATUS).

A sending channel is only in doubt while the sending Message Channel Agent is waiting for an acknowledgment that a batch of messages, which it has sent, has been successfully received. It is not in doubt at all other times, including the period during which messages are being sent, but before an acknowledgment has been requested.

A receiving channel is never in doubt.

The value can be any of the following values:

MQCHIDS_NOT_INDOUBT
Channel is not in-doubt.

MQCHIDS_INDOUBT
Channel is in-doubt.

KeepAliveInterval (MQCFIN)
KeepAlive interval (parameter identifier: MQIACH_KEEP_ALIVE_INTERVAL). This parameter is valid only on z/OS.

LastLUWID (MQCFST)
Logical unit of work identifier for last committed batch (parameter identifier: MQCACH_LAST_LUWID).

The maximum length is MQ_LUWID_LENGTH.

LastMsgDate (MQCFST)
Date last message was sent, or MQI call was handled, in the form `yyyy-mm-dd` (parameter identifier: MQCACH_LAST_MSG_DATE).

The maximum length of the string is MQ_CHANNEL_DATE_LENGTH.

LastMsgTime (MQCFST)
Time last message was sent, or MQI call was handled, in the form `hh.mm.ss` (parameter identifier: MQCACH_LAST_MSG_TIME).

The maximum length of the string is MQ_CHANNEL_TIME_LENGTH.

LastSequenceNumber (MQCFIN)
Sequence number of last message in last committed batch (parameter identifier: MQIACH_LAST_SEQ_NUMBER).

LocalAddress (MQCFST)
Local communications address for the channel (parameter identifier: MQCACH_LOCAL_ADDRESS).

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

LongRetriesLeft (MQCFIN)
Number of long retry attempts remaining (parameter identifier: MQIACH_LONG_RETRIES_LEFT).

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIACH_MAX_MSG_LENGTH). This parameter is valid only on z/OS.

MaxSharingConversations (MQCFIN)
Maximum number of conversations permitted on this channel instance. (parameter identifier: MQIACH_MAX_SHARING_CONVS)

This parameter is returned only for TCP/IP server-connection channels.

A value of zero indicates that the channel instance is running in a mode before IBM WebSphere MQ Version 7.0, regarding:

- Administrator stop-quiesce
- Heartbeating
- Read ahead
- Client asynchronous consumption

MCAJobName (MQCFST)
Name of MCA job (parameter identifier: MQCACH_MCA_JOB_NAME).

The maximum length of the string is MQ_MCA_JOB_NAME_LENGTH.

MCAStatus (MQCFIN)
MCA status (parameter identifier: MQIACH_MCA_STATUS).

The value can be any of the following values:

MQMCAS_STOPPED
Message channel agent stopped.

MQMCAS_RUNNING
Message channel agent running.

MCAUserIdentifier (MQCFST)
The user ID used by the MCA (parameter identifier: MQCACH_MCA_USER_ID).

This parameter applies only to server-connection, receiver, requester, and cluster-receiver channels.

The maximum length of the string is MQ_MCA_USER_ID_LENGTH.

MessageCompression (MQCFIL)
Whether the message data sent by the channel is compressed (parameter identifier: MQIACH_MSG_COMPRESSION). Two values are returned:

- The default message data compression value negotiated for this channel.

- The message data compression value used for the last message sent. The message data compression value can be altered in a sending channels message exit. If no message has been sent, the second value is MQCOMPRESS_NOT_AVAILABLE.

The values can be:

MQCOMPRESS_NONE
No message data compression is performed. MQCOMPRESS_NONE is the default value.

MQCOMPRESS_RLE
Message data compression is performed using run-length encoding.

MQCOMPRESS_ZLIBFAST
Message data compression is performed using ZLIB encoding with speed prioritized.

MQCOMPRESS_ZLIBHIGH
Message data compression is performed using ZLIB encoding with compression prioritized.

MQCOMPRESS_NOT_AVAILABLE
No message has been sent by the channel.

Msgs (MQCFIN)
Number of messages sent or received, or number of MQI calls handled (parameter identifier: MQIACH_MSGS).

MsgsAvailable (MQCFIN)
Number of messages available (parameter identifier: MQIACH_XMITQ_MSGS_AVAILABLE). Number of messages queued on the transmission queue available to the channel for MQGETs.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

This parameter applies to cluster sender channels only.

NetTime (MQCFIL)
Indicator of the time of a network operation (parameter identifier: MQIACH_NETWORK_TIME_INDICATOR). Amount of time, in microseconds, to send a request to the remote end of the channel and receive a response. This time only measures the network time for such an operation. Two values are returned:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

NonPersistentMsgSpeed (MQCFIN)
Speed at which nonpersistent messages are to be sent (parameter identifier: MQIACH_NPM_SPEED).

The value can be any of the following values:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

QMgrName (MQCFST)
Name of the queue manager that owns the channel instance (parameter identifier: MQCA_Q_MGR_NAME). This parameter is valid only on z/OS.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

RemoteApplTag (MQCFST)
The remote partner application name. This parameter is the name of the client application at the remote end of the channel. This parameter applies only to server-connection channels (parameter identifier: MQCACH_REMOTE_APPL_TAG).

RemoteProduct (MQCFST)
The remote partner product identifier. This parameter is the product identifier of the IBM MQ code running at the remote end of the channel (parameter identifier: MQCACH_REMOTE_PRODUCT).

The possible values are shown in the following table:

Table 131. Product Identifier values

Product Identifier	Description
MQMM	Queue Manager (non z/OS Platform)
MQMV	Queue Manager on z/OS
MQCC	IBM MQ C client
MQNM	IBM MQ .NET fully managed client
MQJB	IBM MQ Classes for JAVA
MQJM	IBM MQ Classes for JMS (normal mode)
MQJN	IBM MQ Classes for JMS (migration mode)
MQJU	Common Java interface to the MQI
MQXC	XMS client C/C++ (normal mode)
MQXD	XMS client C/C++ (migration mode)
MQXN	XMS client .NET (normal mode)
MQXM	XMS client .NET (migration mode)
MQXU	IBM MQ .NET XMS client (unmanaged/XA)
MQNU	IBM MQ .NET unmanaged client

RemoteVersion (MQCFST)
The remote partner version. This parameter is the version of the IBM MQ code running at the remote end of the channel (parameter identifier: MQCACH_REMOTE_VERSION).

The remote version is displayed as **VVRRMMFF**, where

VV Version

RR Release

MM Maintenance level

FF Fix level

RemoteQMgrName (MQCFST)
Name of the remote queue manager, or queue-sharing group (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

ShortRetriesLeft (MQCFIN)
Number of short retry attempts remaining (parameter identifier: MQIACH_SHORT_RETRIES_LEFT).

SecurityProtocol (MQCFIN)
Security protocol currently in use (parameter identifier: MQIACH_SECURITY_PROTOCOL).

This parameter does not apply to client-connection channels.

The possible values are:

MQSECPROT_NONE
No security protocol

MQSECPROT_SSLV30
SSL version 3.0

MQSECPROT_TLSV10
TLS version 1.0

MQSECPROT_TLSV12
TLS version 1.2

This parameter is not available on z/OS.

SSLCertRemoteIssuerName (MQCFST)
The full Distinguished Name of the issuer of the remote certificate. The issuer is the certificate authority that issued the certificate (parameter identifier: MQCACH_SSL_CERT_ISSUER_NAME).

The maximum length of the string is MQ_SHORT_DNAME_LENGTH.

SSLCertUserId (MQCFST)
The local user ID associated with the remote certificate (parameter identifier: MQCACH_SSL_CERT_USER_ID).

This parameter is valid only on z/OS.

The maximum length of the string is MQ_USER_ID_LENGTH.

SSLKeyResetDate (MQCFST)
Date of the previous successful TLS secret key reset, in the form `yyyy-mm-dd` (parameter identifier: MQCACH_SSL_KEY_RESET_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

SSLKeyResets (MQCFIN)
TLS secret key resets (parameter identifier: MQIACH_SSL_KEY_RESETS).

The number of successful TLS secret key resets that have occurred for this channel instance since the channel started. If TLS secret key negotiation is enabled, the count is incremented whenever a secret key reset is performed.

SSLKeyResetTime (MQCFST)
Time of the previous successful TLS secret key reset, in the form `hh.mm.ss` (parameter identifier: MQCACH_SSL_KEY_RESET_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

SSLShortPeerName (MQCFST)
Distinguished Name of the peer queue manager or client at the other end of the channel (parameter identifier: MQCACH_SSL_SHORT_PEER_NAME).

The maximum length is MQ_SHORT_DNAME_LENGTH, so longer Distinguished Names are truncated.

StopRequested (MQCFIN)
Whether user stop request is outstanding (parameter identifier: MQIACH_STOP_REQUESTED).

The value can be any of the following values:

MQCHSR_STOP_NOT_REQUESTED
User stop request has not been received.

MQCHSR_STOP_REQUESTED
User stop request has been received.

SubState (MQCFIN)
Current action being performed by the channel (parameter identifier: MQIACH_CHANNEL_SUBSTATE).

The value can be any of the following values:

MQCHSSTATE_CHADEXIT
Running channel auto-definition exit.

MQCHSSTATE_COMPRESSING
Compressing or decompressing data.

MQCHSSTATE_END_OF_BATCH
End of batch processing.

MQCHSSTATE_HANDSHAKING
TLS handshaking.

MQCHSSTATE_HEARTBEATING
Heartbeating with partner.

MQCHSSTATE_IN_MQGET
Performing MQGET.

MQCHSSTATE_IN_MQI_CALL
Executing an IBM MQ API call, other than an MQPUT or MQGET.

MQCHSSTATE_IN_MQPUT
Performing MQPUT.

MQCHSSTATE_MREXIT
Running retry exit.

MQCHSSTATE_MSGEXIT
Running message exit.

MQCHSSTATE_NAME_SERVER
Name server request.

MQCHSSTATE_NET_CONNECTING
Network connect.

MQCHSSTATE_OTHER
Undefined state.

MQCHSSTATE_RCVEXIT
Running receive exit.

MQCHSSTATE_RECEIVING
Network receive.

MQCHSSTATE_RESYNCHING
Resynching with partner.

MQCHSSTATE_SCYEXIT
Running security exit.

MQCHSSTATE_SENDEXIT
Running send exit.

MQCHSSTATE_SENDING
Network send.

MQCHSSTATE_SERIALIZING
Serialized on queue manager access.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

XQTime (MQCFIL)
Transmission queue time indicator (parameter identifier: MQIACH_XMITQ_TIME_INDICATOR). The time, in microseconds, that messages remained on the transmission queue before being retrieved. The

time is measured from when the message is put onto the transmission queue until it is retrieved to be sent on the channel and, therefore, includes any interval caused by a delay in the putting application.

Two values are returned:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

Inquire Channel Status (Response) (MQTT): AIX Linux Windows

The response to the Inquire Channel Status (MQCMD_INQUIRE_CHANNEL_STATUS) command consists of the response header followed by the *ChannelName* structure and the requested combination of attribute parameter structures.

One PCF response message is generated for each channel instance found that matches the criteria that are specified on the command.

If the **ClientIdentifier** parameter is not specified, the output of the Inquire Channel Status command is a summary of statuses of all clients that are connected to the channel. One PCF response message is returned per channel.

Always returned:
ChannelName, *ChannelStatus*, *ChannelType*, *Connections*,

If the **ClientIdentifier** parameter is specified, separate PCF response messages are returned for each client connection. The **ClientIdentifier** parameter might be a wildcard, in which the status for all clients that match the **ClientIdentifier** string is returned (within the limits of**MaxResponses** and **ResponseRestartPoint** if they are set).

Always returned:
ChannelName, *ChannelStatus*, *ChannelType*, *ClientId*

Returned if requested:
ChannelStatusDate, *ChannelStatusTime*, *ClientUser*, *InDoubtInput*, *InDoubtOutput*, *KeepAliveInterval*, *LastMessageSentDate*, *LastMessageSentTime*, *MCAUser*, *MessagesReceived*, *MessagesSent*, *PendingOutbound*, *Protocol*

Response data

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelStartDate (MQCFST)
Date on which the channel started, in the form yyyy-mm-dd (parameter identifier: MQCACH_CHANNEL_START_DATE).

The maximum length of the string is MQ_CHANNEL_DATE_LENGTH.

ChannelStartTime (MQCFST)
Time at which the channel started, in the form hh.mm.ss (parameter identifier: MQCACH_CHANNEL_START_TIME).

The maximum length of the string is MQ_CHANNEL_TIME_LENGTH.

ChannelStatus (MQCFIN)
Channel status (parameter identifier: MQIACH_CHANNEL_STATUS).

The value can be:

MQCHS_DISCONNECTED
Channel is disconnected.

MQCHS_RUNNING
Channel is transferring or waiting for messages.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

The value must be:

MQCHT_MQTT
Telemetry.

ClientUser (MQCFST)
ClientID of the client (parameter identifier: MQCACH_CLIENT_USER_ID).

The maximum length of the string is MQ_CLIENT_USER_ID_LENGTH.

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

Connections (MQCFIN)
Current number of MQTT connections connected to this channel (parameter identifier: MQIACF_NAME_LENGTH).

InDoubtInput (MQCFIN)
The number of inbound messages to the client that are in doubt (parameter identifier: MQIACH_IN_DOUBT_IN).

InDoubtOutput (MQCFIN)
The number of outbound messages from the client that are in doubt (parameter identifier: MQIACH_IN_DOUBT_OUT).

KeepAliveInterval (MQCFIN)
KeepAlive interval (parameter identifier: MQIACH_KEEP_ALIVE_INTERVAL).

The interval in milliseconds after which the client is disconnected because of inactivity. If the MQXR service does not receive any communication from the client within the keep alive interval, it disconnects from the client. This interval is calculated based on the MQTT keep alive time sent by the client when it connects. The maximum size is MQ_MQTT_MAX_KEEP_ALIVE.

LastMsgDate (MQCFST)
Date on which the last message was sent, or the MQI call was handled, in the form `yyyy-mm-dd` (parameter identifier: MQCACH_LAST_MSG_DATE).

The maximum length of the string is MQ_CHANNEL_DATE_LENGTH.

LastMsgTime (MQCFST)
Time at which the last message was sent, or the MQI call was handled, in the form `hh.mm.ss` (parameter identifier: MQCACH_LAST_MSG_TIME).

The maximum length of the string is MQ_CHANNEL_TIME_LENGTH.

MCAUser (MQCFST)
Message channel agent user identifier (parameter identifier: MQCACH_MCA_USER_ID).

The maximum length of the MCA user identifier is MQ_MCA_USER_ID_LENGTH.

MsgsReceived (MQCFIN64)
Number of messages received by the client since it last connected (parameter identifier: MQIACH_MSGS_RECEIVED / MQIACH_MSGS_RCVD).

MsgsSent (MQCFIN64)
Number of messages sent by the client since it last connected (parameter identifier: MQIACH_MSGS_SENT).

PendingOutbound (MQCFIN)
The number of outbound messages pending (parameter identifier: MQIACH_PENDING_OUT).

Protocol (MQCFST)
MQTT protocol supported by this channel (parameter identifier: MQIACH_PROTOCOL).

Specify one or both of the following options. To specify more than one option, either add the values together (do not add the same constant more than once), or combine the values using the bitwise OR operation (if the programming language supports bit operations).

MQTTv3 (constant: MQPROTO_MQTTV3)

HTTP (constant: MQPROTO_HTTP)

MQTTv311 (constant: MQPROTO_MQTTV311)

Inquire Cluster Queue Manager:

The Inquire Cluster Queue Manager (MQCMD_INQUIRE_CLUSTER_Q_MGR) command inquires about the attributes of IBM MQ queue managers in a cluster.

Required parameters

ClusterQMgrName (MQCFST)
Queue manager name (parameter identifier: MQCA_CLUSTER_Q_MGR_NAME).

Generic queue manager names are supported. A generic name is a character string followed by an asterisk "*", for example ABC*. It selects all queue managers having names that start with the selected character string. An asterisk on its own matches all possible names.

The queue manager name is always returned, regardless of the attributes requested.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Optional parameters

Channel (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Specifies that eligible cluster queue managers are limited to those having the specified channel name.

Generic channel names are supported. A generic name is a character string followed by an asterisk "*", for example ABC*. It selects all queue managers having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

If you do not specify a value for this parameter, channel information about *all* queue managers in the cluster is returned.

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

Specifies that eligible cluster queue managers are limited to those having the specified cluster name.

Generic cluster names are supported. A generic name is a character string followed by an asterisk "*", for example ABC*. It selects all queue managers having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

If you do not specify a value for this parameter, cluster information about *all* queue managers inquired is returned.

ClusterQMgrAttrs (MQCFIL)

Attributes (parameter identifier: MQIACF_CLUSTER_Q_MGR_ATTRS).

Some parameters are relevant only for cluster channels of a particular type or types. Attributes that are not relevant for a particular type of channel cause no output, and do not cause an error. To check which attributes apply to which channel types; see Channel attributes and channel types.

The attribute list might specify the following value on its own. If the parameter is not specified, a default value is used.

MQIACF_ALL
All attributes.

Alternative, supply a combination of the following values:

MQCA_ALTERATION_DATE
The date on which the information was last altered.

MQCA_ALTERATION_TIME
The time at which the information was last altered.

MQCA_CLUSTER_DATE
The date on which the information became available to the local queue manager.

MQCA_CLUSTER_NAME
The name of the cluster to which the channel belongs.

MQCA_CLUSTER_Q_MGR_NAME
The name of the cluster to which the channel belongs.

MQCA_CLUSTER_TIME
The time at which the information became available to the local queue manager.

MQCA_Q_MGR_IDENTIFIER
The unique identifier of the queue manager.

MQCA_VERSION
The version of the IBM MQ installation that the cluster queue manager is associated with.

MQCA_XMIT_Q_NAME
The cluster transmission queue used by the queue manager.

MQCACH_CONNECTION_NAME
Connection name.

MQCACH_DESCRIPTION
Description.

MQCACH_LOCAL_ADDRESS
Local communications address for the channel.

MQCACH_MCA_NAME
Message channel agent name.

You cannot use MQCACH_MCA_NAME as a parameter to filter on.

MQCACH_MCA_USER_ID
MCA user identifier.

MQCACH_MODE_NAME
Mode name.

MQCACH_MR_EXIT_NAME
Message-retry exit name.

MQCACH_MR_EXIT_USER_DATA
Message-retry exit user data.

MQCACH_MSG_EXIT_NAME
Message exit name.

MQCACH_MSG_EXIT_USER_DATA
Message exit user data.

MQCACH_PASSWORD
Password.

This parameter is not valid on z/OS.

MQCACH_RCV_EXIT_NAME
Receive exit name.

MQCACH_RCV_EXIT_USER_DATA
Receive exit user data.

MQCACH_SEC_EXIT_NAME
Security exit name.

MQCACH_SEC_EXIT_USER_DATA
Security exit user data.

MQCACH_SEND_EXIT_NAME
Send exit name.

MQCACH_SEND_EXIT_USER_DATA
Send exit user data.

MQCACH_SSL_CIPHER_SPEC
TLS cipher spec.

MQIACH_SSL_CLIENT_AUTH
TLS client authentication.

MQCACH_SSL_PEER_NAME
TLS peer name.

MQCACH_TP_NAME
Transaction program name.

MQCACH_USER_ID
User identifier.

This parameter is not valid on z/OS.

MQIA_MONITORING_CHANNEL
Online monitoring data collection.

MQIA_USE_DEAD_LETTER_Q
Determines whether the dead-letter queue is used when messages cannot be delivered by channels.

MQIACF_Q_MGR_DEFINITION_TYPE
How the cluster queue manager was defined.

MQIACF_Q_MGR_TYPE
The function of the queue manager in the cluster.

MQIACF_SUSPEND
Specifies whether the queue manager is suspended from the cluster.

MQIACH_BATCH_HB
The value being used for the batch heartbeat.

MQIACH_BATCH_INTERVAL
Batch wait interval (seconds).

MQIACH_BATCH_DATA_LIMIT
Batch data limit (kilobytes).

MQIACH_BATCH_SIZE
Batch size.

MQIACH_CHANNEL_STATUS
Channel status.

MQIACH_CLWL_CHANNEL_PRIORITY
Cluster workload channel priority.

MQIACH_CLWL_CHANNEL_RANK
Cluster workload channel rank.

MQIACH_CLWL_CHANNEL_WEIGHT
Cluster workload channel weight.

MQIACH_DATA_CONVERSION
Specifies whether sender must convert application data.

MQIACH_DISC_INTERVAL
Disconnection interval.

MQIACH_HB_INTERVAL
Heartbeat interval (seconds).

MQIACH_HDR_COMPRESSION
The list of header data compression techniques supported by the channel.

MQIACH_KEEP_ALIVE_INTERVAL
KeepAlive interval (valid on z/OS only).

MQIACH_LONG_RETRY
Count of long duration attempts.

MQIACH_LONG_TIMER
Long duration timer.

MQIACH_MAX_MSG_LENGTH
Maximum message length.

MQIACH_MCA_TYPE
MCA type.

MQIACH_MR_COUNT
Count of send message attempts.

MQIACH_MR_INTERVAL
Interval between attempting to resend a message in milliseconds.

MQIACH_MSG_COMPRESSION
List of message data compression techniques supported by the channel.

MQIACH_NETWORK_PRIORITY
Network priority.

MQIACH_NPM_SPEED
Speed of nonpersistent messages.

MQIACH_PUT_AUTHORITY
Put authority.

MQIACH_SEQUENCE_NUMBER_WRAP
Sequence number wrap.

MQIACH_SHORT_RETRY
Count of short duration attempts.

MQIACH_SHORT_TIMER
Short duration timer.

MQIACH_XMIT_PROTOCOL_TYPE
Transmission protocol type.

z/OS

z/OS **CommandScope (MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is processed when the queue manager is a member of a queue-sharing group. You can specify one of the following values:

- Blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- A queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment. The command server must be enabled.
- An asterisk "*". The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *ClusterQMgrAttrs* except MQIACF_ALL and others as noted. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1601 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the **StringFilterCommand** parameter.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *ClusterQMgrAttrs* except MQCA_CLUSTER_Q_MGR_NAME and others as noted. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1608 for information about using this filter condition.

If you specify a string filter for *Channel* or *ClusterName*, you cannot also specify the *Channel* or *ClusterName* parameter.

If you specify a string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter.

Inquire Cluster Queue Manager (Response):

The response to the Inquire Cluster Queue Manager (MQCMD_INQUIRE_CLUSTER_Q_MGR) command consists of three parts. The response header is followed by the *QMgrName* structure and the requested combination of attribute parameter structures.

Always returned:
ChannelName, *ClusterName*, *QMgrName*,

Returned if requested:
AlterationDate, *AlterationTime*, *BatchHeartbeat*, *BatchInterval*, *BatchSize*, *ChannelDesc*, *ChannelMonitoring*, *ChannelStatus*, *ClusterDate*, *ClusterInfo*, *ClusterTime*, *CLWLChannelPriority*, *CLWLChannelRank*, *CLWLChannelWeight*, *ConnectionName*, *DataConversion*, *DiscInterval*, *HeaderCompression*, *HeartbeatInterval*, z/OS *KeepAliveInterval*, *LocalAddress*, *LongRetryCount*, *LongRetryInterval*, *MaxMsgLength*, *MCAName*, *MCAType*, *MCAUserIdentifier*, *MessageCompression*, *ModeName*, *MsgExit*, *MsgRetryCount*, *MsgRetryExit*, *MsgRetryInterval*, *MsgRetryUserData*, *MsgUserData*, *NetworkPriority*, *NonPersistentMsgSpeed*, *Password*, *PutAuthority*, *QMgrDefinitionType*, *QMgrIdentifier*, *QMgrType*, *ReceiveExit*, *ReceiveUserData*, *SecurityExit*, *SecurityUserData*, *SendExit*, *SendUserData*, *SeqNumberWrap*, *ShortRetryCount*, *ShortRetryInterval*, *SSLCipherSpec*, *SSLClientAuth*, *SSLPeerName*, *Suspend*, *TpName*, *TransmissionQName*, *TransportType*, *UseDLQ*, *UserIdentifier*, *Version*

Response data

AlterationDate (MQCFST)
Alteration date, in the form yyyy-mm-dd (parameter identifier: MQCA_ALTERATION_DATE).

The date at which the information was last altered.

AlterationTime (MQCFST)
Alteration time, in the form hh.mm.ss (parameter identifier: MQCA_ALTERATION_TIME).

The time at which the information was last altered.

BatchHeartbeat (MQCFIN)
The value being used for the batch heartbeat (parameter identifier: MQIACH_BATCH_HB).

The value can be 0 - 999,999. A value of 0 indicates that the batch heartbeat is not being used.

BatchInterval (MQCFIN)
Batch interval (parameter identifier: MQIACH_BATCH_INTERVAL).

BatchSize (MQCFIN)
Batch size (parameter identifier: MQIACH_BATCH_SIZE).

ChannelDesc (MQCFST)
Channel description (parameter identifier: MQCACH_DESC).

The maximum length of the string is MQ_CHANNEL_DESC_LENGTH.

ChannelMonitoring (MQCFIN)
Online monitoring data collection (parameter identifier: MQIA_MONITORING_CHANNEL).

The value can be:

MQMON_OFF
Online monitoring data collection is turned off for this channel.

MQMON_Q_MGR
The value of the queue manager's **ChannelMonitoring** parameter is inherited by the channel. MQMON_Q_MGR is the default value.

MQMON_LOW
Online monitoring data collection is turned on, with a low rate of data collection, for this channel unless the queue manager's **ChannelMonitoring** parameter is MQMON_NONE.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate rate of data collection, for this channel unless the queue manager's **ChannelMonitoring** parameter is MQMON_NONE.

MQMON_HIGH
Online monitoring data collection is turned on, with a high rate of data collection, for this channel unless the queue manager's **ChannelMonitoring** parameter is MQMON_NONE.

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelStatus (MQCFIN)
Channel status (parameter identifier: MQIACH_CHANNEL_STATUS).

The value can be:

MQCHS_BINDING
Channel is negotiating with the partner.

MQCHS_INACTIVE
Channel is not active.

MQCHS_STARTING
Channel is waiting to become active.

MQCHS_RUNNING
Channel is transferring or waiting for messages.

MQCHS_PAUSED
Channel is paused.

MQCHS_STOPPING
Channel is in process of stopping.

MQCHS_RETRYING
Channel is reattempting to establish connection.

MQCHS_STOPPED
Channel is stopped.

MQCHS_REQUESTING
Requester channel is requesting connection.

MQCHS_INITIALIZING
Channel is initializing.

This parameter is returned if the channel is a cluster-sender channel (CLUSSDR) only.

ClusterDate (MQCFST)
Cluster date, in the form yyyy-mm-dd (parameter identifier: MQCA_CLUSTER_DATE).

The date at which the information became available to the local queue manager.

ClusterInfo (MQCFIN)
Cluster information (parameter identifier: MQIACF_CLUSTER_INFO).

The cluster information available to the local queue manager.

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

ClusterTime (MQCFST)
Cluster time, in the form hh.mm.ss (parameter identifier: MQCA_CLUSTER_TIME).

The time at which the information became available to the local queue manager.

CLWLChannelPriority (MQCFIN)
Channel priority (parameter identifier: MQIACH_CLWL_CHANNEL_PRIORITY).

CLWLChannelRank (MQCFIN)
Channel rank (parameter identifier: MQIACH_CLWL_CHANNEL_RANK).

CLWLChannelWeight (MQCFIN)
Channel weighting (parameter identifier: MQIACH_CLWL_CHANNEL_WEIGHT).

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH. On z/OS, it is MQ_LOCAL_ADDRESS_LENGTH.

DataConversion (MQCFIN)
Specifies whether sender must convert application data (parameter identifier: MQIACH_DATA_CONVERSION).

The value can be:

MQCDC_NO_SENDER_CONVERSION
No conversion by sender.

MQCDC_SENDER_CONVERSION
Conversion by sender.

DiscInterval (MQCFIN)
Disconnection interval (parameter identifier: MQIACH_DISC_INTERVAL).

HeaderCompression (MQCFIL)
Header data compression techniques supported by the channel (parameter identifier: MQIACH_HDR_COMPRESSION). The values specified are in order of preference.

The value can be one, or more, of

MQCOMPRESS_NONE
No header data compression is performed.

MQCOMPRESS_SYSTEM
Header data compression is performed.

HeartbeatInterval (MQCFIN)
Heartbeat interval (parameter identifier: MQIACH_HB_INTERVAL).

z/OS

KeepAliveInterval (MQCFIN)
KeepAlive interval (parameter identifier: MQIACH_KEEP_ALIVE_INTERVAL). This parameter applies to z/OS only.

LocalAddress (MQCFST)
Local communications address for the channel (parameter identifier: MQCACH_LOCAL_ADDRESS).

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

LongRetryCount (MQCFIN)
Long retry count (parameter identifier: MQIACH_LONG_RETRY).

LongRetryInterval (MQCFIN)
Long timer (parameter identifier: MQIACH_LONG_TIMER).

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIACH_MAX_MSG_LENGTH).

MCAName (MQCFST)
Message channel agent name (parameter identifier: MQCACH_MCA_NAME).

The maximum length of the string is MQ_MCA_NAME_LENGTH.

MCAType (MQCFIN)
Message channel agent type (parameter identifier: MQIACH_MCA_TYPE).

The value can be:

MQMCAT_PROCESS
Process.

MQMCAT_THREAD
Thread (Windows only).

MCAUserIdentifier (MQCFST)
Message channel agent user identifier (parameter identifier: MQCACH_MCA_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH.

MessageCompression (MQCFIL)
Message data compression techniques supported by the channel (parameter identifier: MQIACH_MSG_COMPRESSION). The values specified are in order of preference.

The value can be one, or more, of:

MQCOMPRESS_NONE
No message data compression is performed.

MQCOMPRESS_RLE
Message data compression is performed using run-length encoding.

MQCOMPRESS_ZLIBFAST
Message data compression is performed using ZLIB encoding with speed prioritized.

MQCOMPRESS_ZLIBHIGH
Message data compression is performed using ZLIB encoding with compression prioritized.

ModeName (MQCFST)
Mode name (parameter identifier: MQCACH_MODE_NAME).

The maximum length of the string is MQ_MODE_NAME_LENGTH.

MsgExit (MQCFST)
Message exit name (parameter identifier: MQCACH_MSG_EXIT_NAME).

The maximum length of the string is MQ_EXIT_NAME_LENGTH.

Multi On Multiplatforms, more than one message exit can be defined for a channel. If more than one message exit is defined, the list of names is returned in an MQCFSL structure instead of an MQCFST structure.

z/OS On z/OS, an MQCFSL structure is always used.

MsgRetryCount (MQCFIN)
Message retry count (parameter identifier: MQIACH_MR_COUNT).

MsgRetryExit (MQCFST)
Message retry exit name (parameter identifier: MQCACH_MR_EXIT_NAME).

The maximum length of the string is MQ_EXIT_NAME_LENGTH.

MsgRetryInterval (MQCFIN)
Message retry interval (parameter identifier: MQIACH_MR_INTERVAL).

MsgRetryUserData (MQCFST)
Message retry exit user data (parameter identifier: MQCACH_MR_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

MsgUserData (MQCFST)
Message exit user data (parameter identifier: MQCACH_MSG_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

Multi On Multiplatforms, more than one message exit user data string can be defined for a channel. If more than one string is defined, the list of strings is returned in an MQCFSL structure instead of an MQCFST structure.

z/OS On z/OS, an MQCFSL structure is always used.

NetworkPriority (MQCFIN)
Network priority (parameter identifier: MQIACH_NETWORK_PRIORITY).

NonPersistentMsgSpeed (MQCFIN)
Speed at which non-persistent messages are to be sent (parameter identifier: MQIACH_NPM_SPEED).

The value can be:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

Password (MQCFST)
Password (parameter identifier: MQCACH_PASSWORD). This parameter is not available on z/OS.

If a nonblank password is defined, it is returned as asterisks. Otherwise, it is returned as blanks.

The maximum length of the string is MQ_PASSWORD_LENGTH. However, only the first 10 characters are used.

PutAuthority (MQCFIN)
Put authority (parameter identifier: MQIACH_PUT_AUTHORITY).

The value can be:

MQPA_DEFAULT
Default user identifier is used.

MQPA_CONTEXT
Context user identifier is used.

MQPA_ALTERNATE_OR_MCA
The user identifier from the *UserIdentifier* field of the message descriptor is used. Any user ID received from the network is not used. This value is valid only on z/OS.

MQPA_ONLY_MCA
The default user identifier is used. Any user ID received from the network is not used. This value is valid only on z/OS.

QMgrDefinitionType (MQCFIN)
Queue manager definition type (parameter identifier: MQIACF_Q_MGR_DEFINITION_TYPE).

The value can be:

MQQMDT_EXPLICIT_CLUSTER_SENDER
A cluster-sender channel from an explicit definition.

MQQMDT_AUTO_CLUSTER_SENDER
A cluster-sender channel by auto-definition.

MQQMDT_CLUSTER_RECEIVER
A cluster-receiver channel.

MQQMDT_AUTO_EXP_CLUSTER_SENDER
A cluster-sender channel, both from an explicit definition and by auto-definition.

QMgrIdentifier (MQCFST)
Queue manager identifier (parameter identifier: MQCA_Q_MGR_IDENTIFIER).

The unique identifier of the queue manager.

QMgrName (MQCFST)
Queue manager name (parameter identifier: MQCA_CLUSTER_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QMgrType (MQCFIN)
Queue manager type (parameter identifier: MQIACF_Q_MGR_TYPE).

The value can be:

MQQMT_NORMAL
A normal queue manager.

MQQMT_REPOSITORY
A repository queue manager.

ReceiveExit (MQCFST)
Receive exit name (parameter identifier: MQCACH_RCV_EXIT_NAME).

The maximum length of the string is MQ_EXIT_NAME_LENGTH.

Multi On Multiplatforms, more than one receive exit can be defined for a channel. If more than one receive exit is defined, the list of names is returned in an MQCFSL structure instead of an MQCFST structure.

z/OS On z/OS, an MQCFSL structure is always used.

ReceiveUserData (MQCFST)
Receive exit user data (parameter identifier: MQCACH_RCV_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

Multi On Multiplatforms, more than one receive exit user data string can be defined for a channel. If more than one string is defined, the list of strings is returned in an MQCFSL structure instead of an MQCFST structure.

z/OS On z/OS, an MQCFSL structure is always used.

SecurityExit (MQCFST)
Security exit name (parameter identifier: MQCACH_SEC_EXIT_NAME).

The maximum length of the string is MQ_EXIT_NAME_LENGTH.

SecurityUserData (MQCFST)
Security exit user data (parameter identifier: MQCACH_SEC_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

SendExit (MQCFST)
Send exit name (parameter identifier: MQCACH_SEND_EXIT_NAME).

The maximum length of the string is MQ_EXIT_NAME_LENGTH.

Multi On Multiplatforms, more than one send exit can be defined for a channel. If more than one send exit is defined, the list of names is returned in an MQCFSL structure instead of an MQCFST structure.

z/OS On z/OS, an MQCFSL structure is always used.

SendUserData (MQCFST)
Send exit user data (parameter identifier: MQCACH_SEND_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

Multi On Multiplatforms, more than one send exit user data string can be defined for a channel. If more than one string is defined, the list of names is returned in an MQCFSL structure instead of an MQCFST structure.

z/OS On z/OS, an MQCFSL structure is always used.

SeqNumberWrap (MQCFIN)
Sequence wrap number (parameter identifier: MQIACH_SEQUENCE_NUMBER_WRAP).

ShortRetryCount (MQCFIN)
Short retry count (parameter identifier: MQIACH_SHORT_RETRY).

ShortRetryInterval (MQCFIN)
Short timer (parameter identifier: MQIACH_SHORT_TIMER).

SSLCipherSpec (MQCFST)
CipherSpec (parameter identifier: MQCACH_SSL_CIPHER_SPEC).

The length of the string is MQ_SSL_CIPHER_SPEC_LENGTH.

SSLClientAuth (MQCFIN)
Client authentication (parameter identifier: MQIACH_SSL_CLIENT_AUTH).

The value can be:

MQSCA_REQUIRED
Client authentication required

MQSCA_OPTIONAL
Client authentication is optional.

Defines whether IBM MQ requires a certificate from the TLS client.

SSLPeerName (MQCFST)
Peer name (parameter identifier: MQCACH_SSL_PEER_NAME).

The length of the string is MQ_SSL_PEER_NAME_LENGTH. On z/OS, it is MQ_SHORT_PEER_NAME_LENGTH.

Specifies the filter to use to compare with the distinguished name of the certificate from the peer queue manager or client at the other end of the channel. (A distinguished name is the identifier of the TLS certificate.) If the distinguished name in the certificate received from the peer does not match the SSLPEER filter, the channel does not start.

Suspend (MQCFIN)
Specifies whether the queue manager is suspended (parameter identifier: MQIACF_SUSPEND).

The value can be:

MQSUS_NO
The queue manager is not suspended from the cluster.

MQSUS_YES
The queue manager is suspended from the cluster.

TpName (MQCFST)
Transaction program name (parameter identifier: MQCACH_TP_NAME).

The maximum length of the string is MQ_TP_NAME_LENGTH.

TranmissionQName (MQCFST)
Transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME). The cluster transmission queue used by the queue manager.

The maximum length of the string is MQ_Q_NAME_LENGTH.

TransportType (MQCFIN)
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value can be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

MQXPT_NETBIOS
NetBIOS.

MQXPT_SPX
SPX.

MQXPT_DECNET
DECnet.

UseDLQ (MQCFIN)
Determines whether the dead-letter queue is used when publication messages cannot be delivered to their correct subscriber queue (parameter identifier: MQIA_USE_DEAD_LETTER_Q).

UserIdentifier (MQCFST)
Task user identifier (parameter identifier: MQCACH_USER_ID). This parameter is not available on z/OS.

The maximum length of the string is MQ_USER_ID_LENGTH. However, only the first 10 characters are used.

Version (MQCFST)
The version of the IBM MQ installation that the cluster queue manager is associated with. (parameter identifier: MQCA_VERSION). The version has the format VVRRMMFF:

VV: Version

RR: Release

MM: Maintenance level

FF: Fix level

Inquire Communication Information Object on Multiplatforms: Multi

The Inquire Communication Information Object (MQCMD_INQUIRE_COMM_INFO) command inquires about the attributes of existing IBM MQ communication information objects.

Required parameters:
ComminfoName

Optional parameters:
ComminfoAttrs, **IntegerFilterCommand**, **StringFilterCommand**

Required parameters

ComminfoName (MQCFST)
The name of the communication information definition about which information is to be returned (parameter identifier: MQCA_COMM_INFO_NAME).

The communication information name is always returned regardless of the attributes requested.

The maximum length of the string is MQ_COMM_INFO_NAME_LENGTH.

Optional parameters

ComminfoAttrs (MQCFIL)
Comminfo attributes (parameter identifier: MQIACF_COMM_INFO_ATTRS).

The attribute list might specify the following value on its own - default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQIA_CODED_CHAR_SET_ID
CCSID for transmitted messages.

MQIA_COMM_EVENT
Comminfo event control.

MQIA_MCAST_BRIDGE
Multicast bridging.

MQIA_MONITOR_INTERVAL
Frequency of update for monitoring information.

MQIACF_ENCODING
Encoding for transmitted messages.

MQIACH_MC_HB_INTERVAL
Multicast heartbeat interval.

MQIACH_MSG_HISTORY
Amount of message history being kept.

MQIACH_MULTICAST_PROPERTIES
Multicast properties control.

MQIACH_NEW_SUBSCRIBER_HISTORY
New subscriber history.

MQIACH_PORT
Port Number.

MQCA_ALTERATION_DATE
The date on which the information was last altered.

MQCA_ALTERATION_TIME
The time at which the information was last altered.

MQCA_COMM_INFO_DESC
Comminfo description.

MQCA_COMM_INFO_TYPE
Comminfo type

MQCACH_GROUP_ADDRESS
Group Address.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *CominfoAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1601 for information about using this filter condition.

If you specify an integer filter for *CominfoType* (MQIA_COMM_INFO_TYPE), you cannot also specify the **CominfoType** parameter.

If you specify an integer filter, you cannot also specify a string filter using the **StringFilterCommand** parameter.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *CominfoAttrs* except MQCA_COMM_INFO_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1608 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter.

Inquire Communication Information Object (Response) on Multiplatforms: Multi

The response to the Inquire Communication Information Object (MQCMD_INQUIRE_COMM_INFO) command consists of the response header followed by the CominfoName structure, and the requested combination of attribute parameter structures (where applicable).

If a generic communication information name was specified, one such message is generated for each object found.

Always returned:
CominfoName

Returned if requested:
AlterationDate, *AlterationTime*, *Bridge*, *CCSID*, *CommEvent*, *Description*, *Encoding*, *GrpAddress*, *MonitorInterval*, *MulticastHeartbeat*, *MulticastPropControl*, *MsgHistory*, *NewSubHistory*, *PortNumber*, *Type*

Response data

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered, in the form `yyyy-mm-dd`.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered, in the form `hh.mm.ss`.

Bridge (MQCFIN)
Multicast Bridging (parameter identifier: MQIA_MCAST_BRIDGE).

Controls whether publications from applications not using Multicast are bridged to applications using multicast.

CCSID (MQCFIN)
CCSID that messages are trasmitted in (parameter identifier: MQIA_CODED_CHAR_SET_ID).

The coded character set identifier that messages are transmitted in.

CommEvent (MQCFIN)
Event Control (parameter identifier: MQIA_COMM_EVENT).

Controls whether event messages are generated for multicast handles that are created using this COMMINFO object. The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

MQEVR_EXCEPTION
Reporting of events for message reliability below the reliability threshold enabled.

CominfoName (MQCFST)
The name of the communication information definition (parameter identifier: MQCA_COMM_INFO_NAME).

The maximum length of the string is MQ_COMM_INFO_NAME_LENGTH.

Description (MQCFST)
Description of the communication information definition (parameter identifier: MQCA_COMM_INFO_DESC).

The maximum length of the string is MQ_COMM_INFO_DESC_LENGTH.

Encoding (MQCFIN)
Encoding that messages are transmitted in (parameter identifier: MQIACF_ENCODING).

The encoding that messages are transmitted in. The value can be any of the following values:

MQENC_AS_PUBLISHED
Encoding taken from published message.

MQENC_NORMAL

MQENC_REVERSED

MQENC_S390

MQENC_TNS

GrpAddress (MQCFST)
The group IP address or DNS name (parameter identifier: MQCACH_GROUP_ADDRESS).

The maximum length of the string is MQ_GROUP_ADDRESS_LENGTH.

MonitorInterval (MQCFIN)
Frequency of monitoring (parameter identifier: MQIA_MONITOR_INTERVAL).

How frequently, in seconds, monitoring information is updated and event messages are generated.

MulticastHeartbeat (MQCFIN)
Heartbeat Interval for multicast (parameter identifier: MQIACH_MC_HB_INTERVAL).

The heartbeat interval, in milliseconds, for multicast transmitters.

MulticastPropControl (MQCFIN)
Multicast property control (parameter identifier: MQIACH_MULTICAST_PROPERTIES).

Control which MQMD properties and user properties flow with the message. The value can be any of the following values:

MQMCP_ALL
All MQMD and user properties.

MQMAP_REPLY
Properties related to replying to messages.

MQMAP_USER
Only user properties.

MQMAP_NONE
No MQMD or user properties.

MQMAP_COMPAT
Properties are transmitted in a format compatible with previous Multicast clients.

MsgHistory (MQCFIN)
Message History (parameter identifier: MQIACH_MSG_HISTORY).

The amount of message history, in kilobytes, that is kept by the system to handle retransmissions in the case of NACKS.

NewSubHistory (MQCFIN)
New Subscriber History (parameter identifier: MQIACH_NEW_SUBSCRIBER_HISTORY).

Controls how much historical data a new subscriber receives. The value can be any of the following values:

MQNSH_NONE
Only publications from the time of the subscription are sent.

MQNSH_ALL
As much history as is known is retransmitted.

PortNumber (MQCFIN)
Port Number (parameter identifier: MQIACH_PORT).

The port number to transmit on.

Type (MQCFIN)
The type of the communications information definition (parameter identifier: MQIA_COMM_INFO_TYPE).

The value can be:

MQCIT_MULTICAST
Multicast.

Inquire Connection:

The Inquire connection (MQCMD_INQUIRE_CONNECTION) command inquires about the applications which are connected to the queue manager, the status of any transactions that those applications are running, and the objects which the application has open.

Required parameters

ConnectionId (MQCFBS)
Connection identifier (parameter identifier: MQBACF_CONNECTION_ID).

This parameter is the unique connection identifier associated with an application that is connected to the queue manager. Specify either this parameter **or** *GenericConnectionId*.

All connections are assigned a unique identifier by the queue manager regardless of how the connection is established.

If you need to specify a generic connection identifier, use the **GenericConnectionId** parameter instead.

The length of the string is MQ_CONNECTION_ID_LENGTH.

GenericConnectionId (MQCFBS)
Generic specification of a connection identifier (parameter identifier: MQBACF_GENERIC_CONNECTION_ID).

Specify either this parameter **or** *ConnectionId*.

If you specify a byte string of zero length, or one which contains only null bytes, information about all connection identifiers is returned. This value is the only value permitted for *GenericConnectionId*.

The length of the string is MQ_CONNECTION_ID_LENGTH.

Optional parameters

ByteStringFilterCommand (MQCFBF)
Byte string filter command descriptor. The parameter identifier must be MQBACF_EXTERNAL_UOW_ID, MQBACF_ORIGIN_UOW_ID, or MQBACF_Q_MGR_UOW_ID. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFBF - PCF byte string filter parameter" on page 1596 for information about using this filter condition.

If you specify a byte string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter, or a string filter using the **StringFilterCommand** parameter.

z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_Q_MGR_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

ConnectionAttrs (MQCFIL)
Connection attributes (parameter identifier: MQIACF_CONNECTION_ATTRS).

The attribute list can specify the following value on its own - default value if the parameter is not specified:

MQIACF_ALL
All attributes of the selected *ConnInfoType*.

or, if you select a value of MQIACF_CONN_INFO_CONN for *ConnInfoType*, a combination of the following:

MQBACF_CONNECTION_ID
Connection identifier.

MQBACF_EXTERNAL_UOW_ID
External unit of recovery identifier associated with the connection.

MQBACF_ORIGIN_UOW_ID
Unit of recovery identifier assigned by the originator (valid on z/OS only).

MQBACF_Q_MGR_UOW_ID
Unit of recovery identifier assigned by the queue manager.

MQCACF_APPL_TAG
Name of an application that is connected to the queue manager.

MQCACF_ASID
The 4-character address-space identifier of the application identified in MQCACF_APPL_TAG (valid on z/OS only).

MQCACF_ORIGIN_NAME
Originator of the unit of recovery (valid on z/OS only).

MQCACF_PSB_NAME
The 8-character name of the program specification block (PSB) associated with the running IMS transaction (valid on z/OS only).

MQCACF_PST_ID
The 4-character IMS program specification table (PST) region identifier for the connected IMS region (valid on z/OS only).

MQCACF_TASK_NUMBER
A 7-digit CICS task number (valid on z/OS only).

MQCACF_TRANSACTION_ID
A 4-character CICS transaction identifier (valid on z/OS only).

MQCACF_UOW_LOG_EXTENT_NAME
Name of the first extent required to recover the transaction.
MQCACF_UOW_LOG_EXTENT_NAME is not valid on z/OS.

MQCACF_UOW_LOG_START_DATE
Date on which the transaction associated with the current connection first wrote to the log.

MQCACF_UOW_LOG_START_TIME
Time at which the transaction associated with the current connection first wrote to the log.

MQCACF_UOW_START_DATE
Date on which the transaction associated with the current connection was started.

MQCACF_UOW_START_TIME
Time at which the transaction associated with the current connection was started.

MQCACF_USER_IDENTIFIER
User identifier of the application that is connected to the queue manager.

MQCACH_CHANNEL_NAME
Name of the channel associated with the connected application.

MQCACH_CONNECTION_NAME
Connection name of the channel associated with the application.

MQIA_APPL_TYPE
Type of the application that is connected to the queue manager.

MQIACF_CONNECT_OPTIONS
Connect options currently in force for this application connection.

You cannot use the value MQCNO_STANDARD_BINDING as a filter value.

MQIACF_PROCESS_ID
Process identifier of the application that is currently connected to the queue manager.

This parameter is not valid on z/OS.

MQIACF_THREAD_ID
Thread identifier of the application that is currently connected to the queue manager.

This parameter is not valid on z/OS.

MQIACF_UOW_STATE
State of the unit of work.

MQIACF_UOW_TYPE
Type of external unit of recovery identifier as understood by the queue manager.

or, if you select a value of MQIACF_CONN_INFO_HANDLE for *ConnInfoType*, a combination of the following:

MQCACF_OBJECT_NAME
Name of each object that the connection has open.

MQCACH_CONNECTION_NAME
Connection name of the channel associated with the application.

MQIA_QSG_DISP
Disposition of the object (valid on z/OS only).

You cannot use MQIA_QSG_DISP as a parameter to filter on.

MQIA_READ_AHEAD
The read ahead connection status.

MQIA_UR_DISP
The unit of recovery disposition associated with the connection (valid on z/OS only).

MQIACF_HANDLE_STATE
Whether an API call is in progress.

MQIACF_OBJECT_TYPE
Type of each object that the connection has open.

MQIACF_OPEN_OPTIONS
Options used by the connection to open each object.

or, if you select a value of MQIACF_CONN_INFO_ALL for *ConnInfoType*, any of the previous values.

ConnInfoType (MQCFIN)
Type of connection information to be returned (parameter identifier: MQIACF_CONN_INFO_TYPE).

The value can be any of the following values:

MQIACF_CONN_INFO_CONN
Connection information. On z/OS, MQIACF_CONN_INFO_CONN includes threads which might be logically or actually disassociated from a connection, together with those threads that are in-doubt and for which external intervention is needed to resolve them. MQIACF_CONN_INFO_CONN is the default value used if the parameter is not specified.

MQIACF_CONN_INFO_HANDLE
Information pertaining only to those objects opened by the specified connection.

MQIACF_CONN_INFO_ALL
Connection information and information about those objects that the connection has open.

You cannot use *ConnInfoType* as a parameter to filter on.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *ConnectionAttrs* except as noted and MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. You cannot use the value MQCNO_STANDARD_BINDING on the MQIACF_CONNECT_OPTIONS parameter with either the MQCFOP_CONTAINS or MQCFOP_EXCLUDES operator. See "MQCFIF - PCF integer filter parameter" on page 1601 for information about using this filter condition.

If you filter on MQIACF_CONNECT_OPTIONS or MQIACF_OPEN_OPTIONS, in each case the filter value must have only 1 bit set.

If you specify an integer filter, you cannot also specify a byte string filter using the **ByteStringFilterCommand** parameter or a string filter using the **StringFilterCommand** parameter.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *ConnectionAttrs*. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1608 for information about using this filter condition.

If you specify a string filter, you cannot also specify a byte string filter using the **ByteStringFilterCommand** parameter or an integer filter using the **IntegerFilterCommand** parameter.

URDisposition (MQCFIN)
The unit of recovery disposition associated with the connection (parameter identifier: MQI_UR_DISP). This parameter is valid only on z/OS.

The value can be any of the following values:

MQQSGD_ALL
Specifies that all connections must be returned.

MQQSGD_GROUP
Specifies that only connections with a GROUP unit of recovery disposition must be returned.

MQQSGD_Q_MGR
Specifies that only connections with a QMGR unit of recovery disposition must be returned.

Error code

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CONNECTION_ID_ERROR
Connection identifier not valid.

Inquire Connection (Response):

The response to the Inquire Connection (MQCMD_INQUIRE_CONNECTION) command consists of the response header followed by the *ConnectionId* structure and a set of attribute parameter structures determined by the value of *ConnInfoType* in the Inquire command.

If the value of *ConnInfoType* was MQIACF_CONN_INFO_ALL, there is one message for each connection found with MQIACF_CONN_INFO_CONN, and *n* more messages per connection with MQIACF_CONN_INFO_HANDLE (where *n* is the number of objects that the connection has open).

Always returned:
ConnectionId, *ConnInfoType*

Always returned if *ConnInfoType* is MQIACF_CONN_INFO_HANDLE:
ObjectName, *ObjectType*, z/OS *QSGDisposition*

Returned if requested and *ConnInfoType* is MQIACF_CONN_INFO_CONN:
ApplDesc, *ApplTag*, *ApplType*, z/OS *ASID*, *AsynchronousState*, *ChannelName*, *ConnectionName*, *ConnectionOptions*, z/OS *OriginName*, z/OS *OriginUOWId*, z/OS *ProcessId*, *PSBName*, z/OS *PSTId*, *QMgrUOWId*, *StartUOWLogExtent*, *TaskNumber*, *ThreadId*, z/OS *TransactionId*, *UOWIdentifier*, *UOWLogStartDate*, *UOWLogStartTime*, *UOWStartDate*, *UOWStartTime*, *UOWState*, *UOWType*, z/OS *URDisposition*, *UserId*

Returned if requested and *ConnInfoType* is MQIACF_CONN_INFO_HANDLE:
AsynchronousState, *Destination*, *DestinationQueueManager*, *HandleState*, *OpenOptions*, *ReadAhead*, *SubscriptionID*, *SubscriptionName*, *TopicString*

Response data

ApplDesc (MQCFST)
Application description (parameter identifier: MQCACF_APPL_DESC).

The maximum length is MQ_APPL_DESC_LENGTH.

ApplTag (MQCFST)
Application tag (parameter identifier: MQCACF_APPL_TAG).

The maximum length is MQ_APPL_TAG_LENGTH.

ApplType (MQCFIN)
Application type (parameter identifier: MQIA_APPL_TYPE).

The value can be any of the following values:

MQAT_QMGR
Queue manager process.

MQAT_CHANNEL_INITIATOR
Channel initiator.

MQAT_USER
User application.

MQAT_BATCH
Application using a batch connection (only on z/OS).

MQAT_RRS_BATCH
RRS-coordinated application using a batch connection (only on z/OS).

MQAT_CICS
CICS transaction (only on z/OS).

MQAT_IMS
IMS transaction (only on z/OS).

MQAT_SYSTEM_EXTENSION
Application performing an extension of function that is provided by the queue manager.

z/OS

ASID (MQCFST)
Address space identifier (parameter identifier: MQCACF_ASID).

The four character address-space identifier of the application identified by *ApplTag*. It distinguishes duplicate values of *ApplTag*.

This parameter is valid only on z/OS.

The length of the string is MQ_ASID_LENGTH.

AsynchronousState (MQCFIN)
The state of asynchronous consumption on this handle (parameter identifier: MQIACF_ASYNC_STATE).

The value can be:

MQAS_NONE
If *ConnInfoType* is MQIACF_CONN_INFO_CONN, an MQCTL call has not been issued against the handle. Asynchronous message consumption cannot currently proceed on this connection. If *ConnInfoType* is MQIACF_CONN_INFO_HANDLE, an MQCB call has not been issued against this handle, so no asynchronous message consumption is configured on this handle.

MQAS_SUSPENDED
The asynchronous consumption callback has been suspended so that asynchronous message consumption cannot currently proceed on this handle. This situation can be either because an MQCB or MQCTL call with *Operation* MQOP_SUSPEND has been issued against this object handle by the application, or because it has been suspended by the system. If it has been suspended by the system, as part of the process of suspending asynchronous message consumption the callback function is called with the reason code that describes the problem resulting in suspension. This reason code is reported in the *Reason* field in the MQCBC structure passed to the callback. In order for asynchronous message consumption to proceed, the application must issue an MQCB or MQCTL call with *Operation* MQOP_RESUME. This reason code can be returned if *ConnInfoType* is MQIACF_CONN_INFO_CONN or MQIACF_CONN_INFO_HANDLE.

MQAS_SUSPENDED_TEMPORARY
The asynchronous consumption callback has been temporarily suspended by the system so that asynchronous message consumption cannot currently proceed on this object handle. As part of the process of suspending asynchronous message consumption, the callback function is called with the reason code that describes the problem resulting in suspension. MQAS_SUSPENDED_TEMPORARY is reported in the *Reason* field in the MQCBC structure passed to the callback. The callback function is called again when asynchronous message consumption is resumed by the system when the temporary condition has been resolved. MQAS_SUSPENDED_TEMPORARY is returned only if ConnInfoType is MQIACF_CONN_INFO_HANDLE.

MQAS_STARTED
An MQCTL call with *Operation* MQOP_START has been issued against the connection handle so that asynchronous message consumption can proceed on this connection. MQAS_STARTED is returned only if *ConnInfoType* is MQIACF_CONN_INFO_CONN.

MQAS_START_WAIT
An MQCTL call with *Operation* MQOP_START_WAIT has been issued against the connection

handle so that asynchronous message consumption can proceed on this connection. MQAS_START_WAIT is returned only if *ConnInfoType* is MQIACF_CONN_INFO_CONN.

MQAS_STOPPED
An MQCTL call with *Operation* MQOP_STOP has been issued against the connection handle so that asynchronous message consumption cannot currently proceed on this connection. MQAS_STOPPED is returned only if *ConnInfoType* is MQIACF_CONN_INFO_CONN.

MQAS_ACTIVE
An MQCB call has set up a function to call back to process messages asynchronously and the connection handle has been started so that asynchronous message consumption can proceed. MQAS_ACTIVE is returned only if *ConnInfoType* is MQIACF_CONN_INFO_HANDLE.

MQAS_INACTIVE
An MQCB call has set up a function to call back to process messages asynchronously but the connection handle has not yet been started, or has been stopped or suspended, so that asynchronous message consumption cannot currently proceed. MQAS_INACTIVE is returned only if ConnInfoType is MQIACF_CONN_INFO_HANDLE.

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ConnectionId (MQCFBS)
Connection identifier (parameter identifier: MQBACF_CONNECTION_ID).

The length of the string is MQ_CONNECTION_ID_LENGTH.

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

ConnectionOptions (MQCFIL)
Connect options currently in force for the connection (parameter identifier: MQIACF_CONNECT_OPTIONS).

ConnInfoType (MQCFIN)
Type of information returned (parameter identifier: MQIACF_CONN_INFO_TYPE).

The value can be any of the following values:

MQIACF_CONN_INFO_CONN
Generic information for the specified connection.

MQIACF_CONN_INFO_HANDLE
Information pertinent only to those objects opened by the specified connection.

Destination (MQCFST)
The destination queue for messages published to this subscription (parameter identifier MQCACF_DESTINATION).

This parameter is relevant only for handles of subscriptions to topics.

DestinationQueueManager (MQCFST)
The destination queue manager for messages published to this subscription (parameter identifier MQCACF_DESTINATION_Q_MGR).

This parameter is relevant only for handles of subscriptions to topics. If *Destination* is a queue hosted on the local queue manager, this parameter contains the local queue manager name. If *Destination* is a queue hosted on a remote queue manager, this parameter contains the name of the remote queue manager.

HandleState (MQCFIN)
State of the handle (parameter identifier: MQIACF_HANDLE_STATE).

The value can be any of the following values:

MQHSTATE_ACTIVE
An API call from this connection is currently in progress for this object. If the object is a queue, this condition can arise when an MQGET WAIT call is in progress.

If there is an MQGET SIGNAL outstanding, then this situation does not mean, by itself, that the handle is active.

MQHSTATE_INACTIVE
No API call from this connection is currently in progress for this object. If the object is a queue, this condition can arise when no MQGET WAIT call is in progress.

ObjectName (MQCFST)
Object name (parameter identifier: MQCACF_OBJECT_NAME).

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

ObjectType (MQCFIN)
Object type (parameter identifier: MQIACF_OBJECT_TYPE).

If this parameter is a handle of a subscription to a topic, the SUBID parameter identifies the subscription and can be used with the Inquire Subscription command to find all the details about the subscription.

The value can be any of the following values:

MQOT_Q
Queue.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process.

MQOT_Q_MGR
Queue manager.

MQOT_CHANNEL
Channel.

MQOT_AUTH_INFO
Authentication information object.

MQOT_TOPIC
Topic.

OpenOptions (MQCFIN)
Open options currently in force for the object for connection (parameter identifier: MQIACF_OPEN_OPTIONS).

This parameter is not relevant for a subscription. Use the SUBID field of the DISPLAY SUB command to find all the details about the subscription.

z/OS

OriginName (MQCFST)
Origin name (parameter identifier: MQCACF_ORIGIN_NAME).

Identifies the originator of the unit of recovery, except where *ApplType* is MQAT_RRS_BATCH when it is omitted.

This parameter is valid only on z/OS.

The length of the string is MQ_ORIGIN_NAME_LENGTH.

z/OS

OriginUOWId (MQCFBS)
Origin UOW identifier (parameter identifier: MQBACF_ORIGIN_UOW_ID).

The unit of recovery identifier assigned by the originator. It is an 8-byte value.

This parameter is valid only on z/OS.

The length of the string is MQ_UOW_ID_LENGTH.

z/OS

ProcessId (MQCFIN)
Process identifier (parameter identifier: MQIACF_PROCESS_ID).

PSBName (MQCFST)
Program specification block name (parameter identifier: MQCACF_PSB_NAME).

The 8-character name of the program specification block (PSB) associated with the running IMS transaction.

This parameter is valid only on z/OS.

The length of the string is MQ_PSB_NAME_LENGTH.

z/OS

PSTId (MQCFST)
Program specification table identifier (parameter identifier: MQCACF_PST_ID).

The 4-character IMS program specification table (PST) region identifier for the connected IMS region.

This parameter is valid only on z/OS.

The length of the string is MQ_PST_ID_LENGTH.

QMgrUOWId (MQCFBS)
Unit of recovery identifier assigned by the queue manager (parameter identifier: MQBACF_Q_MGR_UOW_ID).

z/OS On z/OS platforms, this parameter is returned as an 8-byte RBA.

Multi On Multiplatforms, this parameter is an 8-byte transaction identifier.

The maximum length of the string is MQ_UOW_ID_LENGTH.

z/OS

QSGDispositon (MQCFIN)
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object (that is, where it is defined and how it behaves). This parameter is valid only on z/OS. The value can be any of the following values:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED.

ReadAhead (MQCFIN)
The read ahead connection status (parameter identifier: MQIA_READ_AHEAD).

The value can be any of the following values:

MQREADA_NO
Read ahead for browsing messages, or of non-persistent messages is not enabled for the object that the connection has open.

MQREADA_YES
Read ahead for browsing messages, or of non-persistent messages is enabled for the object that the connection has open and is being used efficiently.

MQREADA_BACKLOG
Read ahead for browsing messages, or of non-persistent messages is enabled for this object. Read ahead is not being used efficiently because the client has been sent many messages which are not being consumed.

MQREADA_INHIBITED
Read ahead was requested by the application but has been inhibited because of incompatible options specified on the first MQGET call.

StartUOWLogExtent (MQCFST)
Name of the first extent needed to recover the transaction (parameter identifier: MQCACF_UOW_LOG_EXTENT_NAME).

The 8-character name of the program specification block (PSB) associated with the running IMS transaction.

This parameter is not valid on z/OS.

The maximum length of the string is MQ_LOG_EXTENT_NAME_LENGTH.

SubscriptionID (MQCFBS)
The internal, all time unique identifier of the subscription (parameter identifier MQBACF_SUB_ID).

This parameter is relevant only for handles of subscriptions to topics.

Not all subscriptions can be seen using Inquire Connection; only those subscriptions that have current handles open to the subscriptions can be seen. Use the Inquire Subscription command to see all subscriptions.

SubscriptionName (MQCFST)
The unique subscription name of the application associated with the handle (parameter identifier MQCACF_SUB_NAME).

This parameter is relevant only for handles of subscriptions to topics. Not all subscriptions have a subscription name.

ThreadId (MQCFIN)
Thread identifier (parameter identifier: MQIACF_THREAD_ID).

TopicString (MQCFST)
Resolved topic string (parameter identifier: MQCA_TOPIC_STRING).

This parameter is relevant for handles with an ObjectType of MQOT_TOPIC. For any other object type, this parameter is blank.

z/OS

TransactionId (MQCFST)
Transaction identifier (parameter identifier: MQCACF_TRANSACTION_ID).

The 4-character CICS transaction identifier.

This parameter is valid only on z/OS.

The maximum length of the string is MQ_TRANSACTION_ID_LENGTH.

UOWIdentifier (MQCFBS)
External unit of recovery identifier associated with the connection (parameter identifier: MQBACF_EXTERNAL UOW_ID).

This parameter is the recovery identifier for the unit of recovery. The value of *UOWType* determines its format.

The maximum length of the byte string is MQ_UOW_ID_LENGTH.

UOWLogStartDate (MQCFST)
Logged unit of work start date, in the form yyyy-mm-dd (parameter identifier: MQCACF_UOW_LOG_START_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

UOWLogStartTime (MQCFST)
Logged unit of work start time, in the form hh.mm.ss (parameter identifier: MQCACF_UOW_LOG_START_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

UOWStartDate (MQCFST)
Unit of work creation date (parameter identifier: MQCACF_UOW_START_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

UOWStartTime (MQCFST)
Unit of work creation time (parameter identifier: MQCACF_UOW_START_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

UOWState (MQCFIN)
State of the unit of work (parameter identifier: MQIACF_UOW_STATE).

The value can be any of the following values:

MQUOWST_NONE
There is no unit of work.

MQUOWST_ACTIVE
The unit of work is active.

MQUOWST_PREPARED
The unit of work is in the process of being committed.

MQUOWST_UNRESOLVED
The unit of work is in the second phase of a two-phase commit operation. IBM MQ holds resources on behalf of the unit of work and external intervention is required to resolve it. It might be as simple as starting the recovery coordinator (such as CICS, IMS, or RRS) or it might involve a more complex operation such as using the RESOLVE INDOUBT command. This value can occur only on z/OS.

UOWType (MQCFIN)
Type of external unit of recovery identifier as perceived by the queue manager (parameter identifier: MQIACF_UOW_TYPE).

The value can be any of the following values:

MQUOWT_Q_MGR

MQUOWT_CICS

MQUOWT_RRS

MQUOWT_IMS

MQUOWT_XA

z/OS

URDisposition (MQCFIN)
The unit of recovery disposition associated with the connection.

This parameter is valid only on z/OS.

The value can be:

MQQSGD_GROUP
This connection has a GROUP unit of recovery disposition.

MQQSGD_Q_MGR
This connection has a QMGR unit of recovery disposition.

UserId (MQCFST)
User identifier (parameter identifier: MQCACF_USER_IDENTIFIER).

The maximum length of the string is MQ_MAX_USER_ID_LENGTH.

Inquire Entity Authority on Multiplatforms: Multi

The Inquire Entity Authority (MQCMD_INQUIRE_ENTITY_AUTH) command inquires about authorizations of an entity to a specified object.

Required parameters

EntityName (MQCFST)
Entity name (parameter identifier: MQCACF_ENTITY_NAME).

Depending on the value of *EntityType*, this parameter is either:

- A principal name. This name is the name of a user for whom to retrieve authorizations to the specified object. On IBM MQ for Windows, the name of the principal can optionally include a domain name, specified in this format: `user@domain`.
- A group name. This name is the name of the user group on which to make the inquiry. You can specify one name only and this name must be the name of an existing user group.

  Windows For IBM MQ for Windows only, the group name can optionally include a domain name, specified in the following formats:

  ```
  GroupName@domain
  domain\GroupName
  ```

The maximum length of the string is MQ_ENTITY_NAME_LENGTH.

EntityType (MQCFIN)
Entity type (parameter identifier: MQIACF_ENTITY_TYPE).

The value can be:

MQZAET_GROUP
The value of the **EntityName** parameter refers to a group name.

MQZAET_PRINCIPAL
The value of the **EntityName** parameter refers to a principal name.

ObjectType (MQCFIN)
The type of object referred to by the profile (parameter identifier: MQIACF_OBJECT_TYPE).

The value can be any of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel object.

MQOT_CLNTCONN_CHANNEL
Client-connection channel object.

MQOT_COMM_INFO
Communication information object

MQOT_LISTENER
Listener object.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process.

MQOT_Q
Queue, or queues, that match the object name parameter.

MQOT_Q_MGR
Queue manager.

MQOT_REMOTE_Q_MGR_NAME
Remote queue manager.

MQOT_SERVICE
Service object.

MQOT_TOPIC
Topic object.

Options (MQCFIN)
Options to control the set of authority records that is returned (parameter identifier: MQIACF_AUTH_OPTIONS).

This parameter is required and you must set it to the value MQAUTHOPT_CUMULATIVE. It returns a set of authorities representing the cumulative authority that an entity has to a specified object.

If a user ID is a member of more than one group, this command displays the combined authorizations of all groups.

Optional parameters

ObjectName (MQCFST)
Object name (parameter identifier: MQCACF_OBJECT_NAME).

The name of the queue manager, queue, process definition, or generic profile on which to make the inquiry.

You must include a parameter if the *ObjectType* is not MQOT_Q_MGR. If you do not include this parameter, it is assumed that you are making an inquiry on the queue manager.

You cannot specify a generic object name although you can specify the name of a generic profile.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

ProfileAttrs (MQCFIL)
Profile attributes (parameter identifier: MQIACF_AUTH_PROFILE_ATTRS).

The attribute list might specify the following value on its own - default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCACF_ENTITY_NAME
Entity name.

MQIACF_AUTHORIZATION_LIST
Authorization list.

MQIACF_ENTITY_TYPE
Entity type.

MQIACF_OBJECT_TYPE
Object type.

ServiceComponent (MQCFST)
Service component (parameter identifier: MQCACF_SERVICE_COMPONENT).

If installable authorization services are supported, this parameter specifies the name of the authorization service to which the authorizations apply.

If you omit this parameter, the authorization inquiry is made to the first installable component for the service.

The maximum length of the string is MQ_SERVICE_COMPONENT_LENGTH.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRC_UNKNOWN_ENTITY
User ID not authorized, or unknown.

MQRCCF_OBJECT_TYPE_MISSING
Object type missing.

Inquire Entity Authority (Response) on Multiplatforms: 

Each response to the Inquire Entity Authority (MQCMD_INQUIRE_AUTH_RECS) command consists of the response header followed by the *QMgrName*, *Options*, and *ObjectName* structures and the requested combination of attribute parameter structures.

Always returned:
ObjectName, *Options*, *QMgrName*

Returned if requested:
AuthorizationList, *EntityName*, *EntityType*, *ObjectType*

Response data

AuthorizationList (MQCFIL)
Authorization list(parameter identifier: MQIACF_AUTHORIZATION_LIST).

This list can contain zero or more authorization values. Each returned authorization value means that any user ID in the specified group or principal has the authority to perform the operation defined by that value. The value can be any of the following values:

MQAUTH_NONE
The entity has authority set to 'none'.

MQAUTH_ALT_USER_AUTHORITY
Specify an alternate user ID on an MQI call.

MQAUTH_BROWSE
Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.

MQAUTH_CHANGE
Change the attributes of the specified object, using the appropriate command set.

MQAUTH_CLEAR
Clear a queue.

MQAUTH_CONNECT
Connect the application to the specified queue manager by issuing an MQCONN call.

MQAUTH_CREATE
Create objects of the specified type using the appropriate command set.

MQAUTH_DELETE
Delete the specified object using the appropriate command set.

MQAUTH_DISPLAY
Display the attributes of the specified object using the appropriate command set.

MQAUTH_INPUT
Retrieve a message from a queue by issuing an MQGET call.

MQAUTH_INQUIRE
Make an inquiry on a specific queue by issuing an MQINQ call.

MQAUTH_OUTPUT
Put a message on a specific queue by issuing an MQPUT call.

MQAUTH_PASS_ALL_CONTEXT
Pass all context.

MQAUTH_PASS_IDENTITY_CONTEXT
Pass the identity context.

MQAUTH_SET
Set attributes on a queue from the MQI by issuing an MQSET call.

MQAUTH_SET_ALL_CONTEXT
Set all context on a queue.

MQAUTH_SET_IDENTITY_CONTEXT
Set the identity context on a queue.

MQAUTH_CONTROL
For listeners and services, start and stop the specified channel, listener, or service.

For channels, start, stop, and ping the specified channel.

For topics, define, alter, or delete subscriptions.

MQAUTH_CONTROL_EXTENDED
Reset or resolve the specified channel.

MQAUTH_PUBLISH
Publish to the specified topic.

MQAUTH_SUBSCRIBE
Subscribe to the specified topic.

MQAUTH_RESUME
Resume a subscription to the specified topic.

MQAUTH_SYSTEM
Use queue manager for internal system operations.

MQAUTH_ALL
Use all operations applicable to the object.

MQAUTH_ALL_ADMIN
Use all administration operations applicable to the object.

MQAUTH_ALL_MQI
Use all MQI calls applicable to the object.

Use the *Count* field in the MQCFIL structure to determine how many values are returned.

EntityName (MQCFST)
Entity name (parameter identifier: MQCACF_ENTITY_NAME).

This parameter can either be a principal name or a group name.

The maximum length of the string is MQ_ENTITY_NAME_LENGTH.

EntityType (MQCFIN)
Entity type (parameter identifier: MQIACF_ENTITY_TYPE).

The value can be:

MQZAET_GROUP
The value of the **EntityName** parameter refers to a group name.

MQZAET_PRINCIPAL
The value of the **EntityName** parameter refers to a principal name.

MQZAET_UNKNOWN
On Windows, an authority record still exists from a previous queue manager which did not originally contain entity type information.

ObjectName (MQCFST)
Object name (parameter identifier: MQCACF_OBJECT_NAME).

The name of the queue manager, queue, process definition, or generic profile on which the inquiry is made.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

ObjectType (MQCFIN)
Object type (parameter identifier: MQIACF_OBJECT_TYPE).

The value can be:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel object.

MQOT_CLNTCONN_CHANNEL
Client-connection channel object.

MQOT_COMM_INFO
Communication information object

MQOT_LISTENER
Listener object.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process.

MQOT_Q
Queue, or queues, that match the object name parameter.

MQOT_Q_MGR
Queue manager.

MQOT_REMOTE_Q_MGR_NAME
Remote queue manager.

MQOT_SERVICE
Service object.

QMgrName (MQCFST)
Name of the queue manager on which the Inquire command is issued (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Inquire Group on z/OS: z/OS

The Inquire Group (MQCMD_INQUIRE_QSG) command inquires about the queue-sharing group to which the queue manager is connected.

Note: This command is supported only on z/OS when the queue manager is a member of a queue-sharing group.

Optional parameters

ObsoleteDB2Msgs (MQCFIN)
Whether to look for obsolete Db2 messages (parameter identifier: MQIACF_OBSOLETE_MSGS).

The value can be any of the following values:

MQOM_NO
Obsolete messages in Db2 are not looked for. MQOM_NO is the default value used if the parameter is not specified.

MQOM_YES
Obsolete messages in Db2 are looked for and messages containing information about any found are returned.

Inquire Group (Response) on z/OS: z/OS

The response to the Inquire Group (MQCMD_INQUIRE_QSG) command consists of the response header followed by the *QMgrName* structure and a number of other parameter structures. One such message is generated for each queue manager in the queue-sharing group.

If there are any obsolete Db2 messages, and that information is requested, one message, identified by a value of MQCMDI_DB2_OBSOLETE_MSGS in the **CommandInformation** parameter, is returned for each such message.

Always returned for the queue manager:
CommandLevel, *DB2ConnectStatus*, *DB2Name*, *QmgrCPF*, *QMgrName*, *QmgrNumber*, *QMgrStatus*, *QSGName*

Always returned for obsolete Db2 messages:
CommandInformation, *CFMsgIdentifier*

Response data relating to the queue manager

CommandLevel (MQCFIN)
Command level supported by the queue manager (parameter identifier: MQIA_COMMAND_LEVEL). The value can be any of the following values:

MQCMDL_LEVEL_520
Level 520 of system control commands.

MQCMDL_LEVEL_530
Level 530 of system control commands.

MQCMDL_LEVEL_531
Level 531 of system control commands.

MQCMDL_LEVEL_600
Level 600 of system control commands.

MQCMDL_LEVEL_700
Level 700 of system control commands.

MQCMDL_LEVEL_701
Level 701 of system control commands.

MQCMDL_LEVEL_750
Level 750 of system control commands.

MQCMDL_LEVEL_800
Level 800 of system control commands.

MQCMDL_LEVEL_801
Level 801 of system control commands.

Attention: MQCMDL_LEVEL_801 applies only to UNIX, when you install IBM MQ Version 8.0.0, Fix Pack 2.

MQCMDL_LEVEL_802
Level 802 of system control commands.

MQCMDL_LEVEL_900
Level 900 of system control commands.

MQCMDL_LEVEL_901
Level 901 of system control commands.

MQCMDL_LEVEL_902
Level 902 of system control commands.

MQCMDL_LEVEL_903
Level 903 of system control commands.

MQCMDL_LEVEL_904
Level 904 of system control commands.

DB2ConnectStatus (MQCFIN)
The current status of the connection to Db2 (parameter identifier: MQIACF_DB2_CONN_STATUS).

The current status of the queue manager. The value can be any of the following values:

MQQSGS_ACTIVE
The queue manager is running and is connected to Db2.

MQQSGS_INACTIVE
The queue manager is not running and is not connected to Db2.

MQQSGS_FAILED
The queue manager is running but not connected because Db2 has terminated abnormally.

MQQSGS_PENDING
The queue manager is running but not connected because Db2 has terminated normally.

MQQSGS_UNKNOWN
The status cannot be determined.

DB2Name (MQCFST)
The name of the Db2 subsystem or group to which the queue manager is to connect (parameter identifier: MQCACF_DB2_NAME).

The maximum length is MQ_Q_MGR_CPF_LENGTH.

QMgrCPF (MQCFST)
The command prefix of the queue manager (parameter identifier: MQCA_Q_MGR_CPF).

The maximum length is MQ_Q_MGR_CPF_LENGTH.

QMgrName (MQCFST)
Name of the queue manager (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length is MQ_Q_MGR_NAME_LENGTH.

QmgrNumber (MQCFIN)
The number, generated internally, of the queue manager in the group.(parameter identifier: MQIACF_Q_MGR_NUMBER).

QMgrStatus (MQCFIN)
Recovery (parameter identifier: MQIACF_Q_MGR_STATUS).

The current status of the queue manager. The value can be any of the following values:

MQQSGS_ACTIVE
The queue manager is running.

MQQSGS_INACTIVE
The queue manager is not running, having terminated normally.

MQQSGS_FAILED
The queue manager is not running, having terminated abnormally.

MQQSGS_CREATED
The queue manager has been defined to the group, but has not yet been started.

MQQSGS_UNKNOWN
The status cannot be determined.

QSGName (MQCFST)
The name of the queue-sharing group (parameter identifier: MQCA_QSG_NAME).

The maximum length is MQ_QSG_NAME_LENGTH.

Response data relating to obsolete Db2 messages

CFMsgIdentifier (MQCFBS)
CF list entry identifier (parameter identifier: MQBACF_CF_LEID).

The maximum length is MQ_CF_LEID_LENGTH.

CommandInformation (MQCFIN)
Command information (parameter identifier: MQIACF_COMMAND_INFO). This indicates whether queue managers in the group contain obsolete messages. The value is MQCMDI_DB2_OBSOLETE_MSGS.

Inquire Log on z/OS: z/OS

The Inquire Log (MQCMD_INQUIRE_LOG) command returns log system parameters and information.

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is processed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- a queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Inquire Log (Response) on z/OS: z/OS

The response to the Inquire Log (MQCMD_INQUIRE_LOG) command consists of the response header followed by the *ParameterType* structure and the combination of attribute parameter structures determined by the value of *ParameterType*.

Always returned:
ParameterType. Specifies the type of archive information being returned. The value can be any of the following values:

MQSYSP_TYPE_INITIAL
The initial settings of the log parameters.

MQSYSP_TYPE_SET
The settings of the log parameters if they have been altered since their initial setting.

MQSYSP_TYPE_LOG_COPY
Information relating to the active log copy.

MQSYSP_TYPE_LOG_STATUS
Information relating to the status of the logs.

Returned if *ParameterType* **is MQSYSP_TYPE_INITIAL (one message is returned):**
DeallocateInterval, *DualArchive*, *DualActive*, *DualBSDS*, *InputBufferSize*, *LogArchive*, *LogCompression*, *MaxArchiveLog*, *MaxConcurrentOffloads*, *MaxReadTapeUnits*, *OutputBufferCount*, *OutputBufferSize*, V 9.0.0 *ZHyperWrite*

Returned if *ParameterType* **is MQSYSP_TYPE_SET and any value is set (one message is returned):**
DeallocateInterval, *DualArchive*, *DualActive*, *DualBSDS*, *InputBufferSize*, *LogArchive*, *MaxArchiveLog*, *MaxConcurrentOffloads*, *MaxReadTapeUnits*, *OutputBufferCount*, *OutputBufferSize*

Returned if *ParameterType* **is MQSYSP_TYPE_LOG_COPY (one message is returned for each log copy):** *DataSetName*, *LogCopyNumber*, *LogUsed*, V 9.0.0 *ZHyperWrite*

Returned if *ParameterType* **is MQSYSP_TYPE_LOG_STATUS (one message is returned):**
FullLogs, *LogCompression*, *LogRBA*, *LogSuspend*, *OffloadStatus*, *QMgrStartDate*, *QMgrStartRBA*, *QMgrStartTime*, *TotalLogs*

Response data - log parameter information

DeallocateInterval (MQCFIN)
Deallocation interval (parameter identifier: MQIACF_SYSP_DEALLOC_INTERVAL).

Specifies the length of time, in minutes, that an allocated archive read tape unit is allowed to remain unused before it is deallocated. The value can be in the range zero through 1440. If it is zero, the tape unit is deallocated immediately. If it is 1440, the tape unit is never deallocated.

DualActive (MQCFIN)
Specifies whether dual logging is being used (parameter identifier: MQIACF_SYSP_DUAL_ACTIVE).

The value can be any of the following values:

MQSYSP_YES
Dual logging is being used.

MQSYSP_NO
Dual logging is not being used.

DualArchive (MQCFIN)
Specifies whether dual archive logging is being used (parameter identifier: MQIACF_SYSP_DUAL_ARCHIVE).

The value can be any of the following values:

MQSYSP_YES
Dual archive logging is being used.

MQSYSP_NO
Dual archive logging is not being used.

DualBSDS (MQCFIN)
Specifies whether dual BSDS is being used (parameter identifier: MQIACF_SYSP_DUAL_BSDS).

The value can be any of the following values:

MQSYSP_YES
Dual BSDS is being used.

MQSYSP_NO
Dual BSDS is not being used.

InputBufferSize (MQCFIN)
Specifies the size of input buffer storage for active and archive log data sets (parameter identifier: MQIACF_SYSP_IN_BUFFER_SIZE).

LogArchive (MQCFIN)
Specifies whether archiving is on or off (parameter identifier: MQIACF_SYSP_ARCHIVE).

The value can be any of the following values:

MQSYSP_YES
Archiving is on.

MQSYSP_NO
Archiving is off.

LogCompression (MQCFIN)
Specifies which log compression parameter is used (parameter identifier: MQIACF_LOG_COMPRESSION).

The value can be any of the following values:

MQCOMPRESS_NONE
No log compression is performed.

MQCOMPRESS_RLE
Run-length encoding compression is performed.

MQCOMPRESS_ANY
Enable the queue manager to select the compression algorithm that gives the greatest degree of log record compression. Using this option currently results in RLE compression.

MaxArchiveLog (MQCFIN)
Specifies the maximum number of archive log volumes that can be recorded in the BSDS (parameter identifier: MQIACF_SYSP_MAX_ARCHIVE).

MaxConcurrentOffloads (MQCFIN)
Specifies the maximum number of concurrent log offload tasks (parameter identifier: MQIACF_SYSP_MAX_CONC_OFFLOADS).

OutputBufferCount (MQCFIN)
Specifies the number of output buffers to be filled before they are written to the active log data sets (parameter identifier: MQIACF_SYSP_OUT_BUFFER_COUNT).

OutputBufferSize (MQCFIN)
Specifies the size of output buffer storage for active and archive log data sets (parameter identifier: MQIACF_SYSP_OUT_BUFFER_SIZE).

ZHyperWrite (MQCFIN)
Specifies whether the zHyperWrite feature is enabled (parameter identifier: MQIACF_SYSP_ZHYPERWRITE).

The value can be any of the following values:

MQSYSP_YES
zHyperWrite is enabled.

MQSYSP_NO
zHyperWrite is not enabled.

Response data - to log status information

DataSetName (MQCFST)
The data set name of the active log data set (parameter identifier: MQCACF_DATA_SET_NAME).

If the copy is not currently active, this parameter is returned as blank.

The maximum length of the string is MQ_DATA_DATA_SET_NAME_LENGTH.

FullLogs (MQCFIN)
The total number of full active log data sets that have not yet been archived (parameter identifier: MQIACF_SYSP_FULL_LOGS).

LogCompression (MQCFIN)
Specifies the current log compression option (parameter identifier: MQIACF_LOG_COMPRESSION).

The value can be any of the following values:

MQCOMPRESS_NONE
Log compression is not enabled.

MQCOMPRESS_RLE
Run-length encoding log compression is enabled.

MQCOMPRESS_ANY
Any compression algorithm supported by the queue manager is enabled.

LogCopyNumber (MQCFIN)
Copy number (parameter identifier: MQIACF_SYSP_LOG_COPY).

LogRBA (MQCFST)
The RBA of the most recently written log record (parameter identifier: MQCACF_SYSP_LOG_RBA).

The maximum length of the string is MQ_RBA_LENGTH.

LogSuspend (MQCFIN)
Specifies whether logging is suspended (parameter identifier: MQIACF_SYSP_LOG_SUSPEND).

The value can be any of the following values:

MQSYSP_YES
Logging is suspended.

MQSYSP_NO
Logging is not suspended.

LogUsed (MQCFIN)
The percentage of the active log data set that has been used (parameter identifier: MQIACF_SYSP_LOG_USED).

OffloadStatus (MQCFIN)
Specifies the status of the offload task (parameter identifier: MQIACF_SYSP_OFFLOAD_STATUS).

The value can be any of the following values:

MQSYSP_STATUS_ALLOCATING_ARCHIVE
The offload task is busy, allocating the archive data set. MQSYSP_STATUS_ALLOCATING_ARCHIVE could indicate that a tape mount request is pending.

MQSYSP_STATUS_COPYING_BSDS
The offload task is busy, copying the BSDS data set.

MQSYSP_STATUS_COPYING_LOG
The offload task is busy, copying the active log data set.

MQSYSP_STATUS_BUSY
The offload task is busy with other processing.

MQSYSP_STATUS_AVAILABLE
The offload task is waiting for work.

QMgrStartDate (MQCFST)
The date on which the queue manager was started, in the form `yyyy-mm-dd` (parameter identifier: MQCACF_SYSP_Q_MGR_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

QMgrStartRBA (MQCFST)
The RBA from which logging began when the queue manager was started (parameter identifier: MQCACF_SYSP_Q_MGR_RBA).

The maximum length of the string is MQ_RBA_LENGTH.

QMgrStartTime (MQCFST)
The time that the queue manager was started, in the form `hh.mm.ss` (parameter identifier: MQCACF_SYSP_Q_MGR_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

TotalLogs (MQCFIN)
The total number of active log data sets (parameter identifier: MQIACF_SYSP_TOTAL_LOGS).

V 9.0.0

ZHyperWrite (MQCFIN)
Specifies whether the zHyperWrite feature is enabled (parameter identifier: MQIACF_SYSP_ZHYPERWRITE).

The value can be any of the following values:

MQSYSP_YES
zHyperWrite is enabled.

MQSYSP_NO
zHyperWrite is not enabled.

Inquire Namelist:

The Inquire Namelist (MQCMD_INQUIRE_NAMELIST) command inquires about the attributes of existing IBM MQ namelists.

Required parameters:
NamelistName

Optional parameters:
z/OS *CommandScope*, *IntegerFilterCommand*, *NamelistAttrs*, z/OS *QSGDisposition*, *StringFilterCommand*

Required parameters

NamelistName (MQCFST)
Namelist name (parameter identifier: MQCA_NAMELIST_NAME).

This parameter is the name of the namelist with attributes that are required. Generic namelist names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all namelists having names that start with the selected character string. An asterisk on its own matches all possible names.

The namelist name is always returned regardless of the attributes requested.

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

Optional parameters z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is processed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- a queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *NamelistAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1601 for information about using this filter condition.

If you specify an integer filter for *NamelistType* (MQIA_NAMELIST_TYPE), you cannot also specify the **NamelistType** parameter.

If you specify an integer filter, you cannot also specify a string filter using the **StringFilterCommand** parameter.

NamelistAttrs (MQCFIL)
Namelist attributes (parameter identifier: MQIACF_NAMELIST_ATTRS).

The attribute list might specify the following value on its own - default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_NAMELIST_NAME
Name of namelist object.

MQCA_NAMELIST_DESC
Namelist description.

MQCA_NAMES
Names in the namelist.

MQCA_ALTERATION_DATE
The date on which the information was last altered.

MQCA_ALTERATION_TIME
The time at which the information was last altered.

MQIA_NAME_COUNT
Number of names in the namelist.

MQIA_NAMELIST_TYPE
Namelist type (valid only on z/OS)

NamelistType (MQCFIN)
Namelist attributes (parameter identifier: MQIA_NAMELIST_TYPE). This parameter applies to z/OS only.

Specifies the type of names in the namelist. The value can be any of the following values:

MQNT_NONE
The names are of no particular type.

MQNT_Q
A namelist that holds a list of queue names.

MQNT_CLUSTER
A namelist that is associated with clustering, containing a list of the cluster names.

MQNT_AUTH_INFO
The namelist is associated with TLS, and contains a list of authentication information object names.

z/OS

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined as either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

You cannot use *QSGDisposition* as a parameter to filter on.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *NamelistAttrs* except MQCA_NAMELIST_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See “MQCFSF - PCF string filter parameter” on page 1608 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter.

Inquire Namelist (Response):

The response to the Inquire Namelist (MQCMD_INQUIRE_NAMELIST) command consists of the response header followed by the *NamelistName* structure and the requested combination of attribute parameter structures.

If a generic namelist name was specified, one such message is generated for each namelist found.

Always returned:

NamelistName, z/OS *QSGDisposition*

Returned if requested:

AlterationDate, *AlterationTime*, *NameCount*, *NamelistDesc*, z/OS *NamelistType*, *Names*

Response data

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered, in the form yyyy-mm-dd.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered, in the form hh.mm.ss.

NameCount (MQCFIN)
Number of names in the namelist (parameter identifier: MQIA_NAME_COUNT).

The number of names contained in the namelist.

NamelistDesc (MQCFST)
Description of namelist definition (parameter identifier: MQCA_NAMELIST_DESC).

The maximum length of the string is MQ_NAMELIST_DESC_LENGTH.

NamelistName (MQCFST)
The name of the namelist definition (parameter identifier: MQCA_NAMELIST_NAME).

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

z/OS

NamelistType (MQCFIN)
Type of names in the namelist (parameter identifier: MQIA_NAMELIST_TYPE). This parameter applies to z/OS only.

Specifies the type of names in the namelist. The value can be any of the following values:

MQNT_NONE
The names are of no particular type.

MQNT_Q
A namelist that holds a list of queue names.

MQNT_CLUSTER
A namelist that is associated with clustering, containing a list of the cluster names.

MQNT_AUTH_INFO
The namelist is associated with TLS, and contains a list of authentication information object names.

Names (MQCFSL)
A list of the names contained in the namelist (parameter identifier: MQCA_NAMES).

The number of names in the list is given by the *Count* field in the MQCFSL structure. The length of each name is given by the *StringLength* field in that structure. The maximum length of a name is MQ_OBJECT_NAME_LENGTH.

z/OS

QSGDisposition (MQCFIN)
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object (that is, where it is defined and how it behaves). This parameter applies only to z/OS. The value can be any of the following values:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

Inquire Namelist Names:

The Inquire Namelist Names (MQCMD_INQUIRE_NAMELIST_NAMES) command inquires for a list of namelist names that match the generic namelist name specified.

Required parameters

NamelistName (MQCFST)
Name of namelist (parameter identifier: MQCA_NAMELIST_NAME).

Generic namelist names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

z/OS

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- a queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being processed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined with either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

Inquire Namelist Names (Response):

The response to the Inquire Namelist Names (MQCMD_INQUIRE_NAMELIST_NAMES) command consists of the response header followed by a single parameter structure giving zero or more names that match the specified namelist name.

z/OS
Additionally, on z/OS only, the *QSGDispositions* structure (with the same number of entries as the *NamelistNames* structure) is returned. Each entry in this structure indicates the disposition of the object with the corresponding entry in the *NamelistNames* structure.

Always returned:
NamelistNames, z/OS *QSGDispositions*

Returned if requested:
None

Response data

NamelistNames (MQCFSL)
List of namelist names (parameter identifier: MQCACF_NAMELIST_NAMES).

z/OS

QSGDispositions (MQCFIL)
List of queue-sharing group dispositions (parameter identifier: MQIACF_QSG_DISPS). This parameter is valid only on z/OS. Possible values for fields in this structure are:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

Inquire Policy on Multiplatforms: Multi

The Inquire Policy (MQCMD_INQUIRE_PROT_POLICY) command inquires about the policy, or policies, set on a queue.

Required parameters

generic-policy-name (MQCFST)
Policy name (parameter identifier: MQCA_POLICY_NAME).

This parameter is the name of the policy with attributes that are required. Generic policy names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all policies having names that start with the selected character string. An asterisk on its own matches all possible names.

The policy name is always returned regardless of the attributes requested.

The name of the policy, or policies (or part of the policy name or names) to inquire are the same as the name of the queue, or queues, that the policies control.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Optional parameters

PolicyAttrs (MQCFIL)
Policy attributes (parameter identifier: MQIACF_POLICY_ATTRS).

The attribute list might specify the following value on its own- default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_POLICY_NAME
Name of the policy.

MQIA_SIGNATURE_ALGORITHM
The digital signature algorithm.

MQIA_ENCRYPTION_ALGORITHM
The encryption algorithm.

MQCA_SIGNER_DN
The distinguished name of an authorized signer, or signers.

MQCA_RECIPIENT_DN
The distinguished name of an intended recipient, or recipients.

MQIA_TOLERATE_UNPROTECTED
Whether the policy is enforced or unprotected messages tolerated.

V 9.0.0 **MQIA_KEY_REUSE_COUNT**
The number of times that an encryption key can be re-used.

MQIACF_ACTION
The action taken on the command with regards to signer and recipient parameters.

Inquire Policy (Response) on Multiplatforms: Multi

The response to the Inquire Policy (MQCMD_INQUIRE_PROT_POLICY) command consists of the response header followed by the *PolicyName* structure and the requested combination of attribute parameter structures.

If a generic security policy name was specified, one such message is generated for each policy found.

Always returned:
PolicyName

The name of the policy, or policies (or part of the policy name or names) to inquire are the same as the name of the queue, or queues, that the policies control.

Returned if requested:

Action, *EncAlg*, *Enforce* and *Tolerate*, V 9.0.0 KeyReuse *Recipient*, *Recipient*, *SignAlg*, *Signer*

Response data

Action (MQCFIL)
Action (parameter identifier: MQIACF_ACTION).

The action taken on the command with regards to signer and recipient parameters.

EncAlg (MQCFIL)
Encryption algorithm (parameter identifier: MQIA_ENCRYPTION_ALGORITHM).

The encryption algorithm specified.

Enforce and Tolerate (MQCFST)
Indicates whether the security policy should be enforced or whether unprotected messages are tolerated (parameter identifier: MQIA_TOLERATE_UNPROTECTED).

V 9.0.0

KeyReuse (MQCFIN)
Specifies the number of times that an encryption key can be re-used (parameter identifier MQIA_KEY_REUSE_COUNT)

Recipient (MQCFIL)
Specifies the distinguished name of the intended recipient (parameter identifier: MQCA_RECIPIENT_DN)

This parameter can be specified multiple times.

The maximum length of the string is MQ_DISTINGUISHED_NAME_LENGTH.

SignAlg (MQCFIL)
Specifies the digital signature algorithm (parameter identifier: MQIA_SIGNATURE_ALGORITHM).

Signer (MQCFST)
Specifies the distinguished name of an authorized signer (parameter identifier: MQCA_SIGNER_DN)

This parameter can be specified multiple times.

The maximum length of the string is MQ_DISTINGUISHED_NAME_LENGTH.

Inquire Process:

The Inquire Process (MQCMD_INQUIRE_PROCESS) command inquires about the attributes of existing IBM MQ processes.

Required parameters

ProcessName (MQCFST)
Process name (parameter identifier: MQCA_PROCESS_NAME).

Generic process names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all processes having names that start with the selected character string. An asterisk on its own matches all possible names.

The process name is always returned regardless of the attributes requested.

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

Optional parameters z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *ProcessAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1601 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the **StringFilterCommand** parameter.

ProcessAttrs (MQCFIL)
Process attributes (parameter identifier: MQIACF_PROCESS_ATTRS).

The attribute list might specify the following value on its own - default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_ALTERATION_DATE
The date at which the information was last altered.

MQCA_ALTERATION_TIME
The time at which the information was last altered.

MQCA_APPL_ID
Application identifier.

MQCA_ENV_DATA
Environment data.

MQCA_PROCESS_DESC
Description of process definition.

MQCA_PROCESS_NAME
Name of process definition.

MQCA_USER_DATA
User data.

MQIA_APPL_TYPE
Application type.

z/OS

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined as either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

You cannot use *QSGDisposition* as a parameter to filter on.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed

in *ProcessAttrs* except MQCA_PROCESS_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1608 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter.

Inquire Process (Response):

The response to the Inquire Process (MQCMD_INQUIRE_PROCESS) command consists of the response header followed by the *ProcessName* structure and the requested combination of attribute parameter structures.

If a generic process name was specified, one such message is generated for each process found.

Always returned:

ProcessName, z/OS *QSGDisposition*

Returned if requested:

AlterationDate, *AlterationTime*, *ApplId*, *ApplType*, *EnvData*, *ProcessDesc*, *UserData*

Response data

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered, in the form yyyy-mm-dd.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered, in the form hh.mm.ss.

ApplId (MQCFST)
Application identifier (parameter identifier: MQCA_APPL_ID).

The maximum length of the string is MQ_PROCESS_APPL_ID_LENGTH.

ApplType (MQCFIN)
Application type (parameter identifier: MQIA_APPL_TYPE).

The value can be:

MQAT_AIX
AIX application (same value as MQAT_UNIX)

MQAT_CICS
CICS transaction

MQAT_DOS
DOS client application

MQAT_MVS
z/OS application

MQAT_OS400
IBM i application

MQAT_QMGR
Queue manager

MQAT_UNIX
UNIX application

MQAT_WINDOWS
16-bit Windows application

MQAT_WINDOWS_NT
32-bit Windows application

integer
System-defined application type in the range zero through 65 535 or a user-defined application type in the range 65 536 through 999 999 999

EnvData (MQCFST)
Environment data (parameter identifier: MQCA_ENV_DATA).

The maximum length of the string is MQ_PROCESS_ENV_DATA_LENGTH.

ProcessDesc (MQCFST)
Description of process definition (parameter identifier: MQCA_PROCESS_DESC).

The maximum length of the string is MQ_PROCESS_DESC_LENGTH.

ProcessName (MQCFST)
The name of the process definition (parameter identifier: MQCA_PROCESS_NAME).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

z/OS

QSGDisposition (MQCFIN)
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object (that is, where it is defined and how it behaves). This parameter is valid on z/OS only. The value can be any of the following values:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

UserData (MQCFST)
User data (parameter identifier: MQCA_USER_DATA).

The maximum length of the string is MQ_PROCESS_USER_DATA_LENGTH.

Inquire Process Names:

The Inquire Process Names (MQCMD_INQUIRE_PROCESS_NAMES) command inquires for a list of process names that match the generic process name specified.

Required parameters

ProcessName (MQCFST)
Name of process-definition for queue (parameter identifier: MQCA_PROCESS_NAME).

Generic process names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

z/OS

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined with either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

Inquire Process Names (Response):

The response to the Inquire Process Names (MQCMD_INQUIRE_PROCESS_NAMES) command consists of the response header followed by a single parameter structure giving zero or more names that match the specified process name.

Additionally, on z/OS only, a parameter structure, *QSGDispositions* (with the same number of entries as the *ProcessNames* structure) is returned. Each entry in this structure indicates the disposition of the object with the corresponding entry in the *ProcessNames* structure.

This response is not supported on Windows.

Always returned:
ProcessNames, *QSGDispositions*

Returned if requested:
None

Response data

ProcessNames (MQCFSL)
List of process names (parameter identifier: MQCACF_PROCESS_NAMES).

QSGDispositions (MQCFIL)
List of queue-sharing group dispositions (parameter identifier: MQIACF_QSG_DISPS). This parameter applies only to z/OS. Possible values for fields in this structure are:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

Inquire Pub/Sub Status:

The Inquire Pub/Sub Status (MQCMD_INQUIRE_PUBSUB_STATUS) command inquires about the status of publish/subscribe connections.

Optional parameters

z/OS **CommandScope (MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

blank (or omit the parameter altogether)
The command is executed on the queue manager on which it was entered.

a queue manager name
The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

an asterisk (*)
The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use CommandScope as a parameter to filter on.

PubSubStatusAttrs (MQCFIL)
Publish/subscribe status attributes (parameter identifier: MQIACF_PUBSUB_STATUS_ATTRS).

The attribute list might specify the following value on its own - default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQIA_SUB_COUNT
The total number of subscriptions against the local tree.

MQIA_TOPIC_NODE_COUNT
The total number of topic nodes in the local tree.

MQIACF_PUBSUB_STATUS
Hierarchy status.

MQIACF_PS_STATUS_TYPE
Hierarchy type.

Type (MQCFIN)
Type (parameter identifier: MQIACF_PS_STATUS_TYPE).

The type can specify one of the following:

MQPSST_ALL
Return status of both parent and child connections. MQPSST_ALL is the default value if the parameter is not specified.

MQPSST_LOCAL
Return local status information.

MQPSST_PARENT
Return status of the parent connection.

MQPSST_CHILD
Return status of the child connections.

Inquire Pub/Sub Status (Response):

The response to the Inquire publish/subscribe Status (MQCMD_INQUIRE_PUBSUB_STATUS) command consists of the response header followed by the attribute structures.

A group of parameters is returned containing the following attributes: *Type*, *QueueManagerName*, *Status*, *SubCount*, and *TopicNodeCount*.

Always returned:
QueueManagerName, *Status*, *Type*, *SubCount*, and *TopicNodeCount*.

Returned if requested:
None

Response data

QueueManagerName (MQCFST)
Either the name of the local queue manager when TYPE is LOCAL, or the name of the hierarchically connected queue manager (parameter identifier: MQCA_Q_MGR_NAME).

Type (MQCFIN)
Type of status that is being returned (parameter identifier: MQIACF_PS_ STATUS_TYPE).

The value can be:

MQPSST_CHILD
Publish/subscribe status for a child hierarchical connection.

MQPSST_LOCAL
Publish/subscribe status for the local queue manager.

MQPSST_PARENT
Publish/subscribe status for the parent hierarchical connection.

Status (MQCFIN)
The status of the publish/subscribe engine or the hierarchical connection (parameter identifier: MQIACF_PUBSUB_STATUS).

When TYPE is LOCAL the following values can be returned:

MQPS_STATUS_ACTIVE
The publish/subscribe engine and the queued publish/subscribe interface are running. It is therefore possible to publish or subscribe using the application programming interface and the queues that are monitored by the queued publish/subscribe interface appropriately.

MQPS_STATUS_COMPAT
The publish/subscribe engine is running. It is therefore possible to publish or subscribe using the application programming interface. The queued publish/subscribe interface is not running. Therefore, any message that is put to the queues monitored by the queued publish/subscribe interface is not acted upon by IBM MQ.

MQPS_STATUS_ERROR
The publish/subscribe engine has failed. Check your error logs to determine the reason for the failure.

MQPS_STATUS_INACTIVE
The publish/subscribe engine and the queued publish/subscribe interface are not running. It is therefore not possible to publish or subscribe using the application programming interface. Any publish/subscribe messages that are put to the queues that are monitored by the queued publish/subscribe interface is not acted upon by IBM MQ.

If inactive and you want to start the publish/subscribe engine, on the Change Queue Manager command set PubSubMode to **MQPSM_ENABLED**.

MQPS_STATUS_STARTING
The publish/subscribe engine is initializing and is not yet operational.

MQPS_STATUS_STOPPING
The publish/subscribe engine is stopping.

When TYPE is PARENT, the following values can be returned:

MQPS_STATUS_ACTIVE
The connection with the parent queue manager is active.

MQPS_STATUS_ERROR
This queue manager is unable to initialize a connection with the parent queue manager because of a configuration error.

A message is produced in the queue manager logs to indicate the specific error. If you receive error message AMQ5821 or on z/OS systems CSQT821E, possible causes include:

- Transmit queue is full
- Transmit queue put disabled

If you receive error message AMQ5814 or on z/OS systems CSQT814E, take the following actions:

- Check that the parent queue manager is correctly specified.
- Ensure that broker is able to resolve the queue manager name of the parent broker.

To resolve the queue manager name, at least one of the following resources must be configured:

- A transmission queue with the same name as the parent queue manager name.
- A queue manager alias definition with the same name as the parent queue manager name.
- A cluster with the parent queue manager a member of the same cluster as this queue manager.
- A cluster queue manager alias definition with the same name as the parent queue manager name.
- A default transmission queue.

After you have set up the configuration correctly, modify the parent queue manager name to blank. Then set with the parent queue manager name.

MQPS_STATUS_REFUSED
The connection has been refused by the parent queue manager.

This situation might be caused by the parent queue manager already having another child queue manager of the same name as this queue manager.

Alternatively, the parent queue manager has used the RESET QMGR TYPE(PUBSUB) CHILD command to remove this queue manager as one of its children.

MQPS_STATUS_STARTING
The queue manager is attempting to request that another queue manager is its parent.

If the parent status remains in starting status without progressing to active status, take the following actions:

- Check that the sender channel to parent queue manager is running
- Check that the receiver channel from parent queue manager is running

MQPS_STATUS_STOPPING
The queue manager is disconnecting from its parent.

If the parent status remains in stopping status, take the following actions:
- Check that the sender channel to parent queue manager is running
- Check that the receiver channel from parent queue manager is running

When TYPE is CHILD, the following values can be returned:

MQPS_STATUS_ACTIVE
The connection with the parent queue manager is active.

MQPS_STATUS_ERROR
This queue manager is unable to initialize a connection with the parent queue manager because of a configuration error.

A message is produced in the queue manager logs to indicate the specific error. If you receive error message AMQ5821 or on z/OS systems CSQT821E, possible causes include:
- Transmit queue is full
- Transmit queue put disabled

If you receive error message AMQ5814 or on z/OS systems CSQT814E, take the following actions:
- Check that the child queue manager is correctly specified.
- Ensure that broker is able to resolve the queue manager name of the child broker.

To resolve the queue manager name, at least one of the following resources must be configured:
- A transmission queue with the same name as the child queue manager name.
- A queue manager alias definition with the same name as the child queue manager name.
- A cluster with the child queue manager a member of the same cluster as this queue manager.
- A cluster queue manager alias definition with the same name as the child queue manager name.
- A default transmission queue.

After you have set up the configuration correctly, modify the child queue manager name to blank. Then set with the child queue manager name.

MQPS_STATUS_STARTING
The queue manager is attempting to request that another queue manager is its parent.

If the child status remains in starting status without progressing to active status, take the following actions:
- Check that the sender channel to child queue manager is running
- Check that the receiver channel from child queue manager is running

MQPS_STATUS_STOPPING
The queue manager is disconnecting from its parent.

If the child status remains in stopping status, take the following actions:
- Check that the sender channel to child queue manager is running
- Check that the receiver channel from child queue manager is running

SubCount (MQCFIN)
When *Type* is MQPSST_LOCAL, the total number of subscriptions against the local tree is returned. When

Type is MQPSST_CHILD or MQPSST_PARENT, queue manager relations are not inquired and the value MQPSCT_NONE is returned. (parameter identifier: MQIA_SUB_COUNT).

TopicNodeCount (MQCFIN)
When *Type* is MQPSST_LOCAL, the total number of topic nodes in the local tree is returned. When *Type* is MQPSST_CHILD or MQPSST_PARENT, queue manager relations are not inquired and the value MQPSCT_NONE is returned. (parameter identifier: MQIA_TOPIC_NODE_COUNT).

Inquire Queue:

Use the Inquire Queue command MQCMD_INQUIRE_Q to query the attributes of IBM MQ queues.

Required parameters

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

Generic queue names are supported. A generic name is a character string followed by an asterisk * ; for example ABC*. It selects all queues having names that start with the selected character string. An asterisk on its own matches all possible names.

The queue name is always returned, regardless of the attributes requested.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Optional parameters z/OS

CFStructure (MQCFST)
CF structure (parameter identifier: MQCA_CF_STRUC_NAME). Specifies the name of the CF structure. This parameter is valid only on z/OS.

This parameter specifies that eligible queues are limited to those having the specified *CFStructure* value. If this parameter is not specified, then all queues are eligible.

Generic CF structure names are supported. A generic name is a character string followed by an asterisk * ; for example ABC*. It selects all CF structures having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

ClusterInfo (MQCFIN)
Cluster information (parameter identifier: MQIACF_CLUSTER_INFO).

This parameter requests that cluster information about these queues and other queues in the repository that match the selection criteria is displayed. The cluster information is displayed in addition to information about attributes of queues defined on this queue manager.

In this case, there might be multiple queues with the same name displayed. The cluster information is shown with a queue type of MQQT_CLUSTER.

You can set this parameter to any integer value, the value used does not affect the response to the command.

The cluster information is obtained locally from the queue manager.

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

This parameter specifies that eligible queues are limited to those having the specified *ClusterName* value. If this parameter is not specified, then all queues are eligible.

Generic cluster names are supported. A generic name is a character string followed by an asterisk * ; for example ABC*. It selects all clusters having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

ClusterNamelist (MQCFST)
Cluster namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

This parameter specifies that eligible queues are limited to those having the specified *ClusterNameList* value. If this parameter is not specified, then all queues are eligible.

Generic cluster namelists are supported. A generic name is a character string followed by an asterisk * ; for example ABC*. It selects all cluster namelists having names that start with the selected character string. An asterisk on its own matches all possible names.

z/OS

z/OS **CommandScope (MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is processed when the queue manager is a member of a queue-sharing group. You can specify one of the following values:

- Blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- A queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment. The command server must be enabled.
- An asterisk "*". The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *QAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1601 for information about using this filter condition.

If you specify an integer filter for *Qtype* or *PageSetID*, you cannot also specify the *Qtype* or *PageSetID* parameter.

If you specify an integer filter, you cannot also specify a string filter using the **StringFilterCommand** parameter.

z/OS

PageSetID (MQCFIN)
Page set identifier (parameter identifier: MQIA_PAGESET_ID). This parameter applies to z/OS only.

This parameter specifies that eligible queues are limited to those having the specified *PageSetID* value. If this parameter is not specified, then all queues are eligible.

QAttrs (MQCFIL)
Queue attributes (parameter identifier: MQIACF_Q_ATTRS).

The attribute list might specify the following value on its own. If the parameter is not specified, this value is the default:

MQIACF_ALL
All attributes.

You can also specify a combination of the parameters in the following table:

Table 132. Inquire Queue command, queue attributes

	Local queue	Model queue	Alias queue	Remote queue	Cluster queue
MQCA_ALTERATION_DATE The date on which the information was last altered	✓	✓	✓	✓	✓
MQCA_ALTERATION_TIME The time at which the information was last altered	✓	✓	✓	✓	✓
MQCA_BACKOUT_REQ_Q_NAME Excessive backout requeue name	✓	✓			
MQCA_BASE_NAME Name of queue that alias resolves to			✓		
MQCA_CF_STRUC_NAME Coupling facility structure name. This attribute is valid on z/OS only	✓	✓			
MQCA_CLUS_CHL_NAME The generic name of the cluster-sender channels that use this queue as a transmission queue.	✓	✓			
MQCA_CLUSTER_DATE Date when the definition became available to the local queue manager					✓
MQCA_CLUSTER_NAME Cluster name	✓		✓	✓	✓
MQCA_CLUSTER_NAMELIST Cluster namelist	✓		✓	✓	
MQCA_CLUSTER_Q_MGR_NAME Queue manager name that hosts the queue					✓
MQCA_CLUSTER_TIME Time when the definition became available to the local queue manager					✓
MQCA_CREATION_DATE Queue creation date	✓	✓			
MQCA_CREATION_TIME Queue creation time	✓	✓			
MQCA_CUSTOM The custom attribute for new features	✓	✓	✓	✓	✓
MQCA_INITIATION_Q_NAME Initiation queue name	✓	✓			

Table 132. Inquire Queue command, queue attributes (continued)

	Local queue	Model queue	Alias queue	Remote queue	Cluster queue
MQCA_PROCESS_NAME Name of process definition	✓	✓			
MQCA_Q_DESC Queue description	✓	✓	✓	✓	✓
MQCA_Q_MGR_IDENTIFIER Internally generated queue manager name					✓
MQCA_Q_NAME Queue name	✓	✓	✓	✓	✓
MQCA_REMOTE_Q_MGR_NAME Name of remote queue manager				✓	
MQCA_REMOTE_Q_NAME Name of remote queue as known locally on the remote queue manager				✓	
z/OS MQCA_STORAGE_CLASS Storage class. MQCA_STORAGE_CLASS is valid on z/OS only	✓	✓			
MQCA_TPIPE_NAME The **TPIPE** name used for communication with OTMA using the IBM MQ IMS bridge	✓				
MQCA_TRIGGER_DATA Trigger data	✓	✓			
MQCA_XMIT_Q_NAME Transmission queue name				✓	
MQIA_ACCOUNTING_Q Accounting data collection	✓	✓			
MQIA_BACKOUT_THRESHOLD Backout threshold	✓	✓			
MQIA_BASE_TYPE Type of object	✓	✓	✓	✓	✓
MQIA_CLUSTER_Q_TYPE Cluster queue type					✓
MQIA_CLWL_Q_PRIORITY Cluster workload queue priority	✓		✓	✓	✓

Table 132. Inquire Queue command, queue attributes (continued)

	Local queue	Model queue	Alias queue	Remote queue	Cluster queue
MQIA_CLWL_Q_RANK Cluster workload queue rank	✓		✓	✓	✓
MQIA_CLWL_USEQ Cluster workload use remote setting	✓				
MQIA_CURRENT_Q_DEPTH Number of messages on queue	✓				
MQIA_DEF_BIND Default binding	✓		✓	✓	✓
MQIA_DEF_INPUT_OPEN_OPTION Default open-for-input option	✓	✓			
MQIA_DEF_PERSISTENCE Default message persistence	✓	✓	✓	✓	✓
MQIA_DEF_PRIORITY Default message priority	✓	✓	✓	✓	✓
MQIA_DEF_PUT_RESPONSE_TYPE Default put response type	✓	✓	✓	✓	✓
MQIA_DEF_READ_AHEAD Default put response type	✓	✓	✓	✓	✓
MQIA_DEFINITION_TYPE Queue definition type	✓	✓			
MQIA_DIST_LISTS Distribution list support. MQIA_DIST_LISTS is not valid on z/OS	✓	✓			
MQIA_HARDEN_GET_BACKOUT Whether to harden backout count	✓	✓			
MQIA_INDEX_TYPE Index type. This attribute is valid on z/OS only.	✓	✓			
MQIA_INHIBIT_GET Whether get operations are allowed	✓	✓	✓		
MQIA_INHIBIT_PUT Whether put operations are allowed	✓	✓	✓	✓	✓
MQIA_MAX_MSG_LENGTH Maximum message length	✓	✓			

Table 132. Inquire Queue command, queue attributes (continued)

	Local queue	Model queue	Alias queue	Remote queue	Cluster queue
MQIA_MAX_Q_DEPTH Maximum number of messages allowed on queue	✓	✓			
MQIA_MONITORING_Q Online monitoring data collection	✓	✓			
MQIA_MSG_DELIVERY_SEQUENCE Whether message priority is relevant	✓	✓			
MQIA_NPM_CLASS Level of reliability assigned to non-persistent messages that are put to the queue	✓	✓			
MQIA_OPEN_INPUT_COUNT Number of MQOPEN calls that have the queue open for input	✓				
MQIA_OPEN_OUTPUT_COUNT Number of MQOPEN calls that have the queue open for output	✓				
z/OS MQIA_PAGESET_ID Page set identifier	✓				
MQIA_PROPERTY_CONTROL Property control attribute	✓	✓	✓		
MQIA_Q_DEPTH_HIGH_EVENT Control attribute for queue depth high events. You cannot use MQIA_Q_DEPTH_HIGH_EVENT as a filter attribute.	✓	✓			
MQIA_Q_DEPTH_HIGH_LIMIT High limit for queue depth	✓	✓			
MQIA_Q_DEPTH_LOW_EVENT Control attribute for queue depth low events. You cannot use MQIA_Q_DEPTH_LOW_EVENT as a filter attribute.	✓	✓			
MQIA_Q_DEPTH_LOW_LIMIT Low limit for queue depth	✓	✓			

Table 132. Inquire Queue command, queue attributes (continued)

	Local queue	Model queue	Alias queue	Remote queue	Cluster queue
MQIA_Q_DEPTH_MAX_EVENT Control attribute for queue depth max events	✓	✓			
MQIA_Q_SERVICE_INTERVAL Limit for queue service interval	✓	✓			
MQIA_Q_SERVICE_INTERVAL_ EVENT Control attribute for queue service interval events	✓	✓			
MQIA_Q_TYPE Queue type	✓	✓	✓	✓	✓
MQIA_RETENTION_INTERVAL Queue retention interval	✓	✓			
MQIA_SCOPE Queue definition scope. MQIA_SCOPE is not valid on z/OS or IBM i	✓		✓	✓	
MQIA_SHAREABILITY Whether queue can be shared	✓	✓			
MQIA_STATISTICS_Q Statistics data collection. MQIA_STATISTICS_Q is valid only on Multiplatforms.	✓	✓			
MQIA_TRIGGER_CONTROL Trigger control	✓	✓			
MQIA_TRIGGER_DEPTH Trigger depth	✓	✓			
MQIA_TRIGGER_MSG_PRIORITY Threshold message priority for triggers	✓	✓			
MQIA_TRIGGER_MTYPE Trigger type	✓	✓			
MQIA_USAGE Usage	✓	✓			

z/OS **QSGDisposition (MQCFIN)**

Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned. The meaning of "the disposition of an object" is where the object is defined and how it behaves. The value can be any of the following values:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. In a shared queue manager environment, if the command is run on the queue manager where it was issued, MQQSGD_LIVE also returns information for objects defined with MQQSGD_SHARED. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

In a shared queue manager environment, if the command is run on the queue manager where it was issued, MQQSGD_ALL also displays information for objects defined with MQQSGD_GROUP or MQQSGD_SHARED.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names, with different dispositions.

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined with either MQQSGD_Q_MGR or MQQSGD_COPY.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED. MQQSGD_SHARED is permitted only in a shared queue environment.

You cannot use *QSGDisposition* as a parameter to filter on.

QType (MQCFIN)
Queue type (parameter identifier: MQIA_Q_TYPE).

If this parameter is present, eligible queues are limited to the specified type. Any attribute selector specified in the *QAttrs* list which is valid only for queues of a different type or types is ignored; no error is raised.

If this parameter is not present, or if MQQT_ALL is specified, queues of all types are eligible. Each attribute specified must be a valid queue attribute selector. The attribute can apply to some of the queues returned. It does not have to apply to all the queues. Queue attribute selectors that are valid but not applicable to the queue are ignored, no error messages occur and no attribute is returned. The following lists contains the value of all valid queue attribute selectors:

MQQT_ALL
All queue types.

MQQT_LOCAL
Local queue.

MQQT_ALIAS
Alias queue definition.

MQQT_REMOTE
Local definition of a remote queue.

MQQT_CLUSTER
Cluster queue.

MQQT_MODEL
Model queue definition.

Note: Multi On Multiplatforms, if this parameter is present, it must occur immediately after the **QName** parameter.

z/OS **StorageClass (MQCFST)**
Storage class (parameter identifier: MQCA_STORAGE_CLASS). Specifies the name of the storage class. This parameter is valid only on z/OS.

This parameter specifies that eligible queues are limited to those having the specified *StorageClass* value. If this parameter is not specified, then all queues are eligible.

Generic names are supported. A generic name is a character string followed by an asterisk * ; for example ABC*. It selects all storage classes having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_STORAGE_CLASS_LENGTH.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *QAttrs* except MQCA_Q_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1608 for information about using this filter condition.

If you specify a string filter for *ClusterName*, *ClusterNameList*, *StorageClass*, or *CFStructure*, you cannot also specify that as a parameter.

If you specify a string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter.

Error codes

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_Q_TYPE_ERROR
Queue type not valid.

Inquire Queue (Response):

The response to the Inquire Queue command MQCMD_INQUIRE_Q consists of the response header followed by the *QName* structure. On z/OS only, response includes the *QSGDisposition* structure, and the requested combination of attribute parameter structures.

If a generic queue name was specified, or cluster queues requested, by setting either MQQT_CLUSTER or MQIACF_CLUSTER_INFO, one message is generated for each queue found.

Always returned:
QName, *QSGDisposition*, *QType*

Returned if requested:
AlterationDate, *AlterationTime*, *BackoutRequeueName*, *BackoutThreshold*, *BaseQName*, , *CFStructure*, *ClusterChannelName*, *ClusterDate*, *ClusterName*, *ClusterNamelist*, *ClusterQType*, *ClusterTime*, *CLWLQueuePriority*, *CLWLQueueRank*, *CLWLUseQ*, *CreationDate*, *CreationTime*, *CurrentQDepth*, *Custom*, *DefaultPutResponse*, *DefBind*, *DefinitionType*, *DefInputOpenOption*, *DefPersistence*, *DefPriority*, *DefReadAhead*, *DistLists*, *HardenGetBackout*, V 9.0.2 *Imgrcovq*,

IndexType, *InhibitGet*, *InhibitPut*, *InitiationQName*, *MaxMsgLength*, *MaxQDepth*, *MsgDeliverySequence*, *NonPersistentMessageClass*, *OpenInputCount*, *OpenOutputCount*, *PageSetID*, *ProcessName*, *PropertyControl*, *QDepthHighEvent*, *QDepthHighLimit*, *QDepthLowEvent*, *QDepthLowLimit*, *QDepthMaxEvent*, *QDesc*, *QMgrIdentifier*, *QMgrName*, *QServiceInterval*, *QServiceIntervalEvent*, *QueueAccounting*, *QueueMonitoring*, *QueueStatistics*, *RemoteQMgrName*, *RemoteQName*, *RetentionInterval*, *Scope*, *Shareability*, *StorageClass*, *TpipeNames*, *TriggerControl*, *TriggerData*, *TriggerDepth*, *TriggerMsgPriority*, *TriggerType*, *Usage*, *XmitQName*

Response data

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered, in the form yyyy-mm-dd.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered, in the form hh.mm.ss.

BackoutRequeueName (MQCFST)
Excessive backout requeue name (parameter identifier: MQCA_BACKOUT_REQ_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

BackoutThreshold (MQCFIN)
Backout threshold (parameter identifier: MQIA_BACKOUT_THRESHOLD).

BaseQName (MQCFST)
Queue name to which the alias resolves (parameter identifier: MQCA_BASE_Q_NAME).

The name of a queue that is defined to the local queue manager.

The maximum length of the string is MQ_Q_NAME_LENGTH.

CFStructure (MQCFST)
Coupling facility structure name (parameter identifier: MQCA_CF_STRUC_NAME). This parameter applies to z/OS only.

Specifies the name of the coupling facility structure where you want to store messages when you use shared queues.

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

ClusterChannelName (MQCFST)
Cluster-sender channel name (parameter identifier: MQCA_CLUS_CHL_NAME).

ClusterChannelName is the generic name of the cluster-sender channels that use this queue as a transmission queue.

The maximum length of the channel name is: MQ_CHANNEL_NAME_LENGTH.

ClusterDate (MQCFST)
Cluster date (parameter identifier: MQCA_CLUSTER_DATE).

The date on which the information became available to the local queue manager, in the form yyyy-mm-dd.

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

ClusterNamelist (MQCFST)
Cluster namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

ClusterQType (MQCFIN)
Cluster queue type (parameter identifier: MQIA_CLUSTER_Q_TYPE).

The value can be:

MQCQT_LOCAL_Q
The cluster queue represents a local queue.

MQCQT_ALIAS_Q
The cluster queue represents an alias queue.

MQCQT_REMOTE_Q
The cluster queue represents a remote queue.

MQCQT_Q_MGR_ALIAS
The cluster queue represents a queue manager alias.

ClusterTime (MQCFST)
Cluster time (parameter identifier: MQCA_CLUSTER_TIME).

The time at which the information became available to the local queue manager, in the form hh.mm.ss.

CLWLQueuePriority (MQCFIN)
Cluster workload queue priority (parameter identifier: MQIA_CLWL_Q_PRIORITY).

Priority of the queue in cluster workload management. The value is in the range zero through 9, where zero is the lowest priority and 9 is the highest.

CLWLQueueRank (MQCFIN)
Cluster workload queue rank (parameter identifier: MQIA_CLWL_Q_RANK).

Rank of the queue in cluster workload management. The value is in the range zero through 9, where zero is the lowest rank and 9 is the highest.

CLWLUseQ (MQCFIN)
Cluster workload queue rank (parameter identifier: MQIA_CLWL_USEQ).

The value can be:

MQCLWL_USEQ_AS_Q_MGR
Use the value of the **CLWLUseQ** parameter on the queue manager's definition.

MQCLWL_USEQ_ANY
Use remote and local queues.

MQCLWL_USEQ_LOCAL
Do not use remote queues.

CreationDate (MQCFST)
Queue creation date, in the form yyyy-mm-dd (parameter identifier: MQCA_CREATION_DATE).

The maximum length of the string is MQ_CREATION_DATE_LENGTH.

CreationTime (MQCFST)
Creation time, in the form hh.mm.ss (parameter identifier: MQCA_CREATION_TIME).

The maximum length of the string is MQ_CREATION_TIME_LENGTH.

CurrentQDepth (MQCFIN)
Current queue depth (parameter identifier: MQIA_CURRENT_Q_DEPTH).

Custom (MQCFST)
Custom attribute for new features (parameter identifier: MQCA_CUSTOM).

This attribute is reserved for the configuration of new features before separate attributes are named. It can contain the values of zero or more attributes as pairs of attribute name and value, separated by at least one space. The attribute name-value pairs have the form NAME(VALUE).

This description is updated when features using this attribute are introduced.

DefaultPutResponse (MQCFIN)
Default put response type definition (parameter identifier: MQIA_DEF_PUT_RESPONSE_TYPE).

The parameter specifies the type of response to be used for put operations to the queue when an application specifies MQPMO_RESPONSE_AS_Q_DEF. The value can be any of the following values:

MQPRT_SYNC_RESPONSE
The put operation is issued synchronously, returning a response.

MQPRT_ASYNC_RESPONSE
The put operation is issued asynchronously, returning a subset of MQMD fields.

DefBind (MQCFIN)
Default binding (parameter identifier: MQIA_DEF_BIND).

The value can be:

MQBND_BIND_ON_OPEN
Binding fixed by MQOPEN call.

MQBND_BIND_NOT_FIXED
Binding not fixed.

MQBND_BIND_ON_GROUP
Allows an application to request that a group of messages are all allocated to the same destination instance.

DefinitionType (MQCFIN)
Queue definition type (parameter identifier: MQIA_DEFINITION_TYPE).

The value can be:

MQQDT_PREDEFINED
Predefined permanent queue.

MQQDT_PERMANENT_DYNAMIC
Dynamically defined permanent queue.

MQQDT_SHARED_DYNAMIC
Dynamically defined shared queue. This option is available on z/OS only.

MQQDT_TEMPORARY_DYNAMIC
Dynamically defined temporary queue.

DefInputOpenOption (MQCFIN)
Default input open option for defining whether queues can be shared (parameter identifier: MQIA_DEF_INPUT_OPEN_OPTION).

The value can be:

MQOO_INPUT_EXCLUSIVE
Open queue to get messages with exclusive access.

MQOO_INPUT_SHARED
Open queue to get messages with shared access.

DefPersistence (MQCFIN)
Default persistence (parameter identifier: MQIA_DEF_PERSISTENCE).

The value can be:

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

DefPriority (MQCFIN)
Default priority (parameter identifier: MQIA_DEF_PRIORITY).

DefReadAhead (MQCFIN)
Default read ahead (parameter identifier: MQIA_DEF_READ_AHEAD).

Specifies the default read ahead behavior for non-persistent messages delivered to the client.

The value can be any of the following values:

MQREADA_NO
Non-persistent messages are not sent ahead to the client before an application requests them. A maximum of one non-persistent message can be lost if the client ends abnormally.

MQREADA_YES
Non-persistent messages are sent ahead to the client before an application requests them. Non-persistent messages can be lost if the client ends abnormally or if the client does not consume all the messages it is sent.

MQREADA_DISABLED
Read ahead of non-persistent messages in not enabled for this queue. Messages are not sent ahead to the client regardless of whether read ahead is requested by the client application.

Multi **DistLists (MQCFIN)**
Distribution list support (parameter identifier: MQIA_DIST_LISTS).

The value can be:

MQDL_SUPPORTED
Distribution lists supported.

MQDL_NOT_SUPPORTED
Distribution lists not supported.

This parameter is supported only on Multiplatforms.

HardenGetBackout (MQCFIN)
Harden backout, or not: (parameter identifier: MQIA_HARDEN_GET_BACKOUT).

The value can be:

MQQA_BACKOUT_HARDENED
Backout count remembered.

MQQA_BACKOUT_NOT_HARDENED
Backout count may not be remembered.

V 9.0.2 **ImageRecoverQueue (MQCFST)**
Specifies whether a local or permanent dynamic queue object is recoverable from a media image, if linear logging is being used (parameter identifier: MQIA_MEDIA_IMAGE_RECOVER_Q).

This parameter is not valid on z/OS. Possible values are:

MQIMGRCOV_YES
These queue objects are recoverable.

MQIMGRCOV_NO
Automatic media images, if enabled, are not written for these objects.

MQIMGRCOV_AS_Q_MGR
If the **ImageRecoverQueue** attribute for the queue manager specifies MQIMGRCOV_YES , these queue objects are recoverable.

If the **ImageRecoverQueue** attribute for the queue manager specifies MQIMGRCOV_NO, the "rcdmqimg (record media image)" on page 283 and"rcrmqobj (re-create object)" on page 289 commands are not permitted for these objects, and automatic media images, if enabled, are not written for these objects.

IndexType (MQCFIN)
Index type (parameter identifier: MQIA_INDEX_TYPE). This parameter applies to z/OS only.

Specifies the type of index maintained by the queue manager to expedite MQGET operations on the queue. The value can be any of the following values:

MQIT_NONE
No index.

MQIT_MSG_ID
The queue is indexed using message identifiers.

MQIT_CORREL_ID
The queue is indexed using correlation identifiers.

MQIT_MSG_TOKEN
The queue is indexed using message tokens.

MQIT_GROUP_ID
The queue is indexed using group identifiers.

InhibitGet (MQCFIN)
Get operations are allowed or inhibited: (parameter identifier: MQIA_INHIBIT_GET).

The value can be:

MQQA_GET_ALLOWED
Get operations are allowed.

MQQA_GET_INHIBITED
Get operations are inhibited.

InhibitPut (MQCFIN)
Putt operations are allowed or inhibited: (parameter identifier: MQIA_INHIBIT_PUT).

The value can be:

MQQA_PUT_ALLOWED
Put operations are allowed.

MQQA_PUT_INHIBITED
Put operations are inhibited.

InitiationQName (MQCFST)
Initiation queue name (parameter identifier: MQCA_INITIATION_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

MaxQDepth (MQCFIN)
Maximum queue depth (parameter identifier: MQIA_MAX_Q_DEPTH).

MsgDeliverySequence (MQCFIN)
Messages ordered by priority or sequence: (parameter identifier: MQIA_MSG_DELIVERY_SEQUENCE).

The value can be:

MQMDS_PRIORITY
Messages are returned in priority order.

MQMDS_FIFO
Messages are returned in FIFO order (first in, first out).

NonPersistentMessageClass (MQCFIN)
The level of reliability assigned to non-persistent messages that are put to the queue (parameter identifier: MQIA_NPM_CLASS).

Specifies the circumstances under which non-persistent messages put to the queue may be lost. The value can be any of the following values:

MQNPM_CLASS_NORMAL
Non-persistent messages are limited to the lifetime of the queue manager session. They are discarded in the event of a queue manager restart. MQNPM_CLASS_NORMAL is the default value.

MQNPM_CLASS_HIGH
The queue manager attempts to retain non-persistent messages for the lifetime of the queue. Non-persistent messages may still be lost in the event of a failure.

OpenInputCount (MQCFIN)
Number of MQOPEN calls that have the queue open for input (parameter identifier: MQIA_OPEN_INPUT_COUNT).

OpenOutputCount (MQCFIN)
Number of MQOPEN calls that have the queue open for output (parameter identifier: MQIA_OPEN_OUTPUT_COUNT).

PageSetID (MQCFIN)
Page set identifier (parameter identifier: MQIA_PAGESET_ID).

Specifies the identifier of the page set on which the queue resides.

This parameter applies to z/OS only when the queue is actively associated with a page set.

ProcessName (MQCFST)
Name of process definition for queue (parameter identifier: MQCA_PROCESS_NAME).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

PropertyControl (MQCFIN)
Property control attribute (parameter identifier MQIA_PROPERTY_CONTROL).

Specifies how message properties are handled for messages that are retrieved from queues using the MQGET call with the MQGMO_PROPERTIES_AS_Q_DEF option. The value can be any of the following values:

MQPROP_COMPATIBILITY
If the message contains a property with a prefix of **mcd.**, **jms.**, **usr.** or **mqext.**, all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except properties contained in the message descriptor (or extension), are discarded and are no longer accessible to the application.

MQPROP_COMPATIBILITY is the default value. It allows applications which expect JMS-related properties to be in an MQRFH2 header in the message data to continue to work unmodified.

MQPROP_NONE
All properties of the message are removed from the message before the message is sent to the remote queue manager. Properties in the message descriptor (or extension) are not removed.

MQPROP_ALL
All properties of the message are included with the message when it is sent to the remote queue manager. The properties are placed in one or more MQRFH2 headers in the message data. Properties in the message descriptor (or extension) are not placed in MQRFH2 headers.

MQPROP_FORCE_ MQRFH2
Properties are always returned in the message data in an MQRFH2 header regardless of whether the application specifies a message handle.

A valid message handle supplied in the MsgHandle field of the MQGMO structure on the MQGET call is ignored. Properties of the message are not accessible via the message handle.

This parameter is applicable to local, alias, and model queues.

QDepthHighEvent (MQCFIN)
Controls whether Queue Depth High events are generated (parameter identifier: MQIA_Q_DEPTH_HIGH_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthHighLimit (MQCFIN)
High limit for queue depth (parameter identifier: MQIA_Q_DEPTH_HIGH_LIMIT).

The threshold against which the queue depth is compared to generate a Queue Depth High event.

QDepthLowEvent (MQCFIN)
Controls whether Queue Depth Low events are generated (parameter identifier: MQIA_Q_DEPTH_LOW_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthLowLimit (MQCFIN)
Low limit for queue depth (parameter identifier: MQIA_Q_DEPTH_LOW_LIMIT).

The threshold against which the queue depth is compared to generate a Queue Depth Low event.

QDepthMaxEvent (MQCFIN)
Controls whether Queue Full events are generated (parameter identifier: MQIA_Q_DEPTH_MAX_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDesc (MQCFST)
Queue description (parameter identifier: MQCA_Q_DESC).

The maximum length of the string is MQ_Q_DESC_LENGTH.

QMgrIdentifier (MQCFST)
Queue manager identifier (parameter identifier: MQCA_Q_MGR_IDENTIFIER).

The unique identifier of the queue manager.

QMgrName (MQCFST)
Name of local queue manager (parameter identifier: MQCA_CLUSTER_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

QServiceInterval (MQCFIN)
Target for queue service interval (parameter identifier: MQIA_Q_SERVICE_INTERVAL).

The service interval used for comparison to generate Queue Service Interval High and Queue Service Interval OK events.

QServiceIntervalEvent (MQCFIN)
Controls whether Service Interval High or Service Interval OK events are generated (parameter identifier: MQIA_Q_SERVICE_INTERVAL_EVENT).

The value can be:

MQQSIE_HIGH
Queue Service Interval High events enabled.

MQQSIE_OK
Queue Service Interval OK events enabled.

MQQSIE_NONE
No queue service interval events enabled.

QSGDisposition (MQCFIN)
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object (that is, where it is defined and how it behaves). *QSGDisposition* is valid only on z/OS. The value can be any of the following values:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED.

QType (MQCFIN)
Queue type (parameter identifier: MQIA_Q_TYPE).

The value can be:

MQQT_ALIAS
Alias queue definition.

MQQT_CLUSTER
Cluster queue definition.

MQQT_LOCAL
Local queue.

MQQT_REMOTE
Local definition of a remote queue.

MQQT_MODEL
Model queue definition.

QueueAccounting (MQCFIN)
Controls the collection of accounting (thread-level and queue-level accounting) data (parameter identifier: MQIA_ACCOUNTING_Q).

The value can be:

MQMON_Q_MGR
The collection of accounting data for the queue is performed based upon the setting of the **QueueAccounting** parameter on the queue manager.

MQMON_OFF
Do not collect accounting data for the queue.

MQMON_ON
Collect accounting data for the queue.

QueueMonitoring (MQCFIN)
Online monitoring data collection (parameter identifier: MQIA_MONITORING_Q).

The value can be:

MQMON_OFF
Online monitoring data collection is turned off for this queue.

MQMON_Q_MGR
The value of the queue manager's **QueueMonitoring** parameter is inherited by the queue.

MQMON_LOW
Online monitoring data collection is turned on, with a low rate of data collection, for this queue unless *QueueMonitoring* for the queue manager is MQMON_NONE.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate rate of data collection, for this queue unless *QueueMonitoring* for the queue manager is MQMON_NONE.

MQMON_HIGH
Online monitoring data collection is turned on, with a high rate of data collection, for this queue unless *QueueMonitoring* for the queue manager is MQMON_NONE.

Multi **QueueStatistics (MQCFIN)**
Controls the collection of statistics data (parameter identifier: MQIA_STATISTICS_Q).

The value can be:

MQMON_Q_MGR
The collection of statistics data for the queue is performed based upon the setting of the **QueueStatistics** parameter on the queue manager.

MQMON_OFF
Do not collect statistics data for the queue.

MQMON_ON
Collect statistics data for the queue unless *QueueStatistics* for the queue manager is MQMON_NONE.

This parameter is supported only on Multiplatforms.

RemoteQMgrName (MQCFST)
Name of remote queue manager (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

RemoteQName (MQCFST)
Name of remote queue as known locally on the remote queue manager (parameter identifier: MQCA_REMOTE_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

RetentionInterval (MQCFIN)
Retention interval (parameter identifier: MQIA_RETENTION_INTERVAL).

Scope (MQCFIN)
Scope of the queue definition (parameter identifier: MQIA_SCOPE).

The value can be:

MQSCO_Q_MGR
Queue manager scope.

MQSCO_CELL
Cell scope.

This parameter is not valid on IBM i or z/OS.

Shareability (MQCFIN)
The queue can be shared, or not: (parameter identifier: MQIA_SHAREABILITY).

The value can be:

MQQA_SHAREABLE
Queue is shareable.

MQQA_NOT_SHAREABLE
Queue is not shareable.

StorageClass (MQCFST)
Storage class (parameter identifier: MQCA_STORAGE_CLASS). This parameter applies to z/OS only.

Specifies the name of the storage class.

The maximum length of the string is MQ_STORAGE_CLASS_LENGTH.

TpipeNames (MQCFSL)
TPIPE names (parameter identifier: MQCA_TPIPE_NAME). This parameter applies to local queues on z/OS only.

Specifies the TPIPE names used for communication with OTMA via the IBM MQ IMS bridge, if the bridge is active.

The maximum length of the string is MQ_TPIPE_NAME_LENGTH.

TriggerControl (MQCFIN)
Trigger control (parameter identifier: MQIA_TRIGGER_CONTROL).

The value can be:

MQTC_OFF
Trigger messages not required.

MQTC_ON
Trigger messages required.

TriggerData (MQCFST)
Trigger data (parameter identifier: MQCA_TRIGGER_DATA).

The maximum length of the string is MQ_TRIGGER_DATA_LENGTH.

TriggerDepth (MQCFIN)
Trigger depth (parameter identifier: MQIA_TRIGGER_DEPTH).

TriggerMsgPriority (MQCFIN)
Threshold message priority for triggers (parameter identifier: MQIA_TRIGGER_MSG_PRIORITY).

TriggerType (MQCFIN)
Trigger type (parameter identifier: MQIA_TRIGGER_TYPE).

The value can be:

MQTT_NONE
No trigger messages.

MQTT_FIRST
Trigger message when queue depth goes from 0 to 1.

MQTT_EVERY
Trigger message for every message.

MQTT_DEPTH
Trigger message when depth threshold exceeded.

Usage (MQCFIN)
Usage (parameter identifier: MQIA_USAGE).

The value can be:

MQUS_NORMAL
Normal usage.

MQUS_TRANSMISSION
Transmission queue.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

Inquire Queue Manager:

The Inquire Queue Manager (**MQCMD_INQUIRE_Q_MGR**) command inquires about the attributes of a queue manager.

Optional parameters

z/OS z/OS **CommandScope (MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is processed when the queue manager is a member of a queue-sharing group. You can specify one of the following values:

- Blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- A queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment. The command server must be enabled.
- An asterisk "*". The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

QMgrAttrs (MQCFIL)
Queue manager attributes (parameter identifier: **MQIACF_Q_MGR_ATTRS**).

The attribute list might specify the following value on its own - default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

Or a combination of the following values:

MQCA_ALTERATION_DATE
Date at which the definition was last altered.

MQCA_ALTERATION_TIME
Time at which the definition was last altered.

MQCA_CERT_LABEL
Queue manager certificate label.

MQCA_CHANNEL_AUTO_DEF_EXIT
Automatic channel definition exit name. **MQCA_CHANNEL_AUTO_DEF_EXIT** is not valid on z/OS.

MQCA_CLUSTER_WORKLOAD_DATA
Data passed to the cluster workload exit.

MQCA_CLUSTER_WORKLOAD_EXIT
Name of the cluster workload exit.

MQCA_COMMAND_INPUT_Q_NAME
System command input queue name.

MQCA_CONN_AUTH
Name of the authentication information object that is used to provide the location of user ID and password authentication.

MQCA_CUSTOM
The custom attribute for new features.

MQCA_DEAD_LETTER_Q_NAME
Name of dead-letter queue.

MQCA_DEF_XMIT_Q_NAME
Default transmission queue name.

z/OS **MQCA_DNS_GROUP**
The name of the group that the TCP listener handling inbound transmissions for the queue-sharing group must join when using Workload Manager for Dynamic Domain Name Services support (DDNS). **MQCA_DNS_GROUP** is valid on z/OS only.

z/OS **MQCA_IGQ_USER_ID**
Intra-group queuing user identifier. This parameter is valid on z/OS only.

z/OS **MQCA_LU_GROUP_NAME**
Generic LU name for the LU 6.2 listener. **MQCA_LU_GROUP_NAME** is valid on z/OS only.

z/OS **MQCA_LU_NAME**
LU name to use for outbound LU 6.2 transmissions. **MQCA_LU_NAME** is valid on z/OS only.

z/OS **MQCA_LU62_ARM_SUFFIX**
APPCPM suffix. **MQCA_LU62_ARM_SUFFIX** is valid on z/OS only.

MQCA_PARENT
The name of the hierarchically connected queue manager that is nominated as the parent of this queue manager.

MQCA_Q_MGR_DESC
Queue manager description.

MQCA_Q_MGR_IDENTIFIER
Internally generated unique queue manager name.

MQCA_Q_MGR_NAME
Name of local queue manager.

z/OS **MQCA_QSG_CERT_LABEL**
Queue-sharing group certificate label. This parameter attribute is valid on z/OS only.

z/OS **MQCA_QSG_NAME**
Queue-sharing group name. This parameter attribute is valid on z/OS only.

MQCA_REPOSITORY_NAME
Cluster name for the queue manager repository.

MQCA_REPOSITORY_NAMELIST
Name of the list of clusters for which the queue manager is providing a repository manager service.

MQCA_SSL_CRL_NAMELIST
TLS certificate revocation location namelist.

ULW **MQCA_SSL_CRYPTO_HARDWARE**
Parameters to configure the TLS cryptographic hardware. This parameter is supported only on UNIX, Linux, and Windows.

MQCA_SSL_KEY_REPOSITORY
Location and name of the TLS key repository.

z/OS **MQCA_TCP_NAME**
Name of the TCP/IP system that you are using. **MQCA_TCP_NAME** is valid on z/OS only.

MQCA_VERSION
The version of the IBM MQ installation, the queue manager is associated with. The version has the format *VVRRMMFF*:

VV: Version

RR: Release

MM: Maintenance level

FF: Fix level

ULW **MQIA_ACCOUNTING_CONN_OVERRIDE**
Specifies whether the settings of the **MQIAccounting** and **QueueAccounting** queue manager parameters can be overridden. **MQIA_ACCOUNTING_CONN_OVERRIDE** is valid only on UNIX, Linux, and Windows.

ULW **MQIA_ACCOUNTING_INTERVAL**
Intermediate accounting data collection interval. **MQIA_ACCOUNTING_INTERVAL** is valid only on UNIX, Linux, and Windows.

ULW **MQIA_ACCOUNTING_MQI**
Specifies whether accounting information is to be collected for MQI data. **MQIA_ACCOUNTING_MQI** is valid only on UNIX, Linux, and Windows.

MQIA_ACCOUNTING_Q
Accounting data collection for queues.

z/OS **MQIA_ACTIVE_CHANNELS**
Maximum number of channels that can be active at any time. **MQIA_ACTIVE_CHANNELS** is valid on z/OS only.

MQIA_ACTIVITY_CONN_OVERRIDE
Specifies whether the value of application activity trace can be overridden.

MQIA_ACTIVITY_RECORDING
Specifies whether activity reports can be generated.

MQIA_ACTIVITY_TRACE
Specifies whether application activity trace reports can be generated.

z/OS **MQIA_ADOPTNEWMCA_CHECK**
Elements checked to determine whether an MCA must be adopted when a new inbound channel is detected with the same name as an MCA that is already active. **MQIA_ADOPTNEWMCA_CHECK** is valid on z/OS only.

z/OS **MQIA_ADOPTNEWMCA_TYPE**
Specifies whether an orphaned instance of an MCA must be restarted automatically when a new inbound channel request matching the **AdoptNewMCACheck** parameter is detected. **MQIA_ADOPTNEWMCA_TYPE** is valid on z/OS only.

V 9.0.5 MQ Adv. **MQIA_ADVANCED_CAPABILITY**
Specifies whether IBM MQ Advanced extended capabilities are available for a queue manager.

This parameter is valid as follows:

- z/OS On z/OS from IBM MQ Version 9.0.4.
- Multi On other platforms from IBM MQ Version 9.0.5.

MQIA_AUTHORITY_EVENT
Control attribute for authority events.

z/OS **MQIA_BRIDGE_EVENT**
Control attribute for IMS bridge events. **MQIA_BRIDGE_EVENT** is valid only on z/OS.

ULW **MQIA_CERT_VAL_POLICY**
Specifies which TLS certificate validation policy is used to validate digital certificates received from remote partner systems. This attribute controls how strictly the certificate chain validation conforms to industry security standards. **MQIA_CERT_VAL_POLICY** is valid only on UNIX, Linux, and Windows. For more information, see Certificate validation policies in IBM MQ.

z/OS **MQIA_CHANNEL_AUTO_DEF**
Control attribute for automatic channel definition. **MQIA_CHANNEL_AUTO_DEF** is not valid on z/OS.

z/OS **MQIA_CHANNEL_AUTO_DEF_EVENT**
Control attribute for automatic channel definition events. **MQIA_CHANNEL_AUTO_DEF_EVENT** is not valid on z/OS.

MQIA_CHANNEL_EVENT
Control attribute for channel events.

z/OS **MQIA_CHINIT_ADAPTERS**
Number of adapter subtasks to use for processing IBM MQ calls. **MQIA_CHINIT_ADAPTERS** is valid on z/OS only.

MQIA_CHINIT_CONTROL
Start channel initiator automatically when queue manager starts.

z/OS **MQIA_CHINIT_DISPATCHERS**
Number of dispatchers to use for the channel initiator. **MQIA_CHINIT_DISPATCHERS** is valid on z/OS only.

z/OS **MQIA_CHINIT_SERVICE_PARM**
Reserved for use by IBM. **MQIA_CHINIT_SERVICE_PARM** is valid only on z/OS.

z/OS **MQIA_CHINIT_TRACE_AUTO_START**
Specifies whether the channel initiator trace must start automatically. **MQIA_CHINIT_TRACE_AUTO_START** is valid on z/OS only.

z/OS **MQIA_CHINIT_TRACE_TABLE_SIZE**
Size, in megabytes, of the trace data space of the channel initiator. **MQIA_CHINIT_TRACE_TABLE_SIZE** is valid on z/OS only.

MQIA_CHLAUTH_RECORDS
Control attribute for checking of channel authentication records.

MQIA_CLUSTER_WORKLOAD_LENGTH
Maximum length of the message passed to the cluster workload exit.

MQIA_CLWL_MRU_CHANNELS
Cluster workload most recently used channels.

MQIA_CLWL_USEQ
Cluster workload remote queue use.

MQIA_CMD_SERVER_CONTROL
Start command server automatically when queue manager starts.

MQIA_CODED_CHAR_SET_ID
Coded character set identifier.

MQIA_COMMAND_EVENT
Control attribute for command events.

MQIA_COMMAND_LEVEL
Command level supported by queue manager.

MQIA_CONFIGURATION_EVENT
Control attribute for configuration events.

MQIA_CPI_LEVEL
Reserved for use by IBM.

MQIA_DEF_CLUSTER_XMIT_Q_TYPE
Default transmission queue type to be used for cluster-sender channels.

Multi z/OS **MQIA_DIST_LISTS**
Distribution list support. This parameter is not valid on z/OS.

z/OS **MQIA_DNS_WLM**
Specifies whether the TCP listener that handles inbound transmissions for the queue-sharing group must register with Workload Manager (WLM) for DDNS. **MQIA_DNS_WLM** is valid on z/OS only.

z/OS **MQIA_EXPIRY_INTERVAL**
Expiry interval. This parameter is valid on z/OS only.

z/OS **MQIA_GROUP_UR**
Control attribute for whether transactional applications can connect with a GROUP unit of recovery disposition. This parameter is valid only on z/OS.

z/OS **MQIA_IGQ_PUT_AUTHORITY**
Intra-group queuing put authority. This parameter is valid on z/OS only.

MQIA_INHIBIT_EVENT
Control attribute for inhibit events.

z/OS **MQIA_INTRA_GROUP_queuing**
Intra-group queuing support. This parameter is valid on z/OS only.

MQIA_IP_ADDRESS_VERSION
IP address version selector.

z/OS **MQIA_LISTENER_TIMER**
Listener restart interval. **MQIA_LISTENER_TIMER** is valid on z/OS only.

MQIA_LOCAL_EVENT
Control attribute for local events.

MQIA_LOGGER_EVENT
Control attribute for recovery log events.

z/OS **MQIA_LU62_CHANNELS**
Maximum number of LU 6.2 channels. **MQIA_LU62_CHANNELS** is valid on z/OS only.

MQIA_MSG_MARK_BROWSE_INTERVAL
Interval for which messages that were browsed, remain marked.

z/OS **MQIA_MAX_CHANNELS**
Maximum number of channels that can be current. **MQIA_MAX_CHANNELS** is valid on z/OS only.

MQIA_MAX_HANDLES
Maximum number of handles.

MQIA_MAX_MSG_LENGTH
Maximum message length.

MQIA_MAX_PRIORITY
Maximum priority.

MQIA_MAX_PROPERTIES_LENGTH
Maximum properties length.

MQIA_MAX_UNCOMMITTED_MSGS
Maximum number of uncommitted messages within a unit of work.

MQIA_MONITORING_AUTO_CLUSSDR
Default value of the **ChannelMonitoring** attribute of automatically defined cluster-sender channels.

MQIA_MONITORING_CHANNEL
Specifies whether channel monitoring is enabled.

MQIA_MONITORING_Q
Specifies whether queue monitoring is enabled.

z/OS **MQIA_OUTBOUND_PORT_MAX**
Maximum value in the range for the binding of outgoing channels. **MQIA_OUTBOUND_PORT_MAX** is valid on z/OS only.

z/OS **MQIA_OUTBOUND_PORT_MIN**
Minimum value in the range for the binding of outgoing channels. **MQIA_OUTBOUND_PORT_MIN** is valid on z/OS only.

MQIA_PERFORMANCE_EVENT
Control attribute for performance events.

MQIA_PLATFORM
Platform on which the queue manager resides.

MQIA_PUBSUB_CLUSTER
Controls whether this queue manager participates in the publish/subscribe clustering.

MQIA_PUBSUB_MAXMSG_RETRY_COUNT
The number of retries when processing (under sync point) a failed command message

MQIA_PUBSUB_MODE
Inquires if the publish/subscribe engine and the queued publish/subscribe interface are running, which allow applications to publish/subscribe by using the application programming interface and the queues that are being monitored by the queued publish/subscribe interface.

MQIA_PUBSUB_NP_MSG
Specifies whether to discard (or keep) an undelivered input message.

MQIA_PUBSUB_NP_RESP
The behavior of undelivered response messages.

MQIA_PUBSUB_SYNC_PT
Specifies whether only persistent (or all) messages must be processed under sync point.

z/OS **MQIA_QMGR_CFCONLOS**
Specifies action to be taken when the queue manager loses connectivity to the administration structure, or any CF structure with CFCONLOS set to **ASQMGR**. MQIA_QMGR_CFCONLOS is valid on z/OS only.

z/OS **MQIA_RECEIVE_TIMEOUT**
How long a TCP/IP channel waits to receive data from its partner. **MQIA_RECEIVE_TIMEOUT** is valid on z/OS only.

z/OS **MQIA_RECEIVE_TIMEOUT_MIN**
Minimum length of time that a TCP/IP channel waits to receive data from its partner . **MQIA_RECEIVE_TIMEOUT_MIN** is valid on z/OS only.

z/OS **MQIA_RECEIVE_TIMEOUT_TYPE**
Qualifier to apply to the **ReceiveTimeout** parameter. **MQIA_RECEIVE_TIMEOUT_TYPE** is valid on z/OS only.

MQIA_REMOTE_EVENT
Control attribute for remote events.

z/OS **MQIA_SECURITY_CASE**
Specifies whether the queue manager supports security profile names either in mixed case, or in uppercase only. **MQIA_SECURITY_CASE** is valid only on z/OS.

z/OS **MQIA_SHARED_Q_Q_MGR_NAME**
When a queue manager makes an MQOPEN call for a shared queue and the queue manager that is specified in the **ObjectQmgrName** parameter of the MQOPEN call is in the same queue-sharing group as the processing queue manager, the SQQMNAME attribute specifies whether the **ObjectQmgrName** is used or whether the processing queue manager opens the shared queue directly. **MQIA_SHARED_Q_Q_MGR_NAME** is valid only on z/OS.

MQIA_SSL_EVENT
Control attribute for TLS events.

MQIA_SSL_FIPS_REQUIRED
Specifies whether only FIPS-certified algorithms are to be used if cryptography is executed in IBM MQ rather then in the cryptographic hardware itself.

MQIA_SSL_RESET_COUNT
TLS key reset count.

z/OS **MQIA_SSL_TASKS**
TLS tasks. This parameter is valid on z/OS only.

MQIA_START_STOP_EVENT
Control attribute for start stop events.

MQIA_STATISTICS_AUTO_CLUSSDR
Specifies whether statistics data is to be collected for auto-defined cluster-sender channels and, if so, the rate of data collection.

MQIA_STATISTICS_CHANNEL
Specifies whether statistics monitoring data is to be collected for channels and, if so, the rate of data collection.

ULW **MQIA_STATISTICS_INTERVAL**
Statistics data collection interval. **MQIA_STATISTICS_INTERVAL** is valid only on UNIX, Linux, and Windows.

ULW **MQIA_STATISTICS_MQI**
Specifies whether statistics monitoring data is to be collected for the queue manager. **MQIA_STATISTICS_MQI** is valid only on UNIX, Linux, and Windows.

ULW **MQIA_STATISTICS_Q**
Specifies whether statistics monitoring data is to be collected for queues. **MQIA_STATISTICS_Q** is valid only on UNIX, Linux, and Windows.

MQIA_SUITE_B_STRENGTH
Specifies whether Suite B-compliant cryptography is used and the level of strength employed. For more information about Suite B configuration and its effect on TLS channels, see NSA Suite B Cryptography in IBM MQ.

MQIA_SYNCPOINT
Sync point availability.

MQIA_TCP_CHANNELS
Maximum number of channels that can be current, or clients that can be connected, that use the TCP/IP transmission protocol This is valid only on z/OS.

z/OS **MQIA_TCP_KEEP_ALIVE**
Specifies whether the TCP KEEPALIVE facility is to be used to check whether the other end of a connection is still available. **MQIA_TCP_KEEP_ALIVE** is valid only on z/OS.

z/OS **MQIA_TCP_STACK_TYPE**
Specifies whether the channel initiator can use only the TCP/IP address space specified in the **TCPName** parameter, or can optionally bind to any selected TCP/IP address.
MQIA_TCP_STACK_TYPE is valid only on z/OS.

MQIA_TRACE_ROUTE_RECORDING
Specifies whether trace-route information can be recorded and reply messages generated.

MQIA_TREE_LIFE_TIME
The lifetime of non-administrative topics.

MQIA_TRIGGER_INTERVAL
Trigger interval.

MQIA_XR_CAPABILITY
Specifies whether telemetry commands are supported.

MQIACF_Q_MGR_CLUSTER
All clustering attributes. These attributes are:

- **MQCA_CLUSTER_WORKLOAD_DATA**

- **MQCA_CLUSTER_WORKLOAD_EXIT**
- **MQCA_CHANNEL_AUTO_DEF_EXIT**
- **MQCA_REPOSITORY_NAME**
- **MQCA_REPOSITORY_NAMELIST**
- **MQIA_CLUSTER_WORKLOAD_LENGTH**
- **MQIA_CLWL_MRU_CHANNELS**
- **MQIA_CLWL_USEQ**
- **MQIA_MONITORING_AUTO_CLUSSDR**
- **MQCA_Q_MGR_IDENTIFIER**

MQIACF_Q_MGR_DQM
All distributed queuing attributes. These attributes are:

- **MQCA_CERT_LABEL**
- **MQCA_CHANNEL_AUTO_DEF_EXIT**
- **MQCA_CHANNEL_AUTO_DEF_EXIT**
- **MQCA_DEAD_LETTER_Q_NAME**
- **MQCA_DEF_XMIT_Q_NAME**
- **MQCA_DNS_GROUP**
- **MQCA_IGQ_USER_ID**
- **MQCA_LU_GROUP_NAME**
- **MQCA_LU_NAME**
- **MQCA_LU62_ARM_SUFFIX**
- **MQCA_Q_MGR_IDENTIFIER**
- **MQCA_QSG_CERT_LABEL**
- **MQCA_SSL_CRL_NAMELIST**
- **MQCA_SSL_CRYPTO_HARDWARE**
- **MQCA_SSL_KEY_REPOSITORY**
- **MQCA_TCP_NAME**
- **MQIA_ACTIVE_CHANNELS**
- **MQIA_ADOPTNEWMCA_CHECK**
- **MQIA_ADOPTNEWMCA_TYPE**
- **MQIA_CERT_VAL_POLICY**
- **MQIA_CHANNEL_AUTO_DEF**
- **MQIA_CHANNEL_AUTO_DEF_EVENT**
- **MQIA_CHANNEL_EVENT**
- **MQIA_CHINIT_ADAPTERS**
- **MQIA_CHINIT_CONTROL**
- **MQIA_CHINIT_DISPATCHERS**
- **MQIA_CHINIT_SERVICE_PARM**
- **MQIA_CHINIT_TRACE_AUTO_START**
- **MQIA_CHINIT_TRACE_TABLE_SIZE**
- **MQIA_CHLAUTH_RECORDS**
- **MQIA_INTRA_GROUP_queuing**
- **MQIA_IGQ_PUT_AUTHORITY**
- **MQIA_IP_ADDRESS_VERSION**
- **MQIA_LISTENER_TIMER**

- **MQIA_LU62_CHANNELS**
- **MQIA_MAX_CHANNELS**
- **MQIA_MONITORING_CHANNEL**
- **MQIA_OUTBOUND_PORT_MAX**
- **MQIA_OUTBOUND_PORT_MIN**
- **MQIA_RECEIVE_TIMEOUT**
- **MQIA_RECEIVE_TIMEOUT_MIN**
- **MQIA_RECEIVE_TIMEOUT_TYPE**
- **MQIA_SSL_EVENT**
- **MQIA_SSL_FIPS_REQUIRED**
- **MQIA_SSL_RESET_COUNT**
- **MQIA_SSL_TASKS**
- **MQIA_STATISTICS_AUTO_CLUSSDR**
- **MQIA_TCP_CHANNELS**
- **MQIA_TCP_KEEP_ALIVE**
- **MQIA_TCP_STACK_TYPE**

MQIACF_Q_MGR_EVENT
All event control attributes. These attributes are:

- **MQIA_AUTHORITY_EVENT**
- **MQIA_BRIDGE_EVENT**
- **MQIA_CHANNEL_EVENT**
- **MQIA_COMMAND_EVENT**
- **MQIA_CONFIGURATION_EVENT**
- **MQIA_INHIBIT_EVENT**
- **MQIA_LOCAL_EVENT**
- **MQIA_LOGGER_EVENT**
- **MQIA_PERFORMANCE_EVENT**
- **MQIA_REMOTE_EVENT**
- **MQIA_SSL_EVENT**
- **MQIA_START_STOP_EVENT**

MQIACF_Q_MGR_PUBSUB
All queue manager publish/subscribe attributes. These attributes are:

- **MQCA_PARENT**
- **MQIA_PUBSUB_MAXMSG_RETRY_COUNT**
- **MQIA_PUBSUB_MODE**
- **MQIA_PUBSUB_NP_MSG**
- **MQIA_PUBSUB_NP_RESP**
- **MQIA_PUBSUB_SYNC_PT**
- **MQIA_TREE_LIFE_TIME**

MQIACF_Q_MGR_SYSTEM
All queue manager system attributes. These attributes are:

- **MQCA_COMMAND_INPUT_Q_NAME**
- **MQCA_CUSTOM**
- **MQCA_DEAD_LETTER_Q_NAME**
- **MQCA_Q_MGR_NAME**

- **MQCA_QSG_NAME**
- **MQCA_VERSION**
- **MQIA_ACCOUNTING_CONN_OVERRIDE**
- **MQIA_ACCOUNTING_INTERVAL**
- **MQIA_ACCOUNTING_Q**
- **MQIA_ACTIVITY_CONN_OVERRIDE**
- **MQIA_ACTIVITY_RECORDING**
- **MQIA_ACTIVITY_TRACE**
- **MQCA_ALTERATION_DATE**
- **MQCA_ALTERATION_TIME**
- **MQIA_CMD_SERVER_CONTROL**
- **MQIA_CODED_CHAR_SET_ID**
- **MQIA_COMMAND_LEVEL**
- **MQIA_CPI_LEVEL**
- **MQIA_DIST_LISTS**
- **MQIA_EXPIRY_INTERVAL**
- **MQIA_MAX_HANDLES**
- **MQIA_MAX_MSG_LENGTH**
- **MQIA_MAX_PRIORITY**
- **MQIA_MAX_PROPERTIES_LENGTH**
- **MQIA_MAX_UNCOMMITTED_MSGS**
- **MQIA_MONITORING_Q**
- **MQIA_PLATFORM**
- **MQIA_SHARED_Q_Q_MGR_NAME**
- **MQIA_STATISTICS_INTERVAL**
- **MQIA_STATISTICS_MQI**
- **MQIA_STATISTICS_Q**
- **MQIA_SYNCPOINT**
- **MQIA_TRACE_ROUTE_RECORDING**
- **MQIA_TRIGGER_INTERVAL**
- **MQIA_XR_CAPABILITY**

Inquire Queue Manager (Response):

The response to the Inquire Queue Manager (**MQCMD_INQUIRE_Q_MGR**) command consists of the response header followed by the *QMgrName* structure and the requested combination of attribute parameter structures.

Always returned:
QMgrName

Returned if requested:
AccountingConnOverride, *AccountingInterval*, *ActivityConnOverride*, *ActivityRecording*, *ActivityTrace*, *AdoptNewMCACheck*, *AdoptNewMCAType*, *Advancedcapability*, *AlterationDate*, *AlterationTime*, *AuthorityEvent*, z/OS *BridgeEvent*, *CertificateLabel*, *CertificateValPolicy*, z/OS *CFConlos*, *ChannelAutoDef*, *ChannelAutoDefEvent*, *ChannelAutoDefExit*, *ChannelAuthenticationRecords*, *ChannelEvent*, *ChannelInitiatorControl*, *ChannelMonitoring*, *ChannelStatistics*, z/OS *ChinitAdapters*, z/OS *ChinitDispatchers*, z/OS *ChinitServiceParm*, z/OS *ChinitTraceAutoStart*, z/OS *ChinitTraceTableSize*, *ClusterSenderMonitoringDefault*, *ClusterSenderStatistics*, *ClusterWorkloadData*, *ClusterWorkloadExit*, *ClusterWorkloadLength*, *CLWLMRUChannels*, *CLWLUseQ*, *CodedCharSetId*, *CommandEvent*, *CommandInputQName*, *CommandLevel*, *CommandServerControl*, *ConfigurationEvent*, *ConnAuth*, *CreationDate*, *CreationTime*, *Custom*, *DeadLetterQName*, *DefClusterXmitQueueType*, *DefXmitQName*, *DistLists*, *DNSGroup*, z/OS *DNSWLM*, *EncryptionPolicySuiteB*, *ExpiryInterval*, *GroupUR*, z/OS *IGQPutAuthority*, z/OS *IGQUserId*, V 9.0.2 *ImageInterval*, V 9.0.2 *ImagelogLength*, V 9.0.2 *ImageRecoverObject*, V 9.0.2 *ImageRecoverQueue*, V 9.0.2 *ImageSchedule*, *InhibitEvent*, *IntraGroupqueuing*, *IPAddressVersion*, *ListenerTimer*, *LocalEvent*, *LoggerEvent*, z/OS *LUGroupName*, z/OS *LUName*, z/OS *LU62ARMSuffix*, z/OS *LU62Channels*, z/OS *MaxChannels*, z/OS *MaxActiveChannels*, *MaxHandles*, *MaxMsgLength*, *MaxPriority*, *MaxPropertiesLength*, *MaxUncommittedMsgs*, *MQIAccounting*, *MQIStatistics* z/OS *OutboundPortMax*, z/OS *OutboundPortMin*, *Parent*, *PerformanceEvent*, *Platform*, *PubSubClus*, *PubSubMaxMsgRetryCount*, *PubSubMode*, *QmgrDesc*, *QMgrIdentifier*, z/OS *QSGCertificateLabel*, z/OS *QSGName*, *QueueAccounting*, *QueueMonitoring*, *QueueStatistics*, *ReceiveTimeout*, *ReceiveTimeoutMin*, *ReceiveTimeoutType*, *RemoteEvent*, *RepositoryName*, *RepositoryNamelist*, *RevDns*, z/OS *SecurityCase*, *SharedQQmgrName*, *Splcap*, *SSLCRLNamelist*, *SSLCryptoHardware*, *SSLEvent*, *SSLFIPSRequired*, *SSLKeyRepository*, *SSLKeyResetCount*, *SSLTasks*, *StartStopEvent*, *StatisticsInterval*, *SyncPoint*, *TCPChannels*, *TCPKeepAlive*, *TCPName*, *TCPStackType*, *TraceRouteRecording*, *TreeLifeTime*, *TriggerInterval*, *Version*

Response data

AccountingConnOverride (MQCFIN)
Specifies whether applications can override the settings of the *QueueAccounting* and *MQIAccounting* queue manager parameters (parameter identifier: MQIA_ACCOUNTING_CONN_OVERRIDE).

The value can be any of the following values:

MQMON_DISABLED
Applications cannot override the settings of the **QueueAccounting** and **MQIAccounting** parameters.

MQMON_ENABLED
Applications can override the settings of the **QueueAccounting** and **MQIAccounting** parameters by using the options field of the MQCNO structure of the MQCONNX API call.

This parameter applies only to UNIX, Linux, and Windows.

AccountingInterval (MQCFIN)
The time interval, in seconds, at which intermediate accounting records are written (parameter identifier: MQIA_ACCOUNTING_INTERVAL).

It is a value in the range 1 through 604 000.

This parameter applies only to UNIX, Linux, and Windows.

ActivityConnOverride (MQCFIN)
Specifies whether applications can override the setting of the ACTVTRC value in the queue manager attribute (parameter identifier: MQIA_ACTIVITY_CONN_OVERRIDE).

The value can be any of the following values:

MQMON_DISABLED
Applications cannot override the setting of the ACTVTRC queue manager attribute using the Options field in the MQCNO structure on the MQCONNX call. This is the default value.

MQMON_ENABLED
Applications can override the ACTVTRC queue manager attribute using the Options field in the MQCNO structure.

Changes to this value are only effective for connections to the queue manager after the change to the attribute.

This parameter applies only to IBM i, UNIX, and Windows.

ActivityRecording (MQCFIN)
Whether activity reports can be generated (parameter identifier: MQIA_ACTIVITY_RECORDING).

The value can be:

MQRECORDING_DISABLED
Activity reports cannot be generated.

MQRECORDING_MSG
Activity reports can be generated and sent to the destination specified by the originator of the message causing the report.

MQRECORDING_Q
Activity reports can be generated and sent to SYSTEM.ADMIN.ACTIVITY.QUEUE.

ActivityTrace (MQCFIN)
Whether activity reports can be generated (parameter identifier: MQIA_ACTIVITY_TRACE).

The value can be:

MQMON_OFF
Do not collect IBM MQ MQI application activity trace. This is the default value.

If you set the queue manager attribute ACTVCONO to ENABLED, this value might be overridden for individual connections using the Options field in the MQCNO structure.

MQMON_ON
Collect IBM MQ MQI application activity trace.

Changes to this value are only effective for connections to the queue manager after the change to the attribute.

This parameter applies only to IBM i, UNIX, and Windows.

z/OS **AdoptNewMCACheck (MQCFIN)**
The elements checked to determine whether an MCA must be adopted (restarted) when a new inbound channel is detected. It is adopted if it has the same name as a currently active MCA (parameter identifier: MQIA_ADOPTNEWMCA_CHECK).

The value can be:

MQADOPT_CHECK_Q_MGR_NAME
Check the queue manager name.

MQADOPT_CHECK_NET_ADDR
Check the network address.

MQADOPT_CHECK_ALL
Check the queue manager name and network address.

MQADOPT_CHECK_NONE
Do not check any elements.

This parameter is valid only on z/OS.

z/OS **AdoptNewMCAType (MQCFIL)**
Adoption of orphaned channel instances (parameter identifier: MQIA_ADOPTNEWMCA_TYPE).

The value can be:

MQADOPT_TYPE_NO
Do not adopt orphaned channel instances.

MQADOPT_TYPE_ALL
Adopt all channel types.

This parameter is valid only on z/OS.

MQ Adv. **AdvancedCapability (MQCFIN)**
Whether IBM MQ Advanced extended capabilities are available for a queue manager. (parameter identifier: MQIA_ADVANCED_CAPABILITY).

z/OS V 9.0.4 On z/OS, the queue manager sets the value to be MQCAP_SUPPORTED, only if the value of **QMGRPROD** is ADVANCEDVUE. For any other value of **QMGRPROD**, or if **QMGRPROD** is not set, the queue manager sets the value to MQCAP_NOTSUPPORTED. See "START QMGR on z/OS" on page 1037 for more information.

Multi V 9.0.5 On other platforms, from IBM MQ Version 9.0.5, the queue manager sets the value to be MQCAP_SUPPORTED, only if you have installed Managed File Transfer, XR, or Advanced Message Security. If you have not installed Managed File Transfer, XR, or Advanced Message Security, **AdvancedCapability** is set to MQCAP_NOTSUPPORTED. See IBM MQ components and features for more information.

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date, in the form yyyy-mm-dd, on which the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time, in the form hh.mm.ss, at which the information was last altered.

AuthorityEvent (MQCFIN)
Controls whether authorization (Not Authorized) events are generated (parameter identifier: MQIA_AUTHORITY_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

z/OS **BridgeEvent (MQCFIN)**
Controls whether IMS bridge events are generated (parameter identifier: MQIA_BRIDGE_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

This parameter is valid only on z/OS.

CertificateLabel (MQCFST)
Certificate label for this queue manager to use. The label identifies which personal certificate in the key repository has been selected.

The maximum length of the string is MQ_CERT_LABEL_LENGTH.

CertificateValPolicy (MQCFIN)
Specifies which TLS certificate validation policy is used to validate digital certificates received from remote partner systems (parameter identifier: MQIA_CERT_VAL_POLICY).

This attribute can be used to control how strictly the certificate chain validation conforms to industry security standards. This parameter is valid only on UNIX, Linux, and Windows. For more information, see Certificate validation policies in IBM MQ.

The value can be any of the following values:

MQ_CERT_VAL_POLICY_ANY
Apply each of the certificate validation policies supported by the secure sockets library and accept the certificate chain if any of the policies considers the certificate chain valid. This setting can be used for maximum backwards compatibility with older digital certificates which do not comply with the modern certificate standards.

MQ_CERT_VAL_POLICY_RFC5280
Apply only the RFC 5280 compliant certificate validation policy. This setting provides stricter validation than the ANY setting, but rejects some older digital certificates.

z/OS **CFConlos (MQCFIN)**
Specifies the action to be taken when the queue manager loses connectivity to the administration structure, or any CF structures with CFCONLOS set to ASQMGR (parameter identifier: MQIA_QMGR_CFCONLOS).

The value can be:

MQCFCONLOS_TERMINATE
The queue manager terminates when connectivity to CF structures is lost.

MQCFCONLOS_TOLERATE
The queue manager tolerates loss of connectivity to CF structures without terminating.

This parameter is valid only on z/OS.

ChannelAutoDef (MQCFIN)
Controls whether receiver and server-connection channels can be auto-defined (parameter identifier: MQIA_CHANNEL_AUTO_DEF).

The value can be:

MQCHAD_DISABLED
Channel auto-definition disabled.

MQCHAD_ENABLED
Channel auto-definition enabled.

ChannelAutoDefEvent (MQCFIN)
Controls whether channel auto-definition events are generated (parameter identifier: MQIA_CHANNEL_AUTO_DEF_EVENT), when a receiver, server-connection, or cluster-sender channel is auto-defined.

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

ChannelAutoDefExit (MQCFST)
Channel auto-definition exit name (parameter identifier: MQCA_CHANNEL_AUTO_DEF_EXIT).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

ChannelAuthenticationRecords (MQCFIN)
Controls whether channel authentication records are checked (parameter identifier: MQIA_CHLAUTH_RECORDS).

The value can be:

MQCHLA_DISABLED
Channel authentication records are not checked.

MQCHLA_ENABLED
Channel authentication records are checked.

ChannelEvent (MQCFIN)
Controls whether channel events are generated (parameter identifier: MQIA_CHANNEL_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

MQEVR_EXCEPTION
Reporting of exception channel events enabled.

ChannelInitiatorControl (MQCFIN)
Start the channel initiator during queue manager start (parameter identifier: MQIA_CHINIT_CONTROL). This parameter is not available on z/OS.

The value can be:

MQSVC_CONTROL_MANUAL
The channel initiator is not to be started automatically when the queue manager starts.

MQSVC_CONTROL_Q_MGR
The channel initiator is to be started automatically when the queue manager starts.

ChannelMonitoring (MQCFIN)
Default setting for online monitoring for channels (parameter identifier: MQIA_MONITORING_CHANNEL).

If the *ChannelMonitoring* channel attribute is set to MQMON_Q_MGR, this attribute specifies the value which is assumed by the channel. The value can be any of the following values:

MQMON_OFF
Online monitoring data collection is turned off.

MQMON_NONE
Online monitoring data collection is turned off for channels regardless of the setting of their **ChannelMonitoring** attribute.

MQMON_LOW
Online monitoring data collection is turned on, with a low ratio of data collection.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate ratio of data collection.

MQMON_HIGH
Online monitoring data collection is turned on, with a high ratio of data collection.

z/OS **ChannelStatistics (MQCFIN)**
Specifies whether statistics data is to be collected for channels (parameter identifier: MQIA_STATISTICS_CHANNEL).

The value can be:

MQMON_OFF
Statistics data collection is turned off.

MQMON_LOW
Statistics data collection is turned on, with a low ratio of data collection.

MQMON_MEDIUM
Statistics data collection is turned on, with a moderate ratio of data collection.

MQMON_HIGH
Statistics data collection is turned on, with a high ratio of data collection.

This parameter is valid only on z/OS.

z/OS **ChinitAdapters (MQCFIN)**
Number of adapter subtasks (parameter identifier: MQIA_CHINIT_ADAPTERS).

The number of adapter subtasks to use for processing IBM MQ calls. This parameter is valid only on z/OS.

z/OS **ChinitDispatchers (MQCFIN)**
Number of dispatchers (parameter identifier: MQIA_CHINIT_DISPATCHERS).

The number of dispatchers to use for the channel initiator. This parameter is valid only on z/OS.

z/OS **ChinitServiceParm (MQCFST)**
Reserved for use by IBM (parameter identifier: MQCA_CHINIT_SERVICE_PARM).

z/OS **ChinitTraceAutoStart (MQCFIN)**
Specifies whether the channel initiator trace must start automatically (parameter identifier: MQIA_CHINIT_TRACE_AUTO_START).

The value can be:

MQTRAXSTR_YES
Channel initiator trace is to start automatically.

MQTRAXSTR_NO
Channel initiator trace is not to start automatically.

This parameter is valid only on z/OS.

z/OS **ChinitTraceTableSize (MQCFIN)**
The size, in megabytes, of the trace data space of the channel initiator (parameter identifier: MQIA_CHINIT_TRACE_TABLE_SIZE).

This parameter is valid only on z/OS.

ClusterSenderMonitoringDefault (MQCFIN)
Setting for online monitoring for automatically defined cluster-sender channels (parameter identifier: MQIA_MONITORING_AUTO_CLUSSDR).

The value can be:

MQMON_Q_MGR
Collection of online monitoring data is inherited from the setting of the queue manager's **ChannelMonitoring** parameter.

MQMON_OFF
Monitoring for the channel is disabled.

MQMON_LOW
Specifies a low rate of data collection with a minimal effect on system performance unless **ChannelMonitoring** for the queue manager is MQMON_NONE. The data collected is not likely to be the most current.

MQMON_MEDIUM
Specifies a moderate rate of data collection with limited effect on system performance unless **ChannelMonitoring** for the queue manager is MQMON_NONE.

MQMON_HIGH
Specifies a high rate of data collection with a likely effect on system performance unless **ChannelMonitoring** for the queue manager is MQMON_NONE. The data collected is the most current available.

z/OS On z/OS systems, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying LOW, MEDIUM, or HIGH makes no difference to your results.

ClusterSenderStatistics (MQCFIN)
Specifies whether statistics data is to be collected for auto-defined cluster-sender channels (parameter identifier: MQIA_STATISTICS_AUTO_CLUSSDR).

The value can be:

MQMON_Q_MGR
Collection of statistics data is inherited from the setting of the queue manager's **ChannelStatistics** parameter.

MQMON_OFF
Statistics data collection for the channel is disabled.

MQMON_LOW
Specifies a low rate of data collection with a minimal effect on system performance.

MQMON_MEDIUM
Specifies a moderate rate of data collection.

MQMON_HIGH
Specifies a high rate of data collection.

z/OS On z/OS systems, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying LOW, MEDIUM, or HIGH makes no difference to your results.

ClusterWorkLoadData (MQCFST)
Data passed to the cluster workload exit (parameter identifier: MQCA_CLUSTER_WORKLOAD_DATA).

ClusterWorkLoadExit (MQCFST)
Name of the cluster workload exit (parameter identifier: MQCA_CLUSTER_WORKLOAD_EXIT).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

ClusterWorkLoadLength (MQCFIN)
Cluster workload length (parameter identifier: MQIA_CLUSTER_WORKLOAD_LENGTH).

The maximum length of the message passed to the cluster workload exit.

CLWLMRUChannels (MQCFIN)
Cluster workload most recently used (MRU) channels (parameter identifier: MQIA_CLWL_MRU_CHANNELS).

The maximum number of active most recently used outbound channels.

CLWLUseQ (MQCFIN)
Use of remote queue (parameter identifier: MQIA_CLWL_USEQ).

Specifies whether a cluster queue manager is to use remote puts to other queues defined in other queue managers within the cluster during workload management.

The value can be any of the following values:

MQCLWL_USEQ_ANY
Use remote queues.

MQCLWL_USEQ_LOCAL
Do not use remote queues.

CodedCharSetId (MQCFIN)
Coded character set identifier (parameter identifier: MQIA_CODED_CHAR_SET_ID).

CommandEvent (MQCFIN)
Controls whether command events are generated (parameter identifier: MQIA_COMMAND_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

MQEVR_NODISPLAY
Event reporting enabled for all successful commands except Inquire commands.

CommandInputQName (MQCFST)
Command input queue name (parameter identifier: MQCA_COMMAND_INPUT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

CommandLevel (MQCFIN)
Command level supported by queue manager (parameter identifier: MQIA_COMMAND_LEVEL).

The value can be:

MQCMDL_LEVEL_1
Level 1 of system control commands.

This value is returned by the following platforms:

- MQSeries for AIX V2.2
- MQSeries for OS/400:
  - V2R3
  - V3R1
  - V3R6
- MQSeries for Windows V2.0

MQCMDL_LEVEL_101
MQSeries for Windows V2.0.1

MQCMDL_LEVEL_110
MQSeries for Windows V2.1

MQCMDL_LEVEL_200
MQSeries for Windows NT V2.0

MQCMDL_LEVEL_220
Level 220 of system control commands.

This value is returned by the following platforms:

- MQSeries for AT&T GIS UNIX V2.2
- MQSeries for SINIX and DC/OSx V2.2
- MQSeries for Compaq NonStop Kernel V2.2.0.1

MQCMDL_LEVEL_221
Level 221 of system control commands.

This value is returned by the following platforms:

- MQSeries for AIX Version 2.2.1
- MQSeries for DIGITAL UNIX (Compaq Tru64 UNIX) V2.2.1

MQCMDL_LEVEL_320
MQSeries for OS/400 V3R2 and V3R7

MQCMDL_LEVEL_420
MQSeries for AS/400 V4R2 and R2.1

MQCMDL_LEVEL_500
Level 500 of system control commands.

This value is returned by the following platforms:

- MQSeries for AIX V5.0
- MQSeries for HP-UX V5.0
- MQSeries for Solaris V5.0
- MQSeries for Windows NT V5.0

MQCMDL_LEVEL_510
Level 510 of system control commands.

This value is returned by the following platforms:

- MQSeries for AIX V5.1
- MQSeries for AS/400 V5.1
- MQSeries for HP-UX V5.1
- MQSeries for Compaq Tru64 UNIX, V5.1
- IBM WebSphere MQ for HP Integrity NonStop Server V5.3
- MQSeries for Solaris V5.1
- MQSeries for Windows NT V5.1

MQCMDL_LEVEL_520
Level 520 of system control commands.

This value is returned by the following platforms:

- MQSeries for AIX V5.2
- MQSeries for AS/400 V5.2
- MQSeries for HP-UX V5.2

- MQSeries for Linux V5.2
- MQSeries for Solaris V5.2
- MQSeries for Windows NT V5.2
- MQSeries for Windows 2000 V5.2

MQCMDL_LEVEL_530
Level 530 of system control commands.

This value is returned by the following platforms:
- IBM WebSphere MQ for AIX, V5.3
- IBM WebSphere MQ for IBM i, V5.3
- IBM WebSphere MQ for HP-UX, V5.3
- IBM WebSphere MQ for Linux, V5.3
- IBM WebSphere MQ for Sun Solaris, Version 5.3
- IBM WebSphere MQ for Windows NT and Windows 2000, Version 5.3

MQCMDL_LEVEL_531
Level 531 of system control commands.

MQCMDL_LEVEL_600
Level 600 of system control commands.

MQCMDL_LEVEL_700
Level 700 of system control commands.

MQCMDL_LEVEL_701
Level 701 of system control commands.

MQCMDL_LEVEL_710
Level 710 of system control commands.

MQCMDL_LEVEL_750
Level 750 of system control commands.

This value is returned by the following platforms:
- IBM WebSphere MQ for AIX, V7.5
- IBM WebSphere MQ for HP-UX, V7.5
- IBM WebSphere MQ for Linux, V7.5
- IBM WebSphere MQ for Solaris, V7.5
- IBM WebSphere MQ for Windows V7.5

MQCMDL_LEVEL_800
Level 800 of system control commands.

MQCMDL_LEVEL_801
Level 801 of system control commands.

MQCMDL_LEVEL_900
Level 900 of system control commands.

MQCMDL_LEVEL_901
Level 901 of system control commands.

MQCMDL_LEVEL_901
Level 902 of system control commands.

The set of system control commands that corresponds to a particular value of the **CommandLevel** attribute varies. It varies according to the value of the **Platform** attribute; both must be used to decide which system control commands are supported.

CommandServerControl (MQCFIN)
Start the command server during queue manager start (parameter identifier: MQIA_CMD_SERVER_CONTROL). This parameter is not available on z/OS.

The value can be:

MQSVC_CONTROL_MANUAL
The command server is not to be started automatically when the queue manager starts.

MQSVC_CONTROL_Q_MGR
The command server is to be started automatically when the queue manager starts.

ConfigurationEvent (MQCFIN)
Controls whether configuration events are generated (parameter identifier: MQIA_CONFIGURATION_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

ConnAuth (MQCFST)
Name of the authentication information object that is used to provide the location of user ID and password authentication (parameter identifier: MQCA_CONN_AUTH).

CreationDate (MQCFST)

Queue creation date, in the form yyyy-mm-dd (parameter identifier: MQCA_CREATION_DATE). The maximum length of the string is MQ_CREATION_DATE_LENGTH.

CreationTime (MQCFST)

Creation time, in the form hh.mm.ss (parameter identifier: MQCA_CREATION_TIME). The maximum length of the string is MQ_CREATION_TIME_LENGTH.

Custom (MQCFST)
Custom attribute for new features (parameter identifier: MQCA_CUSTOM).

This attribute is reserved for the configuration of new features before separate attributes are introduced. It can contain the values of zero or more attributes as pairs of attribute name and value, separated by at least one space. The attribute name-value pairs have the form NAME(VALUE).

This description is updated when features using this attribute are introduced.

DeadLetterQName (MQCFST)
Dead letter (undelivered message) queue name (parameter identifier: MQCA_DEAD_LETTER_Q_NAME).

Specifies the name of the local queue that is to be used for undelivered messages. Messages are put on this queue if they cannot be routed to their correct destination.

The maximum length of the string is MQ_Q_NAME_LENGTH.

DefClusterXmitQueueType (MQCFIN)
The DefClusterXmitQueueType attribute controls which transmission queue is selected by default by cluster-sender channels to get messages from, to send the messages to cluster-receiver channels. (Parameter identifier: MQIA_DEF_CLUSTER_XMIT_Q_TYPE.)

The values of DefClusterXmitQueueType are MQCLXQ_SCTQ or MQCLXQ_CHANNEL.

MQCLXQ_SCTQ

All cluster-sender channels send messages from SYSTEM.CLUSTER.TRANSMIT.QUEUE. The correlID of messages placed on the transmission queue identifies which cluster-sender channel the message is destined for.

SCTQ is set when a queue manager is defined. This behavior is implicit in versions of IBM WebSphere MQ, earlier than Version 7.5. In earlier versions, the queue manager attribute DefClusterXmitQueueType was not present.

MQCLXQ_CHANNEL
Each cluster-sender channel sends messages from a different transmission queue. Each transmission queue is created as a permanent dynamic queue from the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE.

DefXmitQName (MQCFST)
Default transmission queue name (parameter identifier: MQCA_DEF_XMIT_Q_NAME).

The default transmission queue is used for the transmission of messages to remote queue managers. It is used if there is no other indication of which transmission queue to use.

The maximum length of the string is MQ_Q_NAME_LENGTH.

DistLists (MQCFIN)
Distribution list support (parameter identifier: MQIA_DIST_LISTS).

The value can be:

MQDL_SUPPORTED
Distribution lists supported.

MQDL_NOT_SUPPORTED
Distribution lists not supported.

z/OS **DNSGroup (MQCFST)**
DNS group name (parameter identifier: MQCA_DNS_GROUP).

This parameter is no longer used. Refer to WLM/DNS no longer supported.

This parameter is valid only on z/OS.

z/OS **DNSWLM (MQCFIN)**
WLM/DNS Control: (parameter identifier: MQIA_DNS_WLM).

This parameter is no longer used. Refer to WLM/DNS no longer supported.

The value can be any of the following values:

MQDNSWLM_NO
MQDNSWLM_NO is the only value supported by the queue manager.

This parameter is valid only on z/OS.

EncryptionPolicySuiteB (MQCFIL)
Specifies whether Suite B-compliant cryptography is used and what level of strength is employed (parameter identifier: MQIA_SUITE_B_STRENGTH). For more information about Suite B configuration and its effect on TLS channels, see NSA Suite B Cryptography in IBM MQ.

The value can be one, or more, of:

MQ_SUITE_B_NONE
Suite B-compliant cryptography is not used.

MQ_SUITE_B_128_BIT
Suite B 128-bit strength security is used.

MQ_SUITE_B_192_BIT
Suite B 192-bit strength security is used.

MQ_SUITE_B_128_BIT,MQ_SUITE_B_192_BIT
Suite B 128-bit and Suite B 192-bit strength security is used.

z/OS **ExpiryInterval (MQCFIN)**
Interval between scans for expired messages (parameter identifier: MQIA_EXPIRY_INTERVAL).

Specifies the frequency with which the queue manager scans the queues looking for expired messages. This parameter is a time interval in seconds in the range 1 through 99 999 999, or the following special value:

MQEXPI_OFF
No scans for expired messages.

This parameter is valid only on z/OS.

z/OS **GroupUR (MQCFIN)**
Identifies whether XA client applications can establish transactions with a GROUP unit of recovery disposition.

The value can be:

MQGUR_DISABLED
XA client applications must connect using a queue manager name.

MQGUR_ENABLED
XA client applications can establish transactions with a group unit of recovery disposition by specifying a queue-sharing group name when they connect.

This parameter is valid only on z/OS.

z/OS **IGQPutAuthority (MQCFIN)**
Type of authority checking used by the intra-group queuing agent (parameter identifier: MQIA_IGQ_PUT_AUTHORITY).

The attribute indicates the type of authority checking that is performed by the local intra-group queuing agent (IGQ agent). The checking is performed when the IGQ agent removes a message from the shared transmission queue and places the message on a local queue. The value can be any of the following values:

MQIGQPA_DEFAULT
Default user identifier is used.

MQIGQPA_CONTEXT
Context user identifier is used.

MQIGQPA_ONLY_IGQ
Only the IGQ user identifier is used.

MQIGQPA_ALTERNATE_OR_IGQ
Alternate user identifier or IGQ-agent user identifier is used.

This parameter is valid only on z/OS.

z/OS **IGQUserId (MQCFST)**
User identifier used by the intra-group queuing agent (parameter identifier: MQCA_IGQ_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH. This parameter is valid only on z/OS.

V 9.0.2 **ImageInterval (MQCFIN)**
The target frequency with which the queue manager automatically writes media images (parameter identifier: MQIA_MEDIA_IMAGE_INTERVAL). This parameter is not valid on z/OS.

The value can be:

The time interval, at which the queue manager automatically writes media images.

MQMEDIMGINTVL_OFF
Automatic media images are not written on a time interval basis.

V 9.0.2 **ImageLogLength (MQCFIN)**
The target size of the recovery log (parameter identifier: MQIA_MEDIA_IMAGE_LOG_LENGTH). This parameter is not valid on z/OS.

The value can be:

The size of the recovery log.

MQMEDIMGLOGLN_OFF
Automatic media images are not written.

V 9.0.2 **ImageRecoverObject (MQCFST)**
Specifies the recoverable objects from a media image, if linear logging is being used (parameter identifier: MQIA_MEDIA_IMAGE_RECOVER_OBJ). This parameter is not valid on z/OS.

The value can be:

MQIMGRCOV_NO
Automatic media images, if enabled, are not written for these objects.

MQIMGRCOV_YES
These objects are recoverable.

V 9.0.2 **ImageRecoverQueue (MQCFST)**
Displays the default **ImageRecoverQueue** attribute for local and permanent dynamic queue objects, when used with this parameter (parameter identifier: MQIA_MEDIA_IMAGE_RECOVER_Q). This parameter is not valid on z/OS.

The value can be:

MQIMGRCOV_NO
The **ImageRecoverQueue** attribute for local and permanent dynamic queue objects is set to MQIMGRCOV_NO .

MQIMGRCOV_YES
The **ImageRecoverQueue** attribute for local and permanent dynamic queue objects is set to MQIMGRCOV_YES .

V 9.0.2 **ImageSchedule (MQCFST)**
Whether the queue manager automatically writes media images (parameter identifier: MQIA_MEDIA_IMAGE_SCHEDUING). This parameter is not valid on z/OS.

The value can be:

MQMEDIMGSCHED_AUTO
The queue manager automatically writes a media image for an object.

MQMEDIMGSCHED_MANUAL
Automatic media images are not written.

InhibitEvent (MQCFIN)
Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated (parameter identifier: MQIA_INHIBIT_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

z/OS **IntraGroupqueuing (MQCFIN)**
Specifies whether intra-group queuing is used (parameter identifier: MQIA_INTRA_GROUP_queuing).

The value can be:

MQIGQ_DISABLED
Intra-group queuing is disabled. All messages destined for other queue managers in the queue-sharing group are transmitted using conventional channels.

MQIGQ_ENABLED
Intra-group queuing is enabled.

This parameter is valid only on z/OS.

IPAddressVersion (MQCFIN)
IP address version selector (parameter identifier: MQIA_IP_ADDRESS_VERSION).

Specifies which IP address version, either IPv4 or IPv6, is used. The value can be:

MQIPADDR_IPv4
IPv4 is used.

MQIPADDR_IPv6
IPv6 is used.

ListenerTimer (MQCFIN)
Listener restart interval (parameter identifier: MQIA_LISTENER_TIMER).

The time interval, in seconds, between attempts by IBM MQ to restart the listener after an APPC or TCP/IP failure.

z/OS **LocalEvent (MQCFIN)**
Controls whether local error events are generated (parameter identifier: MQIA_LOCAL_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

This parameter is valid only on z/OS.

LoggerEvent (MQCFIN)
Controls whether recovery log events are generated (parameter identifier: MQIA_LOGGER_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

This parameter applies only to UNIX, Linux, and Windows.

z/OS **LUGroupName (MQCFST)**
Generic LU name for the LU 6.2 listener (parameter identifier: MQCA_LU_GROUP_NAME).

The generic LU name to be used by the LU 6.2 listener that handles inbound transmissions for the queue-sharing group. This parameter is valid only on z/OS.

z/OS **LUName (MQCFST)**
LU name to use for outbound LU 6.2 transmissions (parameter identifier: MQCA_LU_NAME).

The name of the LU to use for outbound LU 6.2 transmissions. This parameter is valid only on z/OS.

z/OS **LU62ARMSuffix (MQCFST)**
APPCPM suffix (parameter identifier: MQCA_LU62_ARM_SUFFIX).

The suffix of the APPCPM member of SYS1.PARMLIB. This suffix nominates the LUADD for this channel initiator. This parameter is valid only on z/OS.

z/OS **LU62Channels (MQCFIN)**
Maximum number of LU 6.2 channels (parameter identifier: MQIA_LU62_CHANNELS).

The maximum number of channels that can be current, or clients that can be connected, that use the LU 6.2 transmission protocol. This parameter is valid only on z/OS.

z/OS **MaxActiveChannels (MQCFIN)**
Maximum number of channels (parameter identifier: MQIA_ACTIVE_CHANNELS).

The maximum number of channels that can be active at any time. This parameter is valid only on z/OS.

z/OS **MaxChannels (MQCFIN)**
Maximum number of current channels (parameter identifier: MQIA_MAX_CHANNELS).

The maximum number of channels that can be current (including server-connection channels with connected clients). This parameter is valid only on z/OS.

MaxHandles (MQCFIN)
Maximum number of handles (parameter identifier: MQIA_MAX_HANDLES).

Specifies the maximum number of handles that any one connection can have open at the same time.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

MaxPriority (MQCFIN)
Maximum priority (parameter identifier: MQIA_MAX_PRIORITY).

MaxPropertiesLength (MQCFIN)
Maximum properties length (parameter identifier: MQIA_MAX_PROPERTIES_LENGTH).

MaxUncommittedMsgs (MQCFIN)
Maximum number of uncommitted messages within a unit of work (parameter identifier: MQIA_MAX_UNCOMMITTED_MSGS).

This number is the sum of the following number of messages under any one sync point. :

- The number of messages that can be retrieved, plus
- The number of messages that can be put on a queue, plus
- Any trigger messages generated within this unit of work

The limit does not apply to messages that are retrieved or put outside sync point.

MQIAccounting (MQCFIN)
Specifies whether accounting information for MQI data is to be collected (parameter identifier: MQIA_ACCOUNTING_MQI).

The value can be:

MQMON_OFF
MQI accounting data collection is disabled.

MQMON_ON
MQI accounting data collection is enabled.

This parameter applies only to UNIX, Linux, and Windows.

MQIStatistics (MQCFIN)
Specifies whether statistics monitoring data is to be collected for the queue manager (parameter identifier: MQIA_STATISTICS_MQI).

The value can be:

MQMON_OFF
Data collection for MQI statistics is disabled. MQMON_OFF is the initial default value of the queue manager.

MQMON_ON
Data collection for MQI statistics is enabled.

This parameter applies only to UNIX, Linux, and Windows.

MsgMarkBrowseInterval (MQCFIN)
Mark-browse interval (parameter identifier: MQIA_MSG_MARK_BROWSE_INTERVAL).

The time interval in milliseconds after which the queue manager can automatically unmark messages.

z/OS **OutboundPortMax (MQCFIN)**
The maximum value in the range for the binding of outgoing channels (parameter identifier: MQIA_OUTBOUND_PORT_MAX).

The maximum value in the range of port numbers to be used when binding outgoing channels. This parameter is valid only on z/OS.

z/OS **OutboundPortMin (MQCFIN)**
The minimum value in the range for the binding of outgoing channels (parameter identifier: MQIA_OUTBOUND_PORT_MIN).

The minimum value in the range of port numbers to be used when binding outgoing channels. This parameter is valid only on z/OS.

Parent (MQCFST)
The name of the hierarchically connected queue manager nominated as the parent of this queue manager (parameter identifier: MQCA_PARENT).

PerformanceEvent (MQCFIN)
Controls whether performance-related events are generated (parameter identifier: MQIA_PERFORMANCE_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

Platform (MQCFIN)
Platform on which the queue manager resides (parameter identifier: MQIA_PLATFORM).

The value can be:

MQPL_AIX
AIX (same value as MQPL_UNIX).

MQPL_APPLIANCE
IBM MQ Appliance

MQPL_NSK
HP Integrity NonStop Server.

MQPL_OS400
IBM i.

MQPL_UNIX
UNIX.

MQPL_WINDOWS_NT
Windows.

MQPL_ZOS
z/OS

PubSubClus (MQCFIN)
Controls whether the queue manager participates in publish/subscribe clustering (parameter identifier: MQIA_PUBSUB_CLUSTER).

The value can be:

MQPSCLUS_ENABLED
The creating or receipt of clustered topic definitions and cluster subscriptions is permitted.

Note: The introduction of a clustered topic into a large IBM MQ cluster can cause a degradation in performance. This degradation occurs because all partial repositories are notified of all the other members of the cluster. Unexpected subscriptions might be created at all other nodes; for example, where proxysub(FORCE) is specified. Large numbers of channels might be started from a queue manager; for example, on resync after a queue manager failure.

MQPSCLUS_DISABLED
The creating or receipt of clustered topic definitions and cluster subscriptions is inhibited. The creations or receipts are recorded as warnings in the queue manager error logs.

PubSubMaxMsgRetryCount (MQCFIN)
The number of attempts to reprocess a failed command message under sync point (parameter identifier: MQIA_PUBSUB_MAXMSG_RETRY_COUNT).

PubSubMode (MQCFIN)
Specifies whether the publish/subscribe engine and the queued publish/subscribe interface are running. The publish/subscribe engine enables applications to publish or subscribe by using the application programming interface. The publish/subscribe interface monitors the queues used the queued publish/subscribe interface (parameter identifier: MQIA_PUBSUB_MODE).

The values can be as follows:

MQPSM_COMPAT
The publish/subscribe engine is running. It is therefore possible to publish or subscribe by using the application programming interface. The queued publish/subscribe interface is not running. Therefore any message that is put to the queues that are monitored by the queued publish/subscribe interface is not acted on. MQPSM_COMPAT is used for compatibility with versions of IBM Integration Bus, (formerly known as WebSphere Message Broker) prior to version 7 that use this queue manager.

MQPSM_DISABLED
The publish/subscribe engine and the queued publish/subscribe interface are not running. It is therefore not possible to publish or subscribe by using the application programming interface. Any publish/subscribe messages that are put to the queues that are monitored by the queued publish/subscribe interface are not acted on.

MQPSM_ENABLED
The publish/subscribe engine and the queued publish/subscribe interface are running. It is therefore possible to publish or subscribe by using the application programming interface and the queues that are being monitored by the queued publish/subscribe interface. MQPSM_ENABLED is the initial default value of the queue manager.

PubSubNPInputMsg (MQCFIN)
Specifies whether to discard or keep an undelivered input message (parameter identifier: MQIA_PUBSUB_NP_MSG).

The values can be as follows:

MQUNDELIVERED_DISCARD
Non-persistent input messages can be discarded if they cannot be processed. MQUNDELIVERED_DISCARD is the default value.

MQUNDELIVERED_KEEP
Non-persistent input messages are not discarded if they cannot be processed. The queued publish/subscribe interface continues to try the process again at appropriate intervals. It does not continue processing subsequent messages.

PubSubNPResponse (MQCFIN)
Controls the behavior of undelivered response messages (parameter identifier: MQIA_PUBSUB_NP_RESP).

The values can be as follows:

MQUNDELIVERED_NORMAL
Non-persistent responses that cannot be placed on the reply queue are put on the dead letter queue. If they cannot be placed on the dead letter queue, they are discarded.

MQUNDELIVERED_SAFE
Non-persistent responses that cannot be placed on the reply queue are put on the dead letter queue. If the response cannot be sent and cannot be placed on the dead letter queue the queued publish/subscribe interface rolls back the current operation. The operation is tried again at appropriate intervals and does not continue processing subsequent messages.

MQUNDELIVERED_DISCARD
Non-persistent responses that cannot be placed on the reply queue are discarded. MQUNDELIVERED_DISCARD is the default value for new queue managers.

MQUNDELIVERED_KEEP
Non-persistent responses are not placed on the dead letter queue or discarded. Instead, the queued publish/subscribe interface backs out the current operation and then tries it again at appropriate intervals.

PubSubSyncPoint (MQCFIN)
Specifies whether only persistent messages or all messages are processed under sync point (parameter identifier: MQIA_PUBSUB_SYNC_PT).

The values can be as follows:

MQSYNCPOINT_IFPER
This makes the queued publish/subscribe interface receive non-persistent messages outside sync point. If the daemon receives a publication outside sync point, the daemon forwards the publication to subscribers known to it outside sync point. MQSYNCPOINT_IFPER is the default value.

MQSYNCPOINT_YES
MQSYNCPOINT_YES makes the queued publish/subscribe interface receive all messages under sync point.

QMgrDesc (MQCFST)
Queue manager description (parameter identifier: MQCA_Q_MGR_DESC).

This parameter is text that briefly describes the object.

The maximum length of the string is MQ_Q_MGR_DESC_LENGTH.

Use characters from the character set identified by the coded character set identifier (CCSID) for the queue manager on which the command is executing. Using this character set ensures that the text is translated correctly.

QMgrIdentifier (MQCFST)
Queue manager identifier (parameter identifier: MQCA_Q_MGR_IDENTIFIER).

The unique identifier of the queue manager.

QMgrName (MQCFST)
Name of local queue manager (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

z/OS **QSGCertificateLabel (MQCFST)**
Certificate label for this queue-sharing group to use. The label identifies which personal certificate in the key repository has been selected.

The maximum length of the string is MQ_QSG_CERT_LABEL_LENGTH. This parameter is valid only on z/OS.

z/OS **QSGName (MQCFST)**
Queue-sharing group name (parameter identifier: MQCA_QSG_NAME).

The maximum length of the string is MQ_QSG_NAME_LENGTH. This parameter is valid only on z/OS.

QueueAccounting (MQCFIN)
Collection of accounting (thread-level and queue-level accounting) data for queues (parameter identifier: MQIA_ACCOUNTING_Q).

The value can be:

MQMON_NONE
Accounting data collection for queues is disabled.

MQMON_OFF
Accounting data collection is disabled for queues specifying a value of MQMON_Q_MGR in the **QueueAccounting** parameter.

MQMON_ON
Accounting data collection is enabled for queues specifying a value of MQMON_Q_MGR in the **QueueAccounting** parameter.

QueueMonitoring (MQCFIN)
Default setting for online monitoring for queues (parameter identifier: MQIA_MONITORING_Q).

If the **QueueMonitoring** queue attribute is set to MQMON_Q_MGR, this attribute specifies the value which is assumed by the channel. The value can be any of the following values:

MQMON_OFF
Online monitoring data collection is turned off.

MQMON_NONE
Online monitoring data collection is turned off for queues regardless of the setting of their **QueueMonitoring** attribute.

MQMON_LOW
Online monitoring data collection is turned on, with a low ratio of data collection.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate ratio of data collection.

MQMON_HIGH
Online monitoring data collection is turned on, with a high ratio of data collection.

Multi **QueueStatistics (MQCFIN)**
Specifies whether statistics data is to be collected for queues (parameter identifier: MQIA_STATISTICS_Q).

The value can be:

MQMON_NONE
Statistics data collection is turned off for queues regardless of the setting of their **QueueStatistics** parameter.

MQMON_OFF
Statistics data collection is turned off for queues specifying a value of MQMON_Q_MGR in their **QueueStatistics** parameter.

MQMON_ON
Statistics data collection is turned on for queues specifying a value of MQMON_Q_MGR in their **QueueStatistics** parameter.

This parameter is valid only on Multiplatforms.

z/OS **ReceiveTimeout (MQCFIN)**
How long a TCP/IP channel waits to receive data from its partner (parameter identifier: MQIA_RECEIVE_TIMEOUT).

The length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to the inactive state.

This parameter is valid only on z/OS.

z/OS **ReceiveTimeoutMin (MQCFIN)**
The minimum length of time that a TCP/IP channel waits to receive data from its partner (parameter identifier: MQIA_RECEIVE_TIMEOUT_MIN).

The minimum length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to the inactive state. This parameter is valid only on z/OS.

z/OS **ReceiveTimeoutType (MQCFIN)**
The qualifier to apply to *ReceiveTimeout* (parameter identifier: MQIA_RECEIVE_TIMEOUT_TYPE).

The qualifier to apply to *ReceiveTimeoutType* to calculate how long a TCP/IP channel waits to receive data from its partner. The wait includes heartbeats. If the wait interval expires the channel returns to the inactive state. This parameter is valid only on z/OS.

The value can be:

MQRCVTIME_MULTIPLY
The *ReceiveTimeout* value is a multiplier to be applied to the negotiated value of *HeartbeatInterval* to determine how long a channel waits.

MQRCVTIME_ADD
ReceiveTimeout is a value, in seconds, to be added to the negotiated value of *HeartbeatInterval* to determine how long a channel waits.

MQRCVTIME_EQUAL
ReceiveTimeout is a value, in seconds, representing how long a channel waits.

RemoteEvent (MQCFIN)
Controls whether remote error events are generated (parameter identifier: MQIA_REMOTE_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

RepositoryName (MQCFST)
Repository name (parameter identifier: MQCA_REPOSITORY_NAME).

The name of a cluster for which this queue manager is to provide a repository service.

RepositoryNamelist (MQCFST)
Repository name list (parameter identifier: MQCA_REPOSITORY_NAMELIST).

The name of a list of clusters for which this queue manager is to provide a repository service.

RevDns (MQCFIN)
Whether reverse lookup of the host name from a Domain Name Server is carried out. (parameter identifier: MQIA_REVERSE_DNS_LOOKUP).

This attribute has an effect only on channels using a transport type (TRPTYPE) of TCP.

The value can be:

MQRDNS_DISABLED
DNS host names are not reverse looked-up for the IP addresses of inbound channels. With this setting any CHLAUTH rules using host names are not matched.

MQRDNS_ENABLED
DNS host names are reverse looked-up for the IP addresses of inbound channels when this information is required. This setting is required for matching against CHLAUTH rules that contain host names, and for writing out error messages.

z/OS **SecurityCase (MQCFIN)**
Security case supported (parameter identifier: MQIA_SECURITY_CASE).

Specifies whether the queue manager supports security profile names in mixed case, or in uppercase only. The value is activated when a Refresh Security command is run with *SecurityType(MQSECTYPE_CLASSES)* specified.

The value can be:

MQSCYC_UPPER
Security profile names must be in uppercase.

MQSCYC_MIXED
Security profile names can be in uppercase or in mixed case.

This parameter is valid only on z/OS.

z/OS **SharedQQmgrName (MQCFIN)**
Shared-queue queue manager name (parameter identifier: MQIA_SHARED_Q_Q_MGR_NAME).

A queue manager makes an MQOPEN call for a shared queue. The queue manager that is specified in the **ObjectQmgrName** parameter of the MQOPEN call is in the same queue-sharing group as the processing queue manager. The SQQMNAME attribute specifies whether the *ObjectQmgrName* is used or whether the processing queue manager opens the shared queue directly.

The value can be any of the following values:

MQSQQM_USE
ObjectQmgrName is used and the appropriate transmission queue is opened.

MQSQQM_IGNORE
The processing queue manager opens the shared queue directly.

This parameter is valid only on z/OS.

Splcap (MQCFIN)
If the AMS component is installed for the version of IBM MQ that the queue manager is running under, the attribute has a value YES (MQCAP_SUPPORTED). If the AMS component is not installed, the value is NO (MQCAP_NOT_SUPPORTED) (parameter identifier: MQIA_PROT_POLICY_CAPABILITY).

The value can be one of the following values:

MQCAP_SUPPORTED
If the AMS component is installed for the version of IBM MQ that the queue manager is running under.

MQCAP_NOT_SUPPORTED
If the AMS component is not installed.

SSLCRLNamelist (MQCFST)
The TLS certificate revocation location namelist (parameter identifier: MQCA_SSL_CRL_NAMELIST).

The length of the string is MQ_NAMELIST_NAME_LENGTH.

Indicates the name of a namelist of authentication information objects to be used for certificate revocation checking by the queue manager.

Only authentication information objects with types of LDAPCRL or OCSP are allowed in the namelist referred to by *SSLCRLNamelist* (MQCFST). Any other type results in an error message when the list is processed and is subsequently ignored.

Multi **SSLCryptoHardware (MQCFST)**
Parameters to configure the TLS cryptographic hardware (parameter identifier: MQCA_SSL_CRYPTO_HARDWARE).

The length of the string is MQ_SSL_CRYPTO_HARDWARE_LENGTH.

Sets the name of the parameter string required to configure the cryptographic hardware present on the system.

This parameter is valid only on Multiplatforms.

SSLEvent (MQCFIN)
Controls whether TLS events are generated (parameter identifier: MQIA_SSL_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

SSLFipsRequired (MQCFIN)
Controls whether only FIPS-certified algorithms are to be used if cryptography is executed in IBM MQ itself (parameter identifier: MQIA_SSL_FIPS_REQUIRED). This parameter is valid only on z/OS, UNIX, Linux, and Windows.

The value can be:

MQSSL_FIPS_NO
Any supported CipherSpec can be used.

MQSSL_FIPS_YES
Only FIPS-certified cryptographic algorithms are to be used if cryptography is executed in IBM MQ rather than cryptographic hardware.

SSLKeyRepository (MQCFST)
Location and name of the TLS key repository (parameter identifier: MQCA_SSL_KEY_REPOSITORY).

The length of the string is MQ_SSL_KEY_REPOSITORY_LENGTH.

Indicates the name of the Secure Sockets Layer key repository.

The format of the name depends on the environment.

SSLKeyResetCount (MQCFIN)
TLS key reset count (parameter identifier: MQIA_SSL_RESET_COUNT).

The number of unencrypted bytes that initiating TLS channel MCAs send or receive before renegotiating the secret key.

z/OS **SSLTasks (MQCFIN)**
Number of server subtasks used for processing TLS calls (parameter identifier: MQIA_SSL_TASKS).

The number of server subtasks used for processing TLS calls. This parameter is valid only on z/OS.

StartStopEvent (MQCFIN)
Controls whether start and stop events are generated (parameter identifier: MQIA_START_STOP_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

Multi **StatisticsInterval (MQCFIN)**
The time interval, in seconds, at which statistics monitoring data is written to the monitoring queue (parameter identifier: MQIA_STATISTICS_INTERVAL).

This parameter is valid only on Multiplatforms.

SyncPoint (MQCFIN)
Sync point availability (parameter identifier: MQIA_SYNCPOINT).

The value can be:

MQSP_AVAILABLE
Units of work and sync pointing available.

MQSP_NOT_AVAILABLE
Units of work and sync pointing not available.

z/OS **TCPChannels (MQCFIN)**
The maximum number of channels that can be current, or clients that can be connected, that use the TCP/IP transmission protocol (parameter identifier: MQIA_TCP_CHANNELS).

This parameter is valid only on z/OS.

z/OS **TCPKeepAlive (MQCFIN)**
Specifies whether the TCP KEEPALIVE facility is to be used to check whether the other end of the connection is still available (parameter identifier: MQIA_TCP_KEEP_ALIVE).

The value can be:

MQTCPKEEP_YES
The TCP KEEPALIVE facility is to be used as specified in the TCP profile configuration data set. The interval is specified in the *KeepAliveInterval* channel attribute.

MQTCPKEEP_NO
The TCP KEEPALIVE facility is not to be used.

This parameter is valid only on z/OS.

z/OS **TCPName (MQCFST)**
The name of the TCP/IP system that you are using (parameter identifier: MQIA_TCP_NAME).

This parameter is valid only on z/OS.

TCPStackType (MQCFIN)
Specifies whether the channel initiator can use only the TCP/IP address space specified in *TCPName*, or can optionally bind to any selected TCP/IP address (parameter identifier: MQIA_TCP_STACK_TYPE).

The value can be:

MQTCPSTACK_SINGLE
The channel initiator can use only the TCP/IP address space specified in *TCPName*.

MQTCPSTACK_MULTIPLE
The channel initiator can use any TCP/IP address space available to it.

This parameter is valid only on z/OS.

TraceRouteRecording (MQCFIN)
Specifies whether trace-route information can be recorded and a reply message generated (parameter identifier: MQIA_TRACE_ROUTE_RECORDING).

The value can be:

MQRECORDING_DISABLED
Trace-route information cannot be recorded.

MQRECORDING_MSG
Trace-route information can be recorded and sent to the destination specified by the originator of the message causing the trace route record.

MQRECORDING_Q
Trace-route information can be recorded and sent to SYSTEM.ADMIN.TRACE.ROUTE.QUEUE.

TreeLifeTime (MQCFIN)
The lifetime in seconds of non-administrative topics (parameter identifier: MQIA_TREE_LIFE_TIME).

Non-administrative topics are those topics created when an application publishes to, or subscribes on, a topic string that does not exist as an administrative node. When this non-administrative node no longer has any active subscriptions, this parameter determines how long the queue manager waits before removing that node. Only non-administrative topics that are in use by a durable subscription remain after the queue manager it recycled.

The value can be in the range 0 - 604,000. A value of 0 means that non-administrative topics are not removed by the queue manager. The initial default value of the queue manager is 1800.

TriggerInterval (MQCFIN)
Trigger interval (parameter identifier: MQIA_TRIGGER_INTERVAL).

Specifies the trigger time interval, expressed in milliseconds, for use only with queues where *TriggerType* has a value of MQTT_FIRST.

Version (MQCFST)
The version of the IBM MQ code (parameter identifier: MQCA_VERSION).

The version of the IBM MQ code is shown as VVRRMMFF:

VV: Version

RR: Release

MM: Maintenance level

FF: Fix level




XrCapability (MQCFIN)

Specifies whether the MQ Telemetry capability and commands are supported by the queue manager where *XrCapability* has a value of MQCAP_SUPPORTED or MQCAP_NOT_SUPPORTED (parameter identifier: MQIA_XR_CAPABILITY).

This parameter applies only to  IBM i, UNIX, and Windows.

Related information:

Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client

Federal Information Processing Standards (FIPS) for UNIX, Linux and Windows

Inquire Queue Manager Status on Multiplatforms: Multi

The Inquire Queue Manager Status (MQCMD_INQUIRE_Q_MGR_STATUS) command inquires about the status of the local queue manager.

Optional parameters

QMStatusAttrs (MQCFIL)

Queue manager status attributes (parameter identifier: MQIACF_Q_MGR_STATUS_ATTRS).

The attribute list might specify the following value on its own - default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_Q_MGR_NAME
Name of the local queue manager.

MQCA_INSTALLATION_DESC
Description of the installation associated with the queue manager. This parameter is not valid on IBM i.

MQCA_INSTALLATION_NAME
Name of the installation associated with the queue manager. This parameter is not valid on IBM i.

MQCA_INSTALLATION_PATH
Path of the installation associated with the queue manager. This parameter is not valid on IBM i.

V 9.0.2 **MQCACF_ARCHIVE_LOG_EXTENT_NAME)**
Name of the oldest log extent for which the queue manager is waiting for archive notification.

The maximum length of the string is MQ_LOG_EXTENT_NAME_LENGTH.

If the queue manager is not using archive log management, this attribute is blank.

MQCACF_CURRENT_LOG_EXTENT_NAME
Name of the log extent currently being written to by the logger. MQCACF_CURRENT_LOG_EXTENT_NAME is available only on queue managers using linear logging. On other queue managers, MQCACF_CURRENT_LOG_EXTENT_NAME is blank.

MQCACF_LOG_PATH
Location of the recovery log extents.

MQCACF_MEDIA_LOG_EXTENT_NAME
Name of the earliest log extent required to perform media recovery. MQCACF_MEDIA_LOG_EXTENT_NAME is available only on queue managers using linear logging. On other queue managers, MQCACF_MEDIA_LOG_EXTENT_NAME is blank.

MQCACF_RESTART_LOG_EXTENT_NAME
Name of the earliest log extent required to perform restart recovery.

MQCACF_RESTART_LOG_EXTENT_NAME is available only on queue managers using linear logging. On other queue managers, MQCACF_RESTART_LOG_EXTENT_NAME is blank.

MQCACF_Q_MGR_START_DATE
The date on which the queue manager was started (in the form yyyy-mm-dd). The length of this attribute is given by MQ_DATE_LENGTH.

MQCACF_Q_MGR_START_TIME
The time at which the queue manager was started (in the form hh.mm.ss). The length of this attribute is given by MQ_TIME_LENGTH.

V 9.0.2 **MQIACF_ARCHIVE_LOG_SIZE**
Current size of the amount of space occupied, in megabytes, by log extents no longer required for restart or media recovery but waiting to be archived.

This attribute is not valid on IBM i.

MQIACF_CHINIT_STATUS
Current status of the channel initiator.

MQIACF_LDAP_CONNECTION_STATUS
Current status of the connection to the LDAP server.

MQIACF_CMD_SERVER_STATUS
Current status of the command server.

MQIACF_CONNECTION_COUNT
Current number of connections to the queue manager.

V 9.0.2 **MQIACF_LOG_IN_USE**
Current size of the percentage of the primary log space in use for restart recovery at this point in time.

This attribute is not valid on IBM i.

V 9.0.2 **MQIACF_LOG_UTILIZATION**
Current percentage estimate of how well the queue manager workload is contained within the primary log space.

This attribute is not valid on IBM i.

V 9.0.2 **MQIACF_MEDIA_LOG_SIZE**
Current size of the log data required for media recovery in megabytes.

This attribute is not valid on IBM i.

MQIACF_Q_MGR_STATUS
Current status of the queue manager.

V 9.0.2 **MQIACF_Q_MGR_STATUS_LOG**
Current status of all the log attributes. The attributes can be any of the following:

- MQCACF_ARCHIVE_LOG_EXTENT_NAME
- MQIACF_ARCHIVE_LOG_SIZE
- MQCACF_CURRENT_LOG_EXTENT_NAME
- MQIACF_LOG_IN_USE
- MQIACF_LOG_UTILIZATION
- MQCACF_MEDIA_LOG_EXTENT_NAME
- MQIACF_MEDIA_LOG_SIZE
- MQCACF_RESTART_LOG_EXTENT_NAME

- MQIACF_RESTART_LOG_SIZE
- MQIACF_REUSABLE_LOG_SIZE

V 9.0.2 **MQIACF_RESTART_LOG_SIZE**
Size of the log data required for restart recovery in megabytes.

This attribute is not valid on IBM i.

V 9.0.2 **MQIACF_REUSABLE_LOG_SIZE**
The amount of space occupied, in megabytes, by log extents available to be reused.

This attribute is not valid on IBM i.

Inquire Queue Manager Status (Response) on Multiplatforms: Multi

The response to the Inquire Queue Manager Status (MQCMD_INQUIRE_Q_MGR_STATUS) command consists of the response header followed by the *QMgrName* and *QMgrStatus* structures and the requested combination of attribute parameter structures.

Always returned:
QMgrName, *QMgrStatus*

Returned if requested:
V 9.0.2 *ArchiveLog*, V 9.0.2 *ArchiveLogSize*, *ChannelInitiatorStatus*, *CommandServerStatus*, *ConnectionCount*, *CurrentLog*, *InstallationDesc*, *InstallationName*, *InstallationPath*, *LDAPConnectionStatus*, V 9.0.2 *LogInUse*, *LogPath*, V 9.0.2 *LogUtilization*, *MediaRecoveryLog*, V 9.0.2 *MediaRecoveryLogSize*, V 9.0.2 *RestartRecoveryLogSize*, V 9.0.2 *ReusableLogSize*, *StartDate*, *StartTime*

Response data

V 9.0.2 **ArchiveLog (MQCFST)**
Name of the oldest log extent for which the queue manager is waiting for archive notification or blank if they have all been archived (parameter identifier MQCACF_ARCHIVE_LOG_EXTENT_NAME).

V 9.0.2 **ArchiveLogSize (MQCFIN)**
Current size of the amount of space occupied, in megabytes, by log extents no longer required for restart or media recovery but waiting to be archived (parameter identifier MQIACF_ARCHIVE_LOG_SIZE).

ChannelInitiatorStatus (MQCFIN)
Status of the channel initiator reading SYSTEM.CHANNEL.INITQ (parameter identifier: MQIACF_CHINIT_STATUS).

The value can be:

MQSVC_STATUS_STOPPED
The channel initiator is not running.

MQSVC_STATUS_STARTING
The channel initiator is in the process of initializing.

MQSVC_STATUS_RUNNING
The channel initiator is fully initialized and is running.

MQSVC_STATUS_STOPPING
The channel initiator is stopping.

CommandServerStatus (MQCFIN)
Status of the command server (parameter identifier: MQIACF_CMD_SERVER_STATUS).

The value can be:

MQSVC_STATUS_STARTING
The command server is in the process of initializing.

MQSVC_STATUS_RUNNING
The command server is fully initialized and is running.

MQSVC_STATUS_STOPPING
The command server is stopping.

ConnectionCount (MQCFIN)
Connection count (parameter identifier: MQIACF_CONNECTION_COUNT).

The current number of connections to the queue manager.

CurrentLog (MQCFST)
Log extent name (parameter identifier: MQCACF_CURRENT_LOG_EXTENT_NAME).

The name of the log extent that was being written to at the time of the Inquire command. If the queue manager is using circular logging, this parameter is blank.

The maximum length of the string is MQ_LOG_EXTENT_NAME_LENGTH.

InstallationDesc (MQCFST)
Installation Description (parameter identifier: MQCA_INSTALLATION_DESC)

The installation description for this queue manager. Not valid on IBM i.

InstallationName (MQCFST)
Installation Name (parameter identifier: MQCA_INSTALLATION_NAME)

The installation name for this queue manager. Not valid on IBM i.

InstallationPath (MQCFST)
Installation Path (parameter identifier: MQCA_INSTALLATION_PATH)

The installation path for this queue manager. Not valid on IBM i.

LDAPConnectionStatus (MQCFIN)
Current status of the queue manager's connection to the LDAP server (parameter identifier: MQIACF_LDAP_CONNECTION_STATUS).

The value can be:

MQLDAPC_CONNECTED
The queue manager currently has a connection to the LDAP server.

MQLDAPC_ERROR
The queue manager attempted to make a connection to the LDAP server and failed.

MQLDAPC_INACTIVE
The queue manager is not configured to use an LDAP server or has not yet made a connection to the LDAP server.

V 9.0.2 **LogInUse (MQCFIN)**
Current size of the percentage of the primary log space in use for restart recovery at this point in time (parameter identifier MQIACF_LOG_IN_USE).

LogPath (MQCFST)
Location of the recovery log extents (parameter identifier: MQCACF_LOG_PATH).

This parameter identifies the directory where log files are created by the queue manager.

The maximum length of the string is MQ_LOG_PATH_LENGTH.

V 9.0.2 **LogUtilization (MQCFIN)**
Current percentage estimate of how well the queue manager workload is contained within the primary log space (parameter identifier MQIACF_LOG_UTILIZATION).

MediaRecoveryLog (MQCFST)
Name of the oldest log extent required by the queue manager to perform media recovery (parameter identifier: MQCACF_MEDIA_LOG_EXTENT_NAME). This parameter is available only on queue managers using linear logging. If the queue manager is using circular logging, this parameter is blank.

The maximum length of the string is MQ_LOG_EXTENT_NAME_LENGTH.

V 9.0.2 **MediaRecoveryLogSize (MQCFIN)**
Current size of the log data required for media recovery in megabytes (parameter identifier MQIACF_MEDIA_LOG_SIZE).

QMgrName (MQCFST)
Name of the local queue manager (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QMgrStatus (MQCFIN)
Current execution status of the queue manager (parameter identifier: MQIACF_Q_MGR_STATUS).

The value can be:

MQQMSTA_STARTING
The queue manager is initializing.

MQQMSTA_RUNNING
The queue manager is fully initialized and is running.

MQQMSTA_QUIESCING
The queue manager is quiescing.

RestartRecoveryLog (MQCFST)
Name of the oldest log extent required by the queue manager to perform restart recovery (parameter identifier: MQCACF_RESTART_LOG_EXTENT_NAME).

This parameter is available only on queue managers using linear logging. If the queue manager is using circular logging, this parameter is blank.

The maximum length of the string is MQ_LOG_EXTENT_NAME_LENGTH.

V 9.0.2 **RestartRecoveryLogSize (MQCFIN)**
Size of the log data required for restart recovery in megabytes (parameter identifier MQIACF_RESTART_LOG_SIZE).

V 9.0.2 **ReusableLogSize (MQCFIN)**
The amount of space occupied, in megabytes, by log extents available to be reused (parameter identifier MQIACF_REUSABLE_LOG_SIZE).

StartDate (MQCFST)
Date when this queue manager was started (in the form `yyyy-mm-dd`) (parameter identifier: MQCACF_Q_MGR_START_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

StartTime (MQCFST)
Time when this queue manager was started (in the form `hh:mm:ss`) (parameter identifier: MQCACF_Q_MGR_START_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

Inquire Queue Names:

The Inquire Queue Names (MQCMD_INQUIRE_Q_NAMES) command inquires a list of queue names that match the generic queue name, and the optional queue type specified.

Required parameters

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

Generic queue names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_Q_LENGTH.

Optional parameters z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is processed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- a queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

z/OS

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED. MQQSGD_SHARED is permitted only in a shared queue environment.

QType (MQCFIN)
Queue type (parameter identifier: MQIA_Q_TYPE).

If present, this parameter limits the queue names returned to queues of the specified type. If this parameter is not present, queues of all types are eligible. The value can be any of the following values:

MQQT_ALL
All queue types.

MQQT_LOCAL
Local queue.

MQQT_ALIAS
Alias queue definition.

MQQT_REMOTE
Local definition of a remote queue.

MQQT_MODEL
Model queue definition.

The default value if this parameter is not specified is MQQT_ALL.

Inquire Queue Names (Response):

The response to the Inquire Queue Names (MQCMD_INQUIRE_Q_NAMES) command consists of the response header followed by a single parameter structure giving zero or more names that match the specified queue name. The response header is followed by the *QTypes* structure, with the same number of entries as the *QNames* structure. Each entry gives the type of the queue with the corresponding entry in the *QNames* structure.

z/OS
Additionally, on z/OS only, the **QSGDispositions** parameter structure (with the same number of entries as the *QNames* structure) is returned. Each entry in this structure indicates the disposition of the object with the corresponding entry in the *QNames* structure.

Always returned:
QNames, z/OS *QSGDispositions*, *QTypes*

Returned if requested:
None

Response data

QNames (MQCFSL)
List of queue names (parameter identifier: MQCACF_Q_NAMES).

z/OS

QSGDispositions (MQCFIL)
List of queue-sharing group dispositions (parameter identifier: MQIACF_QSG_DISPS). This parameter is valid on z/OS only. Possible values for fields in this structure are:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED.

QTypes (MQCFIL)
List of queue types (parameter identifier: MQIACF_Q_TYPES). Possible values for fields in this structure are:

MQQT_ALIAS
Alias queue definition.

MQQT_LOCAL
Local queue.

MQQT_REMOTE
Local definition of a remote queue.

MQQT_MODEL
Model queue definition.

Inquire Queue Status:

The Inquire Queue Status (MQCMD_INQUIRE_Q_STATUS) command inquires about the status of a local IBM MQ queue. You must specify the name of a local queue for which you want to receive status information.

Required parameters

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

Generic queue names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all queues having names that start with the selected character string. An asterisk on its own matches all possible names.

The queue name is always returned, regardless of the attributes requested.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Optional parameters (Inquire Queue Status)

ByteStringFilterCommand (MQCFBF)
Byte string filter command descriptor. The parameter identifier must be MQBACF_EXTERNAL_UOW_ID or MQBACF_Q_MGR_UOW_ID. Use this parameter to restrict the output from the command by specifying a filter condition. See “MQCFBF - PCF byte string filter parameter” on page 1596 for information about using this filter condition.

If you specify a byte string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter, or a string filter using the **StringFilterCommand** parameter.

z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is initiated when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- Blank (or omit the parameter altogether). The command is initiated on the queue manager on which it was entered.
- Queue manager name. The command is initiated on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be initiated.
- An asterisk (*). The command is initiated on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *QStatusAttrs* except MQIACF_ALL, MQIACF_MONITORING, and MQIACF_Q_TIME_INDICATOR. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1601 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a byte string filter using the **ByteStringFilterCommand** parameter or a string filter using the **StringFilterCommand** parameter.

OpenType (MQCFIN)
Queue status open type (parameter identifier: MQIACF_OPEN_TYPE).

It is always returned, regardless of the queue instance attributes requested.

The value can be:

MQQSOT_ALL
Selects status for queues that are open with any type of access.

MQQSOT_INPUT
Selects status for queues that are open for input.

MQQSOT_OUTPUT
Selects status for queues that are open for output.

The default value if this parameter if not specified is MQQSOT_ALL.

Filtering is not supported for this parameter.

z/OS

QSGDisposition (MQCFIN)
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object (that is, where it is defined and how it behaves). This parameter is valid only on z/OS. The value can be any of the following values:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED.

You cannot use *QSGDisposition* as a parameter to filter on.

QStatusAttrs (MQCFIL)
Queue status attributes (parameter identifier: MQIACF_Q_STATUS_ATTRS).

The attribute list can specify the following value on its own - default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

Where *StatusType* is MQIACF_Q_STATUS:

MQCA_Q_NAME
Queue name.

MQCACF_LAST_GET_DATE
Date of the last message successfully destructively read from the queue.

MQCACF_LAST_GET_TIME
Time of the last message successfully destructively read from the queue.

MQCACF_LAST_PUT_DATE
Date of the last message successfully put to the queue.

MQCACF_LAST_PUT_TIME
Time of the last message successfully put to the queue.

MQCACF_MEDIA_LOG_EXTENT_NAME
Identity of the oldest log extent required to perform media recovery of the queue.

On IBM i, this parameter identifies the name of the oldest journal receiver require to perform media recovery of the queue.

MQIA_CURRENT_Q_DEPTH
The current number of messages on the queue.

MQIA_MONITORING_Q
Current level of monitoring data collection.

MQIA_OPEN_INPUT_COUNT
The number of handles that are currently open for input for the queue.
MQIA_OPEN_INPUT_COUNT does not include handles that are open for browse.

MQIA_OPEN_OUTPUT_COUNT
The number of handles that are currently open for output for the queue.

MQIACF_HANDLE_STATE
Whether an API call is in progress.

MQIACF_MONITORING
All the queue status monitoring attributes. These attributes are:

- MQCACF_LAST_GET_DATE
- MQCACF_LAST_GET_TIME
- MQCACF_LAST_PUT_DATE
- MQCACF_LAST_PUT_TIME
- MQIA_MONITORING_Q
- MQIACF_OLDEST_MSG_AGE
- MQIACF_Q_TIME_INDICATOR

Filtering is not supported for this parameter.

MQIACF_OLDEST_MSG_AGE
Age of oldest message on the queue.

MQIACF_Q_TIME_INDICATOR
Indicator of the time that messages remain on the queue.

MQIACF_UNCOMMITTED_MSGS
The number of uncommitted messages on the queue.

Where *StatusType* is MQIACF_Q_HANDLE:

MQBACF_EXTERNAL_UOW_ID
Unit of recovery identifier assigned by the queue manager.

MQBACF_Q_MGR_UOW_ID
External unit of recovery identifier associated with the connection.

MQCA_Q_NAME
Queue name.

MQCACF_APPL_TAG
This parameter is a string containing the tag of the application connected to the queue manager.

MQCACF_ASID
Address-space identifier of the application identified by *ApplTag*. This parameter is valid on z/OS only.

MQCACF_PSB_NAME
Name of the program specification block (PSB) associated with the running IMS transaction. This parameter is valid on z/OS only.

MQCACF_PSTID
Identifier of the IMS program specification table (PST) for the connected IMS region. This parameter is valid on z/OS only.

MQCACF_TASK_NUMBER
CICS task number. This parameter is valid on z/OS only.

MQCACF_TRANSACTION_ID
CICS transaction identifier. This parameter is valid on z/OS only.

MQCACF_USER_IDENTIFIER
The user name of the application that has opened the specified queue.

MQCACH_CHANNEL_NAME
The name of the channel that has the queue open, if any.

MQCACH_CONNECTION_NAME
The connection name of the channel that has the queue open, if any.

MQIA_APPL_TYPE
The type of application that has the queue open.

MQIACF_OPEN_BROWSE
Open browse.

Filtering is not supported for this parameter.

MQIACF_OPEN_INPUT_TYPE
Open input type.

Filtering is not supported for this parameter.

MQIACF_OPEN_INQUIRE
Open inquire.

Filtering is not supported for this parameter.

MQIACF_OPEN_OPTIONS
The options used to open the queue.

If this parameter is requested, the following parameter structures are also returned:
- *OpenBrowse*
- *OpenInputType*
- *OpenInquire*
- *OpenOutput*
- *OpenSet*

Filtering is not supported for this parameter.

MQIACF_OPEN_OUTPUT
Open output.

Filtering is not supported for this parameter.

MQIACF_OPEN_SET
Open set.

Filtering is not supported for this parameter.

MQIACF_PROCESS_ID
The process identifier of the application that has opened the specified queue.

MQIACF_ASYNC_STATE

MQIACF_THREAD_ID
The thread identifier of the application that has opened the specified queue.

MQIACF_UOW_TYPE
Type of external unit of recovery identifier as seen by the queue manager.

StatusType (MQCFIN)
Queue status type (parameter identifier: MQIACF_Q_STATUS_TYPE).

Specifies the type of status information required.

The value can be any of the following values:

MQIACF_Q_STATUS
Selects status information relating to queues.

MQIACF_Q_HANDLE
Selects status information relating to the handles that are accessing the queues.

The default value, if this parameter is not specified, is MQIACF_Q_STATUS.

You cannot use *StatusType* as a parameter to filter on.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *QStatusAttrs* except MQCA_Q_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1608 for information about using this filter condition.

If you specify a string filter, you cannot also specify a byte string filter using the **ByteStringFilterCommand** parameter or an integer filter using the **IntegerFilterCommand** parameter.

Error codes

This command might return the following error code in the response format header "Error codes applicable to all commands" on page 1075 along with any additional pertinent values.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_Q_TYPE_ERROR
Queue type not valid.

Inquire Queue Status (Response):

The response to the Inquire Queue Status (MQCMD_INQUIRE_Q_STATUS) command consists of the response header followed by the *QName* structure and a set of attribute parameter structures determined by the value of *StatusType* in the Inquire command.

Always returned:
QName, z/OS *QSGDisposition*, *StatusType*

Possible values of *StatusType* are:

MQIACF_Q_STATUS
Returns status information relating to queues.

MQIACF_Q_HANDLE
Returns status information relating to the handles that are accessing the queues.

Returned if requested and *StatusType* is MQIACF_Q_STATUS:
CurrentQDepth, *LastGetDate*, *LastGetTime*, *LastPutDate*, *LastPutTime*, *MediaRecoveryLogExtent*, *OldestMsgAge*, *OnQTime*, *OpenInputCount*, *OpenOutputCount*, *QueueMonitoring*, *UncommittedMsgs*

Returned if requested and *StatusType* is MQIACF_Q_HANDLE:
ApplDesc, *ApplTag*, *ApplType*, z/OS *ASId*, *AsynchronousState*, *ChannelName*, *ConnectionName*, z/OS *ExternalUOWId*, *HandleState*, *OpenOptions*, *ProcessId*, z/OS *PSBName*, z/OS *PSTId*, *QMgrUOWId*, z/OS *TaskNumber*, *ThreadId*, z/OS *TransactionId*, *UOWIdentifier*, *UOWType*, *UserIdentifier*

Response data if StatusType is MQIACF_Q_STATUS

CurrentQDepth (MQCFIN)
Current queue depth (parameter identifier: MQIA_CURRENT_Q_DEPTH).

LastGetDate (MQCFST)
Date on which the last message was destructively read from the queue (parameter identifier: MQCACF_LAST_GET_DATE).

The date, in the form yyyy-mm-dd, on which the last message was successfully read from the queue. The date is returned in the time zone in which the queue manager is running.

The maximum length of the string is MQ_DATE_LENGTH.

LastGetTime (MQCFST)
Time at which the last message was destructively read from the queue (parameter identifier: MQCACF_LAST_GET_TIME).

The time, in the form hh.mm.ss, at which the last message was successfully read from the queue. The time is returned in the time zone in which the queue manager is running.

The maximum length of the string is MQ_TIME_LENGTH.

LastPutDate (MQCFST)
Date on which the last message was successfully put to the queue (parameter identifier: MQCACF_LAST_PUT_DATE).

The date, in the form `yyyy-mm-dd`, on which the last message was successfully put to the queue. The date is returned in the time zone in which the queue manager is running.

The maximum length of the string is MQ_DATE_LENGTH.

LastPutTime (MQCFST)
Time at which the last message was successfully put to the queue (parameter identifier: MQCACF_LAST_PUT_TIME).

The time, in the form `hh.mm.ss`, at which the last message was successfully put to the queue. The time is returned in the time zone in which the queue manager is running.

The maximum length of the string is MQ_TIME_LENGTH.

Multi **MediaRecoveryLogExtent (MQCFST)**
Name of the oldest log extent required to perform media recovery of the queue (parameter identifier: MQCACF_MEDIA_LOG_EXTENT_NAME).

On IBM i, this parameter identifies the name of the oldest journal receiver required to perform media recovery of the queue.

The name returned is of the form `Snnnnnnn.LOG` and is not a fully qualified path name. The use of this parameter provides the ability for the name to be easily correlated with the messages issued, following an **rcdmqimg** command to identify those queues causing the media recovery LSN not to move forwards.

This parameter is valid only on Multiplatforms.

The maximum length of the string is MQ_LOG_EXTENT_NAME_LENGTH.

OldestMsgAge (MQCFIN)
Age of the oldest message (parameter identifier: MQIACF_OLDEST_MSG_AGE). Age, in seconds, of the oldest message on the queue.

If the value is unavailable, MQMON_NOT_AVAILABLE is returned. If the queue is empty, `0` is returned. If the value exceeds 999 999 999, it is returned as 999 999 999.

OnQTime (MQCFIL)
Indicator of the time that messages remain on the queue (parameter identifier: MQIACF_Q_TIME_INDICATOR). Amount of time, in microseconds, that a message spent on the queue. Two values are returned:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned. If the value exceeds 999 999 999, it is returned as 999 999 999.

OpenInputCount (MQCFIN)
Open input count (parameter identifier: MQIA_OPEN_INPUT_COUNT).

OpenOutputCount (MQCFIN)
Open output count (parameter identifier: MQIA_OPEN_OUTPUT_COUNT).

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

z/OS

QSGDisposition (MQCFIN)
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Returns the disposition of the object (that is, where it is defined and how it behaves). This parameter is valid on z/OS only. The value can be any of the following values:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED.

QueueMonitoring (MQCFIN)
Current level of monitoring data collection for the queue (parameter identifier: MQIA_MONITORING_Q). The value can be any of the following values:

MQMON_OFF
Monitoring for the queue is disabled.

MQMON_LOW
Low rate of data collection.

MQMON_MEDIUM
Medium rate of data collection.

MQMON_HIGH
High rate of data collection.

StatusType (MQCFST)
Queue status type (parameter identifier: MQIACF_Q_STATUS_TYPE).

Specifies the type of status information.

UncommittedMsgs (MQCFIN)
The number of uncommitted changes (puts and gets) pending for the queue (parameter identifier: MQIACF_UNCOMMITTED_MSGS). The value can be any of the following values:

MQQSUM_YES
On z/OS, there are one or more uncommitted changes pending.

MQQSUM_NO
There are no uncommitted changes pending.

n  On Multiplatforms, an integer value indicating how many uncommitted changes are pending.

Response data if StatusType is MQIACF_Q_HANDLE

ApplDesc (MQCFST)
Application description (parameter identifier: MQCACF_APPL_DESC).

The maximum length is MQ_APPL_DESC_LENGTH.

ApplTag (MQCFST)
Open application tag (parameter identifier: MQCACF_APPL_TAG).

The maximum length of the string is MQ_APPL_TAG_LENGTH.

ApplType (MQCFIN)
Open application type (parameter identifier: MQIA_APPL_TYPE).

The value can be any of the following values:

MQAT_QMGR
A queue manager process.

MQAT_CHANNEL_INITIATOR
The channel initiator.

MQAT_USER
A user application.

MQAT_BATCH
Application using a batch connection. MQAT_BATCH applies only to z/OS.

MQAT_RRS_BATCH
RRS-coordinated application using a batch connection. MQAT_RRS_BATCH applies only to z/OS.

MQAT_CICS
A CICS transaction. MQAT_CICS applies only to z/OS.

MQAT_IMS
An IMS transaction. MQAT_IMS applies only to z/OS.

MQAT_SYSTEM_EXTENSION
Application performing an extension of function that is provided by the queue manager.

z/OS

ASId (MQCFST)
Address-space identifier (parameter identifier: MQCACF_ASID).

The 4-character address-space identifier of the application identified by *ApplTag*. It distinguishes duplicate values of *ApplTag*. This parameter applies only to z/OS.

The length of the string is MQ_ASID_LENGTH.

AsynchronousState (MQCFIN)
The state of the asynchronous consumer on this queue (parameter identifier: MQIACF_ASYNC_STATE).

The value can be any of the following values:

MQAS_ACTIVE
An MQCB call has set up a function to call back to process messages asynchronously and the connection handle has been started so that asynchronous message consumption can proceed.

MQAS_INACTIVE
An MQCB call has set up a function to call back to process messages asynchronously but the connection handle has not yet been started, or has been stopped or suspended, so that asynchronous message consumption cannot currently proceed.

MQAS_SUSPENDED
The asynchronous consumption callback has been suspended so that asynchronous message consumption cannot currently proceed on this handle. This situation can be either because an MQCB or MQCTL call with *Operation* MQOP_SUSPEND has been issued against this object handle by the application, or because it has been suspended by the system. If it has been suspended by the system, as part of the process of suspending asynchronous message consumption the callback function is called with the reason code that describes the problem resulting in suspension. This situation is reported in the *Reason* field in the MQCBC structure passed to the callback. In order for asynchronous message consumption to proceed, the application must issue an MQCB or MQCTL call with *Operation* MQOP_RESUME.

MQAS_SUSPENDED_TEMPORARY
The asynchronous consumption callback has been temporarily suspended by the system so that asynchronous message consumption cannot currently proceed on this object handle. As

part of the process of suspending asynchronous message consumption the callback function is called with the reason code that describes the problem resulting in suspension. This situation is reported in the *Reason* field in the MQCBC structure passed to the callback. The callback function is called again when asynchronous message consumption is resumed by the system after the temporary condition has been resolved.

MQAS_NONE
An MQCB call has not been issued against this handle, so no asynchronous message consumption is configured on this handle.

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Conname (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

z/OS

ExternalUOWId (MQCFBS)
RRS unit-of-recovery identifier (parameter identifier: MQBACF_EXTERNAL_UOW_ID).

The RRS unit-of-recovery identifier associated with the handle. This parameter is valid only on z/OS only.

The length of the string is MQ_EXTERNAL_UOW_ID_LENGTH.

HandleState (MQCFIN)
State of the handle (parameter identifier: MQIACF_HANDLE_STATE).

The value can be any of the following values:

MQHSTATE_ACTIVE
An API call from a connection is currently in progress for this object. For a queue, this condition can arise when an MQGET WAIT call is in progress.

If there is an MQGET SIGNAL outstanding, it does not mean, by itself, that the handle is active.

MQHSTATE_INACTIVE
No API call from a connection is currently in progress for this object. For a queue, this condition can arise when no MQGET WAIT call is in progress.

OpenBrowse (MQCFIN)
Open browse (parameter identifier: MQIACF_OPEN_BROWSE).

The value can be any of the following values:

MQQSO_YES
The queue is open for browsing.

MQQSO_NO
The queue is not open for browsing.

OpenInputType (MQCFIN)
Open input type (parameter identifier: MQIACF_OPEN_INPUT_TYPE).

The value can be any of the following values:

MQQSO_NO
The queue is not open for inputting.

MQQSO_SHARED
The queue is open for shared input.

MQQSO_EXCLUSIVE
The queue is open for exclusive input.

OpenInquire (MQCFIN)
Open inquire (parameter identifier: MQIACF_OPEN_INQUIRE).

The value can be any of the following values:

MQQSO_YES
The queue is open for inquiring.

MQQSO_NO
The queue is not open for inquiring.

OpenOptions (MQCFIN)
Open options currently in force for the queue (parameter identifier: MQIACF_OPEN_OPTIONS).

OpenOutput (MQCFIN)
Open output (parameter identifier: MQIACF_OPEN_OUTPUT).

The value can be any of the following values:

MQQSO_YES
The queue is open for output.

MQQSO_NO
The queue is not open for output.

OpenSet (MQCFIN)
Open set (parameter identifier: MQIACF_OPEN_SET).

The value can be any of the following values:

MQQSO_YES
The queue is open for setting.

MQQSO_NO
The queue is not open for setting.

ProcessId (MQCFIN)
Open application process ID (parameter identifier: MQIACF_PROCESS_ID).

z/OS

PSBName (MQCFST)
Program specification block (PSB) name (parameter identifier: MQCACF_PSB_NAME).

The 8-character name of the PSB associated with the running IMS transaction. This parameter is valid on z/OS only.

The length of the string is MQ_PSB_NAME_LENGTH.

z/OS

PSTId (MQCFST)
Program specification table (PST) identifier (parameter identifier: MQCACF_PST_ID).

The 4-character identifier of the PST region identifier for the connected IMS region. This parameter is valid on z/OS only.

The length of the string is MQ_PST_ID_LENGTH.

QMgrUOWId (MQCFBS)
The unit of recovery assigned by the queue manager (parameter identifier: MQBACF_Q_MGR_UOW_ID).

On z/OS, this parameter is an 8-byte log RBA, displayed as 16 hexadecimal characters. On platforms other than z/OS, this parameter is an 8-byte transaction identifier, displayed as 16 hexadecimal characters.

The maximum length of the string is MQ_UOW_ID_LENGTH.

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

z/OS

QSGDisposition (MQCFIN)
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Returns the disposition of the object (that is, where it is defined and how it behaves). This parameter is valid on z/OS only. The value can be any of the following values:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED.

StatusType (MQCFST)
Queue status type (parameter identifier: MQIACF_Q_STATUS_TYPE).

Specifies the type of status information.

z/OS

TaskNumber (MQCFST)
CICS task number (parameter identifier: MQCACF_TASK_NUMBER).

A 7-digit CICS task number. This parameter is valid on z/OS only.

The length of the string is MQ_TASK_NUMBER_LENGTH.

ThreadId (MQCFIN)
The thread ID of the open application (parameter identifier: MQIACF_THREAD_ID).

A value of zero indicates that the handle was opened by a shared connection. A handle created by a shared connection is logically open to all threads.

z/OS

TransactionId (MQCFST)
CICS transaction identifier (parameter identifier: MQCACF_TRANSACTION_ID).

A 4-character CICS transaction identifier. This parameter is valid on z/OS only.

The length of the string is MQ_TRANSACTION_ID_LENGTH.

UOWIdentifier (MQCFBS)
The external unit of recovery associated with the connection (parameter identifier: MQBACF_EXTERNAL_UOW_ID).

This parameter is the recovery identifier for the unit of recovery. Its format is determined by the value of *UOWType*.

The maximum length of the string is MQ_UOW_ID_LENGTH.

UOWType (MQCFIN)
Type of external unit of recovery identifier as perceived by the queue manager (parameter identifier: MQIACF_UOW_TYPE).

The value can be any of the following values:

MQUOWT_Q_MGR

MQUOWT_CICS
z/OS Valid only on z/OS.

MQUOWT_RRS
z/OS Valid only on z/OS.

MQUOWT_IMS
z/OS Valid only on z/OS.

MQUOWT_XA

UOWType identifies the *UOWIdentifier* type and not the type of the transaction coordinator. When the value of *UOWType* is MQUOWT_Q_MGR, the associated identifier is in *QMgrUOWId* (and not *UOWIdentifier*).

UserIdentifier (MQCFST)
Open application user name (parameter identifier: MQCACF_USER_IDENTIFIER).

The maximum length of the string is MQ_MAX_USER_ID_LENGTH.

Inquire Security on z/OS: z/OS

The Inquire Security (MQCMD_INQUIRE_SECURITY) command returns information about the current settings for the security parameters.

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- a queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

SecurityAttrs (MQCFIL)
Security parameter attributes (parameter identifier: MQIACF_SECURITY_ATTRS).

The attribute list might specify the following value on its own - default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQIACF_SECURITY_SWITCH
Current setting of the switch profiles. If the subsystem security switch is off, no other switch profile settings are returned.

MQIACF_SECURITY_TIMEOUT
Timeout value.

MQIACF_SECURITY_INTERVAL
Time interval between checks.

Inquire Security (Response) on z/OS: z/OS

The response to the Inquire Security (MQCMD_INQUIRE_SECURITY) command consists of the response header followed by the requested combination of attribute parameter structures.

One message is returned if either **SecurityTimeout** or **SecurityInterval** is specified on the command. If **SecuritySwitch** is specified, one message per security switch found is returned. This message includes the **SecuritySwitch**, **SecuritySwitchSetting**, and **SecuritySwitchProfile** parameter structures.

Returned if requested:
SecurityInterval, **SecuritySwitch**, **SecuritySwitchProfile**, **SecuritySwitchSetting**, **SecurityTimeout**

Response data

SecurityInterval (MQCFIN)
Time interval between checks (parameter identifier: MQIACF_SECURITY_INTERVAL).

The interval, in minutes, between checks for user IDs and their associated resources to determine whether **SecurityTimeout** has expired.

SecuritySwitch (MQCFIN)
Security switch profile (parameter identifier: MQIA_CF_LEVEL). The value can be any of the following values:

MQSECSW_SUBSYSTEM
Subsystem security switch.

MQSECSW_Q_MGR
Queue manager security switch.

MQSECSW_QSG
Queue-sharing group security switch.

MQSECSW_CONNECTION
Connection security switch.

MQSECSW_COMMAND
Command security switch.

MQSECSW_CONTEXT
Context security switch.

MQSECSW_ALTERNATE_USER
Alternate user security switch.

MQSECSW_PROCESS
Process security switch.

MQSECSW_NAMELIST
Namelist security switch.

MQSECSW_TOPIC
Topic security switch.

MQSECSW_Q
Queue security switch.

MQSECSW_COMMAND_RESOURCES
Command resource security switch.

SecuritySwitchProfile (MQCFST)
Security switch profile (parameter identifier: MQCACF_SECURITY_PROFILE).

The maximum length of the string is MQ_SECURITY_PROFILE_LENGTH.

SecuritySwitchSetting (MQCFIN)
Setting of the security switch (parameter identifier: MQIACF_SECURITY_SETTING).

The value can be:

MQSECSW_ON_FOUND
Switch ON, profile found.

MQSECSW_OFF_FOUND
Switch OFF, profile found.

MQSECSW_ON_NOT_FOUND
Switch ON, profile not found.

MQSECSW_OFF_NOT_FOUND
Switch OFF, profile not found.

MQSECSW_OFF_ERROR
Switch OFF, profile error.

MQSECSW_ON_OVERRIDDEN
Switch ON, profile overridden.

SecurityTimeout (MQCFIN)
Timeout value (parameter identifier: MQIACF_SECURITY_TIMEOUT).

How long, in minutes, security information about an unused user ID and associated resources is retained.

Inquire Service on Multiplatforms: 

The Inquire Service (MQCMD_INQUIRE_SERVICE) command inquires about the attributes of existing IBM MQ services.

Required parameters

ServiceName (MQCFST)
Service name (parameter identifier: MQCA_SERVICE_NAME).

This parameter is the name of the service whose attributes are required. Generic service names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all services having names that start with the selected character string. An asterisk on its own matches all possible names.

The service name is always returned regardless of the attributes requested.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Optional parameters

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *ServiceAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1601 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the **StringFilterCommand** parameter.

ServiceAttrs (MQCFIL)
Service attributes (parameter identifier: MQIACF_SERVICE_ATTRS).

The attribute list might specify the following value on its own - default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_ALTERATION_DATE
Date on which the definition was last altered.

MQCA_ALTERATION_TIME
Time at which the definition was last altered.

MQCA_SERVICE_DESC
Description of service definition.

MQCA_SERVICE_NAME
Name of service definition.

MQCA_SERVICE_START_ARGS
Arguments to be passed to the service program.

MQCA_SERVICE_START_COMMAND
Name of program to run to start the service.

MQCA_SERVICE_STOP_ARGS
Arguments to be passed to the stop program to stop the service.

MQCA_STDERR_DESTINATION
Destination of standard error for the process.

MQCA_STDOUT_DESTINATION
Destination of standard output for the process.

MQCA_SERVICE_START_ARGS
Arguments to be passed to the service program.

MQIA_SERVICE_CONTROL
When the queue manager must start the service.

MQIA_SERVICE_TYPE
Mode in which the service is to run.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *ServiceAttrs* except MQCA_SERVICE_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1608 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter.

Inquire Service (Response) on Multiplatforms: Multi

The response to the Inquire Service (MQCMD_INQUIRE_SERVICE) command consists of the response header followed by the *ServiceName* structure and the requested combination of attribute parameter structures.

If a generic service name was specified, one such message is generated for each service found.

Always returned:
ServiceName

Returned if requested:
AlterationDate, *AlterationTime*, *Arguments*, *ServiceDesc*, *ServiceType*, *StartArguments*, *StartCommand*, *StartMode*, *StderrDestination*, *StdoutDestination*, *StopArguments*, *StopCommand*

Response data

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date on which the information was last altered in the form yyyy-mm-dd.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time at which the information was last altered in the form hh.mm.ss.

ServiceDesc (MQCFST)
Description of service definition (parameter identifier: MQCA_SERVICE_DESC).

The maximum length of the string is MQ_SERVICE_DESC_LENGTH.

ServiceName (MQCFST)
Name of service definition (parameter identifier: MQCA_SERVICE_ NAME).

The maximum length of the string is MQ_SERVICE_NAME_LENGTH.

ServiceType (MQCFIN)
The mode in which the service is to run (parameter identifier: MQIA_SERVICE_TYPE).

The value can be:

MQSVC_TYPE_SERVER
Only one instance of the service can be executed at a time, with the status of the service made available by the Inquire Service Status command.

MQSVC_TYPE_COMMAND
Multiple instances of the service can be started.

StartArguments (MQCFST)
The arguments to be passed to the user program at queue manager startup (parameter identifier: MQCA_SERVICE_START_ARGS).

The maximum length of the string is MQ_SERVICE_ARGS_LENGTH.

StartCommand (MQCFST)
Service program name (parameter identifier: MQCA_SERVICE_START_COMMAND).

The name of the program which is to run.

The maximum length of the string is MQ_SERVICE_COMMAND_LENGTH.

StartMode (MQCFIN)
Service mode (parameter identifier: MQIA_SERVICE_CONTROL).

Specifies how the service is to be started and stopped. The value can be any of the following values:

MQSVC_CONTROL_MANUAL
The service is not to be started automatically or stopped automatically. It is to be controlled by user command.

MQSVC_CONTROL_Q_MGR
The service is to be started and stopped at the same time as the queue manager is started and stopped.

MQSVC_CONTROL_Q_MGR_START
The service is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

StderrDestination (MQCFST)
The path to a file to which the standard error (stderr) of the service program is to be redirected (parameter identifier: MQCA_STDERR_DESTINATION).

The maximum length of the string is MQ_SERVICE_PATH_LENGTH.

StdoutDestination (MQCFST)
The path to a file to which the standard output (stdout) of the service program is to be redirected (parameter identifier: MQCA_STDOUT_DESTINATION).

The maximum length of the string is MQ_SERVICE_PATH_LENGTH.

StopArguments (MQCFST)
The arguments to be passed to the stop program when instructed to stop the service (parameter identifier: MQCA_SERVICE_STOP_ARGS).

The maximum length of the string is MQ_SERVICE_ARGS_LENGTH.

StopCommand (MQCFST)
Service program stop command (parameter identifier: MQCA_SERVICE_STOP_COMMAND).

This parameter is the name of the program that is to run when the service is requested to stop.

The maximum length of the string is MQ_SERVICE_COMMAND_LENGTH.

Inquire Service Status on Multiplatforms: 

The Inquire Service Status (MQCMD_INQUIRE_SERVICE_STATUS) command inquires about the status of one or more IBM MQ service instances.

Required parameters

ServiceName (MQCFST)
Service name (parameter identifier: MQCA_SERVICE_NAME).

Generic service names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all services having names that start with the selected character string. An asterisk on its own matches all possible names.

The service name is always returned, regardless of the attributes requested.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Optional parameters (Inquire Service Status)

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *ServiceStatusAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See “MQCFIF - PCF integer filter parameter” on page 1601 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the **StringFilterCommand** parameter.

ServiceStatusAttrs (MQCFIL)
Service status attributes (parameter identifier: MQIACF_SERVICE_STATUS_ATTRS).

The attribute list can specify the following value on its own - is the default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_SERVICE_DESC
Description of service definition.

MQCA_SERVICE_NAME
Name of service definition.

MQCA_SERVICE_START_ARGS
The arguments to pass to the service program.

MQCA_SERVICE_START_COMMAND
The name of the program to run to start the service.

MQCA_SERVICE_STOP_ARGS
The arguments to pass to the stop command to stop the service.

MQCA_SERVICE_STOP_COMMAND
The name of the program to run to stop the service.

MQCA_STDERR_DESTINATION
Destination of standard error for the process.

MQCA_STDOUT_DESTINATION
Destination of standard output for the process.

MQCACF_SERVICE_START_DATE
The date on which the service was started.

MQCACF_SERVICE_START_TIME
The time at which the service was started.

MQIA_SERVICE_CONTROL
How the service is to be started and stopped.

MQIA_SERVICE_TYPE
The mode in which the service is to run.

MQIACF_PROCESS_ID
The process identifier of the operating system task under which this service is executing.

MQIACF_SERVICE_STATUS
Status of the service.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *ServiceStatusAttrs* except MQCA_SERVICE_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See “MQCFSF - PCF string filter parameter” on page 1608 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter.

Error codes

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_SERV_STATUS_NOT_FOUND
Service status not found.

Inquire Service Status (Response) on Multiplatforms: Multi

The response to the Inquire Service Status (MQCMD_INQUIRE_SERVICE_STATUS) command consists of the response header followed by the *ServiceName* structure and the requested combination of attribute parameter structures.

If a generic service name was specified, one such message is generated for each service found.

Always returned:
ServiceName

Returned if requested:
ProcessId, *ServiceDesc*, *StartArguments*, *StartCommand*, *StartDate*, *StartMode*, *StartTime*, *Status*, *StderrDestination*, *StdoutDestination*, *StopArguments*, *StopCommand*

Response data

ProcessId (MQCFIN)
Process identifier (parameter identifier: MQIACF_PROCESS_ID).

The operating system process identifier associated with the service.

ServiceDesc (MQCFST)
Description of service definition (parameter identifier: MQCACH_SERVICE_DESC).

The maximum length of the string is MQ_SERVICE_DESC_LENGTH.

ServiceName (MQCFST)
Name of the service definition (parameter identifier: MQCA_SERVICE_NAME).

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

StartArguments (MQCFST)
Arguments to be passed to the program on startup (parameter identifier: MQCA_SERVICE_START_ARGS).

The maximum length of the string is MQ_SERVICE_ARGS_LENGTH.

StartCommand (MQCFST)
Service program name (parameter identifier: MQCA_SERVICE_START_COMMAND).

Specifies the name of the program which is to run.

The maximum length of the string is MQ_SERVICE_COMMAND_LENGTH.

StartDate (MQCFST)
Start date (parameter identifier: MQIACF_SERVICE_START_DATE).

The date, in the form `yyyy-mm-dd`, on which the service was started.

The maximum length of the string is MQ_DATE_LENGTH

StartMode (MQCFIN)
Service mode (parameter identifier: MQIACH_SERVICE_CONTROL).

How the service is to be started and stopped. The value can be:

MQSVC_CONTROL_MANUAL
The service is not to be started automatically or stopped automatically. It is to be controlled by user command.

MQSVC_CONTROL_Q_MGR
The service is to be started and stopped at the same time as the queue manager is started and stopped.

MQSVC_CONTROL_Q_MGR_START
The service is to be started at the same time as the queue manager is started, but is not request to stop when the queue manager is stopped.

StartTime (MQCFST)
Start date (parameter identifier: MQIACF_SERVICE_START_TIME).

The time, in the form `hh.mm.ss`, at which the service was started.

The maximum length of the string is MQ_TIME_LENGTH

Status (MQCFIN)
Service status (parameter identifier: MQIACF_SERVICE_STATUS).

The status of the service. The value can be any of the following values:

MQSVC_STATUS_STARTING
The service is in the process of initializing.

MQSVC_STATUS_RUNNING
The service is running.

MQSVC_STATUS_STOPPING
The service is stopping.

StderrDestination (MQCFST)
Specifies the path to a file to which the standard error (stderr) of the service program is to be redirected (parameter identifier: MQCA_STDERR_DESTINATION).

The maximum length of the string is MQ_SERVICE_PATH_LENGTH.

StdoutDestination (MQCFST)
Specifies the path to a file to which the standard output (stdout) of the service program is to be redirected (parameter identifier: MQCA_STDOUT_DESTINATION).

The maximum length of the string is MQ_SERVICE_PATH_LENGTH.

StopArguments (MQCFST)
Specifies the arguments to be passed to the stop program when instructed to stop the service (parameter identifier: MQCA_SERVICE_STOP_ARGS).

The maximum length of the string is MQ_SERVICE_ARGS_LENGTH.

StopCommand (MQCFST)
Service program stop command (parameter identifier: MQCA_SERVICE_STOP_COMMAND).

This parameter is the name of the program that is to run when the service is requested to stop.

The maximum length of the string is MQ_SERVICE_COMMAND_LENGTH.

Inquire SMDS on z/OS: z/OS

The Inquire SMDS (MQCMD_INQUIRE_SMDS) command inquires about the attributes of shared message data sets for a CF application structure.

Required parameters

SMDS (qmgr_name)
Specifies the queue manager for which the shared message data set properties are to be displayed, or an asterisk to display the properties for all shared message data sets associated with the specified CFSTRUCT (parameter identifier: MQCACF_CF_SMDS).

CFStrucName (MQCFST)
The name of the CF application structure with SMDS properties that you want to inquire on (parameter identifier: MQCA_CF_STRUC_NAME).

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

Optional parameters

CFSMDSAttrs (MQCFIL)
CF application structure SMDS attributes (parameter identifier: MQIACF_SMDS_ATTRS).

The default value used if this parameter is not specified is:

MQIACF_ALL
All attributes.

The attribute list might specify MQIACF_ALL on its own, or may specify a combination of the following:

MQIA_CF_SMDS_BUFFERS
The shared message data set DSBUFS property.

MQIACF_CF_SMDS_EXPAND
The shared message data set DSEXPAND property.

Inquire SMDS (Response) on z/OS: z/OS

The response to the Inquire SMDS (MQCMD_INQUIRE_SMDS) command returns the attribute parameters of the shared message data set connection.

Response data

SMDS (MQCFST)
The queue manager name for which the shared message data set properties are displayed (parameter identifier: MQCACF_CF_SMDS).

CFStrucName (MQCFST)
CF Structure name (parameter identifier: MQCA_CF_STRUC_NAME).

The maximum length is MQ_CF_STRUC_NAME_LENGTH.

DSBUFS (MQCFIN)
The CF DSBUFS property (parameter identifier: MQIA_CF_SMDS_BUFFERS).

The returned value is in the range 0 - 9999.

The value is the number of buffers to be allocated in each queue manager for accessing shared message data sets. The size of each buffer is equal to the logical block size.

DSEXPAND (MQCFIN)
The CF DSEXPAND property (parameter identifier: MQIACF_CF_SMDS_EXPAND).

MQDSE_YES
The data set can be expanded.

MQDSE_NO
The data set cannot be expanded.

MQDSE_DEFAULT
Only returned on Inquire CF Struct when not explicitly set

Inquire SMDS Connection on z/OS: z/OS

The response to the Inquire SMDS Connection (MQCMD_INQUIRE_SMDSCONN) command returns status and availability information about the connection between the queue manager and the shared message data sets for the specified *CFStrucName*.

Required parameters

SMDSCONN (MQCFST)
Specify the queue manager which owns the SMDS for which the connection information is to be returned, or an asterisk to return the connection information for all shared message data sets associated with the specified *CFStrucName* (parameter identifier: MQCACF_CF_SMDSCONN).

CFStrucName (MQCFST)
The name of the CF application structure with SMDS connections properties that you want to inquire on (parameter identifier: MQCA_CF_STRUC_NAME).

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Inquire SMDS Connection (Response) on z/OS: z/OS

The response to the Inquire SMDS Connection (MQCMD_INQUIRE_SMDSCONN) command returns status and availability information about the connection between the queue manager and the shared message data sets for the specified *CFStrucName*.

Response data

SMDSCONN (MQCFST)
The queue manager which owns the SMDS for which the connection information is returned (parameter identifier: MQCACF_CF_SMDSCONN).

CFStrucName (MQCFST)
The name of the CF application structure with SMDS connections properties that you want to inquire on (parameter identifier: MQCA_CF_STRUC_NAME).

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

Avail (MQCFIN)
The availability of this data set connection as seen by this queue manager. This is one of the following values:

MQS_AVAIL_NORMAL
The connection can be used and no error has been detected.

MQS_AVAIL_ERROR
The connection is unavailable because of an error.

The queue manager may try to enable access again automatically if the error may no longer be present, for example when recovery completes or the status is manually set to RECOVERED. Otherwise, it can be enabled again using the START SMDSCONN command in order to retry the action which originally failed.

MQS_AVAIL_STOPPED
The connection cannot be used because it has been explicitly stopped using the STOP SMDSCONN command. It can only be made available again by using a START SMDSCONN command to enable it.

ExpandST (MQCFIN)
The data set automatic expansion status. This is one of the following values:

MQS_EXPANDST_NORMAL
No problem has been noted which would affect automatic expansion.

MQS_EXPANDST_FAILED
A recent expansion attempt failed, causing the DSEXPAND option to be set to NO for this specific data set. This status is cleared when ALTER SMDS is used to set the DSEXPAND option back to YES or DEFAULT.

MQS_EXPANDST_MAXIMUM
The maximum number of extents has been reached, so future expansion is not possible (except by taking the data set out of service and copying it to larger extents).

OpenMode (MQCFIN)
Indicates the mode in which the shared message data set is currently open by this queue manager.

MQS_OPENMODE_NONE
The shared message data set is not open.

MQS_OPENMODE_READONLY
The shared message data set is owned by another queue manager, and is open for read-only access.

MQS_OPENMODE_UPDATE
The shared message data set is owned by this queue manager, and is open for update access.

MQS_OPENMODE_RECOVERY
The shared message data set is open for recovery processing

Status (MQCFIN)
Indicates the shared message data set connection status as seen by this queue manager.

MQS_STATUS_CLOSED
This data set is not currently open.

MQS_STATUS_CLOSING
This queue manager is currently in the process of closing this data set, including quiescing normal I/O activity and storing the saved space map if necessary.

MQS_STATUS_OPENING
This queue manager is currently in the process of opening and validating this data set (including space map restart processing when necessary).

MQS_STATUS_OPEN
This queue manager has successfully opened this data set and it is available for normal use.

MQS_STATUS_NOTENABLED
The SMDS definition is not in the ACCESS(ENABLED) state so the data set is not currently available for normal use. This status is only set when the SMDSCONN status does not already indicate some other form of failure.

MQS_STATUS_ALLOCFAIL
This queue manager was unable to locate or allocate this data set.

MQS_STATUS_OPENFAIL
This queue manager was able to allocate the data set but was unable to open it, so it has now been deallocated.

MQS_STATUS_STGFAIL
The data set could not be used because the queue manager was unable to allocate associated storage areas for control blocks, or for space map or header record processing.

MQS_STATUS_DATAFAIL
The data set was successfully opened but the data was found to be invalid or inconsistent, or a permanent I/O error occurred, so it has now been closed and deallocated.

This might result in the shared message data set itself being marked as STATUS(FAILED).

Inquire Storage Class on z/OS: z/OS

The Inquire Storage Class (MQCMD_INQUIRE_STG_CLASS) command returns information about storage classes.

Required parameters

StorageClassName (MQCFST)
Storage class name (parameter identifier: MQCA_STORAGE_CLASS).

Generic storage class names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all storage classes having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_STORAGE_CLASS_LENGTH.

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *StgClassAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See “MQCFIF - PCF integer filter parameter” on page 1601 for information about using this filter condition.

If you specify an integer filter for *PageSetId*, you cannot also specify the **PageSetId** parameter.

If you specify an integer filter, you cannot also specify a string filter using the **StringFilterCommand** parameter.

PageSetId (MQCFIN)
Page set identifier that the storage class is associated with (parameter identifier: MQIA_PAGESET_ID).

If you omit this parameter, storage classes with any page set identifiers qualify.

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object (that is, where it is defined and how it behaves). The value can be:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined with either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

You cannot use *QSGDisposition* as a parameter to filter on.

StgClassAttrs (MQCFIL)
Storage class parameter attributes (parameter identifier: MQIACF_STORAGE_CLASS_ATTRS).

The attribute list might specify the following value on its own - is the default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_STORAGE_CLASS
Storage class name.

MQCA_STORAGE_CLASS_DESC
Description of the storage class.

MQIA_PAGESET_ID
The page set identifier to which the storage class maps.

MQCA_XCF_GROUP_NAME
The name of the XCF group of which IBM MQ is a member.

MQIA_XCF_MEMBER_NAME
The XCF member name of the IMS system within the XCF group specified in MQCA_XCF_GROUP_NAME.

MQCA_ALTERATION_DATE
The date on which the definition was last altered.

MQCA_ALTERATION_TIME
The time at which the definition was last altered.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *StgClassAttrs* except MQCA_STORAGE_CLASS. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1608 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter.

Inquire Storage Class (Response) on z/OS: z/OS

The response to the Inquire Storage Class (MQCMD_INQUIRE_STG_CLASS) command consists of the response header followed by the *StgClassName* structure, the *PageSetId* structure and the *QSGDisposition* structure which are followed by the requested combination of attribute parameter structures.

Always returned:
PageSetId, *QSGDisposition*, *StgClassName*

Returned if requested:
AlterationDate, *AlterationTime*, *PassTicketApplication*, *StorageClassDesc*, *XCFGroupName*, *XCFMemberName*,

Response data

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

This parameter is the date, in the form yyyy-mm-dd, on which the definition was last altered.

The maximum length of the string is MQ_DATE_LENGTH.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

This parameter is the time, in the form hh.mm.ss, at which the definition was last altered.

The maximum length of the string is MQ_TIME_LENGTH.

PageSetId (MQCFIN)
Page set identifier (parameter identifier: MQIA_PAGESET_ID).

The page set identifier to which the storage class maps.

PassTicketApplication (MQCFST)
PassTicket application (parameter identifier: MQCA_PASS_TICKET_APPL).

The application name that is passed to RACF when authenticating the PassTicket specified in the MQIIH header.

The maximum length is MQ_PASS_TICKET_APPL_LENGTH.

QSGDisposition (MQCFIN)
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

StorageClassDesc (MQCFST)
Description of the storage class (parameter identifier: MQCA_STORAGE_CLASS_DESC).

The maximum length is MQ_STORAGE_CLASS_DESC_LENGTH.

StgClassName (MQCFST)
Name of the storage class (parameter identifier: MQCA_STORAGE_CLASS).

The maximum length of the string is MQ_STORAGE_CLASS_LENGTH.

XCFGroupName (MQCFST)
Name of the XCF group of which IBM MQ is a member (parameter identifier: MQCA_XCF_GROUP_NAME).

The maximum length is MQ_XCF_GROUP_NAME_LENGTH.

XCFMemberName (MQCFST)
Name of the XCF group of which IBM MQ is a member (parameter identifier: MQCA_XCF_MEMBER_NAME).

The maximum length is MQ_XCF_MEMBER_NAME_LENGTH.

Inquire Storage Class Names on z/OS: z/OS

The Inquire Storage Class Names (MQCMD_INQUIRE_STG_CLASS_NAMES) command inquires a list of storage class names that match the generic storage class name specified.

Required parameters

StorageClassName (MQCFST)
Storage class name (parameter identifier: MQCA_STORAGE_CLASS).

Generic storage class names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all storage classes having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_STORAGE_CLASS_LENGTH.

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined with either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

Inquire Storage Class Names (Response) on z/OS: z/OS

The response to the Inquire Storage Class Names (MQCMD_INQUIRE_STG_CLASS_NAMES) command consists of the response header followed by a parameter structure giving zero or more names that match the specified namelist name.

In addition to this, the *QSGDispositions* structure (with the same number of entries as the *StorageClassNames* structure) is returned. Each entry in this structure indicates the disposition of the object with the corresponding entry in the *StorageClassNames* structure.

Always returned:
StorageClassNames, *QSGDispositions*

Returned if requested:
None

Response data

StorageClassNames (MQCFSL)
List of storage class names (parameter identifier: MQCACF_STORAGE_CLASS_NAMES).

QSGDispositions (MQCFIL)
List of queue-sharing group dispositions (parameter identifier: MQIACF_QSG_DISPS). Possible values for fields in this structure are those permitted for the *QSGDisposition* parameter (MQQSGD_*). Possible values for fields in this structure are:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

Inquire Subscription:

The Inquire Subscription (MQCMD_INQUIRE_SUBSCRIPTION) command inquires about the attributes of a subscription.

Required parameters

SubName (MQCFST)
The unique identifier of the application for a subscription (parameter identifier: MQCACF_SUB_NAME).

If *SubName* is not provided, *SubId* must be specified to identify the subscription to be inquired.

The maximum length of the string is MQ_SUB_NAME_LENGTH.

SubId (MQCFBS)
Subscription identifier (parameter identifier: MQBACF_SUB_ID).

Specifies the unique internal subscription identifier. If the queue manager is generating the CorrelId for a subscription, then the *SubId* is used as the *DestinationCorrelId*.

You must supply a value for *SubId* if you have not supplied a value for *SubName*.

The maximum length of the string is MQ_CORREL_ID_LENGTH.

Optional parameters z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- Blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- A queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- An asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

Durable (MQCFIN)
Specify this attribute to restrict the type of subscriptions which are displayed (parameter identifier: MQIACF_DURABLE_SUBSCRIPTION).

MQSUB_DURABLE_YES
Information about durable subscriptions only is displayed.

MQSUB_DURABLE_NO
Information about nondurable subscriptions only is displayed.

MQSUB_DURABLE_ALL
Information about all subscriptions is displayed.

SubscriptionAttrs (MQCFIL)
Subscription attributes (parameter identifier: MQIACF_SUB_ATTRS).

Use one of the following parameters to select the attributes you want to display:

- ALL to display all attributes.
- SUMMARY to display a subset of the attributes (see MQIACF_SUMMARY for a list).
- Any of the following parameters individually or in combination.

MQIACF_ALL
All attributes.

MQIACF_SUMMARY
Use this parameter to display:
- MQBACF_DESTINATION_CORREL_ID
- MQBACF_SUB_ID
- MQCACF_DESTINATION
- MQCACF_DESTINATION_Q_MGR
- MQCACF_SUB_NAME
- MQCA_TOPIC_STRING
- MQIACF_SUB_TYPE

MQBACF_ACCOUNTING_TOKEN
The accounting token passed by the subscriber for propagation into messages sent to this subscription in the AccountingToken field of the MQMD.

MQBACF_DESTINATION_CORREL_ID
The CorrelId used for messages sent to this subscription.

MQBACF_SUB_ID
The internal unique key identifying a subscription.

MQCA_ALTERATION_DATE
The date of the most recent MQSUB with MQSO_ALTER or ALTER SUB command.

MQCA_ALTERATION_TIME
The time of the most recent MQSUB with MQSO_ALTER or ALTER SUB command.

MQCA_CREATION_DATE
The date of the first MQSUB command that caused this subscription to be created.

MQCA_CREATION_TIME
The time of the first MQSUB that caused this subscription to be created.

MQCA_TOPIC_STRING
The resolved topic string the subscription is for.

MQCACF_APPL_IDENTITY_DATA
The identity data passed by the subscriber for propagation into messages sent to this subscription in the ApplIdentity field of the MQMD.

MQCACF_DESTINATION
The destination for messages published to this subscription.

MQCACF_DESTINATION_Q_MGR
The destination queue manager for messages published to this subscription.

MQCACF_SUB_NAME
The unique identifier of an application for a subscription.

MQCACF_SUB_SELECTOR
The SQL 92 selector string to be applied to messages published on the named topic to select whether they are eligible for this subscription.

MQCACF_SUB_USER_DATA
The user data associated with the subscription.

MQCACF_SUB_USER_ID
The userid that owns the subscription. MQCACF_SUB_USER_ID is either the userid associated with the creator of the subscription, or, if subscription takeover is permitted, the userid which last took over the subscription.

MQCA_TOPIC_NAME
The name of the topic object that identifies a position in the topic hierarchy to which the topic string is concatenated.

MQIACF_DESTINATION_CLASS
Indicated whether this subscription is a managed subscription.

MQIACF_DURABLE_SUBSCRIPTION
Whether the subscription is durable, persisting over queue manager restart.

MQIACF_EXPIRY
The time to live from creation date and time.

MQIACF_PUB_PRIORITY
The priority of the messages sent to this subscription.

MQIACF_PUBSUB_PROPERTIES
The manner in which publish/subscribe related message properties are added to messages sent to this subscription.

MQIACF_REQUEST_ONLY
Indicates whether the subscriber polls for updates by using MQSUBRQ API, or whether all publications are delivered to this subscription.

MQIACF_SUB_TYPE
The type of subscription - how it was created.

MQIACF_SUBSCRIPTION_SCOPE
Whether the subscription forwards messages to all other queue managers directly connected by using a Publish/Subscribe collective or hierarchy, or the subscription forwards messages on this topic within this queue manager only.

MQIACF_SUB_LEVEL
The level within the subscription interception hierarchy at which this subscription is made.

MQIACF_VARIABLE_USER_ID
Users other than the creator of this subscription that can connect to it (subject to topic and destination authority checks).

MQIACF_WILDCARD_SCHEMA
The schema to be used when interpreting wildcard characters in the topic string.

MQIA_DISPLAY_TYPE
Controls the output returned in the **TOPICSTR** and **TOPICOBJ** attributes.

SubscriptionType (MQCFIN)
Specify this attribute to restrict the type of subscriptions which are displayed (parameter identifier: MQIACF_SUB_TYPE).

MQSUBTYPE_ADMIN
Subscriptions which have been created by an admin interface or modified by an admin interface are selected.

MQSUBTYPE_ALL
All subscription types are displayed.

MQSUBTYPE_API
Subscriptions created by applications by way of the IBM MQ API are displayed.

MQSUBTYPE_PROXY
System created subscriptions relating to inter-queue manager subscriptions are displayed.

MQSUBTYPE_USER
USER subscriptions (with SUBTYPE of either ADMIN or API) are displayed. MQSUBTYPE_USER is the default value.

DisplayType (MQCFIN)
Controls the output returned in the **MQCA_TOPIC_STRING** and **MQCA_TOPIC_NAME** attributes (parameter identifier: MQIA_DISPLAY_TYPE).

MQDOPT_RESOLVED
Returns the resolved (full) topic string in the **MQCA_TOPIC_STRING** attribute. The value of the **MQCA_TOPIC_NAME** attribute is also returned.

MQDOPT_DEFINED
Returns the values of the **MQCA_TOPIC_NAME** and **MQCA_TOPIC_STRING** attributes provided when the subscription was created. The **MQCA_TOPIC_STRING** attribute will contain the application part of the topic string only. You can use the values returned with **MQCA_TOPIC_NAME** and **MQCA_TOPIC_STRING** to fully re-create the subscription by using **MQDOPT_DEFINED**.

Inquire Subscription (Response):

The response to the Inquire Subscription (MQCMD_INQUIRE_SUBSCRIPTION) command consists of the response header followed by the *SubId* and *SubName* structures, and the requested combination of attribute parameter structures (where applicable).

Always returned
SubID, *SubName*

Returned if requested
AlterationDate, *AlterationTime*, *CreationDate*, *CreationTime*, *Destination*, *DestinationClass*, *DestinationCorrelId*, *DestinationQueueManager*, *Expiry*, *PublishedAccountingToken*, *PublishedApplicationIdentityData*, *PublishPriority*, *PublishSubscribeProperties*, *Requestonly*, *Selector*, *SelectorType*, *SubscriptionLevel*, *SubscriptionScope*, *SubscriptionType*, *SubscriptionUser*, *TopicObject*, *TopicString*, *Userdata*, *VariableUser*, *WildcardSchema*

Response Data

AlterationDate (MQCFST)
The date of the most recent **MQSUB** or **Change Subscription** command that modified the properties of the subscription (parameter identifier: MQCA_ALTERATION_DATE).

AlterationTime (MQCFST)
The time of the most recent **MQSUB** or **Change Subscription** command that modified the properties of the subscription (parameter identifier: MQCA_ALTERATION_TIME).

CreationDate (MQCFST)
The creation date of the subscription, in the form yyyy-mm-dd (parameter identifier: MQCA_CREATION_DATE).

CreationTime (MQCFST)
The creation time of the subscription, in the form hh.mm.ss (parameter identifier: MQCA_CREATION_TIME).

Destination (MQCFST)
Destination (parameter identifier: MQCACF_DESTINATION).

Specifies the name of the alias, local, remote, or cluster queue to which messages for this subscription are put.

DestinationClass (MQCFIN)
Destination class (parameter identifier: MQIACF_DESTINATION_CLASS).

Whether the destination is managed.

The value can be any of the following values:

MQDC_MANAGED
The destination is managed.

MQDC_PROVIDED
The destination queue is as specified in the *Destination* field.

DestinationCorrelId (MQCFBS)
Destination correlation identifier (parameter identifier: MQBACF_DESTINATION_CORREL_ID).

A correlation identifier that is placed in the *CorrelId* field of the message descriptor for all the messages sent to this subscription.

The maximum length is MQ_CORREL_ID_LENGTH.

DestinationQueueManager (MQCFST)
Destination queue manager (parameter identifier: MQCACF_DESTINATION_Q_MGR).

Specifies the name of the destination queue manager, either local or remote, to which messages for the subscription are forwarded.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

DisplayType (MQCFIN)
The type of output requested for **MQCA_TOPIC_STRING** and **MQCA_TOPIC_NAME** is returned (parameter identifier: MQIA_DISPLAY_TYPE).

MQDOPT_RESOLVED
Returns the resolved (full) topic string in the **MQCA_TOPIC_STRING** attribute. The value of the **MQCA_TOPIC_NAME** attribute is also returned.

MQDOPT_DEFINED
The application portion of the topic string is returned in the **MQCA_TOPIC_STRING** attribute. **MQCA_TOPIC_NAME** contains the name of the **TOPIC** Object used when defining the subscription.

Durable (MQCFIN)
Whether this subscription is a durable subscription (parameter identifier: MQIACF_DURABLE_SUBSCRIPTION).

The value can be any of the following values:

MQSUB_DURABLE_YES
The subscription persists, even if the creating application disconnects from the queue manager or issues an MQCLOSE call for the subscription. The queue manager reinstates the subscription during restart.

MQSUB_DURABLE_NO
The subscription is non-durable. The queue manager removes the subscription when the creating application disconnects from the queue manager or issues an MQCLOSE call for the subscription. If the subscription has a destination class (DESTCLAS) of MANAGED, the queue manager removes any messages not yet consumed when it closes the subscription.

Expiry (MQCFIN)
The time, in tenths of a second, at which a subscription expires after its creation date and time (parameter identifier: MQIACF_EXPIRY).

A value of unlimited means that the subscription never expires.

After a subscription has expired it becomes eligible to be discarded by the queue manager and receives no further publications.

PublishedAccountingToken (MQCFBS)
Value of the accounting token used in the *AccountingToken* field of the message descriptor (parameter identifier: MQBACF_ACCOUNTING_TOKEN).

The maximum length of the string is MQ_ACCOUNTING_TOKEN_LENGTH.

PublishedApplicationIdentityData (MQCFST)
Value of the application identity data used in the *ApplIdentityData* field of the message descriptor (parameter identifier: MQCACF_APPL_IDENTITY_DATA.

The maximum length of the string is MQ_APPL_IDENTITY_DATA_LENGTH.

PublishPriority (MQCFIN)
The priority of messages sent to this subscription (parameter identifier: MQIACF_PUB_PRIORITY).

The value can be any of the following values:

MQPRI_PRIORITY_AS_PUBLISHED
The priority of messages sent to this subscription is taken from that priority supplied to the published message. MQPRI_PRIORITY_AS_PUBLISHED is the supplied default value.

MQPRI_PRIORITY_AS_QDEF
The priority of messages sent to this subscription is determined by the default priority of the queue defined as a destination.

0-9 An integer value providing an explicit priority for messages sent to this subscription.

PublishSubscribeProperties (MQCFIN)
Specifies how publish/subscribe related message properties are added to messages sent to this subscription (parameter identifier: MQIACF_PUBSUB_PROPERTIES).

The value can be any of the following values:

MQPSPROP_NONE
Publish/subscribe properties are not added to the messages. MQPSPROP_NONE is the supplied default value.

MQPSPROP_MSGPROP
Publish/subscribe properties are added as PCF attributes.

MQPSPROP_COMPAT
If the original publication is a PCF message, then the publish/subscribe properties are added as PCF attributes. Otherwise, publish/subscribe properties are added within an MQRFH version 1 header. This method is compatible with applications coded for use with previous versions of IBM MQ.

MQPSPROP_RFH2
Publish/subscribe properties are added within an MQRFH version 2 header. This method is compatible with applications coded for use with IBM Integration Bus brokers.

Requestonly (MQCFIN)
Indicates whether the subscriber polls for updates using the MQSUBRQ API call, or whether all publications are delivered to this subscription (parameter identifier: MQIACF_REQUEST_ONLY).

The value can be:

MQRU_PUBLISH_ALL
All publications on the topic are delivered to this subscription.

MQRU_PUBLISH_ON_REQUEST
Publications are only delivered to this subscription in response to an MQSUBRQ API call.

Selector (MQCFST)
Specifies the selector applied to messages published to the topic (parameter identifier: MQCACF_SUB_SELECTOR).

Only those messages that satisfy the selection criteria are put to the destination specified by this subscription.

SelectorType (MQCFIN)
The type of selector string that has been specified (parameter identifier: MQIACF_SELECTOR_TYPE).

The value can be any of the following values:

MQSELTYPE_NONE
No selector has been specified.

MQSELTYPE_STANDARD
The selector references only the properties of the message, not its content, using the standard IBM MQ selector syntax. Selectors of this type are to be handled internally by the queue manager.

MQSELTYPE_EXTENDED
The selector uses extended selector syntax, typically referencing the content of the message. Selectors of this type cannot be handled internally by the queue manager; extended selectors can be handled only by another program, such as IBM Integration Bus.

SubID (MQCFBS)
The internal, unique key identifying a subscription (parameter identifier: MQBACF_SUB_ID).

SubscriptionLevel (MQCFIN)
The level within the subscription interception hierarchy at which this subscription is made (parameter identifier: MQIACF_SUB_LEVEL).

The value can be:

0 - 9 An integer in the range 0-9. The default value is 1. Subscribers with a subscription level of 9 will intercept publications before they reach subscribers with lower subscription levels.

SubscriptionScope (MQCFIN)
Determines whether this subscription is passed to other queue managers in the network (parameter identifier: MQIACF_SUBSCRIPTION_SCOPE).

The value can be:

MQTSCOPE_ALL
The subscription is forwarded to all queue managers directly connected through a publish/subscribe collective or hierarchy. MQTSCOPE_ALL is the supplied default value.

MQTSCOPE_QMGR
The subscription only forwards messages published on the topic within this queue manager.

SubscriptionType (MQCFIN)
Indicates how the subscription was created (parameter identifier: MQIACF_SUB_TYPE).

MQSUBTYPE_PROXY
An internally created subscription used for routing publications through a queue manager.

MQSUBTYPE_ADMIN
Created using **DEF SUB** MQSC or PCF command. This **SUBTYPE** also indicates that a subscription has been modified using an administrative command.

MQSUBTYPE_API
Created using an **MQSUB** API request.

SubscriptionUser (MQCFST)
The userid that 'owns' this subscription. This parameter is either the userid associated with the creator of the subscription, or, if subscription takeover is permitted, the userid which last took over the subscription. (parameter identifier: MQCACF_SUB_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH.

TopicObject (MQCFST)
The name of a previously defined topic object from which is obtained the topic name for the subscription (parameter identifier: MQCA_TOPIC_NAME).

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

TopicString (MQCFST)
The resolved topic string (parameter identifier: MQCA_TOPIC_STRING).

The maximum length of the string is MQ_TOPIC_STR_LENGTH.

Userdata (MQCFST)
User data (parameter identifier: MQCACF_SUB_USER_DATA).

Specifies the user data associated with the subscription

The maximum length of the string is MQ_USER_DATA_LENGTH.

VariableUser (MQCFIN)
Specifies whether a user other than the one who created the subscription, that is, the user shown in *SubscriptionUser* can take over the ownership of the subscription (parameter identifier: MQIACF_VARIABLE_USER_ID).

The value can be any of the following values:

MQVU_ANY_USER
Any user can take over the ownership. MQVU_ANY_USER is the supplied default value.

MQVU_FIXED_USER
No other user can take over the ownership.

WildcardSchema (MQCFIN)
Specifies the schema to be used when interpreting any wildcard characters contained in the *TopicString* (parameter identifier: MQIACF_WILDCARD_SCHEMA).

The value can be any of the following values:

MQWS_CHAR
Wildcard characters represent portions of strings; it is for compatibility with IBM MQ V6.0 broker.

MQWS_TOPIC
Wildcard characters represent portions of the topic hierarchy; this is for compatibility with IBM Integration Bus brokers. MQWS_TOPIC is the supplied default value.

Inquire Subscription Status:

The Inquire Subscription Status (MQCMD_INQUIRE_SUB_STATUS) command inquires about the status of a subscription.

Required parameters

SubName (MQCFST)
The unique identifier of an application for a subscription (parameter identifier: MQCACF_SUB_NAME).

If *SubName* is not provided, *SubId* must be specified to identify the subscription to be inquired.

The maximum length of the string is MQ_SUB_NAME_LENGTH.

SubId (MQCFBS)
Subscription identifier (parameter identifier: MQBACF_SUB_ID).

Specifies the unique internal subscription identifier. If the queue manager is generating the CorrelId for a subscription, then the *SubId* is used as the *DestinationCorrelId*.

You must supply a value for *SubId* if you have not supplied a value for *SubName*.

The maximum length of the string is MQ_CORREL_ID_LENGTH.

Optional parameters z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is processed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- Blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- A queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- An asterisk (*). The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter on which to filter.

Durable (MQCFIN)
Specify this attribute to restrict the type of subscriptions which are displayed (parameter identifier: MQIACF_DURABLE_SUBSCRIPTION).

MQSUB_DURABLE_YES
Information about durable subscriptions only is displayed. MQSUB_DURABLE_YES is the default.

MQSUB_DURABLE_NO
Information about non-durable subscriptions only is displayed.

SubscriptionType (MQCFIN)
Specify this attribute to restrict the type of subscriptions which are displayed (parameter identifier: MQIACF_SUB_TYPE).

MQSUBTYPE_ADMIN
Subscriptions which have been created by an admin interface or modified by an admin interface are selected.

MQSUBTYPE_ALL
All subscription types are displayed.

MQSUBTYPE_API
Subscriptions created by applications through an IBM MQ API call are displayed.

MQSUBTYPE_PROXY
System created subscriptions relating to inter-queue manager subscriptions are displayed.

MQSUBTYPE_USER
USER subscriptions (with SUBTYPE of either ADMIN or API) are displayed. MQSUBTYPE_USER is the default value.

StatusAttrs (MQCFIL)
Subscription status attributes (parameter identifier: MQIACF_SUB_STATUS_ATTRS).

To select the attributes you want to display you can specify;

- ALL to display all attributes.

- any of the following parameters individually or in combination.

MQIACF_ALL
All attributes.

MQBACF_CONNECTION_ID
The currently active *ConnectionID* that has opened the subscription.

MQIACF_DURABLE_SUBSCRIPTION
Whether the subscription is durable, persisting over queue manager restart.

MQCACF_LAST_MSG_DATE
The date that a message was last sent to the destination specified by the subscription.

MQCACF_LAST_MSG_TIME
The time when a message was last sent to the destination specified by the subscription.

MQIACF_MESSAGE_COUNT
The number of messages put to the destination specified by the subscription.

MQCA_RESUME_DATE
The date of the most recent MQSUB command that connected to the subscription.

MQCA_RESUME_TIME
The time of the most recent MQSUB command that connected to the subscription.

MQIACF_SUB_TYPE
The type of subscription - how it was created.

MQCACF_SUB_USER_ID
The userid owns the subscription.

MQCA_TOPIC_STRING
Returns the fully resolved topic string of the subscription.

Inquire Subscription Status (Response):

The response to the Inquire Subscription Status (MQCMD_INQUIRE_SUB_STATUS) command consists of the response header followed by the *SubId* and *SubName* structures, and the requested combination of attribute parameter structures (where applicable).

Always returned
SubID, *SubName*

Returned if requested
ActiveConnection, *Durable*, *LastPublishDate*, *LastPublishTime*, *MCastRelIndicator*, *NumberMsgs*, *ResumeDate*, *ResumeTime*, *SubType*, *TopicString*

Response Data

ActiveConnection **(MQCFBS)**
The *ConnId* of the *HConn* that currently has this subscription open (parameter identifier: MQBACF_CONNECTION_ID).

Durable **(MQCFIN)**
A durable subscription is not deleted when the creating application closes its subscription handle (parameter identifier: MQIACF_DURABLE_SUBSCRIPTION).

MQSUB_DURABLE_NO
The subscription is removed when the application that created it is closed or disconnected from the queue manager.

MQSUB_DURABLE_YES
The subscription persists even when the creating application is no longer running or has been disconnected. The subscription is reinstated when the queue manager restarts.

LastMessageDate (MQCFST)
The date that a message was last sent to the destination specified by the subscription (parameter identifier: MQCACF_LAST_MSG_DATE).

LastMessageTime (MQCFST)
The time when a message was last sent to the destination specified by the subscription (parameter identifier: MQCACF_LAST_MSG_TIME).

MCastRelIndicator **(MQCFIN)**
The multicast reliability indicator (parameter identifier: MQIACF_MCAST_REL_INDICATOR).

NumberMsgs (MQCFIN)
The number of messages put to the destination specified by this subscription (parameter identifier: MQIACF_MESSAGE_COUNT).

ResumeDate (MQCFST)
The date of the most recent **MQSUB** API call that connected to the subscription (parameter identifier: MQCA_RESUME_DATE).

ResumeTime (MQCFST)
The time of the most recent **MQSUB** API call that connected to the subscription (parameter identifier: MQCA_RESUME_TIME).

SubscriptionUser (MQCFST)
The userid that 'owns' this subscription. This parameter is either the userid associated with the creator of the subscription, or, if subscription takeover is permitted, the userid which last took over the subscription. (parameter identifier: MQCACF_SUB_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH.

SubID (MQCFBS)
The internal, unique key identifying a subscription (parameter identifier: MQBACF_SUB_ID).

SubName (MQCFST)
The unique identifier of a subscription (parameter identifier: MQCACF_SUB_NAME).

SubType (MQCFIN)
Indicates how the subscription was created (parameter identifier: MQIA_SUB_TYPE).

MQSUBTYPE_PROXY
An internally created subscription used for routing publications through a queue manager.

MQSUBTYPE_ADMIN
Created using the **DEF SUB** MQSC or **Create Subscription** PCF command. This Subtype also indicates that a subscription has been modified using an administrative command.

MQSUBTYPE_API
Created using an **MQSUB** API call.

TopicString (MQCFST)
The resolved topic string (parameter identifier: MQCA_TOPIC_STRING). The maximum length of the string is MQ_TOPIC_STR_LENGTH.

Inquire System on z/OS: z/OS

The Inquire System (MQCMD_INQUIRE_SYSTEM) command returns general system parameters and information.

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Inquire System (Response) on z/OS: z/OS

The response to the Inquire System (MQCMD_INQUIRE_SYSTEM) command consists of the response header followed by the *ParameterType* structure and the combination of attribute parameter structures determined by the value of the parameter type.

Always returned:
ParameterType

Possible values of *ParameterType* are:

MQSYSP_TYPE_INITIAL
The initial settings of the system parameters.

MQSYSP_TYPE_SET
The settings of the system parameters if they have been altered since their initial setting.

Returned if *ParameterType* is MQSYSP_TYPE_INITIAL or MQSYSP_TYPE_SET (and a value is set):
CheckpointCount, *ClusterCacheType*, *CodedCharSetId*, *CommandUserId*, *ConnSwap*, *DB2BlobTasks*, *DB2Name*, *DB2Tasks*, *DSGName*, *Exclmsg*, *ExitInterval*, *ExitTasks*, *MULCCapture*, *OpMode*, *OTMADruExit*, *OTMAGroup*, *OTMAInterval*, *OTMAMember*, *OTMSTpipePrefix*, *QIndexDefer*, *QSGName*, *RESLEVELAudit*, *RoutingCode*, *Service*, *SMFAccounting*, *SMFStatistics*, *SMFInterval*, *Splcap*, *TraceClass*, *TraceSize*, *WLMInterval*, *WLMIntervalUnits*

Response data

CheckpointCount (MQCFIN)
The number of log records written by IBM MQ between the start of one checkpoint and the next (parameter identifier: MQIACF_SYSP_CHKPOINT_COUNT).

ClusterCacheType (MQCFIN)
The type of the cluster cache (parameter identifier: MQIACF_SYSP_CLUSTER_CACHE).

The value can be any of the following values:

MQCLCT_STATIC
Static cluster cache.

MQCLCT_DYNAMIC
Dynamic cluster cache.

CodedCharSetId (MQCFIN)
Archive retention period (parameter identifier: MQIA_CODED_CHAR_SET_ID).

The coded character set identifier for the queue manager.

CommandUserId (MQCFST)
Command user ID (parameter identifier: MQCACF_SYSP_CMD_USER_ID).

Specifies the default user ID for command security checks.

The maximum length of the string is MQ_USER_ID_LENGTH.

ConnSwap (MQCFIN)
Specifies whether jobs that are issuing certain MQ API calls are swappable or non-swappable (parameter identifier: MQIACF_CONNECTION_SWAP).

This value can be either MQSYSP_YES or MQSYSP_NO.

DB2BlobTasks (MQCFIN)
The number of Db2 server tasks to be used for BLOBs (parameter identifier: MQIACF_SYSP_DB2_BLOB_TASKS).

DB2Name (MQCFST)
The name of the Db2 subsystem or group attachment to which the queue manager is to connect (parameter identifier: MQCACF_DB2_NAME).

The maximum length of the string is MQ_DB2_NAME_LENGTH.

DB2Tasks (MQCFIN)
The number of Db2 server tasks to use (parameter identifier: MQIACF_SYSP_DB2_TASKS).

DSGName (MQCFST)
The name of the Db2 data-sharing group to which the queue manager is to connect (parameter identifier: MQCACF_DSG_NAME).

The maximum length of the string is MQ_DSG_NAME_LENGTH.

Exclmsg (MQCFSL)
A list of message identifiers to be excluded from being written to any log (parameter identifier: MQCACF_EXCL_OPERATOR_MESSAGES).

The maximum length of each message identifier is MQ_OPERATOR_MESSAGE_LENGTH.

The list can contain a maximum of 16 message identifiers.

ExitInterval (MQCFIN)
The time, in seconds, for which queue manager exits can execute during each invocation (parameter identifier: MQIACF_SYSP_EXIT_INTERVAL).

ExitTasks (MQCFIN)
Specifies how many started server tasks to use to run queue manager exits (parameter identifier: MQIACF_SYSP_EXIT_TASKS).

MULCCapture (MQCFIN)
The Measured Usage Pricing property is used to control the algorithm for gathering data used by Measured Usage License Charging (MULC) (parameter identifier: MQIACF_MULC_CAPTURE).

The returned values can be MQMULC_STANDARD or MQMULC_REFINED.

OpMode (MQCFIL)

An integer item list containing  three elements which describe the current operation mode (parameter identifier: MQIACF_OPERATION_MODE).

1. The first integer element can be one of the following:

MQOPMODE_COMPAT
The queue manager is operating in compatibility mode (COMPAT). Only those functions in the specified level or an earlier level of queue manager are available.

MQOPMODE_NEW_FUNCTION
The queue manager is operating in new function mode (NEWFUNC).

2. The second integer element contains the current compatibility level. The value is in the format of the MQCMDL_LEVEL_* constants. See Constants.
3. CD The third integer element contains the available function level. The value is in the format of the MQCMDL_LEVEL_* constants.

CD The compatibility level indicates which version and fix level the queue manager has been migrated from and thus can fall back to if necessary. The available function level indicates the level of new IBM MQ functions restricted by OPMODE that are currently available. For more details, see OPMODE.

This parameter is valid only on z/OS.

OTMADruExit (MQCFST)
The name of the OTMA destination resolution user exit to be run by IMS (parameter identifier: MQCACF_SYSP_OTMA_DRU_EXIT).

The maximum length of the string is MQ_EXIT_NAME_LENGTH.

OTMAGroup (MQCFST)
The name of the XCF group to which this instance of IBM MQ belongs (parameter identifier: MQCACF_SYSP_OTMA_GROUP).

The maximum length of the string is MQ_XCF_GROUP_NAME_LENGTH.

OTMAInterval (MQCFIN)
The length of time, in seconds, that a user ID from IBM MQ is considered previously verified by IMS (parameter identifier: MQIACF_SYSP_OTMA_INTERVAL).

OTMAMember (MQCFST)
The name of the XCF member to which this instance of IBM MQ belongs (parameter identifier: MQCACF_SYSP_OTMA_MEMBER).

The maximum length of the string is MQ_XCF_MEMBER_NAME_LENGTH.

OTMSTpipePrefix (MQCFST)
The prefix to be used for Tpipe names (parameter identifier: MQCACF_SYSP_OTMA_TPIPE_PFX).

The maximum length of the string is MQ_TPIPE_PFX_LENGTH.

QIndexDefer (MQCFIN)
Specifies whether queue manager restart completes before all indexes are built deferring building to later, or waits until all indexes are built (parameter identifier: MQIACF_SYSP_Q_INDEX_DEFER).

The value can be any of the following values:

MQSYSP_YES
Queue manager restart completes before all indexes are built.

MQSYSP_NO
Queue manager restart waits until all indexes are built.

QSGName (MQCFST)
The name of the queue-sharing group to which the queue manager belongs (parameter identifier: MQCA_QSG_NAME).

The maximum length of the string is MQ_QSG_NAME_LENGTH.

RESLEVELAudit (MQCFIN)
Specifies whether RACF audit records are written for RESLEVEL security checks performed during connection processing (parameter identifier: MQIACF_SYSP_RESLEVEL_AUDIT).

The value can be any of the following values:

MQSYSP_YES
RACF audit records are written.

MQSYSP_NO
RACF audit records are not written.

RoutingCode (MQCFIL)
z/OS routing code list (parameter identifier: MQIACF_SYSP_ROUTING_CODE).

Specifies the list of z/OS routing codes for messages that are not sent in direct response to an MQSC command. There can be in the range 1 through 16 entries in the list.

Service (MQCFST)
Service parameter setting (parameter identifier: MQCACF_SYSP_SERVICE).

The maximum length of the string is MQ_SERVICE_NAME_LENGTH.

SMFAccounting (MQCFIN)
Specifies whether IBM MQ sends accounting data to SMF automatically when the queue manager starts (parameter identifier: MQIACF_SYSP_SMF_ACCOUNTING).

The value can be any of the following values:

MQSYSP_YES
Accounting data is sent automatically.

MQSYSP_NO
Accounting data is not sent automatically.

SMFStatistics (MQCFIN)
Specifies whether IBM MQ sends statistics data to SMF automatically when the queue manager starts (parameter identifier: MQIACF_SYSP_SMF_STATS).

The value can be any of the following values:

MQSYSP_YES
Statistics data is sent automatically.

MQSYSP_NO
Statistics data is not sent automatically.

SMFInterval (MQCFIN)
The default time, in minutes, between each gathering of statistics (parameter identifier: MQIACF_SYSP_SMF_INTERVAL).

Splcap (MQCFIN)
If the AMS component is installed for the version of IBM MQ that the queue manager is running under, the attribute has a value YES (`MQCAP_SUPPORTED`). If the AMS component is not installed, the value is NO (`MQCAP_NOT_SUPPORTED`).

The value can be one of the following values:

MQCAP_SUPPORTED
If the AMS component is installed for the version of IBM MQ that the queue manager is running under.

MQCAP_NOT_SUPPORTED
If the AMS component is not installed.

TraceClass (MQCFIL)
Classes for which tracing is started automatically (parameter identifier: MQIACF_SYSP_TRACE_CLASS). There can be in the range 1 through 4 entries in the list.

TraceSize (MQCFIN)
The size of the trace table, in 4 KB blocks, to be used by the global trace facility (parameter identifier: MQIACF_SYSP_TRACE_SIZE).

WLMInterval (MQCFIN)
The time between scans of the queue index for WLM-managed queues (parameter identifier: MQIACF_SYSP_WLM_INTERVAL).

WLMIntervalUnits (MQCFIN)
Whether the value of *WLMInterval* is given in seconds or minutes (parameter identifier: MQIACF_SYSP_WLM_INT_UNITS). The value can be any of the following values:

MQTIME_UNITS_SEC
The value of *WLMInterval* is given in seconds.

MQTIME_UNITS_MINS
The value of *WLMInterval* is given in minutes.

Inquire Topic:

The Inquire Topic (MQCMD_INQUIRE_TOPIC) command inquires about the attributes of existing IBM MQ administrative topic objects

Required parameters

TopicName (MQCFST)
Administrative topic object name (parameter identifier: MQCA_TOPIC_NAME).

Specifies the name of the administrative topic object about which information is to be returned. Generic topic object names are supported. A generic name is a character string followed by an asterisk (*). For example, ABC* selects all administrative topic objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

Optional parameters

ClusterInfo (MQCFIN)
Cluster information (parameter identifier: MQIACF_CLUSTER_INFO).

This parameter requests that, in addition to information about attributes of topics defined on this queue manager, cluster information about these topics and other topics in the repository that match the selection criteria is returned.

In this case, there might be multiple topics with the same name returned.

You can set this parameter to any integer value: the value used does not affect the response to the command.

The cluster information is obtained locally from the queue manager.

z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *TopicAttrs* except MQIACF_ALL.

Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1601 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the **StringFilterCommand** parameter.

z/OS

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined as either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

You cannot use *QSGDisposition* as a parameter to filter on.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *TopicAttrs* except MQCA_TOPIC_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1608 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter.

TopicAttrs (MQCFIL)
Topic object attributes (parameter identifier: MQIACF_TOPIC_ATTRS).

The attribute list can specify the following value on its own - default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_ALTERATION_DATE
The date on which the information was last altered.

MQCA_ALTERATION_TIME
The time at which the information was last altered.

MQCA_CLUSTER_NAME
The cluster that is to be used for the propagation of publications and subscription to publish/subscribe cluster-connected queue managers for this topic.

MQCA_CLUSTER_DATE
The date on which this information became available to the local queue manager.

MQCA_CLUSTER_TIME
The time at which this information became available to the local queue manager.

MQCA_CLUSTER_Q_MGR_NAME
Queue manager that hosts the topic.

MQCA_CUSTOM
The custom attribute for new features.

MQCA_MODEL_DURABLE_Q
Name of the model queue for durable managed subscriptions.

MQCA_MODEL_NON_DURABLE_Q
Name of the model queue for non-durable managed subscriptions.

MQCA_TOPIC_DESC
Description of the topic object.

MQCA_TOPIC_NAME
Name of the topic object.

MQCA_TOPIC_STRING
The topic string for the topic object.

MQIA_CLUSTER_OBJECT_STATE
The current state of the clustered topic definition.

MQIA_CLUSTER_PUB_ROUTE
The routing behavior of publications between queue managers in a cluster.

MQIA_DEF_PRIORITY
Default message priority.

MQIA_DEF_PUT_RESPONSE_TYPE
Default put response.

MQIA_DURABLE_SUB
Whether durable subscriptions are permitted.

MQIA_INHIBIT_PUB
Whether publications are allowed.

MQIA_INHIBIT_SUB
Whether subscriptions are allowed.

MQIA_NPM_DELIVERY
The delivery mechanism for non-persistent messages.

MQIA_PM_DELIVERY
The delivery mechanism for persistent messages.

MQIA_PROXY_SUB
Whether a proxy subscription is to be sent for this topic, even if no local subscriptions exist.

MQIA_PUB_SCOPE
Whether this queue manager propagates publications to queue managers as part of a hierarchy or a publish/subscribe cluster.

MQIA_SUB_SCOPE
Whether this queue manager propagates subscriptions to queue managers as part of a hierarchy or a publish/subscribe cluster.

MQIA_TOPIC_DEF_PERSISTENCE
Default message persistence.

MQIA_USE_DEAD_LETTER_Q
Determines whether the dead-letter queue is used when publication messages cannot be delivered to their correct subscriber queue.

TopicType (MQCFIN)
Cluster information (parameter identifier: MQIA_TOPIC_TYPE).

If this parameter is present, eligible queues are limited to the specified type. Any attribute selector that is specified in the TopicAttrs list and that is valid only for topics of different type is ignored; no error is raised.

If this parameter is not present (or if MQIACF_ALL is specified), queues of all types are eligible. Each attribute specified must be a valid topic attribute selector (that is, it must be in the following list), but it need not be applicable to all or any of the topics returned. Topic attribute selectors that are valid but not applicable to the queue are ignored; no error messages occur and no attribute is returned.

The value can be any of the following values:

MQTOPT_ALL
All topic types are displayed. MQTOPT_ALL includes cluster topics, if ClusterInfo is also specified. MQTOPT_ALL is the default value.

MQTOPT_CLUSTER
Topics that are defined in publish/subscribe clusters are returned.

MQTOPT_LOCAL
Locally defined topics are displayed.

Inquire Topic (Response):

The response to the Inquire Topic (MQCMD_INQUIRE_TOPIC) command consists of the response header followed by the *TopicName* structure (and on z/OS only, the *QSG Disposition* structure), and the requested combination of attribute parameter structures (where applicable).

Always returned:

TopicName, *TopicType*, z/OS *QSGDisposition*

Returned if requested:

AlterationDate, *AlterationTime*, , *ClusterName*, *ClusterObjectState*, *ClusterPubRoute*, *CommInfo*, *Custom*, *DefPersistence*, *DefPriority*, *DefPutResponse*, *DurableModelQName*, *DurableSubscriptions*, *InhibitPublications*, *InhibitSubscriptions*, *Multicast*, *NonDurableModelQName*, *NonPersistentMsgDelivery*, *PersistentMsgDelivery*, *ProxySubscriptions*, *PublicationScope*, *QMgrName*, *SubscriptionScope*, *TopicDesc*, *TopicString*, *UseDLQ*, *WildcardOperation*

Response data

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered, in the form yyyy-mm-dd.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered, in the form hh.mm.ss.

ClusterName (MQCFST)
The name of the cluster to which this topic belongs. (parameter identifier: **MQCA_CLUSTER_NAME**).

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH. Setting this parameter to a cluster that this queue manager is a member of makes all queue managers in the cluster aware of this topic. Any publication to this topic or a topic string below it put to any queue manager in the cluster is propagated to subscriptions on any other queue manager in the cluster. For more details, see Distributed publish/subscribe networks.

The value can be any of the following values:

Blank If no topic object above this topic in the topic tree has set this parameter to a cluster name, then this topic does not belong to a cluster. Publications and subscriptions for this topic are not propagated to publish/subscribe cluster-connected queue managers. If a topic node higher in the topic tree has a cluster name set, publications and subscriptions to this topic are also propagated throughout the cluster.

This value is the default value for this parameter if no value is specified.

String The topic belongs to this cluster. It is not recommended that this is set to a different cluster from a topic object above this topic object in the topic tree. Other queue managers in the cluster will honor this object's definition unless a local definition of the same name exists on those queue managers.

Additionally, if **PublicationScope** or **SubscriptionScope** are set to MQSCOPE_ALL, this value is the cluster to be used for the propagation of publications and subscriptions, for this topic, to publish/subscribe cluster-connected queue managers.

ClusterObjectState (MQCFIN)
The current state of the clustered topic definition (parameter identifier: MQIA_CLUSTER_OBJECT_STATE).

The value can be any of the following values:

MQCLST_ACTIVE
The cluster topic is correctly configured and being adhered to by this queue manager.

MQCLST_PENDING
Only seen by a hosting queue manager, this state is reported when the topic has been created but the full repository has not yet propagated it to the cluster. This might be because the host queue manager is not connected to a full repository, or because the full repository has deemed the topic to be invalid.

MQCLST_INVALID
This clustered topic definition conflicts with an earlier definition in the cluster and is therefore not currently active.

MQCLST_ERROR
An error has occurred with respect to this topic object.

This parameter is typically used to aid diagnosis when multiple definitions of the same clustered topic are defined on different queue managers, and the definitions are not identical. See Routing for publish/subscribe clusters: Notes on behavior.

ClusterPubRoute (MQCFIN)
The routing behavior of publications between queue managers in a cluster (parameter identifier: MQIA_CLUSTER_PUB_ROUTE).

The value can be any of the following values:

MQCLROUTE_DIRECT
When you configure a direct routed clustered topic on a queue manager, all queue managers in the cluster become aware of all other queue managers in the cluster. When performing publish and subscribe operations, each queue manager can connect direct to any other queue manager in the cluster.

MQCLROUTE_TOPIC_HOST
When you use topic host routing, all queue managers in the cluster become aware of the cluster queue managers that host the routed topic definition (that is, the queue managers on which you have defined the topic object). When performing publish and subscribe operations, queue managers in the cluster connect only to these topic host queue managers, and not directly to each other. The topic host queue managers are responsible for routing publications from queue managers on which publications are published to queue managers with matching subscriptions.

CommInfo (MQCFST)
The name of the communication information object (parameter identifier: MQCA_COMM_INFO_NAME).

Shows the resolved value of the name of the communication information object to be used for this topic node.

The maximum length of the string is MQ_COMM_INFO_NAME_LENGTH.

Custom (MQCFST)
Custom attribute for new features (parameter identifier: MQCA_CUSTOM).

This attribute is reserved for the configuration of new features before separate attributes have been introduced. It can contain the values of zero or more attributes as pairs of attribute name and value, separated by at least one space. The attribute name-value pairs have the form `NAME(VALUE)`.

This description will be updated when features using this attribute are introduced.

DefPersistence (MQCFIN)
Default persistence (parameter identifier: MQIA_TOPIC_DEF_PERSISTENCE).

The value can be:

MQPER_PERSISTENCE_AS_PARENT
The default persistence is based on the setting of the closest parent administrative topic object in the topic tree.

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

DefPriority (MQCFIN)
Default priority (parameter identifier: MQIA_DEF_PRIORITY).

DefPutResponse (MQCFIN)
Default put response (parameter identifier: MQIA_DEF_PUT_RESPONSE_TYPE).

The value can be:

MQPRT_ASYNC_RESPONSE
The put operation is issued asynchronously, returning a subset of MQMD fields.

MQPRT_RESPONSE_AS_PARENT
The default put response is based on the setting of the closest parent administrative topic object in the topic tree.

MQPRT_SYNC_RESPONSE
The put operation is issued synchronously, returning a response.

DurableModelQName (MQCFST)
Name of the model queue to be used for durable managed subscriptions (parameter identifier: MQCA_MODEL_DURABLE_Q).

The maximum length of the string is MQ_Q_NAME_LENGTH.

DurableSubscriptions (MQCFIN)
Whether applications are permitted to make durable subscriptions (parameter identifier: MQIA_DURABLE_SUB).

The value can be:

MQSUB_DURABLE_AS_PARENT
Whether durable subscriptions are permitted is based on the setting of the closest parent administrative topic object in the topic tree.

MQSUB_DURABLE
Durable subscriptions are permitted.

MQSUB_NON_DURABLE
Durable subscriptions are not permitted.

InhibitPublications (MQCFIN)
Whether publications are allowed for this topic (parameter identifier: MQIA_INHIBIT_PUB).

The value can be:

MQTA_PUB_AS_PARENT
Whether messages can be published to this topic is based on the setting of the closest parent administrative topic object in the topic tree.

MQTA_PUB_INHIBITED
Publications are inhibited for this topic.

MQTA_PUB_ALLOWED
Publications are allowed for this topic.

InhibitSubscriptions (MQCFIN)
Whether subscriptions are allowed for this topic (parameter identifier: MQIA_INHIBIT_SUB).

The value can be:

MQTA_SUB_AS_PARENT
Whether applications can subscribe to this topic is based on the setting of the closest parent administrative topic object in the topic tree.

MQTA_SUB_INHIBITED
Subscriptions are inhibited for this topic.

MQTA_SUB_ALLOWED
Subscriptions are allowed for this topic.

Multicast (MQCFIN)
Whether multicast is used for this topic (parameter identifier: MQIA_MULTICAST).

Returned value:

MQMC_ENABLED
Multicast can be used.

MQMC_DISABLED
Multicast is not used.

MQMC_ONLY
Only Multicast publish/subscribe can be used on this topic.

NonDurableModelQName (MQCFST)
Name of the model queue to be used for non-durable managed subscriptions (parameter identifier: MQCA_MODEL_NON_DURABLE_Q).

The maximum length of the string is MQ_Q_NAME_LENGTH.

NonPersistentMsgDelivery (MQCFIN)
The delivery mechanism for non-persistent messages published to this topic (parameter identifier: MQIA_NPM_DELIVERY).

The value can be:

MQDLV_AS_PARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

MQDLV_ALL
Non-persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_DUR
Non-persistent messages must be delivered to all durable subscribers. Failure to deliver a non-persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_AVAIL
Non-persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

PersistentMsgDelivery (MQCFIN)
The delivery mechanism for persistent messages published to this topic (parameter identifier: MQIA_PM_DELIVERY).

The value can be:

MQDLV_AS_PARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

MQDLV_ALL
Persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_DUR
Persistent messages must be delivered to all durable subscribers. Failure to deliver a persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_AVAIL
Persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

ProxySubscriptions (MQCFIN)
Whether a proxy subscription is to be sent for this topic, even if no local subscriptions exist, to directly connected queue managers (parameter identifier: MQIA_PROXY_SUB).

The value can be:

MQTA_PROXY_SUB_FORCE
A proxy subscription is sent to connected queue managers even if no local subscriptions exist.

MQTA_PROXY_SUB_FIRSTUSE
A proxy subscription is sent for this topic only when a local subscription exists.

PublicationScope (MQCFIN)
Whether this queue manager propagates publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: MQIA_PUB_SCOPE).

The value can be:

MQSCOPE_ALL
Publications for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

MQSCOPE_AS_PARENT
Whether this queue manager propagates publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

MQSCOPE_AS_PARENT is the default value for this parameter if no value is specified.

MQSCOPE_QMGR
Publications for this topic are not propagated to other queue managers.

Note: You can override this behavior on a publication-by-publication basis, using MQPMO_SCOPE_QMGR on the Put Message Options.

QMgrName (MQCFST)
Name of local queue manager (parameter identifier: MQCA_CLUSTER_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH

SubscriptionScope (MQCFIN)
Whether this queue manager propagates subscriptions to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: MQIA_SUB_SCOPE).

The value can be:

MQSCOPE_ALL
Subscriptions for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

MQSCOPE_AS_PARENT
Whether this queue manager propagates subscriptions to queue managers as part of a hierarchy or as part of a publish/subscribe cluster is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

MQSCOPE_AS_PARENT is the default value for this parameter if no value is specified.

MQSCOPE_QMGR
Subscriptions for this topic are not propagated to other queue managers.

Note: You can override this behavior on a subscription-by-subscription basis, using MQSO_SCOPE_QMGR on the Subscription Descriptor or SUBSCOPE(QMGR) on DEFINE SUB.

TopicDesc (MQCFST)
Topic description (parameter identifier: MQCA_TOPIC_DESC).

The maximum length is MQ_TOPIC_DESC_LENGTH.

TopicName (MQCFST)
Topic object name (parameter identifier: MQCA_TOPIC_NAME).

The maximum length of the string is MQ_TOPIC_NAME_LENGTH

TopicString (MQCFST)
The topic string (parameter identifier: MQCA_TOPIC_STRING).

The '/' character within this string has special meaning. It delimits the elements in the topic tree. A topic string can start with the '/' character but is not required to. A string starting with the '/' character is not the same as the string which starts without the '/' character. A topic string cannot end with the "/" character.

The maximum length of the string is MQ_TOPIC_STR_LENGTH.

TopicType (MQCFIN)
Whether this object is a local or cluster topic (parameter identifier: MQIA_TOPIC_TYPE).

The value can be:

MQTOPT_LOCAL
This object is a local topic.

MQTOPT_CLUSTER
This object is a cluster topic.

UseDLQ (MQCFIN)
Whether the dead-letter queue (or undelivered message queue) should be used when publication messages cannote be delivered to their correct subscriber queue (parameter identifier: MQIA_USE_DEAD_LETTER_Q).

The value might be:

MQUSEDLQ_NO
Publication messages that cannot be delivered to their correct subscriber queue are treated as a failure to put the message and the application's MQPUT to a topic will fail in accordance with the settings of NPMSGDLV and PMSGDLV.

MQUSEDLQ_YES
If the queue manager DEADQ attribute provides the name of a dead-letter queue then it will be used, otherwise the behvaiour will be as for MQUSEDLQ_NO.

MQUSEDLQ_AS_PARENT
Whether to use the dead-letter queue is based on the setting of the closest administrative topic object in the topic tree.

WildcardOperation (MQCFIN)
Behavior of subscriptions including wildcards made to this topic (parameter identifier: MQIA_WILDCARD_OPERATION).

The value can be:

MQTA_PASSTHRU
Subscriptions made using wildcard topic names that are less specific than the topic string at this topic object receive publications made to this topic and to topic strings more specific than this topic. MQTA_PASSTHRU is the default supplied with IBM MQ.

MQTA_BLOCK
Subscriptions made using wildcard topic names that are less specific than the topic string at this topic object do not receive publications made to this topic or to topic strings more specific than this topic.

Inquire Topic Names:

The Inquire Topic Names (MQCMD_INQUIRE_TOPIC_NAMES) command inquires a list of administrative topic names that match the generic topic name specified.

Required parameters

TopicName (MQCFST)
Administrative topic object name (parameter identifier: MQCA_TOPIC_NAME).

Specifies the name of the administrative topic object that information is to be returned for.

Generic topic object names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

z/OS

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

Inquire Topic Names (Response):

The response to the Inquire Topic Names (MQCMD_INQUIRE_TOPIC_NAMES) command consists of the response header followed by a parameter structure giving zero or more names that match the specified administrative topic name.

z/OS
Additionally, on z/OS only, the **QSGDispositions** parameter structure (with the same number of entries as the *TopicNames* structure) is returned. Each entry in this structure indicates the disposition of the object with the corresponding entry in the *TopicNames* structure.

Always returned:
TopicNames, z/OS *QSGDispositions*

Returned if requested:
None

Response data

TopicNames (MQCFSL)
List of topic object names (parameter identifier: MQCACF_TOPIC_NAMES).

z/OS

QSGDispositions (MQCFIL)
List of queue-sharing group dispositions (parameter identifier: MQIACF_QSG_DISPS). This parameter is valid on z/OS only. The value can be:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

Inquire Topic Status:

The Inquire Topic Status (MQCMD_INQUIRE_TOPIC_STATUS) command inquires the status of a particular topic, or of a topic and its child topics. The Inquire Topic Status command has a required parameter. The Inquire Topic Status command has optional parameters.

Required parameters

TopicString (MQCFST)
The topic string (parameter identifier: MQCA_TOPIC_STRING).

The name of the topic string to display. IBM MQ uses the topic wildcard characters ('#' and '+') and does not treat a trailing asterisk as a wildcard. For more information about using wildcard characters, refer to the related topic.

The maximum length of the string is MQ_TOPIC_STR_LENGTH.

Optional parameters

z/OS **CommandScope (MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- Blank (or omit the parameter altogether). The command runs on the queue manager on which you enter it.
- A queue manager name. The command runs on the queue manager that you specify, if it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which you entered the command, you must be using a queue-sharing group environment, and the command server must be enabled.
- An asterisk (*). The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use CommandScope as a filter parameter.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor that you use to restrict the output from the command. The parameter identifier must be an integer type and must be one of the values allowed for *MQIACF_TOPIC_SUB_STATUS*, *MQIACF_TOPIC_PUB_STATUS* or *MQIACF_TOPIC_STATUS*, except *MQIACF_ALL*.

If you specify an integer filter, you cannot also specify a string filter with the **StringFilterCommand** parameter.

StatusType (MQCFIN)
The type of status to return (parameter identifier: MQIACF_TOPIC_STATUS_TYPE).

The value can be:

MQIACF_TOPIC_STATUS

MQIACF_TOPIC_SUB

MQIACF_TOPIC_PUB

This command ignores any attribute selectors specified in the *TopicStatusAttrs* list that are not valid for the selected *StatusType* and the command raises no error.

The default value if this parameter is not specified is **MQIACF_TOPIC_STATUS**.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed for *MQIACF_TOPIC_SUB_STATUS*, *MQIACF_TOPIC_PUB_STATUS* or *MQIACF_TOPIC_STATUS*, except *MQIACF_ALL*, or the identifier *MQCA_TOPIC_STRING_FILTER* to filter on the topic string.

Use the parameter identifier to restrict the output from the command by specifying a filter condition. Ensure that the parameter is valid for the type selected in StatusType. If you specify a string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter.

TopicStatusAttrs (MQCFIL)
Topic status attributes (parameter identifier: MQIACF_TOPIC_STATUS_ATTRS)

The default value used if the parameter is not specified is:

MQIACF_ALL

You can specify any of the parameter values listed in "Inquire Topic Status (Response)." It is not an error to request status information that is not relevant for a particular status type, but the response contains no information for the value concerned.

Inquire Topic Status (Response):

The response of the Inquire topic (MQCMD_INQUIRE_TOPIC_STATUS) command consists of the response header, followed by the *TopicString* structure, and the requested combination of attribute parameter structures (where applicable). The Inquire Topic Status command returns the values requested when the *StatusType* is MQIACF_TOPIC_STATUS. The Inquire Topic Status command returns the values requested when the *StatusType* is MQIACF_TOPIC_STATUS_SUB. The Inquire Topic Status command returns the values requested when the *StatusType* is MQIACF_TOPIC_STATUS_PUB.

Always returned:
TopicString

Returned if requested and StatusType is MQIACF_TOPIC_STATUS:
Cluster, *ClusterPubRoute*, *CommInfo*, *DefPriority*, *DefaultPutResponse*, *DefPersistence*, *DurableSubscriptions*, *InhibitPublications*, *InhibitSubscriptions*, *AdminTopicName*, *Multicast*, *DurableModelQName*, *NonDurableModelQName*, *PersistentMessageDelivery*, *NonPersistentMessageDelivery*, *RetainedPublication*, *PublishCount*, *SubscriptionScope*, *SubscriptionCount*, *PublicationScope*, *UseDLQ*

Note: The Inquire Topic Status command returns only resolved values for the topic, and no AS_PARENT values.

Returned if requested and StatusType is MQIACF_TOPIC_SUB:
SubscriptionId, *SubscriptionUserId*, *Durable*, *SubscriptionType*, *ResumeDate*, *ResumeTime*, *LastMessageDate*, *LastMessageTime*, *NumberOfMessages*, *ActiveConnection*

Returned if requested and StatusType is MQIACF_TOPIC_PUB:
LastPublishDate, *LastPublishTime*, *NumberOfPublishes*, *ActiveConnection*

Response data (TOPIC_STATUS)

ClusterName (MQCFST)
The name of the cluster to which this topic belongs. (parameter identifier: **MQCA_CLUSTER_NAME**).

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH. Setting this parameter to a cluster that this queue manager is a member of makes all queue managers in the cluster aware of this topic. Any publication to this topic or a topic string below it put to any queue manager in the cluster is propagated to subscriptions on any other queue manager in the cluster. For more details, see Distributed publish/subscribe networks.

The value can be any of the following values:

Blank If no topic object above this topic in the topic tree has set this parameter to a cluster name, then this topic does not belong to a cluster. Publications and subscriptions for this topic are not propagated to publish/subscribe cluster-connected queue managers. If a topic node higher in the topic tree has a cluster name set, publications and subscriptions to this topic are also propagated throughout the cluster.

This value is the default value for this parameter if no value is specified.

String The topic belongs to this cluster. It is not recommended that this is set to a different cluster from a topic object above this topic object in the topic tree. Other queue managers in the cluster will honor this object's definition unless a local definition of the same name exists on those queue managers.

Additionally, if **PublicationScope** or **SubscriptionScope** are set to MQSCOPE_ALL, this value is the cluster to be used for the propagation of publications and subscriptions, for this topic, to publish/subscribe cluster-connected queue managers.

ClusterPubRoute (MQCFIN)
The routing behavior to use for this topic in the cluster (parameter identifier: MQIA_CLUSTER_PUB_ROUTE).

The values can be as follows:

MQCLROUTE_DIRECT
A publication on this topic string, originating from this queue manager, is sent direct to any queue manager in the cluster with a matching subscription.

MQCLROUTE_TOPIC_HOST
A publication on this topic string, originating from this queue manager, is sent to one of the queue managers in the cluster that hosts a definition of the corresponding clustered topic object, and from there to any queue manager in the cluster with a matching subscription.

MQCLROUTE_NONE
This topic node is not clustered.

CommInfo (MQCFST)
The name of the communication information object (parameter identifier: MQCA_COMM_INFO_NAME).

Shows the resolved value of the name of the communication information object to be used for this topic node.

The maximum length of the string is MQ_COMM_INFO_NAME_LENGTH.

DefPersistence (MQCFIN)
Default persistence (parameter identifier: MQIA_TOPIC_DEF_PERSISTENCE).

Returned value:

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

DefaultPutResponse (MQCFIN)
Default put response (parameter identifier: MQIA_DEF_PUT_RESPONSE_TYPE).

Returned value:

MQPRT_SYNC_RESPONSE
The put operation is issued synchronously, returning a response.

MQPRT_ASYNC_RESPONSE
The put operation is issued asynchronously, returning a subset of MQMD fields.

DefPriority (MQCFIN)
Default priority (parameter identifier: MQIA_DEF_PRIORITY).

Shows the resolved default priority of messages published to the topic.

DurableSubscriptions (MQCFIN)
Whether applications are permitted to make durable subscriptions (parameter identifier: MQIA_DURABLE_SUB).

Returned value:

MQSUB_DURABLE_ALLOWED
Durable subscriptions are permitted.

MQSUB_DURABLE_INHIBITED
Durable subscriptions are not permitted.

InhibitPublications (MQCFIN)
Whether publications are allowed for this topic (parameter identifier: MQIA_INHIBIT_PUB).

Returned value:

MQTA_PUB_INHIBITED
Publications are inhibited for this topic.

MQTA_PUB_ALLOWED
Publications are allowed for this topic.

InhibitSubscriptions (MQCFIN)
Whether subscriptions are allowed for this topic (parameter identifier: MQIA_INHIBIT_SUB).

Returned value:

MQTA_SUB_INHIBITED
Subscriptions are inhibited for this topic.

MQTA_SUB_ALLOWED
Subscriptions are allowed for this topic.

AdminTopicName (MQCFST)
Topic object name (parameter identifier: MQCA_ADMIN_TOPIC_NAME).

If the topic is an admin-node, the command displays the associated topic object name containing the node configuration. If the field is not an admin-node the command displays a blank.

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

Multicast (MQCFIN)
Whether multicast is used for this topic (parameter identifier: MQIA_MULTICAST).

Returned value:

MQMC_ENABLED
Multicast can be used.

MQMC_DISABLED
Multicast is not used.

MQMC_ONLY
Only Multicast publish/subscribe can be used on this topic.

DurableModelQName (MQCFST)
The name of the model queue used for managed durable subscriptions (parameter identifier: MQCA_MODEL_DURABLE_Q).

Shows the resolved value of the name of the model queue to be used for durable subscriptions that request the queue manager to manage the destination of publications.

The maximum length of the string is MQ_Q_NAME_LENGTH.

NonDurableModelQName (MQCFST)
The name of the model queue for managed non-durable subscriptions (parameter identifier: MQCA_MODEL_NON_DURABLE_Q).

The maximum length of the string is MQ_Q_NAME_LENGTH.

PersistentMessageDelivery (MQCFIN)
Delivery mechanism for persistent messages published to this topic (parameter identifier: MQIA_PM_DELIVERY).

Returned value:

MQDLV_ALL
Persistent messages must be delivered to all subscribers, irrespective of durability, for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT call fails.

MQDLV_ALL_DUR
Persistent messages must be delivered to all durable subscribers. Failure to deliver a persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no subscribers receive the message and the MQPUT call fails.

MQDLV_ALL_AVAIL
Persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

NonPersistentMessageDelivery (MQCFIN)
Delivery mechanism for non-persistent messages published to this topic (parameter identifier: MQIA_NPM_DELIVERY).

Returned value:

MQDLV_ALL
Non-persistent messages must be delivered to all subscribers, irrespective of durability, for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT call fails.

MQDLV_ALL_DUR
Non-persistent messages must be delivered to all durable subscribers. Failure to deliver a non-persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no subscribers receive the message and the MQPUT call fails.

MQDLV_ALL_AVAIL
Non-persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

RetainedPublication (MQCFIN)
Whether there is a retained publication for this topic (parameter identifier: MQIACF_RETAINED_PUBLICATION).

Returned value:

MQQSO_YES
There is a retained publication for this topic.

MQQSO_NO
There is no retained publication for this topic.

PublishCount (MQCFIN)
Publish count (parameter identifier: MQIA_PUB_COUNT).

The number of applications currently publishing to the topic.

SubscriptionCount (MQCFIN)
Subscription count (parameter identifier: MQIA_SUB_COUNT).

The number of subscribers for this topic string, including durable subscribers who are not currently connected.

SubscriptionScope (MQCFIN)
Determines whether this queue manager propagates subscriptions for this topic to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: MQIA_SUB_SCOPE).

Returned value:

MQSCOPE_QMGR
The queue manager does not propagate subscriptions for this topic to other queue managers.

MQSCOPE_ALL
The queue manager propagates subscriptions for this topic to hierarchically connected queue managers and to publish/subscribe cluster connected queues.

PublicationScope (MQCFIN)
Determines whether this queue manager propagates publications for this topic to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: MQIA_PUB_SCOPE).

Returned value:

MQSCOPE_QMGR
The queue manager does not propagate publications for this topic to other queue managers.

MQSCOPE_ALL
The queue manager propagates publications for this topic to hierarchically connected queue managers and to publish/subscribe cluster connected queues.

UseDLQ (MQCFIN)
Determines whether the dead-letter queue is used when publication messages cannot be delivered to their correct subscriber queue (parameter identifier: MQIA_USE_DEAD_LETTER_Q).

The value can be any of the following values:

MQUSEDLQ_NO
Publication messages that cannot be delivered to their correct subscriber queue are treated as a failure to put the message. The MQPUT of an application to a topic fails in accordance with the settings of MQIA_NPM_DELIVERY and MQIA_PM_DELIVERY.

MQUSEDLQ_YES
If the DEADQ queue manager attribute provides the name of a dead-letter queue then it is used, otherwise the behavior is as for MQUSEDLQ_NO.

Response data (TOPIC_STATUS_SUB)

SubscriptionId (MQCFBS)
Subscription identifier (parameter identifier: MQBACF_SUB_ID).

The queue manager assigns *SubscriptionId* as an all time unique identifier for this subscription.

The maximum length of the string is MQ_CORREL_ID_LENGTH.

SubscriptionUserId (MQCFST)
The user ID that owns this subscription (parameter identifier: MQCACF_SUB_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH.

Durable (MQCFIN)
Whether this subscription is a durable subscription (parameter identifier: MQIACF_DURABLE_SUBSCRIPTION).

MQSUB_DURABLE_YES
The subscription persists, even if the creating application disconnects from the queue manager or issues an MQCLOSE call for the subscription. The queue manager reinstates the subscription during restart.

MQSUB_DURABLE_NO
The subscription is non-durable. The queue manager removes the subscription when the creating application disconnects from the queue manager or issues an MQCLOSE call for the subscription. If the subscription has a destination class (DESTCLAS) of MANAGED, the queue manager removes any messages not yet consumed when it closes the subscription.

SubscriptionType (MQCFIN)
The type of subscription (parameter identifier: MQIACF_SUB_TYPE).

The value can be:

MQSUBTYPE_ADMIN

MQSUBTYPE_API

MQSUBTYPE_PROXY

ResumeDate (MQCFST)
Date of the most recent MQSUB call that connected to this subscription (parameter identifier: MQCA_RESUME_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

ResumeTime (MQCFST)
Time of the most recent MQSUB call that connected to this subscription (parameter identifier: MQCA_RESUME_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

LastMessageDate (MQCFST)
Date on which an MQPUT call last sent a message to this subscription. The queue manager updates the date field after the MQPUT call successfully puts a message to the destination specified by this subscription (parameter identifier: MQCACF_LAST_MSG_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

Note: An **MQSUBRQ** call updates this value.

LastMessageTime (MQCFST)
Time at which an MQPUT call last sent a message to this subscription. The queue manager updates the time field after the MQPUT call successfully puts a message to the destination specified by this subscription (parameter identifier: MQCACF_LAST_MSG_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

Note: An **MQSUBRQ** call updates this value.

NumberOfMessages (MQCFIN)
Number of messages put to the destination specified by this subscription (parameter identifier: MQIACF_MESSAGE_COUNT).

Note: An **MQSUBRQ** call updates this value.

ActiveConnection (MQCFBS)
The currently active *ConnectionId* (CONNID) that opened this subscription (parameter identifier: MQBACF_CONNECTION_ID).

The maximum length of the string is MQ_CONNECTION_ID_LENGTH.

Response data (TOPIC_STATUS_PUB)

LastPublicationDate (MQCFST)
Date on which this publisher last sent a message (parameter identifier: MQCACF_LAST_PUB_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

LastPublicationTime (MQCFST)
Time at which this publisher last sent a message (parameter identifier: MQCACF_LAST_PUB_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

NumberOfPublishes (MQCFIN)
Number of publishes made by this publisher (parameter identifier: MQIACF_PUBLISH_COUNT).

ActiveConnection (MQCFBS)
The currently active *ConnectionId* (CONNID) associated with the handle that has this topic open for publish (parameter identifier: MQBACF_CONNECTION_ID).

The maximum length of the string is MQ_CONNECTION_ID_LENGTH.

Inquire Usage on z/OS: z/OS

The Inquire Usage (MQCMD_INQUIRE_USAGE) command inquires about the current state of a page set, or information about the log data sets.

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

PageSetId (MQCFIN)
Page set identifier (parameter identifier: MQIA_PAGESET_ID). If you omit this parameter, all page set identifiers are returned.

UsageType (MQCFIN)
The type of information to be returned (parameter identifier: MQIACF_USAGE_TYPE).

The value can be any of the following values:

MQIACF_USAGE_PAGESET
Return page set (MQIACF_USAGE_PAGESET) and buffer pool information (MQIACF_USAGE_BUFFER_POOL).

MQIACF_USAGE_DATA_SET
Return data set information for log data sets (MQIACF_USAGE_DATA_SET).

MQIACF_ALL
Return page set, buffer pool, and data set information (MQIACF_USAGE_PAGESET), (MQIACF_USAGE_BUFFER_POOL), and (MQIACF_USAGE_DATA_SET).

MQIACF_USAGE_SMDS
Return shared message data set usage (MQIACF_USAGE_SMDS) and buffer pool information (MQIACF_USAGE_BUFFER_POOL).

This includes the allocated, and used space for each data set, and information about the number of buffers currently active, the number with valid contents, and the number of free buffers.

Inquire Usage (Response) on z/OS: z/OS

The response to the Inquire Usage (MQCMD_INQUIRE_USAGE) command consists of the response header followed by one or more *UsageType* structure and a set of attribute parameter structures determined by the value of *UsageType* in the Inquire command.

Always returned:
UsageType

Possible values of *ParameterType* are:

MQIACF_USAGE_PAGESET
Page set information.

MQIACF_USAGE_BUFFER_POOL
Buffer pool information.

MQIACF_USAGE_DATA_SET
Data set information for log data sets.

MQIACF_USAGE_SMDS
Return shared message data set usage and buffer pool information.

This includes the allocated, and used space for each data set, and information about the number of buffers currently active, the number with valid contents, and the number of free buffers.

Returned if *UsageType* is MQIACF_USAGE_PAGESET:
BufferPoolId, *ExpandCount*, *ExpandType*, *LogRBA*, *NonPersistentDataPages*, *PageSetId*, *PageSetStatus*, *PersistentDataPages*, *TotalPages*, *UnusedPages*

Returned if *UsageType* is MQIACF_USAGE_BUFFER_POOL:
BufferPoolId, *FreeBuffers*, *FreeBuffersPercentage*, *TotalBuffers*, *BufferPoolLocation*, *PageClass*

Returned if *UsageType* is MQIACF_USAGE_DATA_SET:
DataSetName, *DataSetType*, *LogRBA*, *LogLRSN*

Returned if *UsageType* is MQIACF_USAGE_SMDS:
DataSetName, *DataSetType*

Response data if UsageType is MQIACF_USAGE_PAGESET

BufferPoolId (MQCFIN)
Buffer pool identifier (parameter identifier: MQIACF_BUFFER_POOL_ID).

This parameter identifies the buffer pool being used by the page set.

ExpandCount (MQCFIN)
The number of times the page set has been dynamically expanded since restart (parameter identifier: MQIACF_USAGE_EXPAND_COUNT).

ExpandType (MQCFIN)
How the queue manager expands a page set when it becomes nearly full, and further pages are required within it (parameter identifier: MQIACF_USAGE_EXPAND_TYPE).

The value can be:

MQUSAGE_EXPAND_NONE
No further page set expansion is to take place.

MQUSAGE_EXPAND_USER
The secondary extent size that was specified when the page set was defined is used. If no secondary extent size was specified, or it was specified as zero, then no dynamic page set expansion can take place.

At restart, if a previously used page set has been replaced with a data set that is smaller, it is expanded until it reaches the size of the previously used data set. Only one extent is required to reach this size.

MQUSAGE_EXPAND_SYSTEM
A secondary extent size that is approximately 10 per cent of the current size of the page set is used. MQUSAGE_EXPAND_SYSTEM can be rounded up to the nearest cylinder of DASD.

NonPersistentDataPages (MQCFIN)
The number of pages holding nonpersistent data (parameter identifier: MQIACF_USAGE_NONPERSIST_PAGES).

These pages are being used to store nonpersistent message data.

PageSetId (MQCFIN)
Page set identifier (parameter identifier: MQIA_PAGESET_ID).

The string consists of two numeric characters, in the range 00 through 99.

PageSetStatus (MQCFIN)
Current status of the page set (parameter identifier: MQIACF_PAGESET_STATUS).

The value can be any of the following values:

MQUSAGE_PS_AVAILABLE
The page set is available.

MQUSAGE_PS_DEFINED
The page set has been defined but has never been used.

MQUSAGE_PS_OFFLINE
The page set is currently not accessible by the queue manager, for example because the page set has not been defined to the queue manager.

MQUSAGE_PS_NOT_DEFINED
The command was issued for a specific page set that is not defined to the queue manager.

MQUSAGE_PS_SUSPENDED
The page set has been suspended. For further information about suspended page sets, see message CSQP059E.

PersistentDataPages (MQCFIN)
The number of pages holding persistent data (parameter identifier: MQIACF_USAGE_PERSIST_PAGES).

These pages are being used to store object definitions and persistent message data.

TotalPages (MQCFIN)
The total number of 4 KB pages in the page set (parameter identifier: MQIACF_USAGE_TOTAL_PAGES).

UnusedPages (MQCFIN)
The number of pages that are not used (that is, available page sets) (parameter identifier: MQIACF_USAGE_UNUSED_PAGES).

LogRBA (MQCFST)
Log RBA (parameter identifier: MQCACF_USAGE_LOG_RBA).

The maximum length is MQ_RBA_LENGTH.

This response is returned only if PageSetStatus is set to MQUSAGE_PS_NOT_DEFINED or MQUSAGE_SUSPENDED. However, the response is not always returned if PageSetStatus is set to MQUSAGE_PS_NOT_DEFINED.

A value of 'FFFFFFFFFFFFFFFF' indicates that the page set has never been online.

Response data if UsageType is MQIACF_USAGE_BUFFER_POOL

BufferPoolId (MQCFIN)
Buffer pool identifier (parameter identifier: MQIACF_BUFFER_POOL_ID).

This parameter identifies the buffer pool being used by the page set.

FreeBuffers (MQCFIN)
Number of free buffers (parameter identifier: MQIACF_USAGE_FREE_BUFF).

FreeBuffersPercentage (MQCFIN)
Number of free buffers as a percentage of all buffers in the buffer pool (parameter identifier: MQIACF_USAGE_FREE_BUFF_PERC).

TotalBuffers (MQCFIN)
The number of buffers defined for specified buffer pool (parameter identifier: MQIACF_USAGE_TOTAL_BUFFERS).

BufferPoolLocation (MQCFIN)
The location of the buffers in this buffer pool relative to the bar. This is one of the following values:

MQBPLOCATION_ABOVE
All buffer pool buffers are above the bar.

MQBPLOCATION_BELOW
All buffer pool buffers are below the bar.

MQBPLOCATION_SWITCHING_ABOVE
Buffer pool buffers are being moved above the bar.

MQBPLOCATION_SWITCHING_BELOW
Buffer pool buffers are being moved below the bar.

PageClass (MQCFIN)
The type of virtual storage pages used for backing the buffers in the buffer pool. This is one of the following values:

MQPAGECLAS_4KB
Pageable 4 KB pages are used.

MQPAGECLAS_FIXED4KB
Fixed 4 KB pages are used.

Response data if UsageType is MQIACF_USAGE_DATA_SET

DataSetName (MQCFST)
Data set name (parameter identifier: MQCACF_DATA_SET_NAME).

The maximum length is MQ_DATA_SET_NAME_LENGTH.

DataSetType (MQCFIN)
The type of data set, and circumstance (parameter identifier: MQIACF_USAGE_DATA_SET_TYPE).

The value can be:

MQUSAGE_DS_OLDEST_ACTIVE_UOW
The log data set containing the start RBA of the oldest active unit of work for the queue manager

MQUSAGE_DS_OLDEST_PS_RECOVERY
The log data set containing the oldest restart RBA of any page set for the queue manager.

MQUSAGE__DS_OLDEST_CF_RECOVERY
The log data set containing the LRSN which matches the time of the oldest current backup of any CF structure in the queue-sharing group.

LogRBA (MQCFST)
Log RBA (parameter identifier: MQCACF_USAGE_LOG_RBA).

The maximum length is MQ_RBA_LENGTH.

LogLRSN (MQCFST)
Log LRSN (parameter identifier: MQIACF_USAGE_LOG_LRSN).

The length of the string is MQ_LRSN_LENGTH.

Response data if UsageType is MQIACF_USAGE_SMDS

SMDSStatus (MQCFIN)
SMDS status (parameter identifier: MQIACF_SMDS_STATUS).

MQUSAGE_SMDS_NO_DATA
There is no SMDS data available. Nothing further is returned.

MQUSAGE_SMDS_AVAILABLE
For each CF structure two sets of PCF data are returned:

A

CFStrucNames (MQCFSL)
List of CF application structure names (parameter identifier: MQCACF_CF_STRUC_NAME).

MQIACF_USAGE_OFFLOAD_MSGS (MQCFIN)
Description required (parameter identifier: MQIACF_USAGE_OFFLOAD_MSGS).

MQIACF_USAGE_TOTAL_BLOCKS (MQCFIN)
Description required (parameter identifier: MQIACF_USAGE_TOTAL_BLOCKS).

MQIACF_USAGE_DATA_BLOCKS (MQCFIN)
Description required (parameter identifier: MQIACF_USAGE_DATA_BLOCKS).

MQIACF_USAGE_USED_BLOCKS (MQCFIN)
Description required (parameter identifier: MQIACF_USAGE_USED_BLOCKS).

MQIACF_USAGE_USED_RATE (MQCFIN)
Description required (parameter identifier: MQIACF_USAGE_USED_RATE).

MQIACF_SMDS_STATUS (MQCFIN)
Description required (parameter identifier: MQIACF_SMDS_STATUS). The value is MQUSAGE_SMDS_AVAILABLE.

MQIACF_USAGE_TYPE (MQCFIN)
Description required (parameter identifier: MQIACF_USAGE_TYPE).

B

CFStrucNames (MQCFSL)
List of CF application structure names (parameter identifier: MQCACF_CF_STRUC_NAME).

MQIACF_USAGE_BLOCK_SIZE (MQCFIN)
Description required (parameter identifier: MQIACF_USAGE_BLOCK_SIZE).

MQIACF_USAGE_TOTAL_BUFFERS (MQCFIN)
Description required (parameter identifier: MQIACF_USAGE_TOTAL_BUFFERS).

MQIACF_USAGE_INUSE_BUFFERS (MQCFIN)
Description required (parameter identifier: MQIACF_USAGE_INUSE_BUFFERS).

MQIACF_USAGE_SAVED_BUFFERS (MQCFIN)
Description required (parameter identifier: MQIACF_USAGE_SAVED_BUFFERS).

MQIACF_USAGE_EMPTY_BUFFERS (MQCFIN)
Description required (parameter identifier: MQIACF_USAGE_EMPTY_BUFFERS).

MQIACF_USAGE_READS_SAVED (MQCFIN)
Description required (parameter identifier: MQIACF_USAGE_READS_SAVED).

MQIACF_USAGE_LOWEST_FREE (MQCFIN)
Description required (parameter identifier: MQIACF_USAGE_LOWEST_FREE).

MQIACF_USAGE_WAIT_RATE (MQCFIN)
Description required (parameter identifier: MQIACF_USAGE_WAIT_RATE).

MQIACF_SMDS_STATUS (MQCFIN)
Description required (parameter identifier: MQIACF_SMDS_STATUS). The value is MQUSAGE_SMDS_AVAILABLE.

MQIACF_USAGE_TYPE (MQCFIN)
Description required (parameter identifier: MQIACF_USAGE_TYPE).

Move Queue on z/OS: z/OS

The Move Queue (MQCMD_MOVE_Q) command moves all the messages from one local queue to another.

Required parameters

FromQName (MQCFST)
From queue name (parameter identifier: MQCACF_FROM_Q_NAME).

The name of the local queue from which messages are moved. The name must be defined to the local queue manager.

The command fails if the queue contains uncommitted messages.

If an application has this queue open, or has open a queue that eventually resolves to this queue, the command fails. For example, the command fails if this queue is a transmission queue, and any queue that is, or resolves to, a remote queue that references this transmission queue, is open.

An application can open this queue while the command is in progress but the application waits until the command has completed.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Optional parameters (Move Queue)

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_QSG_NAME_LENGTH.

MoveType (MQCFIN)
Move type (parameter identifier: MQIA_QSG_DISP).

Specifies how the messages are moved. The value can be any of the following values:

MQIACF_MOVE_TYPE_MOVE
Move the messages from the source queue to the empty target queue.

The command fails if the target queue already contains one or more messages. The messages are deleted from the source queue. MQIACF_MOVE_TYPE_MOVE is the default value.

MQIACF_MOVE_TYPE_ADD
Move the messages from the source queue and add them to any messages already on the target queue.

The messages are deleted from the source queue.

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be any of the following values:

MQQSGD_PRIVATE
The object is defined as either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE is the default value.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED. MQQSGD_SHARED is valid only in a shared queue environment.

ToQName (MQCFST)
To queue name (parameter identifier: MQCACF_TO_Q_NAME).

The name of the local queue to which messages are moved. The name must be defined to the local queue manager.

The name of the target queue can be the same as the name of the source queue only if the queue exists as both a shared and a private queue. In this case, the command moves messages to the queue that has the opposite disposition (shared or private) from that disposition specified for the source queue on the **QSGDisposition** parameter.

If an application has this queue open, or has open a queue that eventually resolves to this queue, the command fails. The command also fails if this queue is a transmission queue, and any queue that is, or resolves to, a remote queue that references this transmission queue, is open.

No application can open this queue while the command is in progress.

If you specify a value of MQIACF_MOVE_TYPE_MOVE on the **MoveType** parameter, the command fails if the target queue already contains one or more messages.

The **DefinitionType**, **HardenGetBackout**, **Usage** parameters of the target queue must be the same as those parameters of the source queue.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Ping Channel:

The Ping Channel (MQCMD_PING_CHANNEL) command tests a channel by sending data as a special message to the remote message queue manager and checking that the data is returned. The data is generated by the local queue manager.

This command can only be used for channels with a *ChannelType* value of MQCHT_SENDER, MQCHT_SERVER, or MQCHT_CLUSSDR.

Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel.

If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the last channel added to the repository on the local queue manager.

The command is not valid if the channel is running; however it is valid if the channel is stopped or in retry mode.

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be tested. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

DataCount (MQCFIN)
Data count (parameter identifier: MQIACH_DATA_COUNT).

Specifies the length of the data.

Specify a value in the range 16 through 32 768. The default value is 64 bytes.

z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- a queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

z/OS

ChannelDisposition (MQCFIN)
Channel disposition (parameter identifier: MQIACH_CHANNEL_DISP). This parameter applies to z/OS only.

Specifies the disposition of the channels to be tested.

If this parameter is omitted, then the value for the channel disposition is taken from the default channel disposition attribute of the channel object.

The value can be any of the following values:

MQCHLD_PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than MQQSGD_SHARED.

MQCHLD_SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of MQQSGD_SHARED.

MQCHLD_FIXSHARED
Tests shared channels, tied to a specific queue manager.

The combination of the **ChannelDisposition** and **CommandScope** parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.

- On the most suitable queue manager in the group, determined automatically by the queue manager itself.

The various combinations of *ChannelDisposition* and *CommandScope* are summarized in Table 133

Table 133. ChannelDisposition and CommandScope for PING CHANNEL

ChannelDisposition	*CommandScope* **blank or local-qmgr**	*CommandScope* **qmgr-name**	*CommandScope(*)*
MQCHLD_PRIVATE	Ping private channel on the local queue manager	Ping private channel on the named queue manager	Ping private channel on all active queue managers
MQCHLD_SHARED	Ping a shared channel on the most suitable queue manager in the group MQCHLD_SHARED might automatically generate a command using *CommandScope* and send it to the appropriate queue manager. If there is no definition for the channel on the queue manager to which the command is sent, or if the definition is unsuitable for the command, the command fails. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted	Not permitted
MQCHLD_FIXSHARED	Ping a shared channel on the local queue manager	Ping a shared channel on the named queue manager	Not permitted

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_ALLOCATE_FAILED
Allocation failed.

MQRCCF_BIND_FAILED
Bind failed.

MQRCCF_CCSID_ERROR
Coded character-set identifier error.

MQRCCF_CHANNEL_CLOSED
Channel closed.

MQRCCF_CHANNEL_IN_USE
Channel in use.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_CONFIGURATION_ERROR
Configuration error.

MQRCCF_CONNECTION_CLOSED
Connection closed.

MQRCCF_CONNECTION_REFUSED
Connection refused.

MQRCCF_DATA_TOO_LARGE
Data too large.

MQRCCF_ENTRY_ERROR
Connection name not valid.

MQRCCF_HOST_NOT_AVAILABLE
Remote system not available.

MQRCCF_NO_COMMS_MANAGER
Communications manager not available.

MQRCCF_PING_DATA_COMPARE_ERROR
Ping Channel command failed.

MQRCCF_PING_DATA_COUNT_ERROR
Data count not valid.

MQRCCF_PING_ERROR
Ping error.

MQRCCF_RECEIVE_FAILED
Receive failed.

MQRCCF_RECEIVED_DATA_ERROR
Received data error.

MQRCCF_REMOTE_QM_TERMINATING
Remote queue manager terminating.

MQRCCF_REMOTE_QM_UNAVAILABLE
Remote queue manager not available.

MQRCCF_SEND_FAILED
Send failed.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

MQRCCF_TERMINATED_BY_SEC_EXIT
Channel terminated by security exit.

MQRCCF_UNKNOWN_REMOTE_CHANNEL
Remote channel not known.

MQRCCF_USER_EXIT_NOT_AVAILABLE
User exit not available.

Ping Queue Manager on Multiplatforms: Multi

The Ping Queue Manager (MQCMD_PING_Q_MGR) command tests whether the queue manager and its command server is responsive to commands. If the queue manager is responding a positive reply is returned.

Required parameters:
None

Optional parameters:
None

Purge Channel: AIX Linux Windows

The Purge Channel (MQCMD_PURGE_CHANNEL) command stops and purges an IBM MQ telemetry channel.

This command can only be issued to an MQTT channel type.

Purging a telemetry channel disconnects all the MQTT clients connect to it, cleans up the state of the MQTT clients, and stops the telemetry channel. Cleaning the state of a client deletes all the pending publications and removes all the subscriptions from the client.

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be stopped and purged. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType (MQCFIN)
Channel type. This parameter must follow immediately after the **ChannelName** parameter, and the value must be MQTT.

Optional parameters

ClientIdentifier (MQCFST)
Client identifier. The client identifier is a 23-byte string that identifies an MQ Telemetry Transport client. When the Purge Channel command specifies a *ClientIdentifier*, only the connection for the specified client identifier is purged. If the *ClientIdentifier* is not specified, all the connections on the channel are purged.

The maximum length of the string is MQ_CLIENT_ID_LENGTH.

Recover CF Structure on z/OS: z/OS

The Recover CF Structure (MQCMD_RECOVER_CF_STRUC) command initiates recovery of CF application structures.

Note: This command is valid only on z/OS when the queue manager is a member of a queue-sharing group.

Required parameters

CFStrucName (MQCFST)
CF application structure name (parameter identifier: MQCA_CF_STRUC_NAME).

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_Q_MGR_NAME_LENGTH.

Purge (MQCFIN)
Recover to empty CF structure (parameter identifier: MQIACF_PURGE).

Specifies whether the CF application structure is emptied. The value can be any of the following values:

MQPO_YES
Recover to empty CF structure. Any messages in the CF structure are lost.

MQPO_NO
Performs a true recovery of the CF structure. MQPO_NO is the default value.

Refresh Cluster:

The Refresh Cluster (MQCMD_REFRESH_CLUSTER) command discards all locally held cluster information, including any auto-defined channels that are not in doubt, and forces the repository to be rebuilt.

Note: For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.

Required parameters

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster to be refreshed.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

This parameter is the name of the cluster to be refreshed. If an asterisk (*) is specified for the name, the queue manager is refreshed in all the clusters to which it belongs.

If an asterisk (*) is specified with *RefreshRepository* set to MQCFO_REFRESH_REPOSITORY_YES, the queue manager restarts its search for repository queue managers, using information in the local cluster-sender channel definitions.

Optional parameters z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_QSG_NAME_LENGTH.

RefreshRepository (MQCFIN)
Whether repository information is refreshed (parameter identifier: MQIACF_REFRESH_REPOSITORY).

This parameter indicates whether the information about repository queue managers is refreshed.

The value can be:

MQCFO_REFRESH_REPOSITORY_YES
Refresh repository information.

This value cannot be specified if the queue manager is itself a repository queue manager.

MQCFO_REFRESH_REPOSITORY_YES specifies that in addition to MQCFO_REFRESH_REPOSITORY_NO behavior, objects representing full repository cluster queue managers are also refreshed. Do not use this option if the queue manager is itself a full repository.

If it is a full repository, you must first alter it so that it is not a full repository for the cluster in question.

The full repository location is recovered from the manually defined cluster-sender channel definitions. After the refresh with MQCFO_REFRESH_REPOSITORY_YES has been issued the queue manager can be altered so that it is once again a full repository.

MQCFO_REFRESH_REPOSITORY
Do not refresh repository information. MQCFO_REFRESH_REPOSITORY is the default.

If you select MQCFO_REFRESH_REPOSITORY_YES, check that all cluster-sender channels in the relevant cluster are inactive or stopped before you issue the Refresh Cluster command. If there are cluster-sender channels running at the time when the Refresh is processed, and they are used exclusively by the cluster or clusters being refreshed and MQCFO_REFRESH_REPOSITORY_YES is used, the channels are stopped, by using the Stop Channel command with a value of MQMODE_FORCE in the **Mode** parameter if necessary.

This scenario ensures that the Refresh can remove the channel state and that the channel will run with the refreshed version after the Refresh has completed. If the state of a channel cannot be deleted, for example because it is in doubt, or because it is also running as part of another cluster, it is state is not new after the refresh and it does not automatically restart if it was stopped.

Related information:

Clustering: Using REFRESH CLUSTER best practices

Refresh Queue Manager:

Use the Refresh Queue Manager (MQCMD_REFRESH_Q_MGR) command to perform special operations on queue managers.

Required parameters

RefreshType (MQCFIN)
Type of information to be refreshed (parameter identifier: MQIACF_REFRESH_TYPE).

Use this parameter to specify the type of information to be refreshed. The value can be any of the following values:

MQRT_CONFIGURATION
MQRT_CONFIGURATION causes the queue manager to generate configuration event messages for every object definition that matches the selection criteria specified by the **ObjectType**, **ObjectName**, and **RefreshInterval** parameters.

A Refresh Queue Manager command with a **RefreshType** value of MQRT_CONFIGURATION is generated automatically when the value of the queue manager's **ConfigurationEvent** parameter changes from MQEVR_DISABLED to MQEVR_ENABLED.

Use this command with a **RefreshType** of MQRT_CONFIGURATION to recover from problems such as errors on the event queue. In such cases, use appropriate selection criteria, to avoid excessive processing time and event message generation.

MQRT_EXPIRY
This requests that the queue manager performs a scan to discard expired messages for every queue that matches the selection criteria specified by the **ObjectName** parameter.

Note: z/OS Valid only on z/OS.

MQRT_EARLY
Requests that the subsystem function routines (generally known as early code) for the queue manager replace themselves with the corresponding routines in the linkpack area (LPA).

You need to use this command only after you install new subsystem function routines (provided as corrective maintenance or with a new version or release of IBM MQ). This command instructs the queue manager to use the new routines.

z/OS See Task 3: Update the z/OS link list and LPA for more information about IBM MQ early code routines.

MQRT_PROXYSUB
Requests that the queue manager resynchronizes the proxy subscriptions that are held with, and on behalf of, queue managers that are connected in a hierarchy or publish/subscribe cluster.

You should only resynchronize the proxy subscriptions in exceptional circumstances. See Resynchronization of proxy subscriptions.

Optional parameters (Refresh Queue Manager)

z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

ObjectName (MQCFST)
Name of object to be included in the processing of this command (parameter identifier: MQCACF_OBJECT_NAME).

Use this parameter to specify the name of the object to be included in the processing of this command.

Generic names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length is MQ_OBJECT_NAME_LENGTH.

ObjectType (MQCFIN)
Object type for which configuration data is to be refreshed (parameter identifier: MQIACF_OBJECT_TYPE).

Use this parameter to specify the object type for which configuration data is to be refreshed. This parameter is valid only if the value of *RefreshType* is MQRT_CONFIGURATION. The default value, in that case, is MQOT_ALL. The value can be one of:

MQOT_AUTH_INFO
Authentication information object.

MQOT_CF_STRUC
CF structure.

MQOT_CHANNEL
Channel.

MQOT_CHLAUTH
Channel authentication

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_LOCAL_Q
Local queue.

MQOT_MODEL_Q
Model queue.

MQOT_ALIAS_Q
Alias queue.

MQOT_REMOTE_Q
Remote queue.

MQOT_Q_MGR
Queue manager.

MQOT_CFSTRUC
CF structure.

MQOT_SERVICE
Service.

Note: z/OS Not valid on z/OS.

MQOT_STORAGE_CLASS
Storage class.

MQOT_TOPIC
Topic name.

RefreshInterval (MQCFIN)
Refresh interval (parameter identifier: MQIACF_REFRESH_INTERVAL).

Use this parameter to specify a value, in minutes, defining a period immediately before the current time. This requests that only objects that have been created or altered within that period (as defined by their *AlterationDate* and **AlterationTime** attributes) are included.

Specify a value in the range zero through 999 999. A value of zero means there is no time limit (0 is the default).

This parameter is valid only if the value of *RefreshType* is MQRT_CONFIGURATION.

Usage Notes for Refresh Queue Manager

1. Issue this command with *RefreshType*(MQRT_CONFIGURATION) after setting the MQRT_CONFIGURATION queue manager attribute to ENABLED, to bring the queue manager configuration up to date. To ensure that complete configuration information is generated, include all

objects; if you have many objects, it might be preferable to use several commands, each with a different selection of objects, but such that all are included.

2. You can also use the command with *RefreshType*(MQRT_CONFIGURATION) to recover from problems such as errors on the event queue. In such cases, use appropriate selection criteria, to avoid excessive processing time and event messages generation.
3. Issue the command with *RefreshType* (MQRT_EXPIRY) at any time when you believe that a queue could contain numbers of expired messages.
4. If *RefreshType* (MQRT_EARLY) is specified, no other keywords are allowed and the command can be issued only from the z/OS console and only if the queue manager is not active.
5. You are unlikely to use **Refresh Queue Manager RefreshType (MQRT_PROXYSUB)** other than in exceptional circumstances. See Resynchronization of proxy subscriptions.
6. If a **Refresh Queue Manager Object Type(MQRT_PROXYSUB)** command is issued on z/OS when the CHINIT is not running, the command is queued up and will be processed when the CHINIT starts.
7. Running the command Refresh Queue Manager RefreshType (MQRT_CONFIGURATION) Object Type(MQOT_ALL) includes authority records.

   You cannot specify the **Refresh Interval** and **Object Name** parameters if you explicitly specify Authority Record events. If you specify **Object Type**(*MQOT_ALL*) the **Refresh Interval** and **Object Name** parameters are ignored.

Refresh Security:

The Refresh Security (MQCMD_REFRESH_SECURITY) command refreshes the list of authorizations held internally by the authorization service component.

Optional parameters

z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

z/OS

SecurityItem (MQCFIN)
Resource class for which the security refresh is to be performed (parameter identifier: MQIACF_SECURITY_ITEM). This parameter applies to z/OS only.

Use this parameter to specify the resource class for which the security refresh is to be performed. The value can be any of the following values:

MQSECITEM_ALL
A full refresh of the type specified is performed. MQSECITEM_ALL is the default value.

MQSECITEM_MQADMIN
Specifies that administration type resources are to be refreshed. Valid only if the value of *SecurityType* is MQSECTYPE_CLASSES.

MQSECITEM_MQNLIST
Specifies that namelist resources are to be refreshed. Valid only if the value of *SecurityType* is MQSECTYPE_CLASSES.

MQSECITEM_MQPROC
Specifies that process resources are to be refreshed. Valid only if the value of *SecurityType* is MQSECTYPE_CLASSES.

MQSECITEM_MQQUEUE
Specifies that queue resources are to be refreshed. Valid only if the value of *SecurityType* is MQSECTYPE_CLASSES.

MQSECITEM_MXADMIN
Specifies that administration type resources are to be refreshed. Valid only if the value of *SecurityType* is MQSECTYPE_CLASSES.

MQSECITEM_MXNLIST
Specifies that namelist resources are to be refreshed. Valid only if the value of *SecurityType* is MQSECTYPE_CLASSES.

MQSECITEM_MXPROC
Specifies that process resources are to be refreshed. Valid only if the value of *SecurityType* is MQSECTYPE_CLASSES.

MQSECITEM_MXQUEUE
Specifies that queue resources are to be refreshed. Valid only if the value of *SecurityType* is MQSECTYPE_CLASSES.

MQSECITEM_MXTOPIC
Specifies that topic resources are to be refreshed. Valid only if the value of *SecurityType* is MQSECTYPE_CLASSES.

SecurityType (MQCFIN)
Security type (parameter identifier: MQIACF_SECURITY_TYPE).

Use this parameter to specify the type of security refresh to be performed. The value can be any of the following values:

MQSECTYPE_AUTHSERV
The list of authorizations held internally by the authorization services component is refreshed. MQSECTYPE_AUTHSERV is not valid on z/OS.

MQSECTYPE_AUTHSERV is the default on platforms other than z/OS.

MQSECTYPE_CLASSES
Permits you to select specific resource classes for which to perform the security refresh.

z/OS MQSECTYPE_CLASSES is valid only on z/OS where it is the default.

MQSECTYPE_CONNAUTH

Refreshes the cached view of the configuration for connection authentication.

Multi On Multiplatforms this is also a synonym for MQSECTYPE_AUTHSERV.

MQSECTYPE_SSL
MQSECTYPE_SSL refreshes the locations of the LDAP servers to be used for Certified Revocation Lists and the key repository. It also refreshes any cryptographic hardware

parameters specified through IBM MQ and the cached view of the Secure Sockets Layer key repository. It also allows updates to become effective on successful completion of the command.

MQSECTYPE_SSL updates all TLS channels currently running, as follows:

- Sender, server, and cluster-sender channels using TLS are allowed to complete the current batch. In general, they then run the TLS handshake again with the refreshed view of the TLS key repository. However, you must manually restart a requester-server channel on which the server definition has no CONNAME parameter.
- V 9.0.0 AMQP channels using TLS are restarted, with any currently connected clients being forcibly disconnected. The client receives an `amqp:connection:forced` AMQP error message.
- All other channel types using TLS are stopped with a STOP CHANNEL MODE(FORCE) STATUS(INACTIVE) command. If the partner end of the stopped message channel has retry values defined, the channel tries again and the new TLS handshake uses the refreshed view of the contents of the TLS key repository, the location of the LDAP server to be used for Certification Revocation Lists, and the location of the key repository. If there is a server-connection channel, the client application loses its connection to the queue manager and must reconnect in order to continue.

Reset CF Structure on z/OS: z/OS

The Reset coupling facility (CF) Structure (MQCMD_RESET_CF_STRUC) command modifies the status of a specific application structure.

Required parameters

CFStructName (MQCFST)
The name of the coupling facility application structure that you want to reset (parameter identifier: MQCA_CF_STRUC_NAME). The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

Action (MQCFIN)
The action to perform to reset the named application structure (parameter identifier: MQIACF_ACTION).

MQACT_FAIL
A structure failure is simulated and the status of the application structure is set to FAILED.

Reset Channel:

The Reset Channel (MQCMD_RESET_CHANNEL) command resets the message sequence number for an IBM MQ channel with, optionally, a specified sequence number to be used the next time that the channel is started.

This command can be issued to a channel of any type (except MQCHT_SVRCONN and MQCHT_CLNTCONN). However, if it is issued to a sender (MQCHT_SENDER), server (MQCHT_SERVER), or cluster-sender (MQCHT_CLUSSDR) channel, the value at both ends (issuing end and receiver or requester end), is reset when the channel is next initiated or resynchronized. The value at both ends is reset to be equal.

If the command is issued to a receiver (MQCHT_RECEIVER), requester (MQCHT_REQUESTER), or cluster-receiver (MQCHT_CLUSRCVR) channel, the value at the other end is not reset as well; this step must be done separately if necessary.

Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel.

If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the last channel added to the repository on the local queue manager.

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be reset. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_QSG_NAME_LENGTH.

z/OS

ChannelDisposition (MQCFIN)
Channel disposition (parameter identifier: MQIACH_CHANNEL_DISP). This parameter applies to z/OS only.

Specifies the disposition of the channels to be reset.

If this parameter is omitted, then the value for the channel disposition is taken from the default channel disposition attribute of the channel object.

The value can be any of the following values:

MQCHLD_PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than MQQSGD_SHARED.

MQCHLD_SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of MQQSGD_SHARED.

The combination of the **ChannelDisposition** and **CommandScope** parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.

The various combinations of *ChannelDisposition* and *CommandScope* are summarized in Table 134 on page 1540

Table 134. ChannelDisposition and CommandScope for RESET CHANNEL

ChannelDisposition	*CommandScope* **blank or local-qmgr**	*CommandScope* **qmgr-name**
MQCHLD_PRIVATE	Reset private channel on the local queue manager	Reset private channel on the named queue manager
MQCHLD_SHARED	Reset a shared channel on all active queue managers. MQCHLD_SHARED might automatically generate a command using *CommandScope* and send it to the appropriate queue manager. If there is no definition for the channel on the queue manager to which the command is sent, or if the definition is unsuitable for the command, the command fails. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted

MsgSeqNumber (MQCFIN)
Message sequence number (parameter identifier: MQIACH_MSG_SEQUENCE_NUMBER).

Specifies the new message sequence number.

The value must be in the range 1 through 999 999 999. The default value is one.

Error codes

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

Reset Cluster:

The Reset Cluster (MQCMD_RESET_CLUSTER) command forces a queue manager to leave a cluster.

Required parameters

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster to be reset.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

QMgrIdentifier (MQCFST)
Queue manager identifier (parameter identifier: MQCA_Q_MGR_IDENTIFIER).

This parameter is the unique identifier of the queue manager to be forcibly removed from the cluster. Only one of `QMgrIdentifier` and `QMgrName` can be specified. Use `QMgrIdentifier` in preference to `QmgrName`, because `QmgrName` might not be unique.

QMgrName (MQCFST)
Queue manager name (parameter identifier: MQCA_Q_MGR_NAME).

This parameter is the name of the queue manager to be forcibly removed from the cluster. Only one of `QMgrIdentifier` and `QMgrName` can be specified. Use `QMgrIdentifier` in preference to `QmgrName`, because `QmgrName` might not be unique.

`Action (MQCFIN)`
Action (parameter identifier: MQIACF_ACTION).

Specifies the action to take place. This parameter can be requested only by a repository queue manager.

The value can be any of the following values:

MQACT_FORCE_REMOVE
Requests that a queue manager is forcibly removed from a cluster.

Optional parameters z/OS

`CommandScope (MQCFST)`
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- a queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_QSG_NAME_LENGTH.

`RemoveQueues (MQCFIN)`
Whether cluster queues are removed from the cluster (parameter identifier: MQIACF_REMOVE_QUEUES).

This parameter indicates whether the cluster queues that belong to the queue manager being removed from the cluster are to be removed from the cluster. This parameter can be specified even if the queue manager identified by the **`QMgrName`** parameter is not currently in the cluster.

The value can be any of the following values:

MQCFO_REMOVE_QUEUES_YES
Remove queues belonging to the queue manager being removed from the cluster.

MQCFO_REMOVE_QUEUES_NO
Do not remove queues belonging to the queue manager being removed. MQCFO_REMOVE_QUEUES_NO is the default.

Error codes

This command might return the following error code in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 1075.

`Reason (MQLONG)`
The value can be any of the following values:

MQRCCF_ACTION_VALUE_ERROR
Value not valid.

Reset Queue Manager:

Use the Reset Queue Manager (MQCMD_RESET_Q_MGR) command as part of your backup and recovery procedures. V 9.0.2 The **Archive** option enables you to notify the queue manager that all log extents, up to the specified one, have been archived. If the log management type is not **ArchivedLog** the command fails. The **ReduceLog** option enables you to request that the queue manager reduces the number of log extents, provided they are no longer required.

You can use this command to request that the queue manager starts writing to a new log extent, making the previous log extent available for archiving.

Use the Reset Queue Manager (MQCMD_RESET_Q_MGR) command to forcibly remove a publish/subscribe hierarchical connection for which this queue manager is nominated as either the parent or the child in a hierarchical connection. Valid on all supported platforms.

V 9.0.2
Archive option

This option requires change authority on the queue manager object.

The command fails if the log extent is not recognized, or is being written.

If, for some reason, the programmatic way that your enterprise notifies your log extents are archived is not working, and the disk is filling up with log extents, your administrator can use this command.

You need to determine yourself, the name to pass in from your archiving process, as to what has already been archived.

V 9.0.2
ReduceLog option

This option requires change authority on the queue manager object.

You should not need this command in normal circumstances. In general, when using automatic management of log files, you should leave it up to the queue manager to reduce the number of log extents as necessary.

For circular logging, this can remove inactive secondary log extents. The increase in secondary log extents is usually noticed by an increase in disk usage, often due to some specific issue in the past.

Note: For circular logging the command might not be able reduce the log extents by the required number immediately. In that case, the command returns, and the reduction takes place asynchronously at some later point.

For linear logging this can remove log extents that are not required for recovery (and have been archived) as noticed by a high value for ReusableLogSize on the Inquire Queue Manager Status command.

You should run this command only after some specific event that has caused the number of log extents to be extraordinarily large.

The command blocks until the chosen number of extents have been deleted. Note that the command does not return the number of extents that have been removed, but a queue manager error log message is written, indicating what has taken place.

Required parameters

Action (MQCFIN)
Action (parameter identifier: MQIACF_ACTION).

Specifies the action to take place.

The value can be any of the following values, but you can specify one only:

MQACT_ADVANCE_LOG
Requests that the queue manager starts writing to a new log extent, making the previous log extent available for archiving. This command is accepted only if the queue manager is configured to use linear logging.

Note: Not valid on HP Integrity NonStop Server, or z/OS.

MQACT_COLLECT_STATISTICS
Requests that the queue manager ends the current statistics collection period, and writes the statistics collected.

Note: Not valid on HP Integrity NonStop Server, or z/OS.

MQACT_PUBSUB
Requests a publish/subscribe reset. This value requires that one of the optional parameters, ChildName or ParentName, is specified.

V 9.0.2 **MQACT_ARCHIVE_LOG (11)**
Requests that log extents are archived.

The command fails if the log extent is not recognized, or is the current log.

If, for some reason, the programmatic way that your enterprise notifies your log extents are archived is not working, and the disk is filling up with log extents, your administrator can use this command.

V 9.0.2 **MQACT_REDUCE_LOG (10)**
You should not need this command in normal circumstances. In general, when using automatic management of log files, you should leave it up to the queue manager to reduce the number of log extents as necessary.

For circular logging, you can use this option to remove inactive secondary log extents. A growth in secondary log extents is usually noticed by an increase in disk usage, often due to some specific issue in the past.

You should run this command only after some specific event that has caused the number of log extents to be extraordinarily large.

The command blocks until the chosen number of extents have been deleted. Note that the command does not return the number of extents that have been removed, but a queue manager error log message is written, indicating what has taken place.

Optional parameters

V 9.0.2 **ArchivedLog (MQCFST)**
Specifies the name of the log extent to be archived (parameter identifier: MQCACF_ARCHIVE_LOG_EXTENT_NAME).

The maximum length of the string is MQ_LOG_EXTENT_NAME_LENGTH.

ChildName (MQCFST)
The name of the child queue manager for which the hierarchical connection is to be forcibly canceled (parameter identifier: MQCA_CHILD).

This attribute is valid only when the Action parameter has the value MQACT_PUBSUB.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ParentName (MQCFST)
The name of the parent queue manager for which the hierarchical connection is to be forcibly canceled (parameter identifier: MQCA_PARENT).

This attribute is valid only when the Action parameter has the value MQACT_PUBSUB.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

V 9.0.2 **LogReduction (MQCFIN)**
Specifies that the type of log reduction (parameter identifier: MQIACF_LOG_REDUCTION).

The value can be one of:

MQLR_AUTO
-1. The default value. Reduce the log extents by an amount chosen by the queue manager.

MQLR_ONE
1. Reduce the log extents by one extent, if possible.

MQLR_MAX
-2. Reduce the log extents by the maximum number possible.

Error codes

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

V 9.0.2 **MQRCCF_CURRENT_LOG_EXTENT**
The specified log extent is the current log extent, and cannot have been validly archived yet.

V 9.0.2 **MQRCCF_LOG_EXTENT_NOT_FOUND**
The specified log extent was not found or is not valid.

V 9.0.2 **MQRCCF_LOG_NOT_REDUCED**
No log events could be removed.

MQRC_RESOURCE_PROBLEM
Insufficient system resources available.

Reset Queue Statistics:

The Reset Queue Statistics (MQCMD_RESET_Q_STATS) command reports the performance data for a queue and then resets the performance data. Performance data is maintained for each local queue (including transmission queues).

Performance data is reset at the following times:

- When a Reset Queue Statistics command is issued
- When the queue manager is restarted
- When a performance event is generated for a queue

Required parameters

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

The name of the local queue to be tested and reset.

Generic queue names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_Q_NAME_LENGTH.

z/OS

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- a queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_Q_WRONG_TYPE
Action not valid for the queue of specified type.

MQRCCF_EVENTS_DISABLED
The queue manager performance events are disabled (PERFMEV). On z/OS, it is necessary to enable queue manager performance events to use this command. For more details, see the PerformanceEvent property in the “Change Queue Manager” on page 1168 command.

Reset Queue Statistics (Response):

The response to the Reset Queue Statistics (MQCMD_RESET_Q_STATS) command consists of the response header followed by the *QName* structure and the attribute parameter structures shown in the following sections.

If a generic queue name was specified, one such message is generated for each queue found.

Always returned:

HighQDepth, *MsgDeqCount*, *MsgEnqCount*, *QName*, z/OS *QSGDisposition*, *TimeSinceReset*

Response data

HighQDepth (MQCFIN)
Maximum number of messages on a queue (parameter identifier: MQIA_HIGH_Q_DEPTH).

This count is the peak value of the *CurrentQDepth* local queue attribute since the last reset. The *CurrentQDepth* is incremented during an MQPUT call, and during backout of an MQGET call, and is decremented during a (nonbrowse) MQGET call, and during backout of an MQPUT call.

MsgDeqCount (MQCFIN)
Number of messages dequeued (parameter identifier: MQIA_MSG_DEQ_COUNT).

This count includes messages that have been successfully retrieved (with a nonbrowse MQGET) from the queue, even though the MQGET has not yet been committed. The count is not decremented if the MQGET is later backed out.

z/OS On z/OS, if the value exceeds 999 999 999, it is returned as 999 999 999

MsgEnqCount (MQCFIN)
Number of messages enqueued (parameter identifier: MQIA_MSG_ENQ_COUNT).

This count includes messages that have been put to the queue, but have not yet been committed. The count is not decremented if the put is later backed out.

z/OS On z/OS, if the value exceeds 999 999 999, it is returned as 999 999 999

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

z/OS

QSGDisposition (MQCFIN)
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object (that is, where it is defined and how it behaves). This parameter is valid on z/OS only. The value can be any of the following values:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

TimeSinceReset (MQCFIN)
Time since statistics reset in seconds (parameter identifier: MQIA_TIME_SINCE_RESET).

Reset SMDS on z/OS: z/OS

The Reset SMDS (MQCMD_RESET_SMDS) command modifies the availability or status information relating to one or more shared message data sets associated with a specific application structure

Required parameters

SMDS (MQCFST)
Specifies the queue manager for which the shared message data set availability or status information is to be modified or an asterisk to modify the information for all data sets associated with the specified CFSTRUCT. (parameter identifier: MQCACF_CF_SMDS).

The maximum length of the string is 4 characters.

CFStrucName (MQCFST)
The name of the CF application structure with SMDS connections properties that you want to reset (parameter identifier: MQCA_CF_STRUC_NAME).

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

Optional parameters

Access (MQCFIN)
Availability of the share message data set (parameter identifier: MQIACF_CF_STRUC_ACCESS).

MQCFACCESS_ENABLED
The shared message data set is available for use.

MQCFACCESS_DISABLED
The shared message data set is disabled.

Status (MQCFIN)
Status information indicates the state of a resource (parameter identifier: MQIACF_CF_STRUC_STATUS).

MQCFSTATUS_FAILED
The shared message data set is in an unusable state.

MQCFSTATUS_RECOVERED
The data set is set to recovered, and is ready for use again, but requires some restart processing the next time it is opened. This restart processing ensures that obsolete references to any deleted messages have been removed from the coupling facility structure before the data set is made available again. The restart processing also rebuilds the data set space map.

Resolve Channel:

The Resolve Channel (MQCMD_RESOLVE_CHANNEL) command requests a channel to commit or back out in-doubt messages. This command is used when the other end of a link fails during the confirmation stage, and for some reason it is not possible to reestablish the connection. In this situation the sending end remains in an in-doubt state, whether the messages were received. Any outstanding units of work must be resolved using Resolve Channel with either backout or commit.

Care must be exercised in the use of this command. If the resolution specified is not the same as the resolution at the receiving end, messages can be lost or duplicated.

This command can only be used for channels with a *ChannelType* value of MQCHT_SENDER, MQCHT_SERVER, or MQCHT_CLUSSDR.

Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel.

If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the last channel added to the repository on the local queue manager.

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be resolved. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

InDoubt (MQCFIN)
Indoubt resolution (parameter identifier: MQIACH_IN_DOUBT).

Specifies whether to commit or back out the in-doubt messages.

The value can be:

MQIDO_COMMIT
Commit.

MQIDO_BACKOUT
Backout.

z/OS

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_QSG_NAME_LENGTH.

ChannelDisposition (MQCFIN)
Channel disposition (parameter identifier: MQIACH_CHANNEL_DISP). This parameter applies to z/OS only.

Specifies the disposition of the channels to be resolved.

If this parameter is omitted, then the value for the channel disposition is taken from the default channel disposition attribute of the channel object.

The value can be any of the following values:

MQCHLD_PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than MQQSGD_SHARED.

MQCHLD_SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of MQQSGD_SHARED.

The combination of the **ChannelDisposition** and **CommandScope** parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.

The various combinations of *ChannelDisposition* and *CommandScope* are summarized in Table 135

Table 135. ChannelDisposition and CommandScope for RESOLVE CHANNEL

ChannelDisposition	*CommandScope* **blank or local-qmgr**	*CommandScope* **qmgr-name**
MQCHLD_PRIVATE	Resolve private channel on the local queue manager	Resolve private channel on the named queue manager
MQCHLD_SHARED	Resolve a shared channel on all active queue managers. MQCHLD_SHARED might automatically generate a command using *CommandScope* and send it to the appropriate queue manager. If there is no definition for the channel on the queue manager to which the command is sent, or if the definition is unsuitable for the command, the command fails. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_INDOUBT_VALUE_ERROR
In-doubt value not valid.

Resume Queue Manager on z/OS: z/OS

The Resume Queue Manager (MQCMD_RESUME_Q_MGR) command renders the queue manager available again for the processing of IMS or Db2 messages. It reverses the action of the Suspend Queue Manager (MQCMD_SUSPEND_Q_MGR) command.

Required parameters

Facility (MQCFIN)
Facility (parameter identifier: MQIACF_FACILITY).

The type of facility for which activity is to be resumed. The value can be:

MQQMFAC_DB2
Resumes normal activity with Db2.

MQQMFAC_IMS_BRIDGE
Resumes normal IMS bridge activity.

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_QSG_NAME_LENGTH.

Resume Queue Manager Cluster:

The Resume Queue Manager Cluster (MQCMD_RESUME_Q_MGR_CLUSTER) command informs other queue managers in a cluster that the local queue manager is again available for processing, and can be sent messages. It reverses the action of the Suspend Queue Manager Cluster (MQCMD_SUSPEND_Q_MGR_CLUSTER) command.

Required parameters

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster for which availability is to be resumed.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

ClusterNamelist (MQCFST)
Cluster Namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

The name of the namelist specifying a list of clusters for which availability is to be resumed.

z/OS

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- a queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_QSG_NAME_LENGTH.

Error codes

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CLUSTER_NAME_CONFLICT
Cluster name conflict.

Reverify Security on z/OS: z/OS

The Reverify Security (MQCMD_REVERIFY_SECURITY) to set a reverification flag for all specified users. The user is reverified the next time that security is checked for that user.

Required parameters

UserId (MQCFST)
User ID (parameter identifier: MQCACF_USER_IDENTIFIER).

Use this parameter to specify one or more user IDs. Each user ID specified is signed off and signed back on again the next time that a request requiring a security check is issued on behalf of that user.

The maximum length of the string is MQ_USER_ID_LENGTH.

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Set Archive on z/OS: z/OS

Use the Set Archive (MQCMD_SET_ARCHIVE) to dynamically change certain archive system parameter values initially set by your system parameter module at queue manager startup.

Required parameters

ParameterType (MQCFIN)
Parameter type (parameter identifier: MQIACF_SYSP_TYPE).

Specifies how the parameters are to be reset:

MQSYSP_TYPE_INITIAL
The initial settings of the archive system parameters. MQSYSP_TYPE_INITIAL resets all the archive system parameters to the values set at queue manager startup.

MQSYSP_TYPE_SET
MQSYSP_TYPE_SET indicates that you intend to change one, or more, of the archive system parameter settings.

Optional parameters

AllocPrimary (MQCFIN)
Primary space allocation for DASD data sets (parameter identifier: MQIACF_SYSP_ALLOC_PRIMARY).

Specifies the primary space allocation for DASD data sets in the units specified in the **AllocUnits** parameter.

Specify a value greater than zero. This value must be sufficient for a copy of either the log data set or its corresponding BSDS, whichever is the larger.

AllocSecondary (MQCFIN)
Secondary space allocation for DASD data sets (parameter identifier: MQIACF_SYSP_ALLOC_SECONDARY).

Specifies the secondary space allocation for DASD data sets in the units specified in the **AllocUnits** parameter.

Specify a value greater than zero.

AllocUnits (MQCFIN)
Allocation unit (parameter identifier: MQIACF_SYSP_ALLOC_UNIT).

Specifies the unit in which primary and secondary space allocations are made. The value can be any of the following values:

MQSYSP_ALLOC_BLK
Blocks.

MQSYSP_ALLOC_TRK
Tracks.

MQSYSP_ALLOC_CYL
Cylinders.

ArchivePrefix1 (MQCFST)
Specifies the prefix for the first archive log data set name (parameter identifier: MQCACF_SYSP_ARCHIVE_PFX1).

The maximum length of the string is MQ_ARCHIVE_PFX_LENGTH.

ArchivePrefix2 (MQCFST)
Specifies the prefix for the second archive log data set name (parameter identifier: MQCACF_SYSP_ARCHIVE_PFX2).

The maximum length of the string is MQ_ARCHIVE_PFX_LENGTH.

ArchiveRetention (MQCFIN)
Archive retention period (parameter identifier: MQIACF_SYSP_ARCHIVE_RETAIN).

Specifies the retention period, in days, to be used when the archive log data set is created. Specify a value in the range zero through 9999.

For more information, see Discarding archive log data sets.

ArchiveUnit1 (MQCFST)
Specifies the device type or unit name of the device that is used to store the first copy of the archive log data set (parameter identifier: MQCACF_SYSP_ARCHIVE_UNIT1).

Specify a device type or unit name of 1-8 characters.

If you archive to DASD, you can specify a generic device type with a limited volume range.

The maximum length of the string is MQ_ARCHIVE_UNIT_LENGTH.

ArchiveUnit2 (MQCFST)
Specifies the device type or unit name of the device that is used to store the second copy of the archive log data set (parameter identifier: MQCACF_SYSP_ARCHIVE_UNIT2).

Specify a device type or unit name of 1-8 characters.

If this parameter is blank, the value set for the **ArchiveUnit1** parameter is used.

The maximum length of the string is MQ_ARCHIVE_UNIT_LENGTH.

ArchiveWTOR (MQCFIN)
Specifies whether a message is to be sent to the operator and a reply is received before attempting to mount an archive log data set (parameter identifier: MQIACF_SYSP_ARCHIVE_WTOR).

Other IBM MQ users might be forced to wait until the data set is mounted, but they are not affected while IBM MQ is waiting for the reply to the message.

The value can be any of the following values:

MQSYSP_YES
A message is to be sent and a reply received before an attempt to mount an archive log data set.

MQSYSP_NO
A message is not to be sent and a reply received before an attempt to mount an archive log data set.

BlockSize (MQCFIN)
Block size of the archive log data set (parameter identifier: MQIACF_SYSP_BLOCK_SIZE).

The block size you specify must be compatible with the device type you specify in the **ArchiveUnit1** and **ArchiveUnit2** parameters.

Specify a value in the range 4 097 through 28 672. The value you specify is rounded up to a multiple of 4 096.

This parameter is ignored for data sets that are managed by the storage management system (SMS).

Catalog (MQCFIN)
Specifies whether archive log data sets are cataloged in the primary integrated catalog facility (parameter identifier: MQIACF_SYSP_CATALOG).

The value can be:

MQSYSP_YES
Archive log data sets are cataloged.

MQSYSP_NO
Archive log data sets are not cataloged.

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- a queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Compact (MQCFIN)
Specifies whether data written to archive logs is to be compacted (parameter identifier: MQIACF_SYSP_COMPACT).

This parameter applies to a 3480 or 3490 device that has the improved data recording capability (IDRC) feature. When this feature is turned on, hardware in the tape control unit writes data at a much higher density than normal, allowing for more data on each volume. Specify MQSYSP_NO if you do not use a 3480 device with the IDRC feature or a 3490 base model, except for the 3490E. Specify MQSYSP_YES if you want the data to be compacted.

The value can be:

MQSYSP_YES
Data is to be compacted.

MQSYSP_NO
Data is not to be compacted.

Protect (MQCFIN)
Protection by external security manager (ESM) (parameter identifier: MQIACF_SYSP_PROTECT).

Specifies whether archive log data sets are protected by ESM profiles when the data sets are created.

If you specify MQSYSP_YES, ensure that:

- ESM protection is active for IBM MQ.
- The user ID associated with the IBM MQ address space has authority to create these profiles.
- The TAPEVOL class is active if you are archiving to tape.

otherwise, offload processing fails.

The value can be any of the following values:

MQSYSP_YES
Data set profiles are created when logs are offloaded.

MQSYSP_NO
Profiles are not created.

QuiesceInterval (MQCFIN)
Maximum time allowed for the quiesce (parameter identifier: MQIACF_SYSP_QUIESCE_INTERVAL).

Specifies the maximum time, in seconds, allowed for the quiesce.

Specify a value in the range 1 through 999.

RoutingCode (MQCFIL)
z/OS routing code list (parameter identifier: MQIACF_SYSP_ROUTING_CODE).

Specifies the list of z/OS routing codes for messages about the archive log data sets to the operator.

Specify up to 14 routing codes, each with a value in the range zero through 16. You must specify at least one code.

TimeStampFormat (MQCFIN)
Time stamp included (parameter identifier: MQIACF_SYSP_TIMESTAMP).

Specifies whether the archive log data set name has a time stamp in it.

The value can be:

MQSYSP_YES
Names include a time stamp. The archive log data sets are named:
arcpfxi.cyyddd.T *hhmmsst*.A *nnnnnnn*

where *c* is 'D' for the years up to and including 1999 or 'E' for the year 2000 and later, and *arcpfxi* is the data set name prefix specified by *ArchivePrefix1* or *ArchivePrefix2*. *arcpfxi* can have up to 19 characters.

MQSYSP_NO
Names do not include a time stamp. The archive log data sets are named:
arcpfxi.A *nnnnnnn*

Where *arcpfxi* is the data set name prefix specified by *ArchivePrefix1* or *ArchivePrefix2*. *arcpfxi* can have up to 35 characters.

MQSYSP_EXTENDED
Names include a time stamp. The archive log data sets are named:
arcpfxi.D *yyyyddd*.T *hhmmsst*.A *nnnnnnn*

Where *arcpfxi* is the data set name prefix specified by *ArchivePrefix1* or *ArchivePrefix2*. *arcpfxi* can have up to 17 characters.

Set Authority Record on Multiplatforms: Multi

The Set Authority Record (MQCMD_SET_AUTH_REC) command sets the authorizations of a profile, object, or class of objects. Authorizations can be granted to, or revoked from, any number of principals or groups.

Required parameters

ProfileName (MQCFST)
Profile name (parameter identifier: MQCACF_AUTH_PROFILE_NAME).

The authorizations apply to all IBM MQ objects with names that match the profile name specified. You can define a generic profile. If you specify an explicit profile name, the object must exist.

The maximum length of the string is MQ_AUTH_PROFILE_NAME_LENGTH.

ObjectType (MQCFIN)
The type of object for which to set authorizations (parameter identifier: MQIACF_OBJECT_TYPE).

The value can be any of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel object.

MQOT_CLNTCONN_CHANNEL
Client-connection channel object.

MQOT_COMM_INFO
Communication information object

MQOT_LISTENER
Listener object.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process.

MQOT_Q
Queue, or queues, that match the object name parameter.

MQOT_Q_MGR
Queue manager.

MQOT_REMOTE_Q_MGR_NAME
Remote queue manager.

MQOT_SERVICE
Service object.

MQOT_TOPIC
Topic object.

Note: The required parameters must be in the order **ProfileName** followed by **ObjectType**.

Optional parameters

AuthorityAdd (MQCFIL)
Authority values to set (parameter identifier: MQIACF_AUTH_ADD_AUTHS).

This parameter is a list of authority values to set for the named profile. The values can be:

MQAUTH_NONE
The entity has authority set to 'none'.

MQAUTH_ALT_USER_AUTHORITY
Specify an alternate user ID on an MQI call.

MQAUTH_BROWSE
Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.

MQAUTH_CHANGE
Change the attributes of the specified object, using the appropriate command set.

MQAUTH_CLEAR
Clear a queue.

MQAUTH_CONNECT
Connect the application to the specified queue manager by issuing an MQCONN call.

MQAUTH_CREATE
Create objects of the specified type using the appropriate command set.

MQAUTH_DELETE
Delete the specified object using the appropriate command set.

MQAUTH_DISPLAY
Display the attributes of the specified object using the appropriate command set.

MQAUTH_INPUT
Retrieve a message from a queue by issuing an MQGET call.

MQAUTH_INQUIRE
Make an inquiry on a specific queue by issuing an MQINQ call.

MQAUTH_OUTPUT
Put a message on a specific queue by issuing an MQPUT call.

MQAUTH_PASS_ALL_CONTEXT
Pass all context.

MQAUTH_PASS_IDENTITY_CONTEXT
Pass the identity context.

MQAUTH_SET
Set attributes on a queue from the MQI by issuing an MQSET call.

MQAUTH_SET_ALL_CONTEXT
Set all context on a queue.

MQAUTH_SET_IDENTITY_CONTEXT
Set the identity context on a queue.

MQAUTH_CONTROL
For listeners and services, start and stop the specified channel, listener, or service.

For channels, start, stop, and ping the specified channel.

For topics, define, alter, or delete subscriptions.

MQAUTH_CONTROL_EXTENDED
Reset or resolve the specified channel.

MQAUTH_PUBLISH
Publish to the specified topic.

MQAUTH_SUBSCRIBE
Subscribe to the specified topic.

MQAUTH_RESUME
Resume a subscription to the specified topic.

MQAUTH_SYSTEM
Use queue manager for internal system operations.

MQAUTH_ALL
Use all operations applicable to the object.

MQAUTH_ALL_ADMIN
Use all administration operations applicable to the object.

MQAUTH_ALL_MQI
Use all MQI calls applicable to the object.

The contents of the *AuthorityAdd* and *AuthorityRemove* lists must be mutually exclusive. You must specify a value for either *AuthorityAdd* or *AuthorityRemove*. An error occurs if you do not specify either.

AuthorityRemove (MQCFIL)
Authority values to remove (parameter identifier: MQIACF_AUTH_REMOVE_AUTHS).

This parameter is a list of authority values to remove from the named profile. The values can be:

MQAUTH_NONE
The entity has authority set to 'none'.

MQAUTH_ALT_USER_AUTHORITY
Specify an alternate user ID on an MQI call.

MQAUTH_BROWSE
Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.

MQAUTH_CHANGE
Change the attributes of the specified object, using the appropriate command set.

MQAUTH_CLEAR
Clear a queue.

MQAUTH_CONNECT
Connect the application to the specified queue manager by issuing an MQCONN call.

MQAUTH_CREATE
Create objects of the specified type using the appropriate command set.

MQAUTH_DELETE
Delete the specified object using the appropriate command set.

MQAUTH_DISPLAY
Display the attributes of the specified object using the appropriate command set.

MQAUTH_INPUT
Retrieve a message from a queue by issuing an MQGET call.

MQAUTH_INQUIRE
Make an inquiry on a specific queue by issuing an MQINQ call.

MQAUTH_OUTPUT
Put a message on a specific queue by issuing an MQPUT call.

MQAUTH_PASS_ALL_CONTEXT
Pass all context.

MQAUTH_PASS_IDENTITY_CONTEXT
Pass the identity context.

MQAUTH_SET
Set attributes on a queue from the MQI by issuing an MQSET call.

MQAUTH_SET_ALL_CONTEXT
Set all context on a queue.

MQAUTH_SET_IDENTITY_CONTEXT
Set the identity context on a queue.

MQAUTH_CONTROL
For listeners and services, start and stop the specified channel, listener, or service.

For channels, start, stop, and ping the specified channel.

For topics, define, alter, or delete subscriptions.

MQAUTH_CONTROL_EXTENDED
Reset or resolve the specified channel.

MQAUTH_PUBLISH
Publish to the specified topic.

MQAUTH_SUBSCRIBE
Subscribe to the specified topic.

MQAUTH_RESUME
Resume a subscription to the specified topic.

MQAUTH_SYSTEM
Use queue manager for internal system operations.

MQAUTH_ALL
Use all operations applicable to the object.

MQAUTH_ALL_ADMIN
Use all administration operations applicable to the object.

MQAUTH_ALL_MQI
Use all MQI calls applicable to the object.

The contents of the *AuthorityAdd* and *AuthorityRemove* lists must be mutually exclusive. You must specify a value for either *AuthorityAdd* or *AuthorityRemove*. An error occurs if you do not specify either.

GroupNames (MQCFSL)
Group names (parameter identifier: MQCACF_GROUP_ENTITY_NAMES).

The names of groups having their authorizations set. At least one group name or principal name must be specified. An error occurs if neither are specified.

Each member in this list can be a maximum length of MQ_ENTITY_NAME_LENGTH.

PrincipalNames (MQCFSL)
Principal names (parameter identifier: MQCACF_PRINCIPAL_ENTITY_NAMES).

The names of principals having their authorizations set. At least one group name or principal name must be specified. An error occurs if neither are specified.

Each member in this list can be a maximum length of MQ_ENTITY_NAME_LENGTH.

ServiceComponent (MQCFST)
Service component (parameter identifier: MQCACF_SERVICE_COMPONENT).

If installable authorization services are supported, this parameter specifies the name of the authorization service to which the authorizations apply.

If you omit this parameter, the authorization inquiry is made to the first installable component for the service.

The maximum length of the string is MQ_SERVICE_COMPONENT_LENGTH.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRC_UNKNOWN_ENTITY
Userid not authorized, or unknown.

MQRCCF_AUTH_VALUE_ERROR
Invalid authorization.

MQRCCF_AUTH_VALUE_MISSING
Authorization missing.

MQRCCF_ENTITY_NAME_MISSING
Entity name missing.

MQRCCF_OBJECT_TYPE_MISSING
Object type missing.

MQRCCF_PROFILE_NAME_ERROR
Invalid profile name.

Set Channel Authentication Record:

The Set Channel Authentication Record (MQCMD_SET_CHLAUTH_REC) command sets the allowed partner details and mappings to MCAUSER for a channel or set of channels.

Syntax diagram

See the syntax diagram in the MQSC "SET CHLAUTH" on page 1012 command for combinations of parameters and values that are allowed.

Required parameters

The required parameters are valid for the **Action** values of:

- MQACT_ADD or MQACT_REPLACE
- MQACT_REMOVE
- MQACT_REMOVEALL

ProfileName (MQCFST)
The name of the channel or set of channels for which you are setting channel authentication configuration (parameter identifier: MQCACH_CHANNEL_NAME). You can use one or more asterisks (*), in any position, as wildcards to specify a set of channels. If you set Type to MQCAUT_BLOCKADDR, you must set the generic channel name to a single asterisk, which matches all channel names.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Type (MQCFIN)
The **Type** parameter must follow the **ProfileName** parameter.

The type of channel authentication record for which to set allowed partner details or mappings to MCAUSER (parameter identifier: MQIACF_CHLAUTH_TYPE). The following values are valid:

MQCAUT_BLOCKUSER
This channel authentication record prevents a specified user or users from connecting. The MQCAUT_BLOCKUSER parameter must be accompanied by a **UserList**.

MQCAUT_BLOCKADDR
This channel authentication record prevents connections from a specified IP address or addresses. The MQCAUT_BLOCKADDR parameter must be accompanied by an **AddrList**.

MQCAUT_SSLPEERMAP
This channel authentication record maps TLS Distinguished Names (DNs) to MCAUSER values. The MQCAUT_SSLPEERMAP parameter must be accompanied by an **SSLPeer**.

MQCAUT_ADDRESSMAP
This channel authentication record maps IP addresses to MCAUSER values. The MQCAUT_ADDRESSMAP parameter must be accompanied by an **Address**.

MQCAUT_USERMAP
This channel authentication record maps asserted user IDs to MCAUSER values. The MQCAUT_USERMAP parameter must be accompanied by a **ClntUser**.

MQCAUT_QMGRMAP
This channel authentication record maps remote queue manager names to MCAUSER values. The MQCAUT_QMGRMAP parameter must be accompanied by a **QMName**.

Optional parameters

The following table shows which parameters are valid for each value of **Action**:

Parameter	MQACT_ADD or MQACT_REPLACE	MQACT_REMOVE	MQACT_REMOVEALL
z/OS CommandScope	✓	✓	✓
Action	✓	✓	✓
Address	✓	✓	
Addrlist	✓	✓	
CheckClient	✓	✓	
ClntUser	✓	✓	
MCAUser	✓		
QMName	✓	✓	
SSLCertIssuer	✓	✓	
SSLPeer	✓	✓	
UserList	✓	✓	
UserSrc	✓		
Warn	✓		
Description	✓		

Action (MQCFIN)
The action to perform on the channel authentication record (parameter identifier: MQIACF_ACTION). The following values are valid:

MQACT_ADD
Add the specified configuration to a channel authentication record. This is the default value.

For types MQCAUT_SSLPEERMAP, MQCAUT_ADDRESSMAP, MQCAUT_USERMAP and MQCAUT_QMGRMAP, if the specified configuration exists, the command fails.

For types MQCAUT_BLOCKUSER and MQCAUT_BLOCKADDR, the configuration is added to the list.

MQACT_REPLACE
Replace the current configuration of a channel authentication record.

For types MQCAUT_SSLPEERMAP, MQCAUT_ADDRESSMAP, MQCAUT_USERMAP and MQCAUT_QMGRMAP, if the specified configuration exists, it is replaced with the new configuration. If it does not exist it is added.

For types MQCAUT_BLOCKUSER and MQCAUT_BLOCKADDR, the configuration specified replaces the current list, even if the current list is empty. If you replace the current list with an empty list, this acts like MQACT_REMOVEALL.

MQACT_REMOVE
Remove the specified configuration from the channel authentication records. If the configuration does not exist the command fails. If you remove the last entry from a list, this acts like MQACT_REMOVEALL.

MQACT_REMOVEALL
Remove all members of the list and thus the whole record (for MQCAUT_BLOCKADDR and MQCAUT_BLOCKUSER) or all previously defined mappings (for MQCAUT_ADDRESSMAP, MQCAUT_SSLPEERMAP, MQCAUT_QMGRMAP and MQCAUT_USERMAP) from the channel authentication records. This option cannot be combined with specific values supplied in **AddrList**, **UserList**, **Address**, **SSLPeer**, **QMName** or **ClntUser**. If the specified type has no current configuration the command still succeeds.

Address (MQCFST)

Attention: Host names can be specified in this parameter, only on queue managers that have IBM MQ Version 8.0 new functions enabled with OPMODE.
The filter to be used to compare with the IP address, or host name, of the partner queue manager or client at the other end of the channel (parameter identifier: MQCACH_CONNECTION_NAME).

This parameter is mandatory when **Type** is MQCAUT_ADDESSMAP and is also valid when **Type** is MQCAUT_SSLPEERMAP, MQCAUT_USERMAP, or MQCAUT_QMGRMAP and **Action** is MQACT_ADD, MQACT_REPLACE, or MQACT_REMOVE. You can define more than one channel authentication object with the same main identity, for example the same TLS peer name, with different addresses. See "Generic IP addresses for channel authentication records" on page 1019 for more information about filtering IP addresses.

The maximum length of the string is MQ_CONN_NAME_LENGTH.

AddrList (MQCFSL)
A list of up to 100 generic IP addresses which are banned from accessing this queue manager on any channel (parameter identifier: MQCACH_CONNECTION_NAME_LIST).

This parameter is only valid when **Type** is MQCAUT_BLOCKADDR.

The maximum length of each address is MQ_CONN_NAME_LENGTH.

CheckClient (MQCFIN)

Attention: This parameter is valid only on queue managers that have IBM MQ Version 8.0 new functions enabled with OPMODE.
The user ID and password requirements for the client connection to be successful. The following values are valid:

MQCHK_REQUIRED_ADMIN
A valid user ID and password are required for the connection to be allowed if you are using a privileged user ID. The password cannot contain single quotation marks (').

Any connections using a non-privileged user ID are not required to provide a user ID and password.

The user ID and password are checked against the user repository details provided in an authentication information object, and supplied on ALTER QMGR in the CONNAUTH field.

If no user repository details are provided, so that user ID and password checking are not enabled on the queue manager, the connection is not successful.

This option is not valid on z/OS platforms.

MQCHK_REQUIRED
A valid user ID and password are required for the connection to be allowed. The password cannot contain single quotation marks (').

The user ID and password are checked against the user repository details provided in an authentication information object and supplied on ALTER QMGR in the CONNAUTH field.

If no user repository details are provided, so that user ID and password checking are not enabled on the queue manager, the connection is not successful.

MQCHK_AS_Q_MGR
In order for the connection to be allowed, it must meet the connection authentication requirements defined on the queue manager.

If the CONNAUTH field provides an authentication information object, and the value of CHCKCLNT is REQUIRED, the connection fails unless a valid user ID and password are supplied.

If the CONNAUTH field does not provide an authentication information object, or the value of CHCKCLNT is not REQUIRED, the user ID and password are not required.

ClntUser (MQCFST)
The client asserted user ID to be mapped to a new user ID, allowed through unchanged, or blocked (parameter identifier: MQCACH_CLIENT_USER_ID).

This can be the user ID flowed from the client indicating the user ID the client side process is running under, or the user ID presented by the client on an MQCONNX call using MQCSP.

This parameter is valid only with TYPE(USERMAP) and when **Match** is MQMATCH_RUNCHECK.

The maximum length of the string is MQ_CLIENT_USER_ID_LENGTH.

z/OS **CommandScope (MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is run when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is run on the queue manager on which it was entered.
- a queue manager name. The command is run on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which the command was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is run on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

Custom (MQCFST)
Reserved for future use.

Description (MQCFST)
Provides descriptive information about the channel authentication record, which is displayed when you issue the Inquire Channel Authentication Records command (parameter identifier: MQCA_CHLAUTH_DESC).

This parameter must contain only displayable characters. In a DBCS installation, it can contain DBCS characters. The maximum length of the string is MQ_CHLAUTH_DESC_LENGTH.

Note: Use characters from the coded character set identifier (CCSID) for this queue manager. Other characters might be translated incorrectly if the information is sent to another queue manager.

MCAUser (MQCFST)
The user identifier to be used when the inbound connection matches the TLS DN, IP address, client asserted user ID or remote queue manager name supplied (parameter identifier: MQCACH_MCA_USER_ID).

This parameter is mandatory when **UserSrc** is MQUSRC_MAP and is valid when **Type** is MQCAUT_SSLPEERMAP, MQCAUT_ADDRESSMAP, MQCAUT_USERMAP, or MQCAUT_QMGRMAP.

This parameter is valid only when **Action** is MQACT_ADD or MQACT_REPLACE.

The maximum length of the string is MQ_MCA_USER_ID_LENGTH.

QMName (MQCFST)
The name of the remote partner queue manager, or pattern that matches a set of queue manager names, to be mapped to a user ID or blocked (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

This parameter is valid only when **Type** is MQCAUT_QMGRMAP

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

SSLCertIssuer (MQCFST)
This parameter is additional to the **SSLPeer** parameter.

SSLCertIssuer restricts matches to being within certificates issued by a particular Certificate Authority.

Attention: This parameter is valid only on queue managers that have IBM MQ Version 8.0 new functions enabled with OPMODE.

SSLPeer (MQCFST)

The filter to use to compare with the Distinguished Name of the certificate from the peer queue manager or client at the other end of the channel (parameter identifier: MQCACH_SSL_PEER_NAME).

The **SSLPeer** value is specified in the standard form used to specify a Distinguished Name. See Distinguished Names and IBM MQ rules for SSLPEER values.

The maximum length of the string is MQ_SSL_PEER_NAME_LENGTH .

UserList (MQCFSL)
A list of up to 100 user IDs which are banned from using this channel or set of channels (parameter identifier: MQCACH_MCA_USER_ID_LIST).

The following special value can be used:

***MQADMIN**
The exact meaning of this value is determined at runtime. If you are using the OAM supplied with IBM MQ, the meaning depends on platform, as follows:

- On Windows, all members of the mqm group, the Administrators group and SYSTEM
- On UNIX and Linux, all members of the mqm group
- On IBM i, the profiles (users) qmqm and qmqmadm and all members of the qmqmadm group, and any user defined with the *ALLOBJ special setting
- z/OS On z/OS, the user ID that the CHINIT and the user ID that the MSTR address spaces are running under

This parameter is only valid when **TYPE** is MQCAUT_BLOCKUSER.

The maximum length of each user ID is MQ_MCA_USER_ID_LENGTH .

UserSrc (MQCFIN)
The source of the user ID to be used for MCAUSER at run time (parameter identifier: MQIACH_USER_SOURCE).

The following values are valid:

MQUSRC_MAP
Inbound connections that match this mapping use the user ID specified in the **MCAUser** attribute. This is the default value.

MQUSRC_NOACCESS
Inbound connections that match this mapping have no access to the queue manager and the channel ends immediately.

MQUSRC_CHANNEL
Inbound connections that match this mapping use the flowed user ID or any user defined on the channel object in the MCAUSER field.

Note that *Warn* and MQUSRC_CHANNEL, or MQUSRC_MAP are incompatible. This is because channel access is never blocked in these cases, so there is never a reason to generate a warning.

Warn (MQCFIN)
Indicates whether this record operates in warning mode (parameter identifier: MQIACH_WARNING).

MQWARN_NO
This record does not operate in warning mode. Any inbound connection that matches this record is blocked. This is the default value.

MQWARN_YES
This record operates in warning mode. Any inbound connection that matches this record and would therefore be blocked is allowed access. An error message is written and, if events are configured, an event message is created showing the details of what would have been blocked. The connection is allowed to continue. An attempt is made to find another record that is set to WARN(NO) to set the credentials for the inbound channel.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown at "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CHLAUTH_TYPE_ERROR
Channel authentication record type not valid.

MQRCCF_CHLAUTH_ACTION_ERROR
Channel authentication record action not valid.

MQRCCF_CHLAUTH_USERSRC_ERROR
Channel authentication record user source not valid.

MQRCCF_WRONG_CHLAUTH_TYPE
Parameter not allowed for this channel authentication record type.

MQRCCF_CHLAUTH_ALREADY_EXISTS
Channel authentication record already exists

Related information:
Channel authentication records

Set Log: Multi V 9.0.2

The Set Log (MQCMD_SET_LOG) command on Multiplatforms enables you to notify the queue manager that archiving of a log is complete. If the log management type is not **Archive** the command fails. This command requires change authority on the queue manager object.

Required parameters:
ParameterType

Optional parameters:
Archive

Required parameters

ParameterType (MQCFIN)
Specifies the type of the log (parameter identifier: MQIACF_SYSP_TYPE).

The value must be MQSYSP_TYPE_SET

Optional parameters

Archive (MQCFST)
Specifies the log extent that is being marked as archived (parameter identifier: MQCACF_ARCHIVE_LOG_EXTENT_NAME).

The command fails if the log extent is not recognized, or is the current log. The command does not fail if the extent has already been marked as having been archived.

A message is written to the error log if the queue manager is notified about an extent more than once.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown at "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_LOG_EXTENT_NOT_FOUND
The specified log extent was not found or is not valid.

MQRCCF_CURRENT_LOG_EXTENT
The specified log extent is the current log extent, and cannot have been validly archived yet.

MQRCCF_LOG_TYPE_ERROR
The command has been run on a log that is not an archive log.

MQRCCF_LOG_EXTENT_ERROR
The specified log extent is corrupt.

Set Log on z/OS: z/OS

Use the Set Log (MQCMD_SET_LOG) command to dynamically change certain log system parameter values initially set by your system parameter module at queue manager startup.

Required parameters:
ParameterType

Optional parameters (if the value of *ParameterType* is MQSYSP_TYPE_SET:
CommandScope, *DeallocateInterval*, *LogCompression*, *MaxArchiveLog*, *MaxConcurrentOffloads*, *MaxReadTapeUnits*, *OutputBufferCount*

Optional parameters if *ParameterType* type is MQSYSP_INITIAL:
CommandScope

Required parameters

ParameterType (MQCFIN)
Parameter type (parameter identifier: MQIACF_SYSP_TYPE).

Specifies how the parameters are to be set:

MQSYSP_TYPE_INITIAL
The initial settings of the log system parameters. This MQSYSP_TYPE_INITIAL resets all the log system parameters to the values at queue manager startup.

MQSYSP_TYPE_SET
This MQSYSP_TYPE_SETindicates that you intend to change one, or more, of the archive log system parameter settings.

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

DeallocateInterval (MQCFIN)
Deallocation interval (parameter identifier: MQIACF_SYSP_DEALLOC_INTERVAL).

Specifies the length of time, in minutes, that an allocated archive read tape unit is allowed to remain unused before it is deallocated. This parameter, together with the **MaxReadTapeUnits** parameter, allows IBM MQ to optimize archive log reading from tape devices. You are recommended to specify the maximum values, within system constraints, for both parameters, in order to achieve the optimum performance for reading archive tapes.

Specify a value in the range zero and 1440. Zero means that a tape unit is deallocated immediately. If you specify a value of 1440, the tape unit is never deallocated.

LogCompression (MQCFIN)
Log compression parameter (parameter identifier: MQIACF_LOG_COMPRESS).

Specifies the log compression algorithm to enable.

The possible values are:

MQCOMPRESS_NONE
Log compression is disabled.

MQCOMPRESS_RLE
Enable run-length encoding log compression.

MQCOMPRESS_ANY
Enable the queue manager to select the compression algorithm that gives the greatest degree of log record compression.

z/OS For more details see The log files.

MaxArchiveLog (MQCFIN)
Specifies the maximum number of archive log volumes that can be recorded in the BSDS (parameter identifier: MQIACF_SYSP_MAX_ARCHIVE).

When this value is exceeded, recording recommences at the start of the BSDS.

Specify a value in the range 10 through 100.

MaxConcurrentOffloads (MQCFIN)
Specifies the maximum number of concurrent log offload tasks (parameter identifier: MQIACF_SYSP_MAX_CONC_OFFLOADS).

Specify a decimal number between 1 and 31. If no value is specified the default of 31 applies.

Configure a number lower than the default if your archive logs are allocated on a tape device, and there are constraints on the number of such devices that can be concurrently allocated to the queue manager.

MaxReadTapeUnits (MQCFIN)
Specifies the maximum number of dedicated tape units that can be allocated to read archive log tape volumes (parameter identifier: MQIACF_SYSP_MAX_READ_TAPES).

This parameter, together with the *DeallocateInterval* parameter, allows IBM MQ to optimize archive log reading from tape devices.

Specify a value in the range 1 through 99.

If you specify a value that is greater than the current specification, the maximum number of tape units allowable for reading archive logs increases. If you specify a value that is less than the current specification, tape units that are not being used are immediately deallocated to adjust to the new value. Active, or premounted, tapes remain allocated.

OutputBufferCount (MQCFIN)
Specifies the number of 4 KB output buffers to be filled before they are written to the active log data sets (parameter identifier: MQIACF_SYSP_OUT_BUFFER_COUNT).

Specify the number of buffers in the range 1 through 256.

The larger the number of buffers and the less often the write takes place improves the performance of IBM MQ. The buffers might be written before this number is reached if significant events, such as a commit point, occur.

Set Policy: Multi

The Set Policy (MQCMD_CHANGE_PROT_POLICY) command sets the protection policy.

Important: You must have an Advanced Message Security (AMS) license installed to issue this command. If you attempt to issue the **Set Policy** command without an AMS license installed, you receive message AMQ7155 - License file not found or not valid.

Syntax diagram

See the syntax diagram in the MQSC "SET POLICY" on page 1024 command for combinations of parameters and values that are allowed.

Required parameters

PolicyName (MQCFST)
Specifies the name of the policy. The policy name must match the name of the queue which is to be protected (parameter identifier: MQCA_POLICY_NAME).

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Optional parameters

SignAlg (MQCFIN)
Specifies the digital signature algorithm (parameter identifier: MQIA_SIGNATURE_ALGORITHM). The following values are valid:

MQESE_SIGN_ALG_NONE
No digital signature algorithm specified. This is the default value.

MQESE_SIGN_ALG_MD5
MD5 digital signature algorithm specified.

MQESE_SIGN_ALG_SHA1
SHA1 digital signature algorithm specified.

MQESE_SIGN_ALG_SHA256
SHA256 digital signature algorithm specified.

MQESE_SIGN_ALG_SHA384
SHA384 digital signature algorithm specified.

MQESE_SIGN_ALG_SHA512
SHA512 digital signature algorithm specified.

EncAlg (MQCFIN)
Specifies the encryption algorithm (parameter identifier: MQIA_ENCRYPTION_ALGORITHM). The following values are valid:

MQESE_ENC_ALG_NONE
No encryption algorithm specified. This is the default value.

MQESE_ENC_ALG_RC2
RC2 encryption algorithm specified.

MQESE_ENC_ALG_DES
DES encryption algorithm specified.

MQESE_ENC_ALG_3DES
3DES encryption algorithm specified.

MQESE_ENC_ALG_AES128
AES128 encryption algorithm specified.

MQESE_ENC_ALG_AES256
AES256 encryption algorithm specified.

Signer (MQCFST)
Specifies the distinguished name of an authorized signer. This parameter can be specified multiple times (parameter identifier: MQCA_SIGNER_DN).

Recipient (MQCFST)
Specifies the distinguished name of the intended recipient. This parameter can be specified multiple times (parameter identifier: MQCA_RECIPIENT_DN).

Enforce and Tolerate (MQCFST)
Indicates whether the security policy should be enforced or whether unprotected messages are tolerated (parameter identifier: MQIA_TOLERATE_UNPROTECTED). The following values are valid:

MQESE_TOLERATE_NO
Specifies that all message must be protected when retrieved from the queue. Any unprotected message encountered is moved to the SYSTEM.PROTECTION.ERROR.QUEUE. This is the default value.

MQESE_TOLERATE_YES
Specifies that the messages that are not protected when retrieved from the queue can ignore the policy.

Toleration is optional and exists to facilitate staged implementation, where:

- Policies have been applied to queues, but those queues might already contain unprotected messages, or
- Queues might still receive messages from remote systems that do not yet have the policy set.

V 9.0.0 **KeyReuse (MQCFIN)**
Specifies the number of times that an encryption key can be re-used, in the range 1-9,999,999, or the special values *MQKEY_REUSE_DISABLED* or *MQKEY_REUSE_UNLIMITED* (parameter identifier: MQIA_KEY_REUSE_COUNT). The following values are valid:

MQKEY_REUSE_DISABLED
Prevents a symmetric key from being reused. This is the default value.

MQKEY_REUSE_UNLIMITED
Allows a symmetric key to be reused any number of times.

Attention: Key reuse is valid only for CONFIDENTIALITY policies, that is, **SignAlg** set to *MQESE_SIGN_ALG_NONE* and **EncAlg** set to an algorithm value. For all other policy types, you must omit the parameter, or set the **Keyreuse** value to *MQKEY_REUSE_DISABLED*.

Action (MQCFIN)
Specifies the action for the parameters supplied, as they apply to any existing policy (parameter identifier: MQIACF_ACTION). The following values are valid:

MQACT_REPLACE
Has the effect of replacing any existing policy with the parameters supplied. This is the default value.

MQACT_ADD
Has the effect that signers and recipients parameters have an additive effect. That is, if a signer or recipient is specified, and does not already exist in a preexisting policy, the signer or recipient value is added to the existing policy definition.

MQACT_REMOVE
Has the opposite effect of *MQACT_ADD*. That is, if any of the signer or recipient values specified exist in a preexisting policy, those values are removed from the policy definition.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown at "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_POLICY_TYPE_ERROR
Policy type not valid.

Set System on z/OS: z/OS

Use the Set System (MQCMD_SET_SYSTEM) command to dynamically change certain general system parameter values initially set from your system parameter module at queue manager startup.

Required parameters:
ParameterType

Optional parameters (if the value of *ParameterType* is MQSYSP_TYPE_SET:
CheckpointCount, *CommandScope*, *Exclmsg*, *MaxConnects*, *MaxConnectsBackground*, *MaxConnectsForeground*, *Service*, *SMFInterval*, *TraceSize*

Optional parameters if *ParameterType* type is MQSYSP_INITIAL:
CommandScope

Required parameters

ParameterType (MQCFIN)
Parameter type (parameter identifier: MQIACF_SYSP_TYPE).

Specifies how the parameters are to be set:

MQSYSP_TYPE_INITIAL
The initial settings of the system parameters. MQSYSP_TYPE_INITIAL resets the parameters to the values specified in the system parameters at queue manager startup.

MQSYSP_TYPE_SET
MQSYSP_TYPE_SET indicates that you intend to change one, or more, of the system parameter settings.

Optional parameters

CheckpointCount (MQCFIN)
The number of log records written by IBM MQ between the start of one checkpoint and the next (parameter identifier: MQIACF_SYSP_CHKPOINT_COUNT).

IBM MQ starts a new checkpoint after the number of records that you specify has been written.

Specify a value in the range 200 through 16 000 000.

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Exclmsg (MQCFSL)
A list of message identifiers to be excluded from being written to any log (parameter identifier: MQCACF_EXCL_OPERATOR_MESSAGES).

Specify a list of error message identifiers to be excluded from being written to any log. For example, to exclude message CSQX500I, add X500 to this list. Messages in this list are not sent to the z/OS console and hardcopy log. As a result using the EXCLMSG parameter to exclude messages is more efficient from a CPU perspective than using z/OS mechanisms such as the message processing facility list and should be used instead where possible.

The maximum length of each message identifier is MQ_OPERATOR_MESSAGE_LENGTH.

The list can contain a maximum of 16 message identifiers.

Service (MQCFST)
Service parameter setting (parameter identifier: MQIACF_SYSP_SERVICE).

This parameter is reserved for use by IBM.

SMFInterval (MQCFIN)
The default time, in minutes, between each gathering of statistics (parameter identifier: MQIACF_SYSP_SMF_INTERVAL).

Specify a value in the range zero through 1440.

If you specify a value of zero, statistics data and accounting data are both collected at the SMF data collection broadcast.

TraceSize (MQCFIN)
The size of the trace table, in 4 KB blocks, to be used by the global trace facility (parameter identifier: MQIACF_SYSP_TRACE_SIZE).

Specify a value in the range zero through 999.

Start Channel:

The Start Channel (MQCMD_START_CHANNEL) command starts an IBM MQ channel. This command can be issued to a channel of any type (except MQCHT_CLNTCONN). If, however, it is issued to a channel with a *ChannelType* value of MQCHT_RECEIVER, MQCHT_SVRCONN, or MQCHT_CLUSRCVR, the only action is to enable the channel, not start it.

Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel.

If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the last channel added to the repository on the local queue manager.

None of the following attributes are applicable to MQTT channels unless specifically mentioned in the parameter description.

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be started. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

This parameter is required for all channel types including MQTT channels.

z/OS

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

ChannelDisposition (MQCFIN)
Channel disposition (parameter identifier: MQIACH_CHANNEL_DISP). This parameter applies to z/OS only.

Specifies the disposition of the channels to be started.

If this parameter is omitted, then the value for the channel disposition is taken from the default channel disposition attribute of the channel object.

The value can be:

MQCHLD_PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than MQQSGD_SHARED.

MQCHLD_SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of MQQSGD_SHARED.

MQCHLD_FIXSHARED
Shared channels tied to a specific queue manager.

The combination of the **ChannelDisposition** and **CommandScope** parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.
- On every active queue manager in the group.
- On the most suitable queue manager in the group, determined automatically by the queue manager itself.

The various combinations of *ChannelDisposition* and *CommandScope* are summarized in Table 136 on page 1574

Table 136. ChannelDisposition and CommandScope for START CHANNEL

ChannelDisposition	*CommandScope* **blank or local-qmgr**	*CommandScope* **qmgr-name**	*CommandScope(*)*
MQCHLD_PRIVATE	Start as a private channel on the local queue manager	Start as a private channel on the named queue manager	Start as a private channel on all active queue managers
MQCHLD_SHARED	For channels of *ChannelType* MQCHT_SENDER, MQCHT_REQUESTER, and MQCHT_SERVER, start as a shared channel on the most suitable queue manager in the group. For a shared channel of *ChannelType* MQCHT_RECEIVER and MQCHT_SVRCONN, start the channel on all active queue managers. For a shared channel of *ChannelType* MQCHT_CLUSSDR and MQCHT_CLUSRCVR, this option is not permitted. MQCHLD_SHARED might automatically generate a command using *CommandScope* and send it to the appropriate queue manager. If there is no definition for the channel on the queue manager to which the command is sent, or if the definition is unsuitable for the command, the command fails. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted	Not permitted
MQCHLD_FIXSHARED	For a shared channel of *ChannelType* MQCHT_SENDER, MQCHT_REQUESTER, and MQCHT_SERVER, with a nonblank *ConnectionName*, start as a shared channel on the local queue manager.	For a shared channel of *ChannelType* MQCHT_SENDER, MQCHT_REQUESTER, and MQCHT_SERVER, with a nonblank *ConnectionName*, start as a shared channel on the named queue manager.	Not permitted

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CHANNEL_INDOUBT
Channel in-doubt.

MQRCCF_CHANNEL_IN_USE
Channel in use.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_MQCONN_FAILED
MQCONN call failed.

MQRCCF_MQINQ_FAILED
MQINQ call failed.

MQRCCF_MQOPEN_FAILED
MQOPEN call failed.

MQRCCF_NOT_XMIT_Q
Queue is not a transmission queue.

Start Channel (MQTT): 

The Start Channel (MQCMD_START_CHANNEL) command starts an IBM MQ channel. This command can be issued to a channel of type MQCHT_MQTT.

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be started. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

This parameter is required for all channel types including MQTT channels.

ChannelType (MQCFIN)
The type of channel (parameter identifier: MQIACH_CHANNEL_TYPE). This parameter is currently only used with MQTT Telemetry channels, and is required when starting a Telemetry channel. The only value that can currently be given to the parameter is MQCHT_MQTT.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_PARM_SYNTAX_ERROR
The parameter specified contained a syntax error.

MQRCCF_PARM_MISSING
Parameters are missing.

MQRCCF_CHANNEL_NOT_FOUND
The channel specified does not exist.

MQRCCF_CHANNEL_IN_USE
The command did not specify a parameter or parameter value that was required.

MQRCCF_NO_STORAGE
Insufficient storage is available.

MQRCCF_COMMAND_FAILED
The command has failed.

MQRCCF_PORT_IN_USE
The port is in use.

MQRCCF_BIND_FAILED
The bind to a remote system during session negotiation has failed.

MQRCCF_SOCKET_ERROR
Socket error has occurred.

MQRCCF_HOST_NOT_AVAILABLE
An attempt to allocate a conversation to a remote system was unsuccessful. The error might be transitory, and the allocate might succeed later. This reason can occur if the listening program at the remote system is not running.

Start Channel Initiator:

The Start Channel Initiator (MQCMD_START_CHANNEL_INIT) command starts an IBM MQ channel initiator.

Required parameters

InitiationQName (MQCFST)
Initiation queue name (parameter identifier: MQCA_INITIATION_Q_NAME).

The name of the initiation queue for the channel initiation process. That is, the initiation queue that is specified in the definition of the transmission queue.

This parameter is not valid on z/OS.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- a queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_QSG_NAME_LENGTH.

EnvironmentInfo (MQCFST)
Environment information (parameter identifier: MQCACF_ENV_INFO).

The parameters and values to be substituted in the JCL procedure (xxxxCHIN, where xxxx is the queue manager name) that is used to start the channel initiator address space. This parameter applies to z/OS only.

The maximum length of the string is MQ_ENV_INFO_LENGTH.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_MQCONN_FAILED
MQCONN call failed.

MQRCCF_MQGET_FAILED
MQGET call failed.

MQRCCF_MQOPEN_FAILED
MQOPEN call failed.

Start Channel Listener:

The Start Channel Listener (MQCMD_START_CHANNEL_LISTENER) command starts an IBM MQ listener. On z/OS, this command is valid for any transmission protocol; on other platforms, it is valid only for TCP transmission protocols.

Optional parameters z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_Q_MGR_NAME_LENGTH.

z/OS

InboundDisposition (MQCFIN)
Inbound transmission disposition (parameter identifier: MQIACH_INBOUND_DISP). This parameter applies to z/OS only.

Specifies the disposition of the inbound transmissions that are to be handled. The value can be any of the following values:

MQINBD_Q_MGR
Listen for transmissions directed to the queue manager. MQINBD_Q_MGR is the default.

MQINBD_GROUP
Listen for transmissions directed to the queue-sharing group. MQINBD_GROUP is permitted only if there is a shared queue manager environment.

z/OS

IPAddress (MQCFST)
IP address (parameter identifier: MQCACH_IP_ADDRESS). This parameter applies to z/OS only.

The IP address for TCP/IP specified in IPv4 dotted decimal, IPv6 hexadecimal, or alphanumeric form. This parameter is valid only for channels that have a *TransportType* of MQXPT_TCP.

The maximum length of the string is MQ_IP_ADDRESS_LENGTH.

ListenerName (MQCFST)
Listener name (parameter identifier: MQCACH_LISTENER_NAME). This parameter does not apply to z/OS.

The name of the listener definition to be started. On those platforms on which this parameter is valid, if this parameter is not specified, the default listener SYSTEM.DEFAULT.LISTENER is assumed. If this parameter is specified, no other parameters can be specified.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

z/OS

LUName (MQCFST)
LU name (parameter identifier: MQCACH_LU_NAME). This parameter applies to z/OS only.

The symbolic destination name for the logical unit (LU) as specified in the APPC side information data set. The LU must be the same LU that is specified in the channel initiator parameters to be used for outbound transmissions. This parameter is valid only for channels with a *TransportType* of MQXPT_LU62.

The maximum length of the string is MQ_LU_NAME_LENGTH.

z/OS

Port (MQCFIN)
Port number for TCP (parameter identifier: MQIACH_PORT_NUMBER). This parameter applies to z/OS only.

The port number for TCP. This parameter is valid only for channels with a *TransportType* of MQXPT_TCP.

z/OS

TransportType (MQCFIN)
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value can be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

MQXPT_NETBIOS
NetBIOS.

MQXPT_SPX
SPX.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_COMMS_LIBRARY_ERROR
Communications protocol library error.

MQRCCF_LISTENER_NOT_STARTED
Listener not started.

MQRCCF_LISTENER_RUNNING
Listener already running.

MQRCCF_NETBIOS_NAME_ERROR
NetBIOS listener name error.

Start Service on Multiplatforms: Multi

The Start Service (MQCMD_START_SERVICE) command starts an existing IBM MQ service definition.

Required parameters

ServiceName (MQCFST)
Service name (parameter identifier: MQCA_SERVICE_NAME).

This parameter is the name of the service definition to be started. The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_NO_START_CMD
The **StartCommand** parameter of the service is blank.

MQRCCF_SERVICE_RUNNING
Service is already running.

Start SMDS Connection on z/OS: z/OS

Use the Start SMDS Connection (MQCMD_INQUIRE_SMDSCONN) command after connections have been put into the AVAIL(STOPPED) state by a previous STOP SMDSCONN command. It can also be used to signal to the queue manager to retry a connection which is in the AVAIL(ERROR) state after a previous error.

Required parameters

SMDSConn (MQCFST)
Specifies the queue manager name relating to the connection between the shared message data set and the queue manager (parameter identifier: MQCACF_CF_SMDSCONN).

An asterisk value can be used to denote all shared message data sets associated with a specific CFSTRUCT name.

The maximum length of the string is 4 characters.

CFStrucName (MQCFST)
The name of the CF application structure with SMDS connections properties that you want to start (parameter identifier: MQCA_CF_STRUC_NAME).

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Stop Channel:

The Stop Channel (MQCMD_STOP_CHANNEL) command stops an IBM MQ channel.

This command can be issued to a channel of any type (except MQCHT_CLNTCONN).

Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel.

If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the last channel added to the repository on the local queue manager.

None of the following attributes are applicable to MQTT channels unless specifically mentioned in the parameter description.

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be stopped. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

This parameter is required for all channel types..

Optional parameters

z/OS **ChannelDisposition (MQCFIN)**
Channel disposition (parameter identifier: MQIACH_CHANNEL_DISP). This parameter applies to z/OS only.

Specifies the disposition of the channels to be stopped.

If this parameter is omitted, then the value for the channel disposition is taken from the default channel disposition attribute of the channel object.

The value can be any of the following values:

MQCHLD_PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than MQQSGD_SHARED.

MQCHLD_SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of MQQSGD_SHARED.

The combination of the **ChannelDisposition** and **CommandScope** parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.
- On every active queue manager in the group.
- On the most suitable queue manager in the group, determined automatically by the queue manager itself.

The various combinations of *ChannelDisposition* and *CommandScope* are summarized in Table 137

Table 137. ChannelDisposition and CommandScope for STOP CHANNEL

ChannelDisposition	*CommandScope* **blank or local-qmgr**	*CommandScope* **qmgr-name**	*CommandScope(*)*
MQCHLD_PRIVATE	Stop as a private channel on the local queue manager	Stop as a private channel on the named queue manager	Stop as a private channel on all active queue managers

Table 137. ChannelDisposition and CommandScope for STOP CHANNEL (continued)

ChannelDisposition	*CommandScope* **blank or local-qmgr**	*CommandScope* **qmgr-name**	*CommandScope(*)*
MQCHLD_SHARED	For channels of *ChannelType* MQCHT_RECEIVER or MQCHT_SVRCONN, stop as shared channel on all active queue managers. For channels of *ChannelType* MQCHT_SENDER, MQCHT_REQUESTER, and MQCHT_SERVER, stop as a shared channel on the queue manager where it is running. If the channel is in an inactive state (not running), or if it is in RETRY state because the channel initiator on which it was running has stopped, a STOP request for the channel is issued on the local queue manager. MQCHLD_SHARED might automatically generate a command using *CommandScope* and send it to the appropriate queue manager. If there is no definition for the channel on the queue manager to which the command is sent, or if the definition is unsuitable for the command, the command fails. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted	Not permitted

ChannelStatus (MQCFIN)

The new state of the channel after the command is executed (parameter identifier: MQIACH_CHANNEL_STATUS).

The value can be any of the following values:

MQCHS_INACTIVE
Channel is inactive.

MQCHS_STOPPED
Channel is stopped. MQCHS_STOPPED is the default if nothing is specified.

z/OS **CommandScope (MQCFST)**

Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is processed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- a queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

ConnectionName (MQCFST)
Connection name of channel to be stopped (parameter identifier: MQCACH_CONNECTION_NAME).

This parameter is the connection name of the channel to be stopped. If this parameter is omitted, all channels with the specified channel name and remote queue manager name are stopped. On Multiplatforms, the maximum length of the string is MQ_CONN_NAME_LENGTH. On z/OS, the maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

If this parameter is specified, ChannelStatus must be MQCHS_INACTIVE.

Mode (MQCFIN)
How the channel must be stopped (parameter identifier: MQIACF_MODE).

The value can be:

MQMODE_QUIESCE
Quiesce the channel. MQMODE_QUIESCE is the default.

If you issue a `Stop Channel` *`channelname`* `Mode(MQMODE_QUIESCE)` command on a server-connection channel with the sharing conversations feature enabled, the IBM MQ client infrastructure becomes aware of the stop request in a timely manner; this time is dependent upon the speed of the network. The client application becomes aware of the stop request as a result of issuing a subsequent call to IBM MQ.

MQMODE_FORCE
Stop the channel immediately; the thread or process of the channel is not terminated. Stops transmission of any current batch.

For server-connection channels, breaks the current connection, returning MQRC_CONNECTION_BROKEN.

For other types of channels, this situation is likely to result in in-doubt situations.

z/OS On z/OS, this option interrupts any message reallocation in progress, which can leave BIND_NOT_FIXED messages partially reallocated or out of order.

MQMODE_TERMINATE
Multi On Multiplatforms, stop the channel immediately; the thread or process of the channel is terminated.

z/OS On z/OS, MQMODE_TERMINATE is synonymous with FORCE.

z/OS On z/OS, this option interrupts any message reallocation in progress, which can leave BIND_NOT_FIXED messages partially reallocated or out of order.

Note: This parameter was previously called *Quiesce* (MQIACF_QUIESCE), with values MQQO_YES and MQQO_NO. The old names can still be used.

QMgrName (MQCFST)
Name of remote queue manager (parameter identifier: MQCA_Q_MGR_NAME).

This parameter is the name of the remote queue manager to which the channel is connected. If this parameter is omitted, all channels with the specified channel name and connection name are stopped. The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

If this parameter is specified, ChannelStatus must be MQCHS_INACTIVE.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CHANNEL_DISABLED
Channel disabled.

MQRCCF_CHANNEL_NOT_ACTIVE
Channel not active.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_MODE_VALUE_ERROR
Mode value not valid.

MQRCCF_MQCONN_FAILED
MQCONN call failed.

MQRCCF_MQOPEN_FAILED
MQOPEN call failed.

MQRCCF_MQSET_FAILED
MQSET call failed.

Stop Channel (MQTT): 

The Stop Channel (MQCMD_STOP_CHANNEL) command stops an MQ Telemetry channel.

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

This parameter is required.

The name of the channel to be stopped. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType (MQCFIN)
The type of channel (parameter identifier: MQIACH_CHANNEL_TYPE). This parameter is currently only used with MQTT Telemetry channels, and is required when stopping a Telemetry channel. The only value that can currently be given to the parameter is **MQCHT_MQTT**.

Optional parameters

ClientIdentifier (MQCFST)
Client identifier. The client identifier is a 23-byte string that identifies an MQ Telemetry Transport client. When the Stop Channel command specifies a *ClientIdentifier*, only the connection for the specified client identifier is stopped. If the CLIENTID is not specified, all the connections on the channel are stopped.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CHANNEL_DISABLED
Channel disabled.

MQRCCF_CHANNEL_NOT_ACTIVE
Channel not active.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_MODE_VALUE_ERROR
Mode value not valid.

MQRCCF_MQCONN_FAILED
MQCONN call failed.

MQRCCF_MQOPEN_FAILED
MQOPEN call failed.

MQRCCF_MQSET_FAILED
MQSET call failed.

Stop Channel Initiator on z/OS: z/OS

The Stop Channel Initiator (MQCMD_STOP_CHANNEL_INIT) command stops an IBM MQ channel initiator.

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

SharedChannelRestart (MQCFIN)
Shared channel restart (parameter identifier: MQIACH_SHARED_CHANNEL_RESTART).

Specifies whether the channel initiator attempts to restart any active sending channels, started with the **ChannelDisposition** parameter set to MQCHLD_SHARED, that it owns on another queue manager. The value can be:

MQCHSH_RESTART_YES
Shared sending channels are to be restarted. MQCHSH_RESTART_YES is the default.

MQCHSH_RESTART_NO
Shared sending channels are not to be restarted, so become inactive.

Active channels started with the **ChannelDisposition** parameter set to MQCHLD_FIXSHARED are not restarted, and always become inactive.

Stop Channel Listener:

The Stop Channel Listener (MQCMD_STOP_CHANNEL_LISTENER) command stops an IBM MQ listener.

Required parameters

ListenerName (MQCFST)
Listener name (parameter identifier: MQCACH_LISTENER_NAME). This parameter does not apply to z/OS.

The name of the listener definition to be stopped. If this parameter is specified, no other parameters can be specified.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

z/OS

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is processed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- a queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

This parameter is valid only on z/OS.

The maximum length is MQ_QSG_NAME_LENGTH.

InboundDisposition (MQCFIN)
Inbound transmission disposition (parameter identifier: MQIACH_INBOUND_DISP).

Specifies the disposition of the inbound transmissions that the listener handles. The value can be any of the following values:

MQINBD_Q_MGR
Handling for transmissions directed to the queue manager. MQINBD_Q_MGR is the default.

MQINBD_GROUP
Handling for transmissions directed to the queue-sharing group. MQINBD_GROUP is permitted only if there is a shared queue manager environment.

This parameter is valid only on z/OS.

IPAddress (MQCFST)
IP address (parameter identifier: MQCACH_IP_ADDRESS).

The IP address for TCP/IP specified in dotted decimal or alphanumeric form. This parameter is valid on z/OS only where channels have a *TransportType* of MQXPT_TCP.

The maximum length of the string is MQ_IP_ADDRESS_LENGTH.

Port (MQCFIN)
Port number for TCP (parameter identifier: MQIACH_PORT_NUMBER).

The port number for TCP. This parameter is valid only on z/OS where channels have a *TransportType* of MQXPT_TCP.

TransportType (MQCFIN)
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value can be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

This parameter is valid only on z/OS.

Error codes

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_LISTENER_STOPPED
Listener not running.

Stop Connection on Multiplatforms: Multi

The Stop Connection (MQCMD_STOP_CONNECTION) command attempts to break a connection between an application and the queue manager. There might be circumstances in which the queue manager cannot implement this command.

Required parameters

ConnectionId (MQCFBS)
Connection identifier (parameter identifier: MQBACF_CONNECTION_ID).

This parameter is the unique connection identifier associated with an application that is connected to the queue manager.

The length of the byte string is MQ_CONNECTION_ID_LENGTH.

Stop Service on Multiplatforms: Multi

The Stop Service (MQCMD_STOP_SERVICE) command stops an existing IBM MQ service definition that is running.

Required parameters

ServiceName (MQCFST)
Service name (parameter identifier: MQCA_SERVICE_NAME).

This parameter is the name of the service definition to be stopped. The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown on page "Error codes applicable to all commands" on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_NO_STOP_CMD
The **StopCommand** parameter of the service is blank.

MQRCCF_SERVICE_STOPPED
Service is not running.

Stop SMDS Connection on z/OS: z/OS

Use the Stop SMDS Connection (MQCMD_STOP_SMDSCONN) command to terminate the connection from this queue manager to one or more specified shared message data sets (causing them to be closed and deallocated) and to mark the connection as STOPPED.

Required parameters

SMDSConn (MQCFST)
Specifies the queue manager name relating to the connection between the shared message data set and the queue manager (parameter identifier: MQCACF_CF_SMDSCONN).

An asterisk value can be used to denote all shared message data sets associated with a specific CFSTRUCT name.

The maximum length of the string is 4 characters.

CFStrucName (MQCFST)
The name of the CF application structure with SMDS connections properties that you want to stop (parameter identifier: MQCA_CF_STRUC_NAME).

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is processed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- a queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

- an asterisk (*). The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Suspend Queue Manager on z/OS: z/OS

The Suspend Queue Manager (MQCMD_SUSPEND_Q_MGR) command renders the local queue manager unavailable for the processing of IMS or Db2 messages. Its action can be reversed by the Resume Queue Manager command (MQCMD_RESUME_Q_MGR) command.

Required parameters

Facility (MQCFIN)
Facility (parameter identifier: MQIACF_FACILITY).

The type of facility for which activity is to be suspended. The value can be:

MQQMFAC_DB2
The existing connection to Db2 is terminated.

Any in-flight or subsequent MQGET or MQPUT requests are suspended and applications wait until the Db2 connection is re-established by the Resume Queue Manager command, or if the queue manager is stopped.

MQQMFAC_IMS_BRIDGE
Resumes normal IMS bridge activity.

Stops the sending of messages from IMS bridge queues to OTMA. No further messages are sent to IMS until one of these events occurs:

- OTMA is stopped and restarted
- IMS or IBM MQ is stopped or restarted
- A Resume Queue Manager command is processed

Messages returning from IMS OTMA to the queue manager are unaffected.

Optional parameters

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is processed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- a queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_QSG_NAME_LENGTH.

Suspend Queue Manager Cluster:

The Suspend Queue Manager Cluster (MQCMD_SUSPEND_Q_MGR_CLUSTER) command informs other queue managers in a cluster that the local queue manager is not available for processing, and cannot be sent messages. Its action can be reversed by the Resume Queue Manager Cluster (MQCMD_RESUME_Q_MGR_CLUSTER) command.

Required parameters

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster for which availability is to be suspended.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

ClusterNamelist (MQCFST)
Cluster Namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

The name of the namelist specifying a list of clusters for which availability is to be suspended.

Optional parameters z/OS

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is processed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- a queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_QSG_NAME_LENGTH.

Mode (MQCFIN)
How the local queue manager is suspended from the cluster (parameter identifier: MQIACF_MODE).

The value can be:

MQMODE_QUIESCE
Other queue managers in the cluster are told not to send further messages to the local queue manager.

MQMODE_FORCE
All inbound and outbound channels to other queue managers in the cluster are stopped forcibly.

Note: This parameter was previously called *Quiesce* (MQIACF_QUIESCE), with values MQQO_YES and MQQO_NO. The old names can still be used.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 1075.

Reason (MQLONG)
The value can be any of the following values:

MQRCCF_CLUSTER_NAME_CONFLICT
Cluster name conflict.

MQRCCF_MODE_VALUE_ERROR
Mode value not valid.

Structures for commands and responses

PCF commands and responses have a consistent structure including of a header and any number of parameter structures of defined types.

Commands and responses have the form:

- PCF header (MQCFH) structure (described in topic "MQCFH - PCF header" on page 1592), followed by
- Zero or more parameter structures. Each of these is one of the following:
  - PCF byte string filter parameter (MQCFBF, see topic "MQCFBF - PCF byte string filter parameter" on page 1596)
  - PCF byte string parameter (MQCFBS, see topic "MQCFBS - PCF byte string parameter" on page 1599)
  - PCF integer filter parameter (MQCFIF, see topic "MQCFIF - PCF integer filter parameter" on page 1601)
  - PCF integer list parameter (MQCFIL, see topic "MQCFIL - PCF integer list parameter" on page 1604)
  - PCF integer parameter (MQCFIN, see topic "MQCFIN - PCF integer parameter" on page 1606)
  - PCF string filter parameter (MQCFSF, see topic "MQCFSF - PCF string filter parameter" on page 1608)
  - PCF string list parameter (MQCFSL, see topic "MQCFSL - PCF string list parameter" on page 1612)
  - PCF string parameter (MQCFST, see topic "MQCFST - PCF string parameter" on page 1615)

How the structures are shown:

The structures are described in a language-independent form.

The declarations are shown in the following programming languages:

- C
- COBOL
- PL/I
- S/390 assembler
- Visual Basic

Data types

For each field of the structure, the data type is given in brackets after the field name. These data types are the elementary data types described in Data types used in the MQI.

Initial values and default structures

See IBM MQ COPY, header, include, and module files for details of the supplied header files that contain the structures, constants, initial values, and default structures.

Usage notes:

The format of the strings in the PCF message determines the settings of the character set fields in the message descriptor to enable conversion of strings within the message.

If all of the strings in a PCF message have the same coded character-set identifier, the *CodedCharSetId* field in the message descriptor MQMD should be set to that identifier when the message is put, and the *CodedCharSetId* fields in the MQCFST, MQCFSL, and MQCFSF structures within the message should be set to MQCCSI_DEFAULT.

If the format of the PCF message is MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF and some of the strings in the message have different character-set identifiers, the *CodedCharSetId* field in MQMD should be set to MQCCSI_EMBEDDED when the message is put, and the *CodedCharSetId* fields in the MQCFST, MQCFSL, and MQCFSF structures within the message should all be set to the identifiers that apply.

This enables conversions of the strings within the message, to the *CodedCharSetId* value in the MQMD specified on the MQGET call, if the MQGMO_CONVERT option is also specified.

For more information about the MQEPH structure, see MQEPH - Embedded PCF header.

Note: If you request conversion of the internal strings in the message, the conversion will occur only if the value of the *CodedCharSetId* field in the MQMD of the message is different from the *CodedCharSetId* field of the MQMD specified on the MQGET call.

Do not specify MQCCSI_EMBEDDED in MQMD when the message is put, with MQCCSI_DEFAULT in the MQCFST, MQCFSL, or MQCFSF structures within the message, as this will prevent conversion of the message.

MQCFH - PCF header:

The MQCFH structure describes the information that is present at the start of the message data of a command message, or a response to a command message. In either case, the message descriptor *Format* field is MQFMT_ADMIN.

The PCF structures are also used for event messages. In this case the message descriptor *Format* field is MQFMT_EVENT.

The PCF structures can also be used for user-defined message data. In this case the message descriptor *Format* field is MQFMT_PCF (see Message descriptor for a PCF command). Also in this case, not all the fields in the structure are meaningful. The supplied initial values can be used for most fields, but the application must set the *StrucLength* and *ParameterCount* fields to the values appropriate to the data.

Fields for MQCFH

Type (MQLONG)
Structure type.

This field indicates the content of the message. The following values are valid for commands:

MQCFT_COMMAND
Message is a command.

MQCFT_COMMAND_XR
Message is a command to which standard or extended responses might be sent.

This value is required on z/OS.

MQCFT_RESPONSE
Message is a response to a command.

MQCFT_XR_MSG
Message is an extended response to a command. It contains informational or error details.

MQCFT_XR_ITEM
Message is an extended response to an Inquire command. It contains item data.

MQCFT_XR_SUMMARY
Message is an extended response to a command. It contains summary information.

MQCFT_USER
User-defined PCF message.

StrucLength (MQLONG)
Structure length.

This field is the length in bytes of the MQCFH structure. The value must be:

MQCFH_STRUC_LENGTH
Length of command format header structure.

Version (MQLONG)
Structure version number.

For z/OS, the value must be:

MQCFH_VERSION_3
Version number for command format header structure.

The following constant specifies the version number of the current version:

MQCFH_CURRENT_VERSION
Current version of command format header structure.

Command (MQLONG)
Command identifier.

For a command message, this field identifies the function to be performed. For a response message, it identifies the command to which this field is the reply. See the description of each command for the value of this field.

MsgSeqNumber (MQLONG)
Message sequence number.

This field is the sequence number of the message within a set of related messages. For a command, this field must have the value one (because a command is always contained within a single message). For a response, the field has the value one for the first (or only) response to a command, and increases by one for each successive response to that command.

The last (or only) message in a set has the MQCFC_LAST flag set in the *Control* field.

Control (MQLONG)
Control options.

The following values are valid:

MQCFC_LAST
Last message in the set.

For a command, this value must always be set.

MQCFC_NOT_LAST
Not the last message in the set.

CompCode (MQLONG)
Completion code.

This field is meaningful only for a response; its value is not significant for a command. The following values are possible:

MQCC_OK
Command completed successfully.

MQCC_WARNING
Command completed with warning.

MQCC_FAILED
Command failed.

MQCC_UNKNOWN
Whether command succeeded is not known.

Reason (MQLONG)
Reason code qualifying completion code.

This field is meaningful only for a response; its value is not significant for a command.

The possible reason codes that can be returned in response to a command are listed in, "Definitions of the Programmable Command Formats" on page 1069 and in the description of each command.

ParameterCount (MQLONG)
Count of parameter structures.

This field is the number of parameter structures (MQCFBF, MQCFBS, MQCFIF, MQCFIL, MQCFIN, MQCFSL, MQCFSF, and MQCFST) that follow the MQCFH structure. The value of this field is zero or greater.

C language declaration

```
typedef struct tagMQCFH {
  MQLONG  Type;            /* Structure type */
  MQLONG  StrucLength;     /* Structure length */
  MQLONG  Version;         /* Structure version number */
  MQLONG  Command;         /* Command identifier */
  MQLONG  MsgSeqNumber;    /* Message sequence number */
  MQLONG  Control;         /* Control options */
  MQLONG  CompCode;        /* Completion code */
  MQLONG  Reason;          /* Reason code qualifying completion code */
  MQLONG  ParameterCount;  /* Count of parameter structures */
 } MQCFH;
```

COBOL language declaration

```
**   MQCFH structure
  10 MQCFH.
**    Structure type
   15 MQCFH-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFH-STRUCLENGTH    PIC S9(9) BINARY.
**    Structure version number
   15 MQCFH-VERSION        PIC S9(9) BINARY.
**    Command identifier
   15 MQCFH-COMMAND        PIC S9(9) BINARY.
**    Message sequence number
   15 MQCFH-MSGSEQNUMBER   PIC S9(9) BINARY.
**    Control options
   15 MQCFH-CONTROL        PIC S9(9) BINARY.
**    Completion code
   15 MQCFH-COMPCODE       PIC S9(9) BINARY.
```

```
**    Reason code qualifying completion code
   15 MQCFH-REASON         PIC S9(9) BINARY.
**    Count of parameter structures
   15 MQCFH-PARAMETERCOUNT PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
 1 MQCFH based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Version        fixed bin(31), /* Structure version number */
  3 Command        fixed bin(31), /* Command identifier */
  3 MsgSeqNumber   fixed bin(31), /* Message sequence number */
  3 Control        fixed bin(31), /* Control options */
  3 CompCode       fixed bin(31), /* Completion code */
  3 Reason         fixed bin(31), /* Reason code qualifying completion
                                     code */
  3 ParameterCount fixed bin(31); /* Count of parameter structures */
```

System/390 assembler-language declaration (z/OS only)

```
MQCFH                          DSECT
MQCFH_TYPE                     DS   F        Structure type
MQCFH_STRUCLENGTH              DS   F        Structure length
MQCFH_VERSION                  DS   F        Structure version number
MQCFH_COMMAND                  DS   F        Command identifier
MQCFH_MSGSEQNUMBER             DS   F        Message sequence number
MQCFH_CONTROL                  DS   F        Control options
MQCFH_COMPCODE                 DS   F        Completion code
MQCFH_REASON                   DS   F        Reason code qualifying
*                                            completion code
MQCFH_PARAMETERCOUNT           DS   F        Count of parameter
*                                            structures
MQCFH_LENGTH                   EQU  *-MQCFH  Length of structure
                               ORG  MQCFH
MQCFH_AREA                     DS   CL(MQCFH_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFH
  Type As Long             'Structure type
  StrucLength As Long      'Structure length
  Version As Long          'Structure version number
  Command As Long          'Command identifier
  MsgSeqNumber As Long     'Message sequence number
  Control As Long          'Control options
  CompCode As Long         'Completion code
  Reason As Long           'Reason code qualifying completion code
  ParameterCount As Long   'Count of parameter structures
End Type

Global MQCFH_DEFAULT As MQCFH
```

RPG language declaration (IBM i only)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQCFH Structure
D*
D* Structure type
D  FHTYP                  1      4I 0 INZ(1)
D* Structure length
D  FHLEN                  5      8I 0 INZ(36)
D* Structure version number
D  FHVER                  9     12I 0 INZ(1)
D* Command identifier
D  FHCMD                 13     16I 0 INZ(0)
D* Message sequence number
```

```
D  FHSEQ                 17     20I 0 INZ(1)
D* Control options
D  FHCTL                 21     24I 0 INZ(1)
D* Completion code
D  FHCMP                 25     28I 0 INZ(0)
D* Reason code qualifying completion code
D  FHREA                 29     32I 0 INZ(0)
D* Count of parameter structures
D  FHCNT                 33     36I 0 INZ(0)
D*
```

MQCFBF - PCF byte string filter parameter:

The MQCFBF structure describes a byte string filter parameter. The format name in the message descriptor is MQFMT_ADMIN.

The MQCFBF structure is used in Inquire commands to provide a filter description. This filter description is used to filter the results of the Inquire command and return to the user only those objects that satisfy the filter description.

When an MQCFBF structure is present, the Version field in the MQCFH structure at the start of the PCF must be MQCFH_VERSION_3 or higher.

Fields for MQCFBF

Type (MQLONG)
Structure type.

This indicates that the structure is a MQCFBF structure describing a byte string filter parameter. The value must be:

MQCFT_BYTE_STRING_FILTER
Structure defining a byte string filter.

StrucLength (MQLONG)
Structure length.

This is the length, in bytes, of the MQCFBF structure, including the string at the end of the structure (the *FilterValue* field). The length must be a multiple of 4, and must be sufficient to contain the string. Bytes between the end of the string and the length defined by the *StrucLength* field are not significant.

The following constant gives the length of the *fixed* part of the structure, that is the length excluding the *FilterValue* field:

MQCFBF_STRUC_LENGTH_FIXED
Length of fixed part of command format filter string-parameter structure.

Parameter (MQLONG)
Parameter identifier.

This identifies the parameter that is to be filtered on. The value of this identifier depends on the parameter to be filtered on.

The parameter is one of the following:

- MQBACF_EXTERNAL_UOW_ID
- MQBACF_Q_MGR_UOW_ID
- MQBACF_ORIGIN_UOW_ID (on z/OS only)

Operator (MQLONG)
Operator identifier.

This identifies the operator that is being used to evaluate whether the parameter satisfies the filter-value.

Possible values are:

MQCFOP_GREATER
Greater than

MQCFOP_LESS
Less than

MQCFOP_EQUAL
Equal to

MQCFOP_NOT_EQUAL
Not equal to

MQCFOP_NOT_LESS
Greater than or equal to

MQCFOP_NOT_GREATER
Less than or equal to

FilterValueLength (MQLONG)
Length of filter-value string.

This is the length, in bytes, of the data in the *FilterValue* field. This must be zero or greater, and does not need to be a multiple of 4.

FilterValue (MQBYTE x *FilterValueLength*)
Filter value.

This specifies the filter-value that must be satisfied. Use this parameter where the response type of the filtered parameter is a byte string.

Note: If the specified byte string is shorter than the standard length of the parameter in MQFMT_ADMIN command messages, the omitted characters are assumed to be blanks. If the specified string is longer than the standard length, it is an error.

C language declaration

```
typedef struct tagMQCFBF {
   MQLONG  Type;               /* Structure type */
   MQLONG  StrucLength;        /* Structure length */
   MQLONG  Parameter;          /* Parameter identifier */
   MQLONG  Operator;           /* Operator identifier */
   MQLONG  FilterValueLength;  /* Filter value length */
   MQBYTE  FilterValue[1];     /* Filter value -- first byte */
  } MQCFBF;
```

COBOL language declaration

```
**   MQCFBF structure
  10  MQCFBF.
** Structure type
  15  MQCFBF-TYPE PIC S9(9) BINARY.
** Structure length
  15  MQCFBF-STRUCLENGTH PIC S9(9) BINARY.
** Parameter identifier
  15  MQCFBF-PARAMETER PIC S9(9) BINARY.
** Operator identifier
  15  MQCFBF-OPERATOR PIC S9(9) BINARY.
** Filter value length
  15  MQCFBF-FILTERVALUELENGTH PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
  1 MQCFBF based,
   3 Type  fixed bin(31)
    init(MQCFT_BYTE_STRING_FILTER),  /* Structure type */
   3 StrucLength  fixed bin(31)
    init(MQCFBF_STRUC_LENGTH_FIXED),  /* Structure length */
   3 Parameter  fixed bin(31)
    init(0),  /* Parameter identifier */
   3 Operator  fixed bin(31)
    init(0),  /* Operator identifier */
   3 FilterValueLength  fixed bin(31)
    init(0);  /* Filter value length */
```

System/390 assembler-language declaration (z/OS only)

```
MQCFBF                      DSECT
MQCFBF_TYPE                 DS  F    Structure type
MQCFBF_STRUCLENGTH          DS  F    Structure length
MQCFBF_PARAMETER            DS  F    Parameter identifier
MQCFBF_OPERATOR             DS  F    Operator identifier
MQCFBF_FILTERVALUELENGTH    DS  F    Filter value length
MQCFBF_LENGTH               EQU  *-MQCFIF Length of structure
                            ORG  MQCFBF
MQCFBF_AREA                 DS   CL(MQCFBF_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFBF
  Type As Long 'Structure type'
  StrucLength As Long 'Structure length'
  Parameter As Long 'Parameter identifier'
  Operator As Long 'Operator identifier'
  FilterValueLength As Long 'Filter value length'
  FilterValue As 1 'Filter value -- first byte'
End Type
Global MQCFBF_DEFAULT As MQCFBF
```

RPG language declaration (IBM i only)

```
D* MQCFBF Structure
D*
D* Structure type
D  FBFTYP                 1      4I 0 INZ(15)
D* Structure length
D  FBFLEN                 5      8I 0 INZ(20)
D* Parameter identifier
D  FBFPRM                 9     12I 0 INZ(0)
D* Operator identifier
D  FBFOP                 13     16I 0 INZ(0)
D* Filter value length
D  FBFFVL                17     20I 0 INZ(0)
D* Filter value -- first byte
D  FBFFV                 21     21    INZ
```

MQCFBS - PCF byte string parameter:

The MQCFBS structure describes a byte-string parameter in a PCF message. The format name in the message descriptor is MQFMT_ADMIN.

When an MQCFBS structure is present, the *Version* field in the MQCFH structure at the start of the PCF must be MQCFH_VERSION_2 or greater.

In a user PCF message, the *Parameter* field has no significance, and can be used by the application for its own purposes.

The structure ends with a variable-length byte string; see the *String* field in the following section for further details.

Fields for MQCFBS

Type (MQLONG)
Structure type.

This indicates that the structure is an MQCFBS structure describing byte string parameter. The value must be:

MQCFT_BYTE_STRING
Structure defining a byte string.

StrucLength (MQLONG)
Structure length.

This is the length in bytes of the MQCFBS structure, including the variable-length string at the end of the structure (the *String* field). The length must be a multiple of four, and must be sufficient to contain the string; any bytes between the end of the string and the length defined by the *StrucLength* field are not significant.

The following constant gives the length of the *fixed* part of the structure, that is the length excluding the *String* field:

MQCFBS_STRUC_LENGTH_FIXED
Length of fixed part of MQCFBS structure.

Parameter (MQLONG)
Parameter identifier.

This identifies the parameter with a value that is contained in the structure. The values that can occur in this field depend on the value of the *Command* field in the MQCFH structure; see "MQCFH - PCF header" on page 1592 for details. In user PCF messages (MQCFT_USER), this field has no significance.

The parameter is from the MQBACF_* group of parameters.

StringLength (MQLONG)
Length of string.

This is the length in bytes of the data in the *string* field; it must be zero or greater. This length does not need to be a multiple of four.

String (MQBYTE x *StringLength*)
String value.

This is the value of the parameter identified by the *parameter* field. The string is a byte string, and so is not subject to character-set conversion when sent between different systems.

Note: A null character in the string is treated as normal data, and does not act as a delimiter for the string

For MQFMT_ADMIN messages, if the specified string is shorter than the standard length of the *parameter,* the omitted characters are assumed to be nulls. If the specified string is longer than the standard length, it is an error.

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure must be allocated dynamically, and pointers used to address the fields within it.
- For other programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, you must include MQCFBS in a larger structure, and declare additional fields following MQCFBS, to represent the *String* field as required.

C language declaration

```
typedef struct tagMQCFBS {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  StringLength; /* Length of string */
  MQBYTE  String[1];    /* String value - first byte */



   } MQCFBS;
```

COBOL language declaration

```
**    MQCFBS structure
   10 MQCFBS.
**    Structure type
   15 MQCFBS-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFBS-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFBS-PARAMETER   PIC S9(9) BINARY.
**    Length of string
   15 MQCFBS-STRINGLENGTH PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
 1 MQCFBS based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 StringLength fixed bin(31) /* Length of string */
```

System/390 assembler-language declaration (z/OS only)

```
MQCFBS                        DSECT
MQCFBS_TYPE                   DS   F        Structure type
MQCFBS_STRUCLENGTH            DS   F        Structure length
MQCFBS_PARAMETER              DS   F        Parameter identifier
MQCFBS_STRINGLENGTH           DS   F        Length of string
                              ORG  MQCFBS
MQCFBS_AREA                   DS   CL(MQCFBS_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFBS
  Type As Long         ' Structure type
  StrucLength As Long  ' Structure length
  Parameter As Long    ' Parameter identifier
  StringLength As Long ' Operator identifier
```

```
  String as 1          ' String value - first byte
  End Type

Global MQCFBS_DEFAULT As MQCFBS
```

RPG language declaration (IBM i only)

```
D* MQCFBS Structure
D*
D* Structure type
D  BSTYP                   1      4I 0 INZ(3)
D* Structure length
D  BSLEN                   5      8I 0 INZ(16)
D* Parameter identifier
D  BSPRM                   9     12I 0 INZ(0)
D* Length of string
D  BSSTL                  13     16I 0 INZ(0)
D* String value - first byte
D  BSSRA                  17     16
D*
```

MQCFIF - PCF integer filter parameter:

The MQCFIF structure describes an integer filter parameter. The format name in the message descriptor is MQFMT_ADMIN.

The MQCFIF structure is used in Inquire commands to provide a filter condition. This filter condition is used to filter the results of the Inquire command and return to the user only those objects that satisfy the filter condition.

When an MQCFIF structure is present, the Version field in the MQCFH structure at the start of the PCF must be MQCFH_VERSION_3 or higher.

Fields for MQCFIF

Type (MQLONG)
Structure type.

This indicates that the structure is an MQCFIF structure describing an integer filter parameter. The value must be:

MQCFT_INTEGER_FILTER
Structure defining an integer filter.

StrucLength (MQLONG)
Structure length.

This is the length in bytes of the MQCFIF structure. The value must be:

MQCFIF_STRUC_LENGTH
Length of command format integer-parameter structure.

Parameter (MQLONG)
Parameter identifier.

This identifies the parameter that is to be filtered on. The value of this identifier depends on the parameter to be filtered on. Any of the parameters which can be used in the Inquire command can be used in this field.

The parameter is from the following groups of parameters:

- MQIA_*
- MQIACF_*
- MQIAMO_*

- MQIACH_*

Operator (MQLONG)
Operator identifier.

This identifies the operator that is being used to evaluate whether the parameter satisfies the filter-value.

Possible values are:

MQCFOP_GREATER
Greater than

MQCFOP_LESS
Less than

MQCFOP_EQUAL
Equal to

MQCFOP_NOT_EQUAL
Not equal to

MQCFOP_NOT_LESS
Greater than or equal to

MQCFOP_NOT_GREATER
Less than or equal to

MQCFOP_CONTAINS
Contains a specified value. Use MQCFOP_CONTAINS when filtering on lists of values or integers.

MQCFOP_EXCLUDES
Does not contain a specified value. Use MQCFOP_EXCLUDES when filtering on lists of values or integers.

See the *FilterValue* description for details telling you which operators can be used in which circumstances.

FilterValue (MQLONG)
Filter value identifier.

This specifies the filter-value that must be satisfied.

Depending on the parameter, the value and the permitted operators can be:
- An explicit integer value, if the parameter takes a single integer value.

  You can only use the following operators:
  - MQCFOP_GREATER
  - MQCFOP_LESS
  - MQCFOP_EQUAL
  - MQCFOP_NOT_EQUAL
  - MQCFOP_NOT_GREATER
  - MQCFOP_NOT_LESS
- An MQ constant, if the parameter takes a single value from a possible set of values (for example, the value MQCHT_SENDER on the **ChannelType** parameter). You can only use MQCFOP_EQUAL or MQCFOP_NOT_EQUAL.
- An explicit value or an MQ constant, as the case might be, if the parameter takes a list of values. You can use either MQCFOP_CONTAINS or MQCFOP_EXCLUDES. For example, if the value 6 is specified with the operator MQCFOP_CONTAINS, all items where one of the parameter values is 6 are listed.

For example, if you need to filter on queues that are enabled for put operations in your Inquire Queue command, the parameter would be MQIA_INHIBIT_PUT and the filter-value would be MQQA_PUT_ALLOWED.

The filter value must be a valid value for the parameter being tested.

C language declaration

```
typedef struct tagMQCFIF {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Operator;     /* Operator identifier */
  MQLONG  FilterValue;  /* Filter value */
 } MQCFIF;
```

COBOL language declaration

```
**   MQCFIF structure
  10 MQCFIF.
**    Structure type
   15 MQCFIF-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIF-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIF-PARAMETER   PIC S9(9) BINARY.
**    Operator identifier
   15 MQCFIF-OPERATOR    PIC S9(9) BINARY.
**    Filter value
   15 MQCFIF-FILTERVALUE PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
 1 MQCFIF based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Operator    fixed bin(31)  /* Operator identifier */
  3 FilterValue fixed bin(31); /* Filter value */
```

System/390 assembler-language declaration (z/OS only)

```
MQCFIF                      DSECT
MQCFIF_TYPE                 DS   F        Structure type
MQCFIF_STRUCLENGTH          DS   F        Structure length
MQCFIF_PARAMETER            DS   F        Parameter identifier
MQCFIF_OPERATOR             DS   F        Operator identifier
MQCFIF_FILTERVALUE          DS   F        Filter value
MQCFIF_LENGTH               EQU  *-MQCFIF Length of structure
                            ORG  MQCFIF
MQCFIF_AREA                 DS   CL(MQCFIF_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFIF
  Type As Long         ' Structure type
  StrucLength As Long  ' Structure length
  Parameter As Long    ' Parameter identifier
  Operator As Long     ' Operator identifier
  FilterValue As Long  ' Filter value
End Type

Global MQCFIF_DEFAULT As MQCFIF
```

RPG language declaration (IBM i only)

```
D* MQCFIF Structure
D*
D* Structure type
D  FIFTYP                 1      4I 0 INZ(3)
D* Structure length
```

```
D  FIFLEN                  5      8I 0 INZ(16)
D* Parameter identifier
D  FIFPRM                  9     12I 0 INZ(0)
D* Operator identifier
D  FIFOP                  13     16I 0 INZ(0)
D* Condition identifier
D  FIFFV                  17     20I 0 INZ(0)
D*
```

MQCFIL - PCF integer list parameter:

The MQCFIL structure describes an integer-list parameter in a message that is a command or a response to a command. In either case, the format name in the message descriptor is MQFMT_ADMIN.

The MQCFIL structure can also be used for user-defined message data. In this case the message descriptor *Format* field is MQFMT_PCF (see Message descriptor for a PCF command). Also in this case, not all the fields in the structure are meaningful. The supplied initial values can be used for most fields, but the application must set the *StrucLength*, *Count*, and *Values* fields to the values appropriate to the data.

The structure ends with a variable-length array of integers; see the *Values* field in the following section for further details.

Fields for MQCFIL

Type (MQLONG)
Structure type.

This indicates that the structure is an MQCFIL structure describing an integer-list parameter. The value must be:

MQCFT_INTEGER_LIST
Structure defining an integer list.

StrucLength (MQLONG)
Structure length.

This is the length in bytes of the MQCFIL structure, including the array of integers at the end of the structure (the *Values* field). The length must be a multiple of four, and must be sufficient to contain the array; any bytes between the end of the array and the length defined by the *StrucLength* field are not significant.

The following constant gives the length of the *fixed* part of the structure, that is the length excluding the *Values* field:

MQCFIL_STRUC_LENGTH_FIXED
Length of fixed part of command format integer-list parameter structure.

Parameter (MQLONG)
Parameter identifier.

This identifies the parameter with values that are contained in the structure. The values that can occur in this field depend on the value of the *Command* field in the MQCFH structure; see "MQCFH - PCF header" on page 1592 for details.

The parameter is from the following groups of parameters:

- MQIA_*
- MQIACF_*
- MQIAMO_*
- MQIACH_*

Count (MQLONG)
Count of parameter values.

This is the number of elements in the *Values* array; it must be zero or greater.

Values (MQLONG x *Count*)
Parameter values.

This is an array of values for the parameter identified by the *Parameter* field. For example, for MQIACF_Q_ATTRS, this field is a list of attribute selectors (MQCA_* and MQIA_* values).

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure must be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL, PL/I, RPG, and System/390 assembler programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, you must include MQCFIL in a larger structure, and declare additional fields following MQCFIL, to represent the *Values* field as required.

C language declaration

```
typedef struct tagMQCFIL {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Count;        /* Count of parameter values */
  MQLONG  Values[1];    /* Parameter values - first element */
 } MQCFIL;
```

COBOL language declaration

```
**   MQCFIL structure
  10 MQCFIL.
**    Structure type
   15 MQCFIL-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIL-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIL-PARAMETER   PIC S9(9) BINARY.
**    Count of parameter values
   15 MQCFIL-COUNT       PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
 1 MQCFIL based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Count       fixed bin(31); /* Count of parameter values */
```

System/390 assembler-language declaration (z/OS only)

```
MQCFIL                        DSECT
MQCFIL_TYPE                   DS   F        Structure type
MQCFIL_STRUCLENGTH            DS   F        Structure length
MQCFIL_PARAMETER              DS   F        Parameter identifier
MQCFIL_COUNT                  DS   F        Count of parameter values
MQCFIL_LENGTH                 EQU  *-MQCFIL Length of structure
                              ORG  MQCFIL
MQCFIL_AREA                   DS   CL(MQCFIL_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFIL
  Type As Long         ' Structure type
  StrucLength As Long  ' Structure length
  Parameter As Long    ' Parameter identifier
  Count As Long        ' Count of parameter values
End Type

Global MQCFIL_DEFAULT As MQCFIL
```

RPG language declaration (IBM i only)

```
D* MQCFIL Structure
D*
D* Structure type
D  ILTYP                  1      4I 0 INZ(5)
D* Structure length
D  ILLEN                  5      8I 0 INZ(16)
D* Parameter identifier
D  ILPRM                  9     12I 0 INZ(0)
D* Count of parameter values
D  ILCNT                 13     16I 0 INZ(0)
D*
```

MQCFIN - PCF integer parameter:

The MQCFIN structure describes an integer parameter in a message that is a command or a response to a command. In either case, the format name in the message descriptor is MQFMT_ADMIN.

The MQCFIN structure can also be used for user-defined message data. In this case the message descriptor *Format* field is MQFMT_PCF (see Message descriptor for a PCF command). Also in this case, not all the fields in the structure are meaningful. The supplied initial values can be used for most fields, but the application must set the *Value* field to the value appropriate to the data.

Fields for MQCFIN

Type (MQLONG)
Structure type.

This indicates that the structure is an MQCFIN structure describing an integer parameter. The value must be:

MQCFT_INTEGER
Structure defining an integer.

StrucLength (MQLONG)
Structure length.

This is the length in bytes of the MQCFIN structure. The value must be:

MQCFIN_STRUC_LENGTH
Length of command format integer-parameter structure.

Parameter (MQLONG)
Parameter identifier.

This identifies the parameter with a value that is contained in the structure. The values that can occur in this field depend on the value of the *Command* field in the MQCFH structure; see “MQCFH - PCF header” on page 1592 for details.

The parameter is from the following groups of parameters:

- MQIA_*
- MQIACF_*

- MQIAMO_*
- MQIACH_*

Value (MQLONG)
Parameter value.

This is the value of the parameter identified by the *Parameter* field.

C language declaration

```
typedef struct tagMQCFIN {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Value;        /* Parameter value */
 } MQCFIN;
```

COBOL language declaration

```
**   MQCFIN structure
  10 MQCFIN.
**    Structure type
   15 MQCFIN-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIN-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIN-PARAMETER   PIC S9(9) BINARY.
**    Parameter value
   15 MQCFIN-VALUE       PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
 1 MQCFIN based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Value       fixed bin(31); /* Parameter value */
```

System/390 assembler-language declaration (z/OS only)

```
MQCFIN                        DSECT
MQCFIN_TYPE                   DS   F        Structure type
MQCFIN_STRUCLENGTH            DS   F        Structure length
MQCFIN_PARAMETER              DS   F        Parameter identifier
MQCFIN_VALUE                  DS   F        Parameter value
MQCFIN_LENGTH                 EQU  *-MQCFIN Length of structure
                              ORG  MQCFIN
MQCFIN_AREA                   DS   CL(MQCFIN_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFIN
  Type As Long         ' Structure type
  StrucLength As Long  ' Structure length
  Parameter As Long    ' Parameter identifier
  Value As Long        ' Parameter value
End Type

Global MQCFIN_DEFAULT As MQCFIN
```

RPG language declaration (IBM i only)

```
D* MQCFIN Structure
D*
D* Structure type
D  INTYP                 1      4I 0 INZ(3)
D* Structure length
```

```
D  INLEN                  5      8I 0 INZ(16)
D* Parameter identifier
D  INPRM                  9     12I 0 INZ(0)
D* Parameter value
D  INVAL                 13     16I 0 INZ(0)
D*
```

MQCFSF - PCF string filter parameter:

The MQCFSF structure describes a string filter parameter. The format name in the message descriptor is MQFMT_ADMIN.

The MQCFSF structure is used in Inquire commands to provide a filter condition. This filter condition is used to filter the results of the Inquire command and return to the user only those objects that satisfy the filter condition.

The results of filtering character strings on EBCDIC-based systems might be different from those results achieved on ASCII-based systems. This difference is because comparison of character strings is based on the collating sequence of the internal built-in values representing the characters.

When an MQCFSF structure is present, the Version field in the MQCFH structure at the start of the PCF must be MQCFH_VERSION_3 or higher.

Fields for MQCFSF

Type (MQLONG)
Structure type.

This indicates that the structure is an MQCFSF structure describing a string filter parameter. The value must be:

MQCFT_STRING_FILTER
Structure defining a string filter.

StrucLength (MQLONG)
Structure length.

This is the length in bytes of the MQCFSF structure. The value must be:

MQCFSF_STRUC_LENGTH
MQCFSF_STRUC_LENGTH is the length, in bytes, of the MQCFSF structure, including the string at the end of the structure (the *FilterValue* field). The length must be a multiple of 4, and must be sufficient to contain the string. Bytes between the end of the string and the length defined by the *StrucLength* field are not significant.

The following constant gives the length of the *fixed* part of the structure, that is the length excluding the *FilterValue* field:

MQCFSF_STRUC_LENGTH_FIXED
Length of fixed part of command format filter string-parameter structure.

Parameter (MQLONG)
Parameter identifier.

This identifies the parameter that is to be filtered on. The value of this identifier depends on the parameter to be filtered on. Any of the parameters which can be used in the Inquire command can be used in this field.

The parameter is from the following groups of parameters:

- MQCA_*
- MQCACF_*
- MQCAMO_*

- MQCACH_*

Operator (MQLONG)
Operator identifier.

This identifies the operator that is being used to evaluate whether the parameter satisfies the filter-value.

Possible values are:

MQCFOP_GREATER
Greater than

MQCFOP_LESS
Less than

MQCFOP_EQUAL
Equal to

MQCFOP_NOT_EQUAL
Not equal to

MQCFOP_NOT_LESS
Greater than or equal to

MQCFOP_NOT_GREATER
Less than or equal to

MQCFOP_LIKE
Matches a generic string

MQCFOP_NOT_LIKE
Does not match a generic string

MQCFOP_CONTAINS
Contains a specified string. Use MQCFOP_CONTAINS when filtering on lists of strings.

MQCFOP_EXCLUDES
Does not contain a specified string. Use MQCFOP_EXCLUDES when filtering on lists of strings.

MQCFOP_CONTAINS_GEN
Contains an item which matches a generic string. Use MQCFOP_CONTAINS_GEN when filtering on lists of strings.

MQCFOP_EXCLUDES_GEN
Does not contain any item which matches a generic string. Use MQCFOP_EXCLUDES_GEN when filtering on lists of strings.

See the *FilterValue* description for details telling you which operators can be used in which circumstances.

CodedCharSetId (MQLONG)
Coded character set identifier.

This specifies the coded character set identifier of the data in the *FilterValue* field. The following special value can be used:

MQCCSI_DEFAULT
Default character set identifier.

The string data is in the character set defined by the *CodedCharSetId* field in the MQ header structure that *precedes* the MQCFH structure, or by the *CodedCharSetId* field in the MQMD if the MQCFH structure is at the start of the message.

FilterValueLength (MQLONG)
Length of filter-value string.

This is the length, in bytes, of the data in the *FilterValue* field. This parameter must be zero or greater, and does not need to be a multiple of 4.

Note: z/OS On z/OS there is a 256 character limit for the filter-value of the MQSC **WHERE** clause. This limit is not in place for other platforms.

FilterValue (MQCHAR x *FilterValueLength*)
Filter value.

This specifies the filter-value that must be satisfied. Depending on the parameter, the value and the permitted operators can be:

- An explicit string value.

  You can only use the following operators:
  - MQCFOP_GREATER
  - MQCFOP_LESS
  - MQCFOP_EQUAL
  - MQCFOP_NOT_EQUAL
  - MQCFOP_NOT_GREATER
  - MQCFOP_NOT_LESS
- A generic string value. This field is a character string with an asterisk at the end, for example ABC*. The operator must be either MQCFOP_LIKE or MQCFOP_NOT_LIKE. The characters must be valid for the attribute you are testing. If the operator is MQCFOP_LIKE, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is MQCFOP_NOT_LIKE, all items where the attribute value does not begin with the string are listed.
- If the parameter takes a list of string values, the operator can be:
  - MQCFOP_CONTAINS
  - MQCFOP_EXCLUDES
  - MQCFOP_CONTAINS_GEN
  - MQCFOP_EXCLUDES_GEN

  An item in a list of values. The value can be explicit or generic. If it is explicit, use MQCFOP_CONTAINS or MQCFOP_EXCLUDES as the operator. For example, if the value DEF is specified with the operator MQCFOP_CONTAINS, all items where one of the attribute values is DEF are listed. If it is generic, use MQCFOP_CONTAINS_GEN or MQCFOP_EXCLUDES_GEN as the operator. If ABC* is specified with the operator MQCFOP_CONTAINS_GEN, all items where one of the attribute values begins with ABC are listed.

Note:

1. If the specified string is shorter than the standard length of the parameter in MQFMT_ADMIN command messages, the omitted characters are assumed to be blanks. If the specified string is longer than the standard length, it is an error.
2. When the queue manager reads an MQCFSF structure in an MQFMT_ADMIN message from the command input queue, the queue manager processes the string as though it had been specified on an MQI call. This processing means that within the string, the first null and the characters following it (up to the end of the string) are treated as blanks.
3. On z/OS there is a 256 character limit for the filter-value of the MQSC **WHERE** clause. This limit is not in place for other platforms.

The filter value must be a valid value for the parameter being tested.

C language declaration

```
typedef struct tagMQCFSF {
  MQLONG  Type;             /* Structure type */
  MQLONG  StrucLength;      /* Structure length */
  MQLONG  Parameter;        /* Parameter identifier */
```

```
  MQLONG  Operator;         /* Operator identifier */
  MQLONG  CodedCharSetId;   /* Coded character set identifier */
  MQLONG  FilterValueLength /* Filtervalue length */
  MQCHAR[1]  FilterValue;   /* Filter value */
 } MQCFSF;
```

COBOL language declaration

```
**    MQCFSF structure
   10 MQCFSF.
**    Structure type
   15 MQCFSF-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFSF-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFSF-PARAMETER   PIC S9(9) BINARY.
**    Operator identifier
   15 MQCFSF-OPERATOR PIC S9(9) BINARY.
**    Coded character set identifier
   15 MQCFSF-CODEDCHARSETID PIC S9(9) BINARY.
**    Filter value length
   15 MQCFSF-FILTERVALUE PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
 1 MQCFSF based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Operator    fixed bin(31)  /* Operator identifier */
  3 CodedCharSetId    fixed bin(31)  /* Coded character set identifier */
  3 FilterValueLength fixed bin(31); /* Filter value length */
```

System/390 assembler-language declaration (z/OS only)

```
MQCFSF                      DSECT
MQCFSF_TYPE                 DS   F        Structure type
MQCFSF_STRUCLENGTH          DS   F        Structure length
MQCFSF_PARAMETER            DS   F        Parameter identifier
MQCFSF_OPERATOR             DS   F        Operator identifier
MQCFSF_CODEDCHARSETID       DS   F        Coded character set identifier
MQCFSF_FILTERVALUELENGTH    DS   F        Filter value length
MQCFSF_LENGTH               EQU  *-MQCFSF Length of structure
                            ORG  MQCFSF
MQCFSF_AREA                 DS   CL(MQCFSF_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFSF
  Type As Long         ' Structure type
  StrucLength As Long  ' Structure length
  Parameter As Long    ' Parameter identifier
  Operator As Long     ' Operator identifier
  CodedCharSetId As Long     ' Coded character set identifier
  FilterValueLength As Long  ' Operator identifier
  FilterValue As String*1  ' Condition value -- first character
End Type

Global MQCFSF_DEFAULT As MQCFSF
```

RPG language declaration (IBM i only)

```
D* MQCFSF Structure
D*
D* Structure type
D  FISTYP                 1      4I 0 INZ(3)
D* Structure length
```

```
D  FSFLEN                  5      8I 0 INZ(16)
D* Parameter identifier
D  FSFPRM                  9     12I 0 INZ(0)
D* Reserved field
D  FSFRSV                 13     16I 0 INZ(0)
D* Parameter value
D  FSFVAL                 17     16
D* Structure type
D  FSFTYP                 17     20I 0
D* Structure length
D  FSFLEN                 21     24I 0
D* Parameter value
D  FSFPRM                 25     28I 0
D* Operator identifier
D  FSFOP                  29     32I 0
D* Coded character set identifier
D  FSFCSI                 33     36I 0
D* Length of condition
D  FSFFVL                 37     40 0
D* Condition value -- first character
D  FSFFV                  41     41
D*
```

MQCFSL - PCF string list parameter:

The MQCFSL structure describes a string-list parameter in a message which is a command or a response to a command. In either case, the format name in the message descriptor is MQFMT_ADMIN.

The MQCFSL structure can also be used for user-defined message data. In this case the message descriptor *Format* field is MQFMT_PCF (see Message descriptor for a PCF command). Also in this case, not all the fields in the structure are meaningful. The supplied initial values can be used for most fields, but the application must set the *StrucLength*, *Count*, *StringLength*, and *Strings* fields to the values appropriate to the data.

The structure ends with a variable-length array of character strings; see the *Strings* field section for further details.

See "Usage notes" on page 1592 for further information about how to use the structure.

Fields for MQCFSL

Type (MQLONG)
Structure type.

This indicates that the structure is an MQCFSL structure describing a string-list parameter. The value must be:

MQCFT_STRING_LIST
Structure defining a string list.

StrucLength (MQLONG)
Structure length.

This is the length in bytes of the MQCFSL structure, including the data at the end of the structure (the *Strings* field). The length must be a multiple of four, and must be sufficient to contain all the strings; any bytes between the end of the strings and the length defined by the *StrucLength* field are not significant.

The following constant gives the length of the *fixed* part of the structure, that is the length excluding the *Strings* field:

MQCFSL_STRUC_LENGTH_FIXED
Length of fixed part of command format string-list parameter structure.

Parameter (MQLONG)
Parameter identifier.

This identifies the parameter with values that are contained in the structure. The values that can occur in this field depend on the value of the *Command* field in the MQCFH structure; see "MQCFH - PCF header" on page 1592 for details.

The parameter is from the following groups of parameters:

- MQCA_*
- MQCACF_*
- MQCAMO_*
- MQCACH_*

CodedCharSetId (MQLONG)
Coded character set identifier.

This specifies the coded character set identifier of the data in the *Strings* field. The following special value can be used:

MQCCSI_DEFAULT
Default character set identifier.

The string data is in the character set defined by the *CodedCharSetId* field in the MQ header structure that *precedes* the MQCFH structure, or by the *CodedCharSetId* field in the MQMD if the MQCFH structure is at the start of the message.

Count (MQLONG)
Count of parameter values.

This is the number of strings present in the *Strings* field; it must be zero or greater.

StringLength (MQLONG)
Length of one string.

This is the length in bytes of one parameter value, that is the length of one string in the *Strings* field; all the strings are this length. The length must be zero or greater, and need not be a multiple of four.

Strings (MQCHAR x *StringLength* x *Count*)
String values.

This is a set of string values for the parameter identified by the *Parameter* field. The number of strings is given by the *Count* field, and the length of each string is given by the *StringLength* field. The strings are concatenated together, with no bytes skipped between adjacent strings. The total length of the strings is the length of one string multiplied by the number of strings present (that is, *StringLength* x *Count*).

- In MQFMT_ADMIN command messages, if the specified string is shorter than the standard length of the parameter, the omitted characters are assumed to be blanks. If the specified string is longer than the standard length, it is an error.
- In MQFMT_ADMIN response messages, string parameters might be returned padded with blanks to the standard length of the parameter.
- In MQFMT_EVENT messages, trailing blanks might be omitted from string parameters (that is, the string might be shorter than the standard length of the parameter).

In all cases, *StringLength* gives the length of the string present in the message.

The strings can contain any characters that are in the character set defined by *CodedCharSetId*, and that are valid for the parameter identified by *Parameter*.

Note: When the queue manager reads an MQCFSL structure in an MQFMT_ADMIN message from the command input queue, the queue manager processes each string in the list as though it had been

specified on an MQI call. This processing means that within each string, the first null, and the characters following it (up to the end of the string) are treated as blanks.

In responses and all other cases, a null character in a string is treated as normal data, and does not act as a delimiter for the string. This treatment means that when a receiving application reads a MQFMT_PCF, MQFMT_EVENT, or MQFMT_ADMIN message, the receiving application receives all the data specified by the sending application.

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure must be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL, PL/I, RPG, and System/390 assembler programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, you must include MQCFSL in a larger structure, and declare additional fields following MQCFSL, to represent the *Strings* field as required.

C language declaration

```
typedef struct tagMQCFSL {
  MQLONG  Type;            /* Structure type */
  MQLONG  StrucLength;     /* Structure length */
  MQLONG  Parameter;       /* Parameter identifier */
  MQLONG  CodedCharSetId;  /* Coded character set identifier */
  MQLONG  Count;           /* Count of parameter values */
  MQLONG  StringLength;    /* Length of one string */
  MQCHAR  Strings[1];      /* String values - first
                              character */
 } MQCFSL;
```

COBOL language declaration

```
**   MQCFSL structure
  10 MQCFSL.
**    Structure type
   15 MQCFSL-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFSL-STRUCLENGTH    PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFSL-PARAMETER      PIC S9(9) BINARY.
**    Coded character set identifier
   15 MQCFSL-CODEDCHARSETID PIC S9(9) BINARY.
**    Count of parameter values
   15 MQCFSL-COUNT          PIC S9(9) BINARY.
**    Length of one string
   15 MQCFSL-STRINGLENGTH   PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
 1 MQCFSL based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Parameter      fixed bin(31), /* Parameter identifier */
  3 CodedCharSetId fixed bin(31), /* Coded character set identifier */
  3 Count          fixed bin(31), /* Count of parameter values */
  3 StringLength   fixed bin(31); /* Length of one string */
```

System/390 assembler-language declaration (z/OS only)

```
MQCFSL                          DSECT
MQCFSL_TYPE                     DS   F        Structure type
MQCFSL_STRUCLENGTH              DS   F        Structure length
MQCFSL_PARAMETER                DS   F        Parameter identifier
MQCFSL_CODEDCHARSETID           DS   F        Coded character set
*                                             identifier
```

```
MQCFSL_COUNT                    DS   F        Count of parameter values
MQCFSL_STRINGLENGTH             DS   F        Length of one string
MQCFSL_LENGTH                   EQU  *-MQCFSL Length of structure
                                ORG  MQCFSL
MQCFSL_AREA                     DS   CL(MQCFSL_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFSL
  Type As Long             ' Structure type
  StrucLength As Long      ' Structure length
  Parameter As Long        ' Parameter identifier
  CodedCharSetId As Long   ' Coded character set identifier
  Count As Long            ' Count of parameter values
  StringLength As Long     ' Length of one string
End Type

Global MQCFSL_DEFAULT As MQCFSL
```

RPG language declaration (IBM i only)

```
D* MQCFSL Structure
D*
D* Structure type
D  SLTYP                  1      4I 0 INZ(6)
D* Structure length
D  SLLEN                  5      8I 0 INZ(24
D* Parameter identifier
D  SLPRM                  9     12I 0 INZ(0)
D* Coded character set identifier
D  SLCSI                 13     16I 0 INZ(0)
D* Count of parameter values
D  SLCNT                 17     20I 0 INZ(0)
D* Length of one string
D  SLSTL                 21     24I 0 INZ(0)
```

MQCFST - PCF string parameter:

The MQCFST structure describes a string parameter in a message that is a command or a response to a command. In either case, the format name in the message descriptor is MQFMT_ADMIN.

The MQCFST structure can also be used for user-defined message data. In this case the message descriptor *Format* field is MQFMT_PCF (see Message descriptor for a PCF command). Also in this case, not all the fields in the structure are meaningful. The supplied initial values can be used for most fields, but the application must set the *StrucLength*, *StringLength*, and *String* fields to the values appropriate to the data.

The structure ends with a variable-length character string; see the *String* field section for further details.

See “Usage notes” on page 1592 for further information about how to use the structure.

Fields for MQCFST

Type (MQLONG)
Structure type.

This indicates that the structure is an MQCFST structure describing a string parameter. The value must be:

MQCFT_STRING
Structure defining a string.

StrucLength (MQLONG)
Structure length.

This is the length in bytes of the MQCFST structure, including the string at the end of the structure (the *String* field). The length must be a multiple of four, and must be sufficient to contain the string; any bytes between the end of the string and the length defined by the *StrucLength* field are not significant.

The following constant gives the length of the *fixed* part of the structure, that is the length excluding the *String* field:

MQCFST_STRUC_LENGTH_FIXED
Length of fixed part of command format string-parameter structure.

Parameter (MQLONG)
Parameter identifier.

This identifies the parameter with a value that is contained in the structure. The values that can occur in this field depend on the value of the *Command* field in the MQCFH structure; see "MQCFH - PCF header" on page 1592 for details.

The parameter is from the following groups of parameters:

- MQCA_*
- MQCACF_*
- MQCAMO_*
- MQCACH_*

CodedCharSetId (MQLONG)
Coded character set identifier.

This specifies the coded character set identifier of the data in the *String* field. The following special value can be used:

MQCCSI_DEFAULT
Default character set identifier.

The string data is in the character set defined by the *CodedCharSetId* field in the MQ header structure that *precedes* the MQCFH structure, or by the *CodedCharSetId* field in the MQMD if the MQCFH structure is at the start of the message.

StringLength (MQLONG)
Length of string.

This is the length in bytes of the data in the *String* field; it must be zero or greater. This length does not need to be a multiple of four.

String (MQCHAR x *StringLength*)
String value.

This is the value of the parameter identified by the *Parameter* field:

- In MQFMT_ADMIN command messages, if the specified string is shorter than the standard length of the parameter, the omitted characters are assumed to be blanks. If the specified string is longer than the standard length, it is an error.
- In MQFMT_ADMIN response messages, string parameters might be returned padded with blanks to the standard length of the parameter.
- In MQFMT_EVENT messages, trailing blanks might be omitted from string parameters (that is, the string can be shorter than the standard length of the parameter).

The value of *StringLength* depends on whether, when the specified string is shorter than the standard length, padding blanks have been added to the string. If so, the value of *StringLength* is the sum of the actual length of the string plus the padded blanks.

The string can contain any characters that are in the character set defined by *CodedCharSetId*, and that are valid for the parameter identified by *Parameter*.

Note: When the queue manager reads an MQCFST structure in an MQFMT_ADMIN message from the command input queue, the queue manager processes the string as though it had been specified on an MQI call. This processing means that within the string, the first null and the characters following it (up to the end of the string) are treated as blanks.

In responses and all other cases, a null character in the string is treated as normal data, and does not act as a delimiter for the string. This treatment means that when a receiving application reads a MQFMT_PCF, MQFMT_EVENT, or MQFMT_ADMIN message, the receiving application receives all the data specified by the sending application.

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure must be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL, PL/I, and System/390 assembler programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, the user must include MQCFST in a larger structure, and declare an additional field or additional fields following MQCFST, to represent the *String* field as required.

C language declaration

```
typedef struct tagMQCFST {
  MQLONG  Type;            /* Structure type */
  MQLONG  StrucLength;     /* Structure length */
  MQLONG  Parameter;       /* Parameter identifier */
  MQLONG  CodedCharSetId;  /* Coded character set identifier */
  MQLONG  StringLength;    /* Length of string */
  MQCHAR  String[1];       /* String value - first
                              character */
 } MQCFST;
```

COBOL language declaration

```
**    MQCFST structure
   10 MQCFST.
**    Structure type
   15 MQCFST-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFST-STRUCLENGTH    PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFST-PARAMETER      PIC S9(9) BINARY.
**    Coded character set identifier
   15 MQCFST-CODEDCHARSETID PIC S9(9) BINARY.
**    Length of string
   15 MQCFST-STRINGLENGTH   PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
 1 MQCFST based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Parameter      fixed bin(31), /* Parameter identifier */
  3 CodedCharSetId fixed bin(31), /* Coded character set identifier */
  3 StringLength   fixed bin(31); /* Length of string */
```

System/390 assembler-language declaration (z/OS only)

```
MQCFST                        DSECT
MQCFST_TYPE                   DS   F        Structure type
MQCFST_STRUCLENGTH            DS   F        Structure length
MQCFST_PARAMETER              DS   F        Parameter identifier
MQCFST_CODEDCHARSETID         DS   F        Coded character set
*                                           identifier
```

```
MQCFST_STRINGLENGTH           DS   F        Length of string
MQCFST_LENGTH                 EQU  *-MQCFST Length of structure
                              ORG  MQCFST
MQCFST_AREA                   DS   CL(MQCFST_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFST
  Type As Long            ' Structure type
  StrucLength As Long     ' Structure length
  Parameter As Long       ' Parameter identifier
  CodedCharSetId As Long  ' Coded character set identifier
  StringLength As Long    ' Length of string
End Type

Global MQCFST_DEFAULT As MQCFST
```

RPG language declaration (IBM i only)

```
D* MQCFST Structure
D*
D* Structure type
D  STTYP                  1      4I 0 INZ(4)
D* Structure length
D  STLEN                  5      8I 0 INZ(20)
D* Parameter identifier
D  STPRM                  9     12I 0 INZ(0)
D* Coded character set identifier
D  STCSI                 13     16I 0 INZ(0)
D* Length of string
D  STSTL                 17     20I 0 INZ(0)
D*
```

PCF example

The compiled program, written in C language, in the example uses IBM MQ for Windows. It inquires of the default queue manager about a subset of the attributes for all local queues defined to it. It then produces an output file, SAVEQMGR.TST, in the directory from which it was run for use with RUNMQSC.

Inquire local queue attributes

This following section provides an example of how Programmable Command Formats can be used in a program for administration of IBM MQ queues.

The program is given as an example of using PCFs and has been limited to a simple case. This program is of most use as an example if you are considering the use of PCFs to manage your IBM MQ environment.

Program listing

```
/*============================================================================*/
/*                                                                            */
/* This is a program to inquire of the default queue manager about the        */
/* local queues defined to it.                                                */
/*                                                                            */
/* The program takes this information and appends it to a file                */
/* SAVEQMGR.TST which is of a format suitable for RUNMQSC. It could,          */
/* therefore, be used to re-create or clone a queue manager.                  */
/*                                                                            */
/* It is offered as an example of using Programmable Command Formats (PCFs)   */
/* as a method for administering a queue manager.                             */
/*                                                                            */
/*============================================================================*/

/* Include standard libraries */
```

```
#include <memory.h>
#include <stdio.h>

/* Include MQSeries headers */
#include <cmqc.h>
#include <cmqcfc.h>
#include <cmqxc.h>

typedef struct LocalQParms {
   MQCHAR48    QName;
   MQLONG      QType;
   MQCHAR64    QDesc;
   MQLONG      InhibitPut;
   MQLONG      DefPriority;
   MQLONG      DefPersistence;
   MQLONG      InhibitGet;
   MQCHAR48    ProcessName;
   MQLONG      MaxQDepth;
   MQLONG      MaxMsgLength;
   MQLONG      BackoutThreshold;
   MQCHAR48    BackoutReqQName;
   MQLONG      Shareability;
   MQLONG      DefInputOpenOption;
   MQLONG      HardenGetBackout;
   MQLONG      MsgDeliverySequence;
   MQLONG      RetentionInterval;
   MQLONG      DefinitionType;
   MQLONG      Usage;
   MQLONG      OpenInputCount;
   MQLONG      OpenOutputCount;
   MQLONG      CurrentQDepth;
   MQCHAR12    CreationDate;
   MQCHAR8     CreationTime;
   MQCHAR48    InitiationQName;
   MQLONG      TriggerControl;
   MQLONG      TriggerType;
   MQLONG      TriggerMsgPriority;
   MQLONG      TriggerDepth;
   MQCHAR64    TriggerData;
   MQLONG      Scope;
   MQLONG      QDepthHighLimit;
   MQLONG      QDepthLowLimit;
   MQLONG      QDepthMaxEvent;
   MQLONG      QDepthHighEvent;
   MQLONG      QDepthLowEvent;
   MQLONG      QServiceInterval;
   MQLONG      QServiceIntervalEvent;
} LocalQParms;

MQOD  ObjDesc = { MQOD_DEFAULT };
MQMD  md      = { MQMD_DEFAULT };
MQPMO pmo     = { MQPMO_DEFAULT };
MQGMO gmo     = { MQGMO_DEFAULT };

void ProcessStringParm( MQCFST *pPCFString, LocalQParms *DefnLQ );

void ProcessIntegerParm( MQCFIN *pPCFInteger, LocalQParms *DefnLQ );

void AddToFileQLOCAL( LocalQParms DefnLQ );

void MQParmCpy( char *target, char *source, int length );

void PutMsg( MQHCONN   hConn       /* Connection to queue manager           */
           , MQCHAR8   MsgFormat   /* Format of user data to be put in msg  */
           , MQHOBJ    hQName      /* handle of queue to put the message to */
           , MQCHAR48  QName       /* name of queue to put the message to   */
           , MQBYTE   *UserMsg     /* The user data to be put in the message */
```

```
          , MQLONG    UserMsgLen  /*                                      */
          );

void GetMsg( MQHCONN   hConn          /* handle of queue manager           */
           , MQLONG    MQParm         /* Options to specify nature of get  */
           , MQHOBJ    hQName         /* handle of queue to read from      */
           , MQBYTE   *UserMsg        /* Input/Output buffer containing msg */
           , MQLONG    ReadBufferLen  /* Length of supplied buffer         */
           );
MQHOBJ OpenQ( MQHCONN     hConn
            , MQCHAR48    QName
            , MQLONG      OpenOpts
            );

int main( int argc, char *argv[] )
{
  MQCHAR48             QMgrName;         /* Name of connected queue mgr     */
  MQHCONN              hConn;            /* handle to connected queue mgr   */
  MQOD                 ObjDesc;          /*                                 */
  MQLONG               OpenOpts;         /*                                 */
  MQLONG               CompCode;         /* MQ API completion code          */
  MQLONG               Reason;           /* Reason qualifying CompCode      */
                                         /*                                 */
  MQHOBJ               hAdminQ;          /* handle to output queue          */
  MQHOBJ               hReplyQ;          /* handle to input queue           */
                                         /*                                 */
  MQLONG               AdminMsgLen;      /* Length of user message buffer   */
  MQBYTE              *pAdminMsg;        /* Ptr to outbound data buffer     */
  MQCFH               *pPCFHeader;       /* Ptr to PCF header structure     */
  MQCFST              *pPCFString;       /* Ptr to PCF string parm block    */
  MQCFIN              *pPCFInteger;      /* Ptr to PCF integer parm block   */
  MQLONG              *pPCFType;         /* Type field of PCF message parm  */
  LocalQParms          DefnLQ;           /*                                 */
                                         /*                                 */
  char                 ErrorReport[40];  /*                        */
  MQCHAR8              MsgFormat;        /* Format of inbound message       */
  short                Index;            /* Loop counter                    */

  /* Connect to default queue manager */
  QMgrName[0] =  '\0';                   /* set to null   default QM */
  if ( argc > 1 )
    strcpy(QMgrName, argv[1]);

  MQCONN(  QMgrName                      /*  use default queue manager  */
        , &hConn                         /*  queue manager handle       */
        , &CompCode                      /*  Completion code            */
        , &Reason                        /*  Reason qualifying CompCode */
        );

  if ( CompCode != MQCC_OK ) {
     printf( "MQCONN failed for %s, CC=%d RC=%d\n"
           , QMgrName
           , CompCode
           , Reason
           );
     exit( -1 );
  } /* endif */

  /* Open all the required queues */
  hAdminQ = OpenQ( hConn, "SYSTEM.ADMIN.COMMAND.QUEUE\0", MQOO_OUTPUT );

  hReplyQ = OpenQ( hConn, "SAVEQMGR.REPLY.QUEUE\0", MQOO_INPUT_EXCLUSIVE );

  /* ************************************************************* */
  /* Put a message to the SYSTEM.ADMIN.COMMAND.QUEUE to inquire all     */
  /* the local queues defined on the queue manager.                     */
  /*                                                                    */
```

```
/* The request consists of a Request Header and a parameter block     */
/* used to specify the generic search. The header and the parameter   */
/* block follow each other in a contiguous buffer which is pointed    */
/* to by the variable pAdminMsg. This entire buffer is then put to    */
/* the queue.                                                         */
/*                                                                    */
/* The command server, (use STRMQCSV to start it), processes the      */
/* SYSTEM.ADMIN.COMMAND.QUEUE and puts a reply on the application     */
/* ReplyToQ for each defined queue.                                   */
/* ****************************************************************** */

/* Set the length for the message buffer */
AdminMsgLen = MQCFH_STRUC_LENGTH
            + MQCFST_STRUC_LENGTH_FIXED + MQ_Q_NAME_LENGTH
            + MQCFIN_STRUC_LENGTH
            ;

/* ------------------------------------------------------------------ */
/* Set pointers to message data buffers                               */
/*                                                                    */
/* pAdminMsg points to the start of the message buffer                */
/*                                                                    */
/* pPCFHeader also points to the start of the message buffer. It is   */
/* used to indicate the type of command we wish to execute and the    */
/* number of parameter blocks following in the message buffer.        */
/*                                                                    */
/* pPCFString points into the message buffer immediately after the    */
/* header and is used to map the following bytes onto a PCF string    */
/* parameter block. In this case the string is used to indicate the   */
/* nameof the queue we want details about, * indicating all queues.   */
/*                                                                    */
/* pPCFInteger points into the message buffer immediately after the   */
/* string block described above. It is used to map the following      */
/* bytes onto a PCF integer parameter block. This block indicates     */
/* the type of queue we wish to receive details about, thereby        */
/* qualifying the generic search set up by passing the previous       */
/* string parameter.                                                  */
/*                                                                    */
/* Note that this example is a generic search for all attributes of   */
/* all local queues known to the queue manager. By using different,   */
/* or more, parameter blocks in the request header it is possible     */
/* to narrow the search.                                              */
/* ------------------------------------------------------------------ */

pAdminMsg   = (MQBYTE *)malloc( AdminMsgLen );

pPCFHeader  = (MQCFH *)pAdminMsg;

pPCFString  = (MQCFST *)(pAdminMsg
                         + MQCFH_STRUC_LENGTH
                         );

pPCFInteger = (MQCFIN *)( pAdminMsg
                         + MQCFH_STRUC_LENGTH
                         + MQCFST_STRUC_LENGTH_FIXED + MQ_Q_NAME_LENGTH
                         );


/* Set up request header */
pPCFHeader->Type           = MQCFT_COMMAND;
pPCFHeader->StrucLength    = MQCFH_STRUC_LENGTH;
pPCFHeader->Version        = MQCFH_VERSION_1;
pPCFHeader->Command        = MQCMD_INQUIRE_Q;
pPCFHeader->MsgSeqNumber   = MQCFC_LAST;
pPCFHeader->Control        = MQCFC_LAST;
pPCFHeader->ParameterCount = 2;
```

```
/* Set up parameter block */
pPCFString->Type           = MQCFT_STRING;
pPCFString->StrucLength    = MQCFST_STRUC_LENGTH_FIXED + MQ_Q_NAME_LENGTH;
pPCFString->Parameter      = MQCA_Q_NAME;
pPCFString->CodedCharSetId = MQCCSI_DEFAULT;
pPCFString->StringLength   = MQ_Q_NAME_LENGTH;
memset( pPCFString->String, ' ', MQ_Q_NAME_LENGTH );
memcpy( pPCFString->String, "*", 1 );

/* Set up parameter block */
pPCFInteger->Type        = MQCFT_INTEGER;
pPCFInteger->StrucLength = MQCFIN_STRUC_LENGTH;
pPCFInteger->Parameter   = MQIA_Q_TYPE;
pPCFInteger->Value       = MQQT_LOCAL;

PutMsg(  hConn                    /* Queue manager handle             */
      ,  MQFMT_ADMIN              /* Format of message                */
      ,  hAdminQ                  /* Handle of command queue          */
      ,  "SAVEQMGR.REPLY.QUEUE\0" /* reply to queue                   */
      ,  (MQBYTE *)pAdminMsg      /* Data part of message to put      */
      ,  AdminMsgLen
      );

free( pAdminMsg );

/* ****************************************************************** */
/* Get and process the replies received from the command server onto  */
/* the applications ReplyToQ.                                         */
/*                                                                    */
/* There will be one message per defined local queue.                 */
/*                                                                    */
/* The last message will have the Control field of the PCF header     */
/* set to MQCFC_LAST. All others will be MQCFC_NOT_LAST.              */
/*                                                                    */
/* An individual Reply message consists of a header followed by a     */
/* number a parameters, the exact number, type and order will depend  */
/* upon the type of request.                                          */
/*                                                                    */
/* ------------------------------------------------------------------ */
/*                                                                    */
/* The message is retrieved into a buffer pointed to by pAdminMsg.    */
/* This buffer has been allocated enough memory to hold every         */
/* parameter needed for a local queue definition.                     */
/*                                                                    */
/* pPCFHeader is then allocated to point also to the beginning of     */
/* the buffer and is used to access the PCF header structure. The     */
/* header contains several fields. The one we are specifically        */
/* interested in is the ParameterCount. This tells us how many        */
/* parameters follow the header in the message buffer. There is       */
/* one parameter for each local queue attribute known by the          */
/* queue manager.                                                     */
/*                                                                    */
/* At this point we do not know the order or type of each parameter   */
/* block in the buffer, the first MQLONG of each block defines its    */
/* type; they may be parameter blocks containing either strings or    */
/* integers.                                                          */
/*                                                                    */
/* pPCFType is used initially to point to the first byte beyond the   */
/* known parameter block. Initially then, it points to the first byte */
/* after the PCF header. Subsequently it is incremented by the length */
/* of the identified parameter block and therefore points at the      */
/* next. Looking at the value of the data pointed to by pPCFType we   */
/* can decide how to process the next group of bytes, either as a     */
/* string, or an integer.                                             */
/*                                                                    */
/* In this way we parse the message buffer extracting the values of   */
/* each of the parameters we are interested in.                       */
```

```
/*                                                                  */
/* ***************************************************************** */

/* AdminMsgLen is to be set to the length of the expected reply      */
/* message. This structure is specific to Local Queues.             */
AdminMsgLen =   MQCFH_STRUC_LENGTH
            + ( MQCFST_STRUC_LENGTH_FIXED * 7 )
            + ( MQCFIN_STRUC_LENGTH       * 39 )
            + ( MQ_Q_NAME_LENGTH          * 6  )
            + ( MQ_Q_MGR_NAME_LENGTH      * 2  )
            +   MQ_Q_DESC_LENGTH
            +   MQ_PROCESS_NAME_LENGTH
            +   MQ_CREATION_DATE_LENGTH
            +   MQ_CREATION_TIME_LENGTH
            +   MQ_TRIGGER_DATA_LENGTH + 100
            ;

/* Set pointers to message data buffers */
pAdminMsg = (MQBYTE *)malloc( AdminMsgLen );

do {

   GetMsg(  hConn                      /* Queue manager handle           */
         ,  MQGMO_WAIT
         ,  hReplyQ                    /* Get queue handle               */
         ,  (MQBYTE *)pAdminMsg        /* pointer to message area        */
         ,  AdminMsgLen                /* length of get buffer           */
         );

   /* Examine Header */
   pPCFHeader = (MQCFH *)pAdminMsg;

   /* Examine first parameter */
   pPCFType = (MQLONG *)(pAdminMsg + MQCFH_STRUC_LENGTH);

   Index = 1;

   while ( Index <= pPCFHeader->ParameterCount ) {

      /* Establish the type of each parameter and allocate  */
      /* a pointer of the correct type to reference it.     */
      switch ( *pPCFType ) {
      case MQCFT_INTEGER:
         pPCFInteger = (MQCFIN *)pPCFType;
         ProcessIntegerParm( pPCFInteger, &DefnLQ );
         Index++;
         /* Increment the pointer to the next parameter by the */
         /* length of the current parm.                        */
         pPCFType = (MQLONG *)( (MQBYTE *)pPCFType
                              + pPCFInteger->StrucLength
                              );
         break;
      case MQCFT_STRING:
         pPCFString = (MQCFST *)pPCFType;
         ProcessStringParm( pPCFString, &DefnLQ );
         Index++;
         /* Increment the pointer to the next parameter by the */
         /* length of the current parm.                        */
         pPCFType = (MQLONG *)( (MQBYTE *)pPCFType
                              + pPCFString->StrucLength
                              );
         break;
      } /* endswitch */

   } /* endwhile */

   /* ***************************************************** */
```

```
      /* Message parsed, append to output file                     */
      /* ********************************************************* */
      AddToFileQLOCAL( DefnLQ );


      /* ********************************************************* */
      /* Finished processing the current message, do the next one. */
      /* ********************************************************* */

  } while ( pPCFHeader->Control == MQCFC_NOT_LAST ); /* enddo */

  free( pAdminMsg );

  /* ************************************* */
  /* Processing of the local queues complete */
  /* ************************************* */

}

void ProcessStringParm( MQCFST *pPCFString, LocalQParms *DefnLQ )
{
   switch ( pPCFString->Parameter ) {
   case MQCA_Q_NAME:
      MQParmCpy( DefnLQ->QName, pPCFString->String, 48 );
      break;
   case MQCA_Q_DESC:
      MQParmCpy( DefnLQ->QDesc, pPCFString->String, 64 );
      break;
   case MQCA_PROCESS_NAME:
      MQParmCpy( DefnLQ->ProcessName, pPCFString->String, 48 );
      break;
   case MQCA_BACKOUT_REQ_Q_NAME:
      MQParmCpy( DefnLQ->BackoutReqQName, pPCFString->String, 48 );
      break;
   case MQCA_CREATION_DATE:
      MQParmCpy( DefnLQ->CreationDate, pPCFString->String, 12 );
      break;
   case MQCA_CREATION_TIME:
      MQParmCpy( DefnLQ->CreationTime, pPCFString->String, 8 );
      break;
   case MQCA_INITIATION_Q_NAME:
      MQParmCpy( DefnLQ->InitiationQName, pPCFString->String, 48 );
      break;
   case MQCA_TRIGGER_DATA:
      MQParmCpy( DefnLQ->TriggerData, pPCFString->String, 64 );
      break;
   } /* endswitch */
}

void ProcessIntegerParm( MQCFIN *pPCFInteger, LocalQParms *DefnLQ )
{
   switch ( pPCFInteger->Parameter ) {
   case MQIA_Q_TYPE:
      DefnLQ->QType = pPCFInteger->Value;
      break;
   case MQIA_INHIBIT_PUT:
      DefnLQ->InhibitPut = pPCFInteger->Value;
      break;
   case MQIA_DEF_PRIORITY:
      DefnLQ->DefPriority = pPCFInteger->Value;
      break;
   case MQIA_DEF_PERSISTENCE:
      DefnLQ->DefPersistence = pPCFInteger->Value;
      break;
   case MQIA_INHIBIT_GET:
      DefnLQ->InhibitGet = pPCFInteger->Value;
      break;
```

```
case MQIA_SCOPE:
   DefnLQ->Scope = pPCFInteger->Value;
   break;
case MQIA_MAX_Q_DEPTH:
   DefnLQ->MaxQDepth = pPCFInteger->Value;
   break;
case MQIA_MAX_MSG_LENGTH:
   DefnLQ->MaxMsgLength = pPCFInteger->Value;
   break;
case MQIA_BACKOUT_THRESHOLD:
   DefnLQ->BackoutThreshold = pPCFInteger->Value;
   break;
case MQIA_SHAREABILITY:
   DefnLQ->Shareability = pPCFInteger->Value;
   break;
case MQIA_DEF_INPUT_OPEN_OPTION:
   DefnLQ->DefInputOpenOption = pPCFInteger->Value;
   break;
case MQIA_HARDEN_GET_BACKOUT:
   DefnLQ->HardenGetBackout = pPCFInteger->Value;
   break;
case MQIA_MSG_DELIVERY_SEQUENCE:
   DefnLQ->MsgDeliverySequence = pPCFInteger->Value;
   break;
case MQIA_RETENTION_INTERVAL:
   DefnLQ->RetentionInterval = pPCFInteger->Value;
   break;
case MQIA_DEFINITION_TYPE:
   DefnLQ->DefinitionType = pPCFInteger->Value;
   break;
case MQIA_USAGE:
   DefnLQ->Usage = pPCFInteger->Value;
   break;
case MQIA_OPEN_INPUT_COUNT:
   DefnLQ->OpenInputCount = pPCFInteger->Value;
   break;
case MQIA_OPEN_OUTPUT_COUNT:
   DefnLQ->OpenOutputCount = pPCFInteger->Value;
   break;
case MQIA_CURRENT_Q_DEPTH:
   DefnLQ->CurrentQDepth = pPCFInteger->Value;
   break;
case MQIA_TRIGGER_CONTROL:
   DefnLQ->TriggerControl = pPCFInteger->Value;
   break;
case MQIA_TRIGGER_TYPE:
   DefnLQ->TriggerType = pPCFInteger->Value;
   break;
case MQIA_TRIGGER_MSG_PRIORITY:
   DefnLQ->TriggerMsgPriority = pPCFInteger->Value;
   break;
case MQIA_TRIGGER_DEPTH:
   DefnLQ->TriggerDepth = pPCFInteger->Value;
   break;
case MQIA_Q_DEPTH_HIGH_LIMIT:
   DefnLQ->QDepthHighLimit = pPCFInteger->Value;
   break;
case MQIA_Q_DEPTH_LOW_LIMIT:
   DefnLQ->QDepthLowLimit = pPCFInteger->Value;
   break;
case MQIA_Q_DEPTH_MAX_EVENT:
   DefnLQ->QDepthMaxEvent = pPCFInteger->Value;
   break;
case MQIA_Q_DEPTH_HIGH_EVENT:
   DefnLQ->QDepthHighEvent = pPCFInteger->Value;
   break;
case MQIA_Q_DEPTH_LOW_EVENT:
```

```
      DefnLQ->QDepthLowEvent = pPCFInteger->Value;
      break;
   case MQIA_Q_SERVICE_INTERVAL:
      DefnLQ->QServiceInterval = pPCFInteger->Value;
      break;
   case MQIA_Q_SERVICE_INTERVAL_EVENT:
      DefnLQ->QServiceIntervalEvent = pPCFInteger->Value;
      break;
   } /* endswitch */
}

/* ------------------------------------------------------------------------ */
/*                                                                          */
/* This process takes the attributes of a single local queue and adds them  */
/* to the end of a file, SAVEQMGR.TST, which can be found in the current    */
/* directory.                                                               */
/*                                                                          */
/* The file is of a format suitable for subsequent input to RUNMQSC.        */
/*                                                                          */
/* ------------------------------------------------------------------------ */
void AddToFileQLOCAL( LocalQParms DefnLQ )
{
   char    ParmBuffer[120];   /* Temporary buffer to hold for output to file */
   FILE   *fp;                /* Pointer to a file                           */

   /* Append these details to the end of the current SAVEQMGR.TST file */
   fp = fopen( "SAVEQMGR.TST", "a" );

   sprintf( ParmBuffer, "DEFINE QLOCAL ('%s') REPLACE +\n", DefnLQ.QName );
   fputs( ParmBuffer, fp );

   sprintf( ParmBuffer, "       DESCR('%s') +\n" , DefnLQ.QDesc );
   fputs( ParmBuffer, fp );

   if ( DefnLQ.InhibitPut == MQQA_PUT_ALLOWED ) {
      sprintf( ParmBuffer, "       PUT(ENABLED) +\n" );
      fputs( ParmBuffer, fp );
   } else {
      sprintf( ParmBuffer, "       PUT(DISABLED) +\n" );
      fputs( ParmBuffer, fp );
   } /* endif */

   sprintf( ParmBuffer, "       DEFPRTY(%d) +\n", DefnLQ.DefPriority );
   fputs( ParmBuffer, fp );

   if ( DefnLQ.DefPersistence == MQPER_PERSISTENT ) {
      sprintf( ParmBuffer, "       DEFPSIST(YES) +\n" );
      fputs( ParmBuffer, fp );
   } else {
      sprintf( ParmBuffer, "       DEFPSIST(NO) +\n" );
      fputs( ParmBuffer, fp );
   } /* endif */

   if ( DefnLQ.InhibitGet == MQQA_GET_ALLOWED ) {
      sprintf( ParmBuffer, "       GET(ENABLED) +\n" );
      fputs( ParmBuffer, fp );
   } else {
      sprintf( ParmBuffer, "       GET(DISABLED) +\n" );
      fputs( ParmBuffer, fp );
   } /* endif */

   sprintf( ParmBuffer, "       MAXDEPTH(%d) +\n", DefnLQ.MaxQDepth );
   fputs( ParmBuffer, fp );

   sprintf( ParmBuffer, "       MAXMSGL(%d) +\n", DefnLQ.MaxMsgLength );
   fputs( ParmBuffer, fp );
```

```
if ( DefnLQ.Shareability == MQQA_SHAREABLE ) {
   sprintf( ParmBuffer, "       SHARE +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       NOSHARE +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.DefInputOpenOption == MQOO_INPUT_SHARED ) {
   sprintf( ParmBuffer, "       DEFSOPT(SHARED) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       DEFSOPT(EXCL) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.MsgDeliverySequence == MQMDS_PRIORITY ) {
   sprintf( ParmBuffer, "       MSGDLVSQ(PRIORITY) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       MSGDLVSQ(FIFO) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.HardenGetBackout == MQQA_BACKOUT_HARDENED ) {
   sprintf( ParmBuffer, "       HARDENBO +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       NOHARDENBO +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.Usage == MQUS_NORMAL ) {
   sprintf( ParmBuffer, "       USAGE(NORMAL) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       USAGE(XMIT) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.TriggerControl == MQTC_OFF ) {
   sprintf( ParmBuffer, "       NOTRIGGER +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       TRIGGER +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

switch ( DefnLQ.TriggerType ) {
case MQTT_NONE:
   sprintf( ParmBuffer, "       TRIGTYPE(NONE) +\n" );
   fputs( ParmBuffer, fp );
   break;
case MQTT_FIRST:
   sprintf( ParmBuffer, "       TRIGTYPE(FIRST) +\n" );
   fputs( ParmBuffer, fp );
   break;
case MQTT_EVERY:
   sprintf( ParmBuffer, "       TRIGTYPE(EVERY) +\n" );
   fputs( ParmBuffer, fp );
   break;
case MQTT_DEPTH:
   sprintf( ParmBuffer, "       TRIGTYPE(DEPTH) +\n" );
   fputs( ParmBuffer, fp );
   break;
} /* endswitch */
```

```
sprintf( ParmBuffer, "       TRIGDPTH(%d) +\n", DefnLQ.TriggerDepth );
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       TRIGMPRI(%d) +\n", DefnLQ.TriggerMsgPriority);
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       TRIGDATA('%s') +\n", DefnLQ.TriggerData );
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       PROCESS('%s') +\n", DefnLQ.ProcessName );
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       INITQ('%s') +\n", DefnLQ.InitiationQName );
fputs( ParmBuffer, fp );


sprintf( ParmBuffer, "       RETINTVL(%d) +\n", DefnLQ.RetentionInterval );
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       BOTHRESH(%d) +\n", DefnLQ.BackoutThreshold );
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       BOQNAME('%s') +\n", DefnLQ.BackoutReqQName );
fputs( ParmBuffer, fp );

if ( DefnLQ.Scope == MQSCO_Q_MGR ) {
   sprintf( ParmBuffer, "       SCOPE(QMGR) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       SCOPE(CELL) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

sprintf( ParmBuffer, "       QDEPTHHI(%d) +\n", DefnLQ.QDepthHighLimit );
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       QDEPTHLO(%d) +\n", DefnLQ.QDepthLowLimit );
fputs( ParmBuffer, fp );

if ( DefnLQ.QDepthMaxEvent == MQEVR_ENABLED ) {
   sprintf( ParmBuffer, "       QDPMAXEV(ENABLED) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       QDPMAXEV(DISABLED) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.QDepthHighEvent == MQEVR_ENABLED ) {
   sprintf( ParmBuffer, "       QDPHIEV(ENABLED) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       QDPHIEV(DISABLED) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.QDepthLowEvent == MQEVR_ENABLED ) {
   sprintf( ParmBuffer, "       QDPLOEV(ENABLED) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       QDPLOEV(DISABLED) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

sprintf( ParmBuffer, "       QSVCINT(%d) +\n", DefnLQ.QServiceInterval );
fputs( ParmBuffer, fp );

switch ( DefnLQ.QServiceIntervalEvent ) {
```

```
   case MQQSIE_OK:
      sprintf( ParmBuffer, "       QSVCIEV(OK)\n" );
      fputs( ParmBuffer, fp );
      break;
   case MQQSIE_NONE:
      sprintf( ParmBuffer, "       QSVCIEV(NONE)\n" );
      fputs( ParmBuffer, fp );
      break;
   case MQQSIE_HIGH:
      sprintf( ParmBuffer, "       QSVCIEV(HIGH)\n" );
      fputs( ParmBuffer, fp );
      break;
   } /* endswitch */

   sprintf( ParmBuffer, "\n" );
   fputs( ParmBuffer, fp );

   fclose(fp);

}

/* ------------------------------------------------------------------------ */
/*                                                                          */
/* The queue manager returns strings of the maximum length for each         */
/* specific parameter, padded with blanks.                                  */
/*                                                                          */
/* We are interested in only the nonblank characters so will extract them   */
/* from the message buffer, and terminate the string with a null, \0.       */
/*                                                                          */
/* ------------------------------------------------------------------------ */
void MQParmCpy( char *target, char *source, int length )
{
   int   counter=0;

   while ( counter < length && source[counter] != ' ' ) {
      target[counter] = source[counter];
      counter++;
   } /* endwhile */

   if ( counter < length) {
      target[counter] = '\0';
   } /* endif */
}

MQHOBJ OpenQ( MQHCONN hConn, MQCHAR48 QName, MQLONG OpenOpts)
{
   MQHOBJ Hobj;
   MQLONG CompCode, Reason;

   ObjDesc.ObjectType = MQOT_Q;
   strncpy(ObjDesc.ObjectName, QName, MQ_Q_NAME_LENGTH);

   MQOPEN(hConn,     /* connection handle                     */
          &ObjDesc,  /* object descriptor for queue           */
          OpenOpts,  /* open options                          */
          &Hobj,     /* object handle                         */
          &CompCode, /* MQOPEN completion code                */
          &Reason);  /* reason code                           */

   /* report reason, if any; stop if failed                    */
   if (Reason != MQRC_NONE)
   {
     printf("MQOPEN for %s ended with Reason Code %d and Comp Code %d\n",
      QName,
      Reason,
      CompCode);
     exit( -1 );
```

```
   }

   return Hobj;
}

void PutMsg(MQHCONN hConn,
      MQCHAR8 MsgFormat,
      MQHOBJ hQName,
      MQCHAR48 QName,
      MQBYTE *UserMsg,
      MQLONG UserMsgLen)
{
   MQLONG CompCode, Reason;

   /* set up the message descriptor prior to putting the message */
   md.Report         = MQRO_NONE;
   md.MsgType        = MQMT_REQUEST;
   md.Expiry         = MQEI_UNLIMITED;
   md.Feedback       = MQFB_NONE;
   md.Encoding       = MQENC_NATIVE;
   md.Priority       = MQPRI_PRIORITY_AS_Q_DEF;
   md.Persistence    = MQPER_PERSISTENCE_AS_Q_DEF;
   md.MsgSeqNumber   = 1;
   md.Offset         = 0;
   md.MsgFlags       = MQMF_NONE;
   md.OriginalLength = MQOL_UNDEFINED;

   memcpy(md.GroupId,  MQGI_NONE, sizeof(md.GroupId));
   memcpy(md.Format,   MsgFormat, sizeof(md.Format) );
   memcpy(md.ReplyToQ, QName,     sizeof(md.ReplyToQ) );

   /* reset MsgId and CorrelId to get a new one                */
   memcpy(md.MsgId,    MQMI_NONE, sizeof(md.MsgId) );
   memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId) );

   MQPUT(hConn,             /* connection handle               */
         hQName,            /* object handle                   */
         &md,               /* message descriptor              */
         &pmo,              /* default options                 */
         UserMsgLen,        /* message length                  */
         (MQBYTE *)UserMsg, /* message buffer                  */
         &CompCode,         /* completion code                 */
         &Reason);          /* reason code                     */

   if (Reason != MQRC_NONE) {
      printf("MQPUT ended with with Reason Code %d and Comp Code %d\n",
       Reason, CompCode);
      exit( -1 );
   }
}

void GetMsg(MQHCONN hConn, MQLONG MQParm, MQHOBJ hQName,
      MQBYTE *UserMsg, MQLONG ReadBufferLen)
{
   MQLONG CompCode, Reason, msglen;

   gmo.Options      = MQParm;
   gmo.WaitInterval = 15000;

   /* reset MsgId and CorrelId to get a new one                */
   memcpy(md.MsgId,    MQMI_NONE, sizeof(md.MsgId) );
   memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId) );

   MQGET(hConn,             /* connection handle               */
         hQName,            /* object handle                   */
         &md,               /* message descriptor              */
         &gmo,              /* get message options             */
```

```
         ReadBufferLen,     /* Buffer length                   */
         (MQBYTE *)UserMsg, /* message buffer                  */
         &msglen,           /* message length                  */
         &CompCode,         /* completion code                 */
         &Reason);          /* reason code                     */

   if (Reason != MQRC_NONE) {
      printf("MQGET ended with Reason Code %d and Comp Code %d\n",
       Reason, CompCode);
      exit( -1 );
}
}
```

Administrative REST API reference

Reference information about the administrative REST API.

For more information about using the administrative REST API, see Administration using the REST API.

REST API resources

This collection of topics provides reference information for each of the administrative REST API resources.

For more information about using the administrative REST API, see Administration using the REST API.

/admin/action/qmgr/{qmgrName}/mqsc: V 9.0.4

You can use the HTTP POST method with the `/admin/action/qmgr/{qmgrName}/mqsc` resource to execute an arbitrary MQSC command on a queue manager.

V 9.0.5 From Version 9.0.5, you can use the administrative REST API gateway with this resource URL.

POST: V 9.0.4

Use the HTTP POST method with this resource to submit administrative commands directly to a queue manager.

You can use this REST API command with HTTP to run any MQSC command that is not part of the administrative REST API.

On UNIX, Linux, and Windows, this REST API command is similar to the **Escape** PCF command.

On z/OS, this REST API command is similar to submitting commands directly to the command server:

- Messages are put to a request queue. These messages have:
  - `MsgType` set to `MQMT_REQUEST`.
  - `Format` set to `MQFMT_STRING` or `MQFMT_NONE`.
  - Payload set to the text of an MQSC command.
- The command server running in the queue manager:
  - Reads the messages.
  - Validates them.
  - Passes valid commands to the command processor.
- The command processor:
  - Executes the commands.

– Puts replies to the commands as messages on the reply-to queues specified in the incoming messages.

This REST API is deliberately lightweight in approach. Commands are submitted to the specified queue manager and the results are returned in an unprocessed format. For examples of this format, see the "Example - z/OS" on page 1635 section.

Resource URL

Version 9.0.4 and later:

`https://host:port/ibmmq/rest/v1/admin/action/qmgr/qmgrName/mqsc`

qmgrName
Specifies the name of the queue manager on which to execute the command.

V 9.0.5 From Version 9.0.5, you can use the administrative REST API gateway with this resource URL. That is, you can specify a remote queue manager as the queue manager name.

The queue manager name is case-sensitive.

If the queue manager name includes a forward slash, a period, or a percent sign, these characters must be URL encoded:

- A forward slash (/) must be encoded as `%2F`.
- A percent sign (%) must be encoded as `%25`.

V 9.0.1 You can use HTTP instead of HTTPS if you enable HTTP connections. For more information about enabling HTTP, see Configuring HTTP and HTTPS ports.

Request headers

The following headers must be sent with the request:

Content-Type
This header must be sent with a value of `application/json;charset=utf-8`.

ibm-mq-rest-csrf-token
This header must be sent with a value that is the content of the `csrfToken` cookie. The content of the `csrfToken` cookie is used to confirm that the credentials that are being used to authenticate the request are being used by the owner of the credentials. That is, the token is used to prevent cross-site request forgery attacks.

The `csrfToken` cookie is returned after a request is made with an HTTP GET method. You cannot use a cached version of the content of the cookie because the content of the cookie can change. You must use the latest value of the cookie for each request.

V 9.0.5 The preceding information applies to releases up to and including IBM MQ Version 9.0.4. From IBM MQ Version 9.0.5, this header must be set, but the value can be anything, including

being blank.
The `csrfToken` cookie is no longer sent in responses from the REST API in Version 9.0.5 and later.

`Authorization`
This header must be sent if you are using basic authentication. For more information, see Using HTTP basic authentication with the REST API.

Request body format

The request body must be in JSON format in UTF-8 encoding. Within the request body attributes are defined, and named JSON objects are created to specify extra attributes. Any attributes that are not specified use the default value.

The following attributes can be included in the request body:

`type`
Required.

String.

Specifies the type of action to be performed.

`runCommand`
Specifies that an MQSC command is to be executed

`parameters`
Required.

Nested JSON Object.

Specifies the parameters for the action.

This nested object contains only one attribute.

`command`
Required.

A valid MQSC command to be executed.

Security requirements

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. For more information about security for the administrative REST API, see IBM MQ Console and REST API security.

The security principal of the caller must be granted the ability to issue such MQSC commands against the specified queue manager as they specify.

ULW On UNIX, Linux, and Windows, you can grant authority to security principals to use IBM MQ resources by using the **`mqsetaut`** command. For more information, see mqsetaut.

z/OS On z/OS, see Setting up security on z/OS.

Response status codes

200 The specified command was executed successfully.

400 Invalid data provided.

For example, an invalid MQSC command is specified.

401 Not authenticated.

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. The `ibm-mq-rest-csrf-token` header must also be specified. For more information, see "Security requirements" on page 1633.

403 Not authorized.

The caller is authenticated to the mqweb server and is associated with a valid principal. However, the principal does not have access to all, or a subset of the required IBM MQ resources. For more information about the access that is required, see "Security requirements" on page 1633.

500 Server issue or error code from IBM MQ.

503 Queue manager not running.

Response headers

None.

Response body format

If an error occurs, the response body contains an error message. For more information, see REST API error handling.

The format of the response body is standardized, with a consistent JSON schema. However, the content is platform-dependent, reflecting the underlying mechanism for executing MQSC commands.

The response body has the following JSON structure:

```
{
  "commandResponse" : [
    {
      "completionCode" : number,
      "reasonCode" :  number,
      "text" : [
        "string",
        ...
        ]
    },
    ...
  ]
  "overallCompletionCode" : number,
  "overAllReasonCode" :  number
}
```

The fields in the response have the following meanings:

commandResponse
A JSON array of JSON objects that represent individual responses from the execution of the command.

Each response contains the following data:

completionCode
The completion code that is associated with the operation for this instance.

reasonCode
The reason code that is associated with the operation for this instance.

text A JSON array of strings that contain the response text that is associated with the operation for this instance. Note that embedded newlines will be stripped from this text.

On UNIX, Linux, and Windows, this field contains a single string that contains the response from the command, with any newlines escaped in the usual JSON manner.

On z/OS, this field contains multiple entries. For further information, see Interpreting the reply messages from the command server.

overallCompletionCode
The completion code that is associated with the operation as a whole.

overallReasonCode
The reason code that is associated with the operation for this instance.

Example - z/OS

The following sequence shows how to create a new server-connection channel that is called NEWSVRCONN on a z/OS queue manager - our example queue manager is called QM21.

- First check that the channel does not exist. The following URL is used with the HTTP POST method:

  Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/action/qmgr/QM21/mqsc
  ```

  The following JSON payload is sent:

  ```
  {
    "type": "runCommand",
    "parameters": {
      "command": "DISPLAY CHANNEL(NEWSVRCONN)"
    }
  }
  ```

  A response code of 200 is returned, as the REST command succeeded. The response body that is returned contains the following JSON.

  ```
  {
    "commandResponse": [
      {
        "completionCode": 0,
        "reasonCode": 0,
        "text": [
          "CSQN205I   COUNT=       3, RETURN=00000000, REASON=00000000",
          "CSQM297I ]MQ21 CSQMDRTS NO CHANNEL FOUND MATCHING REQUEST CRITERIA ",
          "CSQ9022I ]MQ21 CSQMDRTS ' DISPLAY CHANNEL' NORMAL COMPLETION "
        ]
      }
    ],
    "overallCompletionCode": 0,
    "overallReasonCode": 0
  }
  ```

  The completion and reason codes here are zero, as on z/OS the command is regarded as succeeding, although no matching channel was.

- Now create the channel. The same URL is used with the HTTP POST method:

  Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/action/qmgr/QM21/mqsc
  ```

  The following JSON payload is sent:

  ```
  {
    "type": "runCommand",
    "parameters": {
      "command": "DEFINE CHANNEL(NEWSVRCONN) CHLTYPE(SVRCONN)"
    }
  ```

  A response code of 200 is returned, as the REST command succeeded. The response body that is returned contains the following JSON.

```
{
  "commandResponse": [
    {
      "completionCode": 0,
      "reasonCode": 0,
      "text": [
        "CSQN205I   COUNT=       2, RETURN=00000000, REASON=00000000",
        "CSQ9022I ]MQ21 CSQMACHL ' DEFINE CHANNEL' NORMAL COMPLETION"
      ]
    }
  ],
  "overallCompletionCode": 0,
  "overallReasonCode": 0
}
```

- Finally, check that the channel does exist. Again the same URL is used with the HTTP POST method:

  Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/action/qmgr/QM21/mqsc
  ```

  The following JSON payload is sent:

  ```
  {
    "type": "runCommand",
    "parameters": {
      "command": "DISPLAY CHANNEL(NEWSVRCONN) ALL"
    }
  }
  ```

  A response code of 200 is returned, as the REST command succeeded. The response body that is returned contains the following JSON. The response body is edited for brevity after the TRPTYPE attribute.

  ```
  {
    "commandResponse": [
      {
        "completionCode": 0,
        "reasonCode": 0,
        "text": [
          "CSQN205I   COUNT=       3, RETURN=00000000, REASON=00000000",
          "CSQM415I ]MQ21 CHANNEL(NEWSVRCONN          ) CHLTYPE(SVRCONN   ) QSGDISP(QMGR      ) DEFCDISP(PRIVATE   ) TRPTYP
          "CSQ9022I ]MQ21 CSQMDRTS ' DISPLAY CHANNEL' NORMAL COMPLETION "
        ]
      }
    ],
    "overallCompletionCode": 0,
    "overallReasonCode": 0
  }
  ```

Example - UNIX, Linux, and Windows

The following sequence shows how to create a new server-connection channel that is called NEWSVRCONN on UNIX, Linux, and Windows queue managers - our example queue manager is called QM_T1.

- First check that the channel does not exist. The following URL is used with the HTTP POST method:

  Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/action/qmgr/QM_T1/mqsc
  ```

  The following JSON payload is sent:

  ```
  {
    "type": "runCommand",
    "parameters": {
      "command": "DISPLAY CHANNEL(NEWSVRCONN)"
    }
  }
  ```

A response code of 200 is returned, as the REST command succeeded. The response body that is returned contains the following JSON.

```
{
  "commandResponse": [
    {
      "completionCode": 2,
      "reasonCode": 2085,
      "text": [
        "AMQ8147: IBM MQ object NEWSVRCONN not found."
      ]
    }
  ],
  "overallCompletionCode": 2,
  "overallReasonCode": 3008
}
```

The individual response shows a reason code of 2085 (MQRC_UNKNOWN_OBJECT_NAME) and the MQSC command has an overall reason code of 3008 (MQRCCF_COMMAND_FAILED) as it failed to display the requested channel's details.

- Now create the channel. The same URL is used with the HTTP POST method:

  Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/action/qmgr/QM_T1/mqsc
  ```

  The following JSON payload is sent:

  ```
  {
    "type": "runCommand",
    "parameters": {
      "command": "DEFINE CHANNEL(NEWSVRCONN) CHLTYPE(SVRCONN)"
    }
  ```

  A response code of 200 is returned, as the REST command succeeded. The response body that is returned contains the following JSON.

  ```
  {
    "commandResponse": [
      {
        "completionCode": 0,
        "reasonCode": 0,
        "text": [
          "AMQ8014: IBM MQ channel created."
        ]
      }
    ],
    "overallCompletionCode": 0,
    "overallReasonCode": 0
  }
  ```

- Finally, check that the channel does exist. Again the same URL is used with the HTTP POST method:

  Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/action/qmgr/QM_T1/mqsc
  ```

  The following JSON payload is sent:

  ```
  {
    "type": "runCommand",
    "parameters": {
      "command": "DISPLAY CHANNEL(NEWSVRCONN) ALL"
    }
  }
  ```

A response code of 200 is returned, as the REST command succeeded. The response body that is returned contains the following JSON. The response body is edited for brevity after the CHLTYPE attribute.

```
{
  "commandResponse": [
    {
      "completionCode": 0,
      "reasonCode": 0,
      "text": [
        "AMQ8414: Display Channel details.   CHANNEL(NEWSVRCONN)                    CHLTYPE(SVRCONN)"
      ]
    }
  ],
  "overallCompletionCode": 0,
  "overallReasonCode": 0
}
```

/admin/installation: V 9.0.1

You can use the HTTP GET method with the `installation` resource to request information about installations.

V 9.0.4 You cannot use the administrative REST API gateway with this resource URL.

GET: V 9.0.1

Use the HTTP GET method with the `installation` resource to request information about the installation that the administrative REST API runs in.

The information that is returned is similar to the information that is returned by the **dspmqver** control command.

- Resource URL
- Optional query parameters
- V 9.0.2 "Request headers" on page 1640
- Request body format
- V 9.0.2 "Security requirements" on page 1640
- Response status codes
- "Response headers" on page 1640
- Response body format
- Examples

Resource URL

V 9.0.4 Version 9.0.4 and later:

`https://host:port/ibmmq/rest/v1/admin/installation/{installationName}`

Version 9.0.3 and earlier:

`https://host:port/ibmmq/rest/v1/installation/{installationName}`

installationName
Optionally specifies the name of the installation to query. This name must be the name of the installation that the REST API is running in.

V 9.0.1 You can use HTTP instead of HTTPS if you enable HTTP connections. For more information about enabling HTTP, see Configuring HTTP and HTTPS ports.

Optional query parameters

attributes={extended|*|extended.*attributeName*,...}

extended
Specifies that all extended attributes are returned.

***** Specifies all attributes. This parameter is equivalent to **extended**.

extended.*attributeName*,...
Specifies a comma-separated list of extended attributes to return:

level String.

IBM MQ build level.

operatingSystem
ULW z/OS This attribute is only available on z/OS, UNIX, Linux, and Windows.

String.

Full descriptive text of the operating system.

hostName
String.

System host name.

If the system has multiple hosts, only one name is returned.

description
ULW This attribute is only available on UNIX, Linux, and Windows.

String.

Installation description.

installationPath
ULW This attribute is only available on UNIX, Linux, and Windows.

String.

The path to the installation.

dataPath
ULW This attribute is only available on UNIX, Linux, and Windows.

String.

The path to where the data for the installation is stored.

maximumCommandLevel
ULW IBM MQ Appliance This attribute is only available on the IBM MQ Appliance, UNIX, Linux, and Windows.

Integer.

Maximum command level that is supported.

primary
ULW This attribute is only available on UNIX, Linux, and Windows.

Boolean.

Primary installation status.

V 9.0.2

Request headers

The following headers must be sent with the request:

`Authorization`
This header must be sent if you are using basic authentication. For more information, see Using HTTP basic authentication with the REST API.

Request body format

None.

Security requirements

V 9.0.2 The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. For more information about security for the administrative REST API, see IBM MQ Console and REST API security.

There are no specific authorization requirements for an HTTP GET on the `installation` resource.

Response status codes

200 Installation information retrieved successfully.

400 Invalid data provided.

For example, invalid installation attributes specified.

401 Not authenticated.

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. For more information, see "Security requirements."

404 Installation does not exist.

500 Server issue or error code from IBM MQ.

Response headers

The following headers are returned with the response:

`Content-Type`
This header is returned with a value of `application/json;charset=utf-8`.

Response body format

The response is in JSON format in UTF-8 encoding. The response contains an outer JSON object that contains a single JSON array called `installation`. Each element in the array is a JSON object that represents information about an installation. Each JSON object contains the following attributes:

`name`

ULW This attribute is only available on UNIX, Linux, and Windows.

String.

The installation name.

version
String.

The version of IBM MQ for the installation.

platform
String.

One of the following values:

- appliance
- ibm-i
- unix
- windows
- z/os

extended
JSON object.

If requested, contains one or more of the following extra properties:

level String.

IBM MQ build level.

operatingSystem
ULW z/OS This attribute is only available on z/OS, UNIX, Linux, and Windows.

String.

Full descriptive text of the operating system.

hostName
String.

System host name.

If the system has multiple hosts, only one name is returned.

description
ULW This attribute is only available on UNIX, Linux, and Windows.

String.

Installation description.

installationPath
ULW This attribute is only available on UNIX, Linux, and Windows.

String.

The path to the installation.

dataPath
ULW This attribute is only available on UNIX, Linux, and Windows.

String.

The path to where the data for the installation is stored.

maximumCommandLevel
ULW IBM MQ Appliance This attribute is only available on the IBM MQ Appliance, UNIX, Linux, and Windows.

Integer.

Maximum command level that is supported.

primary

ULW This attribute is only available on UNIX, Linux, and Windows.

Boolean.

Primary installation status.

If an error occurs, the response body contains an error message. For more information, see REST API error handling.

ULW

Examples for UNIX, Linux, and Windows

- The following example gets basic information about the installation that the REST API is running in. The following URL is used with the HTTP GET method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/installation
  ```

  Version 9.0.3 and earlier:

  ```
  https://localhost:9443/ibmmq/rest/v1/installation
  ```

  The following JSON response is returned:

  ```
  {
      "installation":
      [{
         "name": "Installation1",
         "platform": "windows",
         "version": "9.0.0.0"
      }]
  }
  ```

- The following example gets extended information about the installation `Installation1`. The following URL is used with the HTTP GET method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/installation/Installation1?attributes=*
  ```

  Version 9.0.3 and earlier:

  ```
  https://localhost:9443/ibmmq/rest/v1/installation/Installation1?attributes=*
  ```

  The following JSON response is returned:

  ```
  {
      "installation":
      [{
           "extended": {
               "dataPath": "C:\\Program Files (x86)\\IBM\\WebSphere MQ",
               "description": "My MQ installation",
               "hostName": "exampleHost",
               "installationPath": "C:\\Program Files\\IBM\\WebSphere MQ",
               "level": "p900-L160614",
               "maximumCommandLevel": 900,
               "operatingSystem": "Windows 7 Professional x64 Edition, Build 7601: SP1",
               "primary": true
           },
           "name": "Installation1",
           "platform": "windows",
           "version": "9.0.0.0"
      }]
  }
  ```

- The following example gets the installation path and host name for `Installation1`. The following URL is used with the HTTP GET method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/installation/Installation1?attributes=extended.installationPath,extended.ho
  ```

  Version 9.0.3 and earlier:

  ```
  https://localhost:9443/ibmmq/rest/v1/installation/Installation1?attributes=extended.installationPath,extended.hostName
  ```

  The following JSON response is returned:

  ```
  {
      "installation": [{
          "extended": {
              "hostName": "exampleHost",
              "installationPath": "C:\\Program Files\\IBM\\MQ"
           },
           "name": "Installation1",
           "platform": "windows",
           "version": "9.0.1.0"
      }]
  }
  ```

z/OS

Examples for z/OS

- The following example gets basic information about the installation. The following URL is used with the HTTP GET method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://REST.example.com:9443/ibmmq/rest/v1/admin/installation
  ```

  Version 9.0.3 and earlier:

  ```
  https://REST.example.com:9443/ibmmq/rest/v1/installation
  ```

  The following JSON response is returned:

  ```
  {
      "installation": [{
         "platform": "z/os",
         "version": "9.0.1"
      }]
  }
  ```

- The following example gets extended information about the installation. The following URL is used with the HTTP GET method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://REST.example.com:9443/ibmmq/rest/v1/admin/installation?attributes=extended
  ```

  Version 9.0.3 and earlier:

  ```
  https://REST.example.com:9443/ibmmq/rest/v1/installation?attributes=extended
  ```

  The following JSON response is returned:

  ```
  {
      "installation": [{
         "extended": {
             "hostName": "REST.example.com",
             "level": "V901-L161011",
             "operatingSystem": "z/OS 01.00 02"
         },
         "platform": "z/os",
         "version": "9.0.1"
     }]
  }
  ```

/login: V 9.0.2

You can use the HTTP GET method together with the `login` resource to get information about the user that is logged in. You can use the HTTP POST method to log in a user and get an LTPA token and V 9.0.5, for releases before Version 9.0.5, a CSRF token. You can use the HTTP DELETE method to log out a user and end the session.

POST: V 9.0.2

Use the HTTP POST method with the `login` resource to log in a user and start a token-based authentication session. An LTPA token is returned for the user to authenticate further REST requests.

For more information about how to use token based authentication, see Using token based authentication with the REST API.

- Resource URL
- Optional query parameters
- "Request headers"
- Request body format
- Response status codes
- "Response headers" on page 1645
- Response body format
- Examples

Resource URL

`https://host:port/ibmmq/rest/v1/login`

Optional query parameters

None.

Request headers

The following headers must be sent with the request:

`Content-Type`
This header must be sent with a value of `application/json;charset=utf-8`.

Request body format

The request body must be in JSON format in UTF-8 encoding. Within the request body attributes are defined. The following attributes can be included in the request body:

`username`
String.

Specifies the user name to authenticate with.

The user name that is specified must be defined within the mqweb server user registry, and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles.

`password`
String.

Specifies the password of the user that is specified by the **`username`** attribute.

Response status codes

204 User logged in successfully.

400 Invalid data provided.

For example, an integer value is specified for the user name.

401 Not authenticated.

An invalid user name or password was provided.

500 Server issue or error code from IBM MQ.

Response headers

None.

Response body format

The response body is empty if the login is successful. If an error occurs, the response body contains an error message. For more information, see REST API error handling.

V 9.0.5 From IBM MQ Version 9.0.5, with a successful login, a security token, `LtpaToken2`, which is used to authenticate all further REST requests, is returned.

For IBM MQ Version 9.0.4 and earlier, with a successful login, two cookies are returned:

- A security token, `LtpaToken2`, which is used to authenticate all further REST requests.
- A CSRF token, `csrfToken`, which is used in the `ibm-mq-rest-csrf-token` HTTP header for REST requests that use the POST, PATCH, or DELETE HTTP methods.

Examples

The following example logs in a user called `mqadmin` with the password `mqadmin`. The following URL is used with the HTTP POST method:

```
https://localhost:9443/ibmmq/rest/v1/login
```

The following JSON payload is sent:

```
{
    "username" : "mqadmin",
    "password" : "mqadmin"
}
```

In cURL, the log in request might look like the following Windows example. The LTPA token is stored in the `cookiejar.txt` file by using the `-c` flag:

```
curl -k "https://localhost:9443/ibmmq/rest/v1/login" -X POST
-H "Content-Type: application/json" --data "{\"username\":\"mqadmin\",\"password\":\"mqadmin\"}"
-c c:\cookiejar.txt
```

After the user is logged in, the LTPA token and `ibm-mq-rest-csrf-token` HTTP header are used to authenticate further requests. For example, to create a local queue, `Q1`, the following cURL might be used. The LTPA token is retrieved from the `cookiejar.txt` file by using the `-b` flag. The required contents of the `ibm-mq-rest-csrf-token` HTTP header varies depending on the version of IBM MQ.

V 9.0.5 From Version 9.0.5, its value can be anything including blank.

For IBM MQ Version 9.0.4 and earlier, after the user is logged in, the LTPA token and CSRF token are used to authenticate further requests. For example, to create a local queue, `Q1`, the following cURL might

be used. The LTPA token is retrieved from the `cookiejar.txt` file by using the `-b` flag. The CSRF token is included in an `ibm-mq-rest-csrf-token` HTTP header. The value of the CSRF token is copied from the `cookiejar.txt` file:

V 9.0.5 Version 9.0.5

```
curl -k "https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue" -X POST
-b c:\cookiejar.txt
-H "ibm-mq-rest-csrf-token: value" -H "Content-Type: application/json"
--data "{\"name\":\"Q1\"}"
```

V 9.0.4 Version 9.0.4:

```
curl -k "https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue" -X POST -b c:\cookiejar.txt
-H "ibm-mq-rest-csrf-token: 416E144A02E19E515ED5709A77FB07B4EF550FD1FE1CC44CF82C5774088A041928486A
BE9597618938B9F51D12FE4A0DFC1CB41D0C7567E9AB890F0FDB0EE43A27756F32341E712EFB82305F8603E566D3F1D041
2BADDF60AEEE656A2F3D06034FEF535BB67D52ACE265B3B6FB0D1B7F5EC83354F2118226C89FAC200724963FBA9BDA30376
DD84331933E300E543D01AEFE4AE638A6284DBA0210932CF00F376E1501615910926BA38D612682F22DC92391776B013C38
E73516CDC958F3D20661765097E4E0F4FC36DC13871C6BDE06D95E33D0EF4B41742D95F54DF962BE28FCDE04963DF77EB9A3
FEFB27CD2597415DDB9D1427602DDF517D4E07C092BEA3" -H "Content-Type: application/json"
--data "{\"name\":\"Q1\"}"
```

Version 9.0.3 and earlier:

```
curl -k "https://localhost:9443/ibmmq/rest/v1/qmgr/QM1/queue" -X POST -b c:\cookiejar.txt
-H "ibm-mq-rest-csrf-token: 416E144A02E19E515ED5709A77FB07B4EF550FD1FE1CC44CF82C5774088A041928486A
BE9597618938B9F51D12FE4A0DFC1CB41D0C7567E9AB890F0FDB0EE43A27756F32341E712EFB82305F8603E566D3F1D041
2BADDF60AEEE656A2F3D06034FEF535BB67D52ACE265B3B6FB0D1B7F5EC83354F2118226C89FAC200724963FBA9BDA30376
DD84331933E300E543D01AEFE4AE638A6284DBA0210932CF00F376E1501615910926BA38D612682F22DC92391776B013C38
E73516CDC958F3D20661765097E4E0F4FC36DC13871C6BDE06D95E33D0EF4B41742D95F54DF962BE28FCDE04963DF77EB9A3
FEFB27CD2597415DDB9D1427602DDF517D4E07C092BEA3" -H "Content-Type: application/json"
--data "{\"name\":\"Q1\"}"
```

GET: V 9.0.2

Use the HTTP GET method with the `login` resource to request information about the user that is authenticated.

- Resource URL
- Optional query parameters
- "Request headers"
- Request body format
- "Security requirements" on page 1647
- Response status codes
- "Response headers" on page 1647
- Response body format
- Examples

Resource URL

`https://`*`host`*`:`*`port`*`/ibmmq/rest/v1/login`

Optional query parameters

None.

Request headers

The following headers must be sent with the request:

Authorization
This header must be sent if you are using basic authentication. For more information, see Using HTTP basic authentication with the REST API.

Request body format

None.

Security requirements

The request must be authenticated by using one of the following authentication mechanisms:

- For HTTP basic authentication, you must provide the user name and password to authenticate. For more information, see Using HTTP basic authentication with the REST API.
- For token based authentication, you must provide the LTPA token to authenticate. For more information, see Using token based authentication with the REST API.
- For client certificate authentication, you must provide the client certificate to authenticate. For more information, see Using client certificate authentication with the REST API.

Response status codes

200 User queried successfully.

400 Invalid data provided.

401 Not authenticated.

An invalid credential was provided.

404 Resource was not found.

500 Server issue or error code from IBM MQ.

Response headers

The following headers are returned with the response:

Content-Type
This header is returned with a value of `application/json;charset=utf-8`.

Response body format

The response is in JSON format in UTF-8 encoding. The response contains an outer JSON object that contains a single JSON array called `user`. This array contains the following attributes:

authenticationMechanism
String.

Specifies how the user was authenticated.

The value is one of the following values:

form
The user is authenticated with token authentication.

basic
The user is authenticated with HTTP basic authentication.

clientCertificate
The user is authenticated with client certificate authentication (X.509).

noSecurity
Security is not enabled.

name
String.

Specifies the name of the user that is used to check for authorization.

This name might be different from the credentials that are specified using, for example, LDAP user mapping or client certificate user mapping.

role
JSON array.

Specifies which roles the user is granted.

The value is one or more of the following values:

- `MQWebAdmin`
- `MQWebAdminRO`
- `MQWebUser`

Examples

The following example queries the user. The following URL is used with the HTTP GET method:

```
https://localhost:9443/ibmmq/rest/v1/login
```

The following JSON response is returned:

```
{
    "user" :
    [{
       "name" : "reader",
       "role" : [
           "MQWebAdminRO",
           "MQWebUser"
       ],
       "authenticationMechanism" : "form"
    }]
}
```

In cURL, the log in query might look like the following Windows example that uses token based authentication. The LTPA token is retrieved from the `cookiejar.txt` file by using the `-b` flag:

```
curl -k "https://localhost:9443/ibmmq/rest/v1/login" -X GET
-b c:\cookiejar.txt
```

DELETE: V 9.0.2

Use the HTTP DELETE method with the `login` resource to log out a user and end a token-based authentication session.

For more information about how to use token based authentication, see Using token based authentication with the REST API.

- Resource URL
- Optional query parameters
- “Request headers” on page 1649
- Request body format
- “Security requirements” on page 1649
- Response status codes
- “Response headers” on page 1650
- Response body format

- Examples

Resource URL

`https://host:port/ibmmq/rest/v1/login`

Optional query parameters

None.

Request headers

The following headers must be sent with the request:

ibm-mq-rest-csrf-token
This header must be sent with a value that is the content of the `csrfToken` cookie. The content of the `csrfToken` cookie is used to confirm that the credentials that are being used to authenticate the request are being used by the owner of the credentials. That is, the token is used to prevent cross-site request forgery attacks.

The `csrfToken` cookie is returned after a request is made with an HTTP GET method. You cannot use a cached version of the content of the cookie because the content of the cookie can change. You must use the latest value of the cookie for each request.

V 9.0.5 The preceding information applies to releases up to and including IBM MQ Version 9.0.4. From IBM MQ Version 9.0.5, this header must be set, but the value can be anything, including being blank.
The `csrfToken` cookie is no longer sent in responses from the REST API in Version 9.0.5 and later.

Request body format

None.

Security requirements

The following tokens must be provided with the request to authenticate:
- The LTPA token that is used to authenticate the user must be provided as a cookie.

With the response to the REST request, an instruction to delete the LTPA token from the local cookie store is included. Ensure that you process this instruction. If the instruction is not processed, and the LTPA token remains in the local cookie store, then the LTPA token can be used to authenticate future REST requests. That is, when the user attempts to authenticate with the LTPA token after the session is ended, a new session is created that uses the existing token.

Response status codes

204 User logged out successfully.

400 Invalid data provided.

401 Not authenticated.

An invalid LTPA token was provided; the contents of the `ibm-mq-rest-csrf-token` HTTP header was incorrect, or the `ibm-mq-rest-csrf-token` header was missing.

404 Resource was not found.

500 Server issue or error code from IBM MQ.

Response headers

None.

Response body format

The response body is empty if the logout is successful. If an error occurs, the response body contains an error message. For more information, see REST API error handling.

Examples

The following cURL example for Windows logs out a user.

V 9.0.5 From IBM MQ Version 9.0.5, the LTPA token is retrieved from the `cookiejar.txt` file by using the `-b` flag. CSRF protection is provided by the presence of the `ibm-mq-rest-csrf-token` HTTP header. The location of the `cookiejar.txt` file is specified by the `-c` flag so that the LTPA token is deleted from the file:

```
curl -k "https://localhost:9443/ibmmq/rest/v1/login" -X DELETE
-H "ibm-mq-rest-csrf-token: value" -b c:\cookiejar.txt
-c c:\cookiejar.txt
```

For IBM MQ Version 9.0.4 and earlier, the LTPA token is retrieved from the `cookiejar.txt` file by using the `-b` flag. The CSRF token is included in an `ibm-mq-rest-csrf-token` HTTP header. The value of the CSRF token is copied from the `cookiejar.txt` file. The location of the `cookiejar.txt` file is specified by the `-c` flag so that the LTPA token is deleted from the file:

```
curl -k "https://localhost:9443/ibmmq/rest/v1/login" -X DELETE
-H "ibm-mq-rest-csrf-token: 416E144A02E19E515ED5709A77FB07B4EF550FD1FE1
CC44CF82C5774088A041928486ABE9597618938B9F51D12FE4A0DFC1CB41D0C7567E9AB8
90F0FDB0EE43A27756F32341E712EFB82305F8603E566D3F1D0412BADDF60AEEE656A2F3
D06034FEF535BB67D52ACE265B3B6FB0D1B7F5EC83354F2118226C89FAC200724963FBA9
BDA30376DD84331933E300E543D01AEFE4AE638A6284DBA0210932CF00F376E150161591
0926BA38D612682F22DC92391776B013C38E73516CDC958F3D20661765097E4E0F4FC36D
C13871C6BDE06D95E33D0EF4B41742D95F54DF962BE28FCDE04963DF77EB9A3FEFB27CD2
597415DDB9D1427602DDF517D4E07C092BEA3" -b c:\cookiejar.txt
-c c:\cookiejar.txt
```

/admin/qmgr: V 9.0.1

You can use the HTTP GET method with the `qmgr` resource to request information about queue managers V 9.0.3 , including status information.

V 9.0.4 You can use the administrative REST API gateway with this resource URL.

For more information about the PCF equivalents to the queue manager REST API parameters and attributes, see “REST API and PCF equivalents for queue managers” on page 1796.

GET: V 9.0.1

Use the HTTP GET method with the `qmgr` resource to request basic information and status information about queue managers.

The information that is returned is similar to the information that is returned by the **dspmq** control command, the **DISPLAY QMSTATUS** MQSC command, and the **Inquire Queue Manager Status** PCF command.

- Resource URL
- Optional query parameters
- "Request headers" on page 1653
- Request body format
- V 9.0.2 "Security requirements" on page 1653
- Response status codes
- "Response headers" on page 1654
- Response body format
- Examples

Resource URL

V 9.0.4 Version 9.0.4 and later:

`https://host:port/ibmmq/rest/v1/admin/qmgr/{qmgrName}`

Version 9.0.3 and earlier:

`https://host:port/ibmmq/rest/v1/qmgr/{qmgrName}`

qmgrName
Optionally specifies the name of the queue manager to query.

V 9.0.4 You can specify a remote queue manager as the **qmgrName**. If you specify a remote queue manager, you must configure a gateway queue manager. For more information, see Remote administration using the REST API.

If you specify a remote queue manager, only the following attributes are returned:

- name
- started
- channelInitiatorState
- ldapConnectionState
- connectionCount
- publishSubscribeState

The queue manager name is case-sensitive.

If the queue manager name includes a forward slash, a period, or a percent sign, these characters must be URL encoded:

- A forward slash (/) must be encoded as `%2F`.
- A percent sign (%) must be encoded as `%25`.

V 9.0.1 You can use HTTP instead of HTTPS if you enable HTTP connections. For more information about enabling HTTP, see Configuring HTTP and HTTPS ports.

Optional query parameters

attributes={extended|*|extended.*attributeName*,...}

ULW IBM MQ Appliance This parameter is only available on the IBM MQ Appliance, UNIX, Linux, and Windows.

This parameter is not valid if you specify a remote queue manager in the resource URL.

extended
Specifies that all extended attributes are retrieved.

***** Specifies all attributes. This parameter is equivalent to **extended**.

extended.*attributeName*,...
Specifies a comma-separated list of extended attributes to return.

For example, to return the `installationName` attribute, specify `extended.installationName`.

For a full list of extended attributes, see Extended attributes for queue managers.

V 9.0.3 **status={status|*|status.*attributeName*,...}**

Note: In Version 9.0.1, the **status** optional query parameter was used to filter returned queues based on the running state of the queue. From Version 9.0.2, this optional query parameter is known as **state**.

status
Specifies that all status attributes are returned.

***** Specifies all attributes. This parameter is equivalent to **status**.

status.*attributeName*,...
Specifies a comma-separated list of queue manager status attributes to return.

The queue manager must be running to return the status attributes.

For example, to return the `connectionCount` attribute, specify `status.connectionCount`.

For a full list of status attributes, see Status attributes for queue managers.

V 9.0.2 **state=*state***

Note: In Version 9.0.1, the **state** optional query parameter was known as **status**. From Version 9.0.3, the **status** optional query parameter is used to retrieve status attributes for the queue manager.

Specifies that only queue managers with the specified state are returned. The following values are valid values:
On all platforms:

- running
- ended

ULW On UNIX, Linux, and Windows:

- endedImmediately
- endedPreemptively
- endedUnexpectedly
- starting
- quiescing
- endingImmediately
- endingPreemptively
- beingDeleted

- stateNotAvailable
- runningAsStandby
- runningElsewhere

You can specify the `state=state` optional query parameter only if you do not specify a queue manager name within the resource URL. That is, you cannot request information about a specific queue manager in a specific state.

Request headers

The following headers must be sent with the request:

`Authorization`
This header must be sent if you are using basic authentication. For more information, see Using HTTP basic authentication with the REST API.

V 9.0.4 The following headers can optionally be sent with the request:

`ibm-mq-rest-gateway-qmgr`
This header specifies the queue manager that is to be used as the gateway queue manager. The gateway queue manager is used to connect to a remote queue manager. For more information, see Remote administration using the REST API.

Request body format

None.

Security requirements

V 9.0.2 The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. For more information about security for the administrative REST API, see IBM MQ Console and REST API security.

When the **`status`** optional query parameter is specified, the ability to issue certain PCF commands is required. If only a subset of the status attributes is to be returned, only the permissions for the corresponding PCF commands are required. The security principal of the caller must be granted the ability to issue the following PCF commands for the specified queue manager:

- ULW IBM MQ Appliance On the IBM MQ Appliance, UNIX, Linux, and Windows:
  - To return the `started`, `channelInitatorState`, `ldapConnectionState`, or `connectionCount` attributes, authority to issue the **`MQCMD_INQUIRE_Q_MGR_STATUS`** PCF command must be granted.
  - To return the `publishSubscribeState` attribute, authority to issue the **`MQCMD_INQUIRE_PUBSUB_STATUS`** PCF command must be granted.
- z/OS On z/OS:
  - To return the `started` attribute, authority to issue the **`MQCMD_INQUIRE_LOG`** PCF command must be granted.
  - To return the `channelInitiatorState` attribute, authority to issue the **`MQCMD_INQUIRE_CHANNEL_INIT`** PCF command must be granted.
  - To return the `connectionCount` attribute, authority to issue the **`MQCMD_INQUIRE_CONNECTION`** PCF command must be granted.
  - To return the `publishSubscribeState` attribute, authority to issue the **`MQCMD_INQUIRE_PUBSUB_STATUS`** PCF command must be granted.

ULW On UNIX, Linux, and Windows, you can grant authority to security principals to use IBM MQ resources by using the **mqsetaut** command. For more information, see mqsetaut.

z/OS On z/OS, see Setting up security on z/OS.

Response status codes

200 Queue manager information retrieved successfully.

400 Invalid data provided.

For example, invalid queue manager specified.

401 Not authenticated.

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. For more information, see "Security requirements" on page 1653.

404 Queue manager does not exist.

500 Server issue or error code from IBM MQ.

Response headers

The following headers are returned with the response:

`Content-Type`
This header is returned with a value of `application/json;charset=utf-8`.

V 9.0.4 **`ibm-mq-rest-gateway-qmgr`**
This header is returned if a remote queue manager is specified in the resource URL. The value of this header is the name of the queue manager that is used as the gateway queue manager.

Response body format

The response is in JSON format in UTF-8 encoding. The response contains an outer JSON object that contains a single JSON array called `qmgr`. Each element in the array is a JSON object that represents information about a queue manager. Each JSON object contains the following attributes:

`name`
String.

The queue manager name.

V 9.0.2 **`state`**
String.

Note: In Version 9.0.1, the **`state`** attribute was known as **`status`**. From Version 9.0.3, the **`status`** object contains status attributes for the queue manager.

This attribute is not returned if the queue manager that is specified in the resource URL is a remote queue manager.

One of the following values:
On all platforms:

- running
- ended

ULW On UNIX, Linux, and Windows:

- endedImmediately

- endedPreemptively
- endedUnexpectedly
- starting
- quiescing
- endingImmediately
- endingPreemptively
- beingDeleted
- stateNotAvailable
- runningAsStandby
- runningElsewhere

The following objects can be included in the JSON object that represents information about a queue. Which objects and attributes are returned depends on the URL that was specified for the request:

V 9.0.3 **status**
Contains attributes that are related to status information for the queue manager.

Note: In Version 9.0.1, the **status** attribute returned information about the running state of the queue. From Version 9.0.2, this attribute is known as **state**.

extended
ULW IBM MQ Appliance These attributes are only available on the IBM MQ Appliance, UNIX, Linux, and Windows.

These attributes are not returned if the queue manager that is specified in the resource URL is a remote queue manager.

Contains extended attributes.

For more information, see "Response body attributes for queue managers" on page 1657.

If an error occurs, the response body contains an error message. For more information, see REST API error handling.

ULW

Examples for UNIX, Linux, and Windows

- The following example gets basic information about all queue managers. The following URL is used with the HTTP GET method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr
  ```

  Version 9.0.3:

  ```
  https://localhost:9443/ibmmq/rest/v1/qmgr
  ```

  The following JSON response is returned:

  ```
  {
      "qmgr": [{
          "name": "QM_T1",
          "state": "endedImmediately"
      }, {
         "name": "RESTQM0",
         "state": "endedUnexpectedly"
      }]
  }
  ```

- The following example gets extended information about the queue manager QM_T1. The following URL is used with the HTTP GET method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM_T1?attributes=extended
  ```

  Version 9.0.3 and earlier:

  ```
  https://localhost:9443/ibmmq/rest/v1/qmgr/QM_T1?attributes=extended
  ```

  The following JSON response is returned:

  ```
  {
      "qmgr": [{
          "extended": {
             "installationName": "Installation1",
             "isDefaultQmgr": false,
             "permitStandby": "notApplicable"
          },
          "name": "QM_T1",
          "state": "endedImmediately"
      }]
  }
  ```

- The following example gets specific information about all queue managers. The following URL is used with the HTTP GET method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr?attributes=extended.permitStandby
  ```

  Version 9.0.3 and earlier:

  ```
  https://localhost:9443/ibmmq/rest/v1/qmgr?attributes=extended.permitStandby
  ```

  The following JSON response is returned:

  ```
  {
      "qmgr": [{
          "extended": {
              "permitStandby": "notApplicable"
          },
          "name": "QM_T1",
          "state": "endedImmediately"
      }, {
          "extended": {
              "permitStandby": "notApplicable"
          },
          "name": "RESTQM0",
          "state": "endedUnexpectedly"
      }]
  }
  ```

- V 9.0.3 The following example gets status for the queue manager QM1. The following URL is used with the HTTP GET method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  http://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1?status=*
  ```

  Version 9.0.3 and earlier:

  ```
  http://localhost:9443/ibmmq/rest/v1/qmgr/QM1?status=*
  ```

  The following JSON response is returned:

  ```
  {
      "qmgr":
      [{
          "name": "QM1",
          "state": "running",
          "status":
  ```

```
        {
            "started":"2016-11-08T11:02:29.000Z",
            "channelInitiatorState":"running",
            "ldapConnectionState":"disconnected",
            "connectionCount":23,
            "publishSubscribeState":"running"
        }
    }]
}
```

z/OS

Examples for z/OS

- The following example gets basic information about all queue managers. The following URL is used with the HTTP GET method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://REST.example.com:9443/ibmmq/rest/v1/admin/qmgr
  ```

  Version 9.0.3 and earlier:

  ```
  https://REST.example.com:9443/ibmmq/rest/v1/qmgr
  ```

  The following JSON response is returned:

  ```
  {
      "qmgr": [{
          "name": "MQ5B",
          "state": "ended"
      }]
  }
  ```

Response body attributes for queue managers: V 9.0.1

When you use the HTTP GET method with the `qmgr` object to request information about queue managers, the following attributes are returned within named JSON objects.

The following objects are available:

- V 9.0.3 "status"
- "extended" on page 1659

For more information about the PCF equivalents to the queue manager REST API parameters and attributes, see "REST API and PCF equivalents for queue managers" on page 1796.

V 9.0.3

status

The `status` object contains status information about queue managers:

`started`
String.

Specifies the date and time at which the queue manager was started.

For more information about the time stamp format that is used to return the date and time, see REST API time stamps.

`channelInitiatorState`
String.

Specifies the current state of the channel initiator.

On all platforms, the value is one of the following values:

- stopped
- running

ULW IBM MQ Appliance On the IBM MQ Appliance, UNIX, Linux, and Windows, the value can also be one of the following values:

- starting
- stopping

z/OS On z/OS, the value can also be one of the following values:

- unknown

  This value indicates that the channel initiator did not return a response to the status request. The channel initiator might be running, but busy. Retry the request after a short time to resolve the issue.

ldapConnectionState

ULW IBM MQ Appliance This attribute is only available on the IBM MQ Appliance, UNIX, Linux, and Windows.

String.

Specifies the current state of the connection to the LDAP server.

The value is one of the following values:

- connected
- error
- disconnected

connectionCount

Integer.

Specifies the current number of connections to the queue manager.

On z/OS, this attribute includes threads that might be disassociated from a connection, together with connections that are in-doubt and connections where external intervention is required.

publishSubscribeState

String.

Specifies the current state of the publish/subscribe engine of the queue manager.

The value is one of the following values:

stopped
Specifies that the publish/subscribe engine, and the queued publish/subscribe interface is not running.

starting
Specifies that the publish/subscribe engine is initializing.

running
Specifies that the publish/subscribe engine, and the queued publish/subscribe interface are running.

compatibility
Specifies that the publish/subscribe engine is running, but that the publish/subscribe interface is not running. Therefore, it is possible to publish or subscribe by using the application programming interface. However, any message that is put to the queues that are monitored by the queued publish/subscribe interface are not acted upon.

error
The publish/subscribe engine failed.

stopping
The publish/subscribe engine is stopping.

extended



This object is only available on the IBM MQ Appliance, UNIX, Linux, and Windows. This object is not returned if the queue manager that is specified in the resource URL is a remote queue manager. The `extended` object contains extended information about queue managers:

isDefaultQmgr
Boolean.

Specifies whether the queue manager is the default queue manager.

The value is `true` if the queue manager is the default queue manager.

permitStandby

This attribute is only available on UNIX, Linux, and Windows.

String.

Specifies the permissible standby state.

The value can be one of the following values:
- permitted
- notPermitted
- notApplicable

installationName
String.

Specifies the name of the installation that the queue manager is associated with.

/admin/qmgr/{qmgrName}/channel: 

You can use the HTTP GET method with the `channel` resource to request information about channels.


From Version 9.0.5, you can use the administrative REST API gateway with this resource URL.

For more information about the PCF equivalents to the channel REST API parameters and attributes, see "REST API and PCF equivalents for channels" on page 1803.

GET: V 9.0.4

Use the HTTP GET method with the `channel` resource to request information about channels.

The information that is returned is similar to the information returned by the "Inquire Channel" on page 1266 and "Inquire Channel Status" on page 1307 PCF commands, and the "DISPLAY CHANNEL" on page 759 and "DISPLAY CHSTATUS" on page 783 MQSC commands.

Note: z/OS On z/OS, the channel initiator must be running before you use the `channel` resource with the HTTP GET method specifying the **`status`** parameter.

Note: The REST API supports only the following channels:

- Channels that have a transport type of TCP.
- Sender, receiver, server, requester, cluster-sender, and cluster-receiver channels.

Other channels are not returned.

Resource URL

`https://host:port/ibmmq/rest/v1/admin/qmgr/{qmgrName}/channel/{channelName}`

`qmgrName`
Specifies the name of the queue manager on which to query the channels.

V 9.0.5 From Version 9.0.5, you can use the administrative REST API gateway with this resource URL. That is, you can specify a remote queue manager as the queue manager name.

The queue manager name is case-sensitive.

If the queue manager name includes a forward slash, a period, or a percent sign, these characters must be URL encoded:

- A forward slash (/) must be encoded as `%2F`.
- A percent sign (%) must be encoded as `%25`.

`channelName`
Optionally specifies the name of a channel to query. This channel must exist on the specified queue manager.

The channel name is case-sensitive.

If the channel name includes a forward slash or a percent sign, these characters must be URL encoded:

- A forward slash, /, must be encoded as `%2F`.
- A percent sign, %, must be encoded as `%25`.

You can use HTTP instead of HTTPS if you enable HTTP connections. For more information about enabling HTTP, see Configuring HTTP and HTTPS ports.

Optional query parameters

attributes={*object*,...|*|*object.attributeName*,...}

object,...
Specifies a comma-separated list of JSON objects that contain related channel configuration attributes to return.

For example, to return all channel configuration attributes that are related to time stamps, specify `timestamps`. To return all channel configuration attributes that are related to compression and to connection management, specify `compression,connectionManagement`.

The `status` objects cannot be specified with this query parameter. Use the **status** query parameter to return these attributes.

You cannot specify the same object more than once. If you request objects that are not valid for a particular channel, the attributes are not returned for that channel. However, if you specify a value for the **type** parameter that is not `all`, and request objects that are not valid for that channel type, an error is returned.

For a full list of objects and associated attributes, see Attributes for channels.

***** Specifies all attributes.

object.attributeName,...
Specifies a comma-separated list of channel configuration attributes to return.

Each attribute must specify the JSON object that contains the attribute, in the form `object.attributeName`. For example, to return the `keepAliveInterval` attribute, which is contained in the `connectionManagement` object, specify `connectionManagement.keepAliveInterval`.

Attributes can be nested inside multiple JSON objects, such as `exits.message.name`, which is an attribute inside a message object inside an exits object.

The keyword `[type]` can be used as a wildcard to include multiple channel-type-specific sections that contain the same attribute. For example, `[type].clusterName` is equivalent to `clusterSender.clusterName,clusterReceiver.clusterName`.

Attributes from the `status` object cannot be specified with this query parameter. Use the **status** query parameter to return these attributes.

You cannot specify the same attribute more than once. If you request attributes that are not valid for a particular channel, the attributes are not returned for that channel. However, if you specify the **type** parameter and request attributes that are not valid for that channel type, an error is returned.

For a full list of attributes and associated objects, see Attributes for channels.

status={*|currentStatus|savedStatus|currentStatus.*attributeName*,savedStatus.*attributeName*,...}

***** Specifies that all savedStatus and currentStatus attributes are returned.

currentStatus
Specifies that all currentStatus attributes are returned.

savedStatus
Specifies that all savedStatus attributes are returned.

currentStatus.*attributeName*,savedStatus.*attributeName*,...
Specifies a comma-separated list of current status and saved status attributes to return.

For example, to return the `state` attribute, specify `currentStatus.state`.

For a full list of status attributes, see Current status attributes for channels and Saved status attributes for channels.

filter=*filterValue*

This query parameter cannot be used if you specify a channel name in the resource URL.

Specifies a filter for the channel definitions that are returned.

You can specify only one filter. If you filter on a status attribute, you must specify the corresponding **status** query parameter.

filterValue has the following format:

attribute:*operator*:*value*

where:

attribute

Specifies one of the applicable attributes. For a full list of attributes, see Attributes for channels. The following attributes cannot be specified:

- z/OS queueSharingGroup.disposition
- [type].connection.port
- connectionManagement.localAddress.port
- connectionManagement.localAddress.portRange
- currentStatus.general.connection.port
- currentStatus.connectionManagement.localAddress.port

The keyword [type] can be used as a wildcard to include multiple channel-type-specific sections that contain the same attribute, such as sender.connection and clusterReceiver.connection.

To filter on any attributes that are time stamps, the filter can specify any portion of the time stamp, with a trailing asterisk, *. The format of a time stamp is, YYYY-MM-DDThh:mm:ss. For example, you can specify 2001-11-1* to filter on dates in the range 2001-11-10 to 2001-11-19, or 2001-11-12T14:* to filter any minute in the specified hour of the specified day.

Valid values for the YYYY section of the date are in the range 1900 - 9999.

The time stamp is a string. Therefore, only the equalTo and notEqualTo operators can be used with the time stamp.

operator

Specifies one of the following operators:

lessThan

Use this operator only with integer attributes.

greaterThan

Use this operator only with integer attributes.

equalTo

Use this operator with any attribute except string array attributes and integer array attributes.

notEqualTo

Use this operator with any attribute except string array attributes and integer array attributes.

lessThanOrEqualTo

Use this operator only with integer attributes.

greaterThanOrEqualTo

Use this operator only with integer attributes.

contains

Use this operator only with integer array attributes and string array attributes.

doesNotContain
Use this operator only with integer array attributes and string array attributes.

value
Specifies the constant value to test against the attribute.

The value type is determined by the attribute type.

For string and boolean attributes, you can omit the value field after the colon. For string attributes, omit the value to return channels with no value for the specified attribute. For boolean attributes, omit the value to return any channels that have the specified attribute set to false. For example, the following filter returns all channels where the description attribute is not specified:

```
filter=general.description:equalTo:
```

You can use a single asterisk, *, at the end of the value as a wildcard. You cannot use only an asterisk.

If the value includes a space, a forward slash, a percent sign, or an asterisk that is not a wildcard, these characters must be URL encoded:

- A space must be encoded as `%20`
- A plus, +, must be encoded as `%2B`
- A forward slash, /, must be encoded as `%2F`.
- A percent sign, %, must be encoded as `%25`.
- An asterisk, *, must be encoded as `%2A`.

name=*name*
This query parameter cannot be used if you specify a channel name in the resource URL.

Specifies a wildcard channel name to filter on.

The *name* specified must include an asterisk, *, as a wildcard. You can specify one of the following combinations:

***** Specifies that all channels are returned.

prefix*
Specifies that all channels with the specified prefix in the channel name are returned.

***suffix**
Specified that all channels with the specified suffix in the channel name are returned.

prefix*suffix
Specifies that all channels with the specified prefix and the specified suffix in the channel name are returned.

type=type
Specifies the type of channel to return information about.

The value can be one of the following values:

all
Specifies that information about all channels is returned.

sender
Specifies that information about sender channels is returned.

receiver
Specifies that information about receiver channels is returned.

server
Specifies that information about server channels is returned.

requester
Specifies that information about requester channels is returned.

clusterSender
Specifies that information about cluster sender channels is returned.

clusterReceiver
Specifies that information about cluster receiver channels is returned.

The default value is `all`.

queueSharingGroupDisposition=*disposition*

z/OS This parameter is only available on z/OS.

Specifies the disposition of the channels for which information is to be returned.

The value can be one of the following values:

live
Return channels defined with `qmgr` or `copy` disposition.

all
Return channels defined with `qmgr`, `copy` or `group` disposition.

copy
Return channels defined with `copy` disposition.

group
Return channels defined with `group` disposition.

private
Return channels defined with `copy` or `qmgr` disposition.

qmgr
Return channels defined with `qmgr` disposition.

The default value is `live`.

Request headers

The following headers must be sent with the request:

Authorization
This header must be sent if you are using basic authentication. For more information, see Using HTTP basic authentication with the REST API.

Request body format

None.

Security requirements

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. For more information about security for the administrative REST API, see IBM MQ Console and REST API security.

The security principal of the caller must be granted the ability to issue the following PCF commands for the specified queue manager:

- If the **status** query parameter is not specified:
  - For the channel that is specified by the *{channelName}* portion of the resource URL, or for channels that match the specified query parameters, authority to issue the **MQCMD_INQUIRE_CHANNEL** PCF command must be granted.
- If the **status** query parameter is specified:

- For the channel that is specified by the *{channelName}* portion of the resource URL, or for channels that match the specified query parameters, authority to issue the **MQCMD_INQUIRE_CHANNEL** PCF command must be granted.
- For the channel that is specified by the *{channelName}* portion of the resource URL, or for channels that match the specified query parameters, authority to issue the **MQCMD_INQUIRE_CHSTATUS** PCF command must be granted.

A principal has display authority if the principal can issue one or both of the **MQCMD_INQUIRE_CHANNEL** and **MQCMD_INQUIRE_CHSTATUS** PCF commands. If the principal has display authority for only some of the channels that are specified by the resource URL and query parameters, then the array of channels that is returned from the REST request is limited to those channels that the principal has authority to display. No information is returned about channels that cannot be displayed. If the principal does not have display authority for any of the channels that are specified by the resource URL and query parameters, an HTTP status code of 403 is returned.

Multi On Multiplatforms, if the attribute `currentStatus.monitoring.messagesAvailable` is to be returned, authority to issue the MQCMD_INQUIRE_Q on the transmission queues used by cluster sender channels is required.

ULW On UNIX, Linux, and Windows, you can grant authority to security principals to use IBM MQ resources by using the **mqsetaut** command. For more information, see mqsetaut.

z/OS On z/OS, see Setting up security on z/OS.

Response status codes

200 Channel information retrieved successfully.

400 Invalid data provided.

For example, invalid channel attributes specified.

401 Not authenticated.

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. For more information, see “Security requirements” on page 1664.

403 Not authorized.

The caller is authenticated to the mqweb server and is associated with a valid principal. However, the principal does not have access to all, or a subset of the required IBM MQ resources. For more information about the access that is required, see “Security requirements” on page 1664.

404 Channel does not exist.

500 Server issue or error code from IBM MQ.

503 Queue manager not running.

Response headers

The following headers are returned with the response:

`Content-Type`
This header is returned with a value of `application/json;charset=utf-8`.

Response body format

The response is in JSON format in UTF-8 encoding. The response contains an outer JSON object that contains a single JSON array called `channel`. Each element in the array is a JSON object that represents information about a channel. Each of these JSON objects contains the following attributes:

name
String.

Specifies the name of the channel.

This attribute is always returned.

type
String.

Specifies the type of channel.

The value is one of the following values:

- `sender`
- `receiver`
- `server`
- `requester`
- `clusterSender`
- `clusterReceiver`

This attribute is always returned.

The following objects can be included in the JSON object that represents information about a channel. Which objects and attributes are returned depends on the URL that was specified for the request:

sender
Contains attributes that are related to sender channels.

server
Contains attributes that are related to server channels.

requester
Contains attributes that are related to requester channels.

clusterSender
Contains attributes that are related to cluster sender channels.

clusterReceiver
Contains attributes that are related to cluster receiver channels.

clusterRouting
Contains attributes that are related to the routing of messages in a cluster.

connectionManagement
Contains attributes that are related to connection management including:

- A JSON array of connection objects that are labeled connectionManagement, which contain host and port information
- longRetry and shortRetry objects, containing count and interval attributes

compression
Contains attributes that are related to compression

dataCollection
Contains attributes that are related to monitoring and statistics

exits
Contains exit objects and arrays of exit objects, each containing:

- Exit name attribute
- User data attribute

extended
Contains attributes that are related to extended channel properties, such as data conversion, and sequence numbers.

failedDelivery
Contains attributes that are related to message delivery failure, such as retry options.

general
Contains attributes that are related to general channel properties, such as the description of the channel.

batch
Contains attributes that are related to message batches.

queueSharingGroup
Contains attributes that are related to queue-sharing groups on z/OS.

receiverSecurity
Contains attributes that are related to security for receiving channels.

transmissionSecurity
Contains attributes that are related to transmission security and encryption.

For more information, see "Response body attributes for channels" on page 1670.

If a damaged object is found, and the REST request did not specify a channel name within the resource URL, an extra JSON array that is called `damaged` is returned. This JSON array contains a list of the objects that are damaged, specifying the object names. If the REST request specifies a channel name within the resource URL, but the object is damaged, an error is returned.

If an error occurs, the response body contains an error message. For more information, see REST API error handling.

Examples

- The following example lists all channels on the queue manager `QM1`. The following URL is used with the HTTP GET method:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/channel
  ```

  The following JSON response is returned:

  ```
  {
      "channel":
      [{
          "name": "RECEIVER.CHL",
          "type": "receiver"
      }, {
          "name": "SENDER.CHL",
          "type": "sender",
          "sender": {
              "connection": [{
                  "host":"example.com",
                  "port": "1414"
              }],
              "transmissionQueueName": "XMIT.Q"
          }
      }, {
          "name": "SERVER.CHL",
  ```

```
        "type": "server",
        "server": {
            "transmissionQueueName": "XMIT.Q"
        }
    }, {
        "name": "REQUESTER.CHL",
        "type": "requester",
        "requester": {
            "connection": [{
                "host": "example.com",
                "port": 1414
            }]
        }
    }, {
        "name": "CLUSSDR.CHL",
        "type": "clusterSender",
        "clusterSender": {
            "connection": [{
                "host": "example.com",
                "port": 1414
            }],
            "clusterName": "CUSTER1"
        }
    }, {
        "name": "CLUSRCVR.CHL",
        "type": "clusterReceiver",
        "clusterReceiver": {
            "connection": [{
                "host": "example.com",
                "port": 1414
            }],
            "clusterName": "CUSTER1"
        }
    }]
}
```

- The following example lists all receiver channels on the queue manager QM1, showing their connection retry attempts information. The following URL is used with the HTTP GET method:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QMGR2/channel?type=sender&attributes=connectionManagement.shortRetry,conn
  ```

  The following JSON response is returned:

  ```
  {
      "channel":
      [{
          "name": "SENDER.CHL",
          "type": "sender",
          "connectionManagement": {
              "longRetry": {
                  "count": 999999999,
                  "interval": 1200
              },
              "shortRetry": {
                  "count": 10,
                  "interval": 60
              }
          },
          "sender": {
              "connection": [{
                  "host": "example.com",
                  "port": 1414
              }],
              "transmissionQueueName": "XMIT.Q"
      }, {
          "name": "SYSTEM.DEF.SENDER",
          "type": "sender",
          "connectionManagement": {
  ```

```
            "longRetry": {
                "count": 999999999,
                "interval": 1200
            },
            "shortRetry": {
                "count": 10,
                "interval": 60
            }
        },
        "sender": {
            "connection": [],
            "transmissionQueueName": ""
    }]
}
```

- The following example lists some status attributes for the channel CHL1, on channel manager QM1. The following URL is used with the HTTP GET method:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/channel/CHL1?status=currentStatus.timestamps,currentStatus.batch.c
  ```

  The following JSON response is returned:

  ```
  {
   "channel":
   [{
    "name": "CHL1",
    "type": "sender",
    "currentStatus": [{
        "inDoubt": false,
     "state": "running",
     "batch": {
            "currentMessages": 10
        },
     "timestamps": {
      "lastMessage": "2017-10-02T09:17:42.314Z",
      "started": "1993-12-31T23:59:59.000Z"
     }
    }],
    "savedStatus": [{
        "inDoubt": false,
        "batch": {
            "currentMessages": 5
        }
    }, {
        "inDoubt": false,
        "batch": {
            "currentMessages": 7
        }
    }]
   }]
  }
  ```

- The following example shows how to get all information, including current status and saved status, for the channel CHL2 on queue manager QM1. The following URL is used with the HTTP GET method:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/channel/CHL2?attributes=*&status=*
  ```

- The following example shows how to get all channel configuration and status information for channels that are currently running, for the queue manager QM1. The following URL is used with the HTTP GET method:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/channel?attributes=*&status=*&filter=currentStatus.state:equalTo:r
  ```

Response body attributes for channels: V 9.0.4

When you receive the response body from using the HTTP verb GET with the `channel` object to request information about channels, attributes for the channels are returned within named JSON objects.

The following objects are available:
- "sender"
- "server" on page 1671
- "requester" on page 1671
- "clusterSender" on page 1671
- "clusterReceiver" on page 1672
- "clusterRouting" on page 1673
- "connectionManagement" on page 1673
- "compression" on page 1675
- "dataCollection" on page 1675
- "exits" on page 1676
- "extended" on page 1677
- "failedDelivery" on page 1678
- "general" on page 1679
- "batch" on page 1679
- "queueSharingGroup" on page 1679
- "receiverSecurity" on page 1680
- "transmissionSecurity" on page 1681
- "currentStatus" on page 1681
- "savedStatus" on page 1692

For more information about the PCF equivalents to the queue REST API parameters and attributes, see REST API and PCF equivalents for channels.

Note: The REST API supports only channels that have TCP as their transport type, and are of type sender, receiver, server, requester, cluster-sender, or cluster-receiver. Other channels are not returned.

sender

The `sender` object contains information about sender channels and is returned only for sender channels:

`connection`
An array of JSON objects that can contain the following attributes that define the channel connection:

`host`
String.

Specifies the host that this channel connects to.

`port`
Integer.

Specifies the port that this channel uses on this host.

These attributes are always returned if they are available. If no connection information is available, an empty array is returned. If the connection does not conform to the expected syntax, an array containing a single host attribute having the value of the entire connection is returned.

transmissionQueueName
String.

Specifies the name of the transmission queue in use by this channel.

This attribute is always returned.

server

The `server` object contains information about server channels and is returned only for server channels:

connection
An array of JSON objects that can contain the following attributes that define the channel connection:

host
String.

Specifies the host that this channel connects to.

port
Integer.

Specifies the port that this channel uses on this host.

These attributes are always returned if they are available. If no connection information is available, an empty array is returned. If the connection does not conform to the expected syntax, an array containing a single host attribute having the value of the entire connection is returned.

transmissionQueueName
String.

Specifies the name of the transmission queue in use by this channel.

This attribute is always returned.

requester

The `requester` object contains information about requester channels and is returned only for requester channels:

connection
An array of JSON objects that can contain the following attributes that define the channel connection:

host
String.

Specifies the host that this channel connects to.

port
Integer.

Specifies the port that this channel uses on this host.

If no connection information is available, an empty array is returned.

If the connection does not conform to the expected syntax, an array containing a single host attribute having the value of the entire connection is returned.

clusterSender

The `clusterSender` object contains information about cluster sender channels and is returned only for cluster sender channels:

connection
An array of JSON objects that can contain the following attributes that define the channel connections:

host
String.

Specifies the host that this channel connects to.

port
Integer.

Specifies the port that this channel uses on this host.

These attributes are always returned if they are not empty. If no connection information is available, an empty array is returned.

If the connection does not conform to the expected syntax, an array containing a single host attribute having the value of the entire connection is returned.

clusterName
String.

Specifies the name of the cluster to which the channel belongs.

This attribute is always returned if it is not empty.

clusterNamelist
String.

Specifies a list of clusters to which the channel belongs.

This attribute is always returned if it is not empty.

clusterReceiver

The `clusterReceiver` object contains information about cluster receiver channels and is returned only for cluster receiver channels:

connection
An array of JSON objects that can contain the following attributes that define the channel connections:

host
String.

Specifies the host that this channel connects to.

port
Integer.

Specifies the port that this channel uses on this host.

These attributes are always returned if they are not empty. If no connection information is available, an empty array is returned.

If the connection does not conform to the expected syntax, an array containing a single host attribute having the value of the entire connection is returned.

clusterName
String.

Specifies the name of the cluster to which the channel belongs.

This attribute is always returned if it is not empty.

clusterNamelist
String.

Specifies a list of clusters to which the channel belongs.

This attribute is always returned if it is not empty.

clusterRouting

The `clusterRouting` object contains information about routing within clusters and is returned only for cluster receiver and cluster sender channels:

workloadPriority
Integer.

Specifies the channel priority for cluster workload distribution.

A value of 0 specifies the lowest priority and a value of 9 specifies the highest priority.

workloadRank
Integer.

Specifies the channel rank for cluster workload distribution.

A value of 0 specifies the lowest rank and a value of 9 specifies the highest rank.

workloadWeight
Integer.

Specifies channel weighting for cluster workload distribution.

A value of 1 specifies the lowest weight and a value of 99 specifies the highest weight.

networkPriority
Integer.

Specifies priority for the network connection. If there are multiple paths available, distributed queuing selects the path with the highest priority.

A value of 0 specifies the lowest priority and a value of 9 specifies the highest priority.

connectionManagement

The `connectionManagement` object contains information about connection management:

heartbeatInterval
Integer.

Specifies the time, in seconds, between heartbeat flows that are passed from the sending MCA when there are no messages on the transmission queue. This interval gives the receiving MCA the opportunity to quiesce the channel.

disconnectInterval
Integer.

Specifies the maximum number of seconds that the channel waits for messages to be put on a transmission queue before the channel ends.

A value of zero causes the message channel agent to wait indefinitely.

keepAliveInterval
Integer.

Specifies the value that is passed to the communications stack for KeepAlive timing for the channel.

localAddress
An array of JSON objects that can contain the following attributes that define the local communications address of the channel:

host
String.

Specifies the local IP address or host name.

This value is returned if the local address in the channel definition contains a host name or IP address.

port
Integer.

Specifies the local port number.

This value is returned if the local address in the channel definition contains a port number.

portRange
JSON object that contains a range of local ports:

low
Integer.

Specifies the start of the port range.

high
Integer.

Specifies the end of the port range.

Returned if a port range is specified in the local address in the channel definition.

If no local address information is available, an empty array is returned.

If the local address does not conform to the expected syntax, an array containing a single host attribute having the value of the entire local address is returned.

shortRetry
JSON object.

Specifies the maximum number and interval of attempts that are made to establish a connection to the remote machine before the `longRetry.count` and`longRetry.interval` are used:

count
Integer.

Specifies the maximum number of attempts to connect to the remote machine.

interval
Integer.

Specifies the interval in seconds between attempts to connect to the remote machine.

longRetry
JSON object.

Specifies the maximum number of attempts and interval of attempts that are made to establish a connection to the remote machine after the count by `shortRetry.count` is exhausted:

count
Integer.

Specifies the maximum number of attempts to connect to the remote machine.

interval
Integer.

Specifies the interval in seconds between attempts to connect to the remote machine.

compression

The `compression` object contains attributes that are related to data compression:

header
String array.

Specifies the header data compression techniques that are supported by the channel. The values that are returned are in order of preference.

The value is one of the following values:

none
Specifies that no header data compression is performed.

system
Specifies that header data compression is performed.

message
String array.

Specifies the message data compression techniques that are supported by the channel. The values that are returned are in order of preference.

The value is one of the following values:

none
Specifies that no header data compression is performed.

runLengthEncoding
Specifies that message data compression is performed by using run-length encoding.

zlibFast
Specifies that message data compression is performed by using ZLIB encoding with speed prioritized.

zlibHigh
Specifies that message data compression is performed by using ZLIB encoding with compression prioritized.

any
Specifies that any compression technique that is supported by the queue manager can be used.

This value is only valid for channels of type receiver and requester.

dataCollection

The `dataCollection` object contains attributes that are related to data collection, monitoring, and statistics:

monitoring
String.

Specifies whether online monitoring data is collected, and if so, the rate at which the data is collected.

The value is one of the following values:

off
Specifies that online monitoring data is not collected for the channel.

asQmgr
Specifies that the queue inherits the value from the queue manager `MONCHL` MQSC parameter.

low
Specifies that online monitoring data is collected for the channel if the MONCHL MQSC parameter on the queue manager is not set to none. The rate of data collection is low.

medium
Specifies that online monitoring data is collected for the channel if the MONCHL MQSC parameter on the queue manager is not set to none. The rate of data collection is moderate.

high
Specifies that online monitoring data is collected for the channel if the MONCHL MQSC parameter on the queue manager is not set to none. The rate of data collection is high.

statistics
String.

Specifies whether statistics data is collected for the channel.

The value is one of the following values:

off
Specifies that statistics data is not collected for the channel.

asQmgr
Specifies that the channel inherits the value from the queue manager STATCHL MQSC parameter.

low
Specifies that statistics data is collected for the channel if the STATCHL MQSC parameter on the channel manager is not set to none. The rate of data collection is low.

medium
Specifies that statistics data is collected for the channel if the STATCHL MQSC parameter on the channel manager is not set to none. The rate of data collection is moderate.

high
Specifies that statistics data is collected for the channel if the STATCHL MQSC parameter on the channel manager is not set to none. The rate of data collection is high.

exits

The exits object contains information about channel exits:

message
An array of JSON objects that contain the following attributes that define the channel message exits:

name
String.

Specifies the message exit name.

userData
String.

Specifies the user data that is passed to the message exit.

messageRetry
A JSON object that contains the following attributes that define the channel message retry exit:

name
String.

Specifies the message retry exit name.

userData
String.

Specifies the user data that is passed to the message retry exit.

receive
An array of JSON objects that contain the following attributes that define the channel receive exits:

name
String.

Specifies the receive exit name.

userData
String.

Specifies the user data that is passed to the receive exit.

security
A JSON object that contains the following attributes that define the channel security exit:

name
String.

Specifies the security exit name.

userData
String.

Specifies the user data that is passed to the security exit.

send
An array of JSON objects that contain the following attributes that define the channel send exits:

name
String.

Specifies the send exit name.

userData
String.

Specifies the user data that is passed to the send exit.

extended

The `extended` object contains attributes that are related to extended channel properties, such as data conversion and sequence number settings:

channelAgentType
String.

Specifies the type of the message channel agent program.

The value is one of the following values:

process

thread

messagePropertyControl
String.

Specifies what happens to message properties when the message is about to be sent to a V6 or earlier queue manager, which does not understand the concept of a property descriptor.

The value is one of the following values:

compatible
If the message contains a property with a prefix of mcd., jms., usr. or mqext., all message properties are delivered to the application in an MQRFH2 header. Otherwise, all properties of the

message, except those properties that are contained in the message descriptor (or extension), are discarded and are no longer accessible to the application.

none
All properties of the message, except those properties in the message descriptor (or extension), are removed from the message before the message is sent to the remote queue manager.

all
All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except those properties in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

senderDataConversion
Boolean.

Specifies whether the sender must convert application data.

sequenceNumberWrap
Integer.

Specifies the maximum message sequence number.

When the maximum is reached, sequence numbers wrap to start again at 1.

resetSequenceNumber
Integer.

Specifies the pending reset sequence number.

A nonzero value indicates that a reset channel request is outstanding. The value is in the range 1 - 999999999.

failedDelivery

The `failedDelivery` object contains attributes that are related to channel behavior when delivery of a message fails:

retry
JSON object.

Specifies the maximum number of attempts and the interval of attempts that are made to establish a connection to the remote machine before the `longRetry.count` and `longRetry.interval` are used:

count
Integer.

Specifies the maximum number of attempts to redeliver the message.

interval
Integer.

Specifies the interval, in milliseconds, between attempts to redeliver the message.

This attribute is only returned for channels of type receiver, requester, and clusterReceiver.

useDeadLetterQueue
Boolean.

Specifies whether the dead-letter queue is used when messages cannot be delivered by channels:

false
Specifies that messages that cannot be delivered by a channel are treated as a failure. The channel either discards the message, or the channel ends, in accordance with the `nonPersistentMessageSpeedFast` setting.

true
Specifies that when the DEADQ attribute of a queue manager provides the name of a dead-letter queue, then the dead letter queue is used. Otherwise, the behavior is as for `false`.

general

The `general` object contains attributes that are related to more generic channel properties, such as description:

description
String.

Specifies the description of the channel.

maximumMessageLength
Integer.

Specifies the maximum message length that can be transmitted on the channel. This value is compared with the value for the remote channel, and the actual maximum is the lower of the two values.

batch

The `batch` object contains attributes that are related to batches of messages that are sent through the channel:

preCommitHeartbeat
Integer.

Specifies whether batch heartbeats are used.

The value is the length of the heartbeat in milliseconds.

timeExtend
Integer.

Specifies the approximate time, in milliseconds, that a channel keeps a batch open if fewer than `batch.messageLimit` messages have been transmitted in the current batch.

dataLimit
Integer.

Specifies the limit, in KB, of the amount of data that can be sent through a channel before a sync point is taken.

messageLimit
Integer.

Specifies the maximum number of messages that can be sent through a channel before a sync point is taken.

nonPersistentMessageSpeedFast
Boolean.

Specifies whether fast speed is used to send nonpersistent messages.

Fast speed means that nonpersistent messages on a channel need not wait for a syncpoint before the messages are made available for retrieval.

queueSharingGroup

The `queueSharingGroup` object contains attributes that are related to queue sharing groups on z/OS:

disposition
String.

z/OS This attribute is only available on z/OS.

Specifies the disposition of the channel. That is, where it is defined and how it behaves.

This value is always returned if the queue manager is a member of the queue-sharing group.

The value is one of the following values:

qmgr
Specifies that the channel definition exists on the page set of the queue manager that runs the command.

group
Specifies that the channel definition exists in the shared repository.

copy
Specifies that the channel definition exists on the page set of the queue manager that runs the command, copying its definition from the channel of the same name defined in the shared repository.

defaultChannelDisposition
String.

z/OS This attribute is only available on z/OS.

Specifies the intended disposition of a channel when it is activated or started.

The value is one of the following values:

private
Specifies that the intended use of the object is as a private channel.

fixShared
Specifies that the intended use of the object is as a fixshared channel.

shared
Specifies that the intended use of the object is as a shared channel.

receiverSecurity

The `receiverSecurity` object contains attributes that are related to security for receiving channels:

channelAgentUserId
String.

Specifies the user identifier that is to be used by the message channel agent for authorization to access IBM MQ resources, including authorization to put the message to the destination queue for receiver or requester channels.

If the value is blank, the message channel agent uses its default user identifier.

putAuthority
String.

Specifies which user identifiers are used to establish authority to put messages to the destination queue.

The value is one of the following values:

default
Specifies that the default user identifier is used.

context
Specifies that the user ID from the `UserIdentifier` field of the message descriptor is used.

alternateOrChannelAgent
Specifies that the user ID from the `UserIdentifier` field of the message descriptor is used.

z/OS This value is only supported on z/OS.

onlyChannelAgent
Specifies that the user ID derived from MCAUSER is used.

transmissionSecurity

The `transmissionSecurity` object contains attributes that are related to security for message transmission:

certificateLabel
String.

Specifies which personal certificate in the key repository is sent to the remote peer.

If this attribute is blank, the certificate is determined by the queue manager **CERTLABL** parameter.

cipherSpecification
String.

Specifies the name of the CipherSpec for the channel to use.

requirePartnerCertificate
Boolean.

Specifies whether IBM MQ requires a certificate from the TLS client.

certificatePeerName
String.

Specifies the filter to use to compare with the Distinguished Name of the certificate from the peer queue manager or client at the other end of the channel. A Distinguished Name is the identifier of the TLS certificate.

currentStatus

The `currentStatus` object contains attributes that are related to current status information:

inDoubt
Boolean.

Specifies whether the channel is in doubt.

A sending channel is in doubt only while the sending message channel agent is waiting for an acknowledgment that a batch of sent messages has been successfully received.

state
String.

Specifies the current status of the channel.

The value is one of the following values:

binding
Specifies that the channel is negotiating with the partner.

starting
Specifies that the channel is waiting to become active.

running
Specifies that the channel is transferring or waiting for messages.

paused
Specifies that the channel is paused.

stopping
Specifies that the channel is in process of stopping.

retrying
Specifies that the channel is reattempting to establish connection.

stopped
Specifies that the channel is stopped.

requesting
Specifies that the requester channel is requesting connection.

switching
Specifies that the channel is switching transmission queues.

initializing
Specifies that the channel is initializing.

agent
A JSON object that contains attributes that are related to the message channel agent:

jobName
String.

Specifies the name of the MCA job.

running
Boolean.

Specifies whether the MCA is running or not.

state
String.

Specifies the current action being performed by the MCA.

The value is one of the following values:

runningChannelAutoDefinitionExit
Specifies that the MCA is running a channel auto-definition exit.

compressingData
Specifies that the MCA is compressing or decompressing data.

processingEndOfBatch
Specifies that the MCA is performing end of batch processing.

performingSecurityHandshake
Specifies that the MCA is performing TLS handshaking.

heartbeating
Specifies that the MCA is heartbeating with a partner.

executingMQGET
Specifies that the MCA is performing an MQGET.

executingMQI
Specifies that the MCA is executing an IBM MQ API call, other than an MQPUT or MQGET.

executingMQPUT
Specifies that the MCA is performing an MQPUT.

runningRetryExit
Specifies that the MCA is running a retry exit.

runningMessageExit
Specifies that the MCA is running a message exit.

communicatingWithNameServer
Specifies that the MCA is processing a name server request.

connectingToNetwork
Specifies that the MCA is connecting to the network.

undefined
Specifies that the MCA is in an undefined state.

runningReceiveExit
Specifies that the MCA is running a receive exit.

receivingFromNetwork
Specifies that the MCA is receiving from the network.

resynchingWithPartner
Specifies that the MCA is resynching with a partner.

runningSecurityExit
Specifies that the MCA is running a security exit.

runningSendExit
Specifies that the MCA is running a send exit.

sendingToNetwork
Specifies that the MCA is performing a network send.

serializingAccessToQmgr
Specifies that the MCA is serialized on queue manager access.

userId
Specifies the user ID that is in use by the MCA.

This attribute is only applicable to receiver, requester, and cluster receiver channels.

batch
JSON Object containing attributes that are related to batches of messages:

count
Integer.

Specifies the number of completed batches.

currentMessages
Integer.

Specifies the number of messages that are sent or received in the current batch.

When a sending channel becomes in-doubt, it specifies the number of the messages that are in-doubt.

The number is reset to 0 when the batch is committed.

luwid
JSON object that contains attributes that are related to logical units of work:

current
Hex string.

Specifies the logical unit of work identifier that is associated with the current batch.

For a sending channel, when the channel is in-doubt it is the LUWID of the in-doubt batch.

last
String. This identifier is represented as 2 hexadecimal digits for each byte.

Specifies the logical unit of work identifier that is associated with the last committed batch.

nonPersistentMessageSpeedFast
Boolean.

Specifies whether non-persistent messages are to be sent at fast speed.

sequenceNumber
JSON object that contains attributes that are related to sequence numbers:

current
Integer.

Specifies the message sequence number of the last message sent or received.

When a sending channel becomes in-doubt, it is the message sequence number of the last message in the in-doubt batch.

last
Integer.

Specifies the sequence number of last message in last committed batch.

size
Integer.

Specifies the negotiated batch size.

compression
JSON Object that contains attributes that are related to data compression:

header
JSON object that contains attributes that are related to header data compression:

default
String.

Specifies the default header data compression value that is negotiated for this channel.

The value is one of the following values:

none
Specifies that no header data compression is performed.

system
Specifies that header data compression is performed.

lastMessage
String.

Specifies the header data compression value that was used for the last message sent.

The value is one of the following values:

none
Specifies that no header data compression was performed.

system
Specifies that header data compression was performed.

unavailable
Specifies that no message was sent.

message
JSON object that contains attributes that are related to message data compression:

default
String.

Specifies the default message data compression value that was negotiated for this channel.

The value is one of the following values:

none
Specifies that no message data compression is performed.

runLengthEncoding
Specifies that message data compression is performed by using run-length encoding.

zlibFast
Specifies that message data compression is performed by using ZLIB encoding with speed prioritized.

zlibHigh
Specifies that message data compression is performed by using ZLIB encoding with compression prioritized.

lastMessage
String.

Specifies the message data compression value that was used for the last message sent.

The value is one of the following values:

none
Specifies that no message data compression was performed.

runLengthEncoding
Specifies that message data compression was performed by using run-length encoding.

zlibFast
Specifies that message data compression was performed by using ZLIB encoding with speed prioritized.

zlibHigh
Specifies that message data compression was performed by using ZLIB encoding with compression prioritized.

unavailable
Specifies that no message was sent.

connectionManagement
JSON Object that contains attributes that are related to connection management:

heartbeatInterval
Integer.

Specifies the heartbeat interval in seconds.

keepAliveInterval
Integer.

Specifies the value that is passed to the communications stack for KeepAlive timing for the channel.

z/OS This parameter is only available on the z/OS

localAddress
An array of JSON objects that can contain the following attributes that define the local communications address of the channel:

host
String.

Specifies the IP address or host name that is used for local communications.

port
Integer.

Specifies the port number that is used for local communications.

If no local address information is available, an empty array is returned.

remainingRetries
JSON object that contains attributes that are related to connection retry attempts:

long
Integer.

Specifies the number of long retry attempts remaining.

last
Integer.

Specifies the number of short retry attempts remaining.

This object is applicable only to sender, server, and cluster-sender channels.

extended
JSON object that contains attributes that are related to extended channel status properties:

buffers
JSON object that contains the following attributes that are related to buffers:

received
Integer.

Specifies the number of buffers received.

sent
Integer.

Specifies the number of buffers sent.

bytes
JSON object that contains the following attributes that are related to data transmission:

received
Integer.

Specifies the number of bytes received.

sent
Integer.

Specifies the number of bytes sent.

messageCount
Integer.

Specifies the total number of messages that are sent or received, or the number of MQI calls handled.

general
JSON Object containing more generic attributes that are related to channels:

heartbeatInterval
Integer.

Specifies the heartbeat interval in seconds.

keepAliveInterval
Integer.

Specifies the value that is passed to the communications stack for KeepAlive timing for the channel.

z/OS This parameter is only available on the z/OS

connection
An array of JSON objects that can contain the following attributes that define the remote communications address of the channel:

host
String.

Specifies the remote IP address or host name.

port
Integer.

Specifies the remote port number.

If no connection information is available, an empty array is returned.

If the connection does not conform to the expected syntax, an array containing a single host attribute having the value of the entire connection is returned.

maximumMessageLength
Integer.

Specifies the maximum length of a message.

statistics
String.

Specifies the rate at which statistics data is collected for the channel.

The value is one of the following values:

off
Specifies that no data is collected.

low
Specifies a low rate of data collection.

medium
Specifies a medium rate of data collection.

high
Specifies a high rate of data collection.

stopRequested
Boolean.

Specifies whether a stop request from the user has been received.

transmissionQueueName
String.

Specifies the name of the transmission queue in use by the channel.

monitoring
JSON object that contains more generic attributes that are related to channel monitoring:

messagesInBatch
JSON object that contains information about the number of messages in a batch:

shortSamplePeriod
Specifies the number of messages in a batch, based on recent activity over a short period.

longSamplePeriod
Specifies the number of messages in a batch, based on activity over a long period.

rate
String.

Specifies the rate at which monitoring data is collected for the channel.

The value is one of the following values:

off
Specifies that no data is collected.

low
Specifies a low rate of data collection.

medium
Specifies a medium rate of data collection.

high
Specifies a high rate of data collection.

compressionRate
JSON object that contains information about data compression rates:

shortSamplePeriod
Specifies the compression rate as a percentage, based on recent activity over a short period.

If no measurement is available, a value of -1 is returned.

longSamplePeriod
Specifies the compression rate as a percentage, based on activity over a long period.

If no measurement is available, a value of -1 is returned.

compressionTime
JSON object that contains information about data compression rates:

shortSamplePeriod
Specifies the compression speed as the time in microseconds spent compressing or decompressing each message, based on recent activity over a short period.

If no measurement is available, a value of -1 is returned.

longSamplePeriod
Specifies the compression speed as the time in microseconds spent compressing or decompressing each message, based on activity over a long period.

If no measurement is available, a value of -1 is returned.

exitTime
JSON object that contains information about exit processing speed:

shortSamplePeriod
Specifies the exit processing speed as the time in microseconds spent processing user exits for each message, based on recent activity over a short period.

If no measurement is available, a value of -1 is returned.

longSamplePeriod
Specifies the exit processing speed as the time in microseconds spent processing user exits for each message, based on activity over a long period.

If no measurement is available, a value of -1 is returned.

messagesAvailable
Integer.

Specifies the number of messages currently queued on the transmission queue and available for MQGETs.

networkTime
JSON object that contains information about network performance:

shortSamplePeriod
Specifies the time, in microseconds, to send a request to the remote end of the channel and receive a response, based on recent activity over a short period.

If no measurement is available, a value of -1 is returned.

longSamplePeriod
Specifies the time, in microseconds, to send a request to the remote end of the channel and receive a response, based on activity over a long period.

If no measurement is available, a value of -1 is returned.

transmissionQueueTime
JSON object that contains information about transmission queue delay:

shortSamplePeriod
Specifies the time, in microseconds, that messages remain on the transmission queue before being retrieved, based on recent activity over a short period.

If no measurement is available, a value of -1 is returned.

longSamplePeriod
Specifies the time, in microseconds, that messages remain on the transmission queue before being retrieved, based on activity over a long period.

If no measurement is available, a value of -1 is returned.

This attribute is only applicable to sender, server, and cluster sender channels.

partner
JSON Object that contains attributes that are related to the remote end queue manager:

productIdentifier
String.

Specifies the product identifier for the IBM MQ version that is running at the remote end of the channel.

The value is one of the following values:

MQMM
Queue Manager (non z/OS Platform)

MQMV
Queue Manager on z/OS

MQCC
IBM MQ C client

MQNM
IBM MQ .NET fully managed client

MQJB
IBM MQ Classes for Java

MQJM
IBM MQ Classes for JMS (normal mode)

MQJN
IBM MQ Classes for JMS (migration mode)

MQJU
Common Java interface to the MQI

MQXC
XMS client C/C++ (normal mode)

MQXD
XMS client C/C++ (migration mode)

MQXN
XMS client .NET (normal mode)

MQXM
XMS client .NET (migration mode)

MQXU
IBM MQ .NET XMS client (unmanaged/XA)

MQNU
IBM MQ .NET unmanaged client

qmgrName
String.

Specifies the name of the remote queue manager or queue-sharing group.

version
String.

Specifies the version of IBM MQ running at the remote end of the channel, in the form V.R.M.F.

maximumMessageLength
Integer.

Specifies the maximum length of a message.

queueSharingGroup
JSON Object that contains attributes that are related to the queue-sharing group this channel belongs to:

channelDisposition
String.

z/OS This attribute is only available on z/OS.

Specifies the disposition of the channel. That is, where it is defined and how it behaves.

The value is one of the following values:

qmgr
Specifies that the channel definition exists on the page set of the queue manager that runs the command.

group
Specifies that the channel definition exists in the shared repository.

copy
Specifies that the channel definition exists on the page set of the queue manager that runs the command, copying its definition from the channel of the same name defined in the shared repository.

timestamps
JSON object that contains attributes that are related to date and time information:

started
String.

Specifies the date and time at which the channel was started.

For more information about the time stamp format that is used to return the date and time, see REST API time stamps.

lastMessage
String.

Specifies the date and time at which the last message was sent over the channel.

For more information about the time stamp format that is used to return the date and time, see REST API time stamps.

transmissionSecurity
JSON object that contains attributes that are related to transmission security:

certificateIssuerName
String.

Specifies the full Distinguished Name of the issuer of the remote certificate.

certificateUserId
String.

Specifies the local user ID that is associated with the remote certificate.

keyLastReset
String.

Specifies the date and time of the last successful TLS secret key reset.

For more information about the time stamp format that is used to return the date and time, see REST API time stamps.

keyResetCount
String.

Specifies the number of successful TLS secret key resets since the channel started.

protocol
String.

This parameter is only available on the IBM MQ Appliance, UNIX, Linux, and Windows.

Specifies the security protocol currently in use.

The value is one of the following values:

none
Specifies that no security protocol is in use.

sslV30
Specifies that SSL version 3.0 is in use.

tlsV10
Specifies that TLS version 1.0 is in use.

tlsV12
Specifies that TLS version 1.2 is in use.

shortPeerName
String.

Specifies the Distinguished Name of the peer queue manager or client at the other end of the channel.

savedStatus

The `savedStatus` object contains attributes that are related to saved status information:

`inDoubt`
Boolean.

Specifies whether the channel was in doubt.

A sending channel is only in doubt while the sending message channel agent is waiting for an acknowledgment that a batch of messages, which it has sent, has been successfully received.

`batch`
JSON Object that contains attributes that are related to batches of messages:

`currentMessages`
Integer.

Specifies the number of messages that are sent or received in the current batch or, if the channel was in-doubt, the number of messages that were in-doubt.

In the context of saved status, this number is only meaningful if the channel was in-doubt, but this value is returned regardless.

`luwid`
JSON object that contains attributes that are related to logical units of work:

`current`
String. This identifier is represented as 2 hexadecimal digits for each byte.

Specifies the logical unit of work identifier that is associated with the current batch.

For a sending channel, if the channel was in-doubt, it specifies the LUWID of the in-doubt batch.

In the context of saved status, this number is only meaningful if the channel was in-doubt, but this value is returned regardless.

`last`
Hex string.

Specifies the logical unit of work identifier that is associated with the last committed batch.

`sequenceNumber`
JSON object that contains attributes that are related to sequence numbers:

`current`
Integer.

Specifies the message sequence number of the last message that is sent or received.

When a sending channel is in-doubt, it specifies the sequence number of the last message in the in-doubt batch.

`last`
Integer.

Specifies the sequence number of the last message in the last committed batch.

`general`
JSON Object that contains more generic attributes that are related to channels:

`connection`
An array of JSON objects that can contain the following attributes that define the remote communications address of the channel:

host
String.

Specifies the remote IP address or host name.

port
Integer.

Specifies the remote port number.

If no connection information is available, an empty array is returned.

If the connection does not conform to the expected syntax, an array containing a single host attribute having the value of the entire connection is returned.

transmissionQueueName
String.

Specifies the name of the transmission queue in use by the channel.

queueSharingGroup
JSON Object that contains attributes that are related to the queue-sharing group this channel belonged to:

channelDisposition
String.

z/OS This attribute is only available on z/OS.

Specifies the disposition of the channel. That is, where it was defined and how it behaved.

The value is one of the following values:

qmgr
Specifies that the channel definition existed on the page set of the queue manager that runs the command.

group
Specifies that the channel definition existed in the shared repository.

copy
Specifies that the channel definition existed on the page set of the queue manager that runs the command, copying its definition from the channel of the same name defined in the shared repository.

/admin/qmgr/{qmgrName}/queue: V 9.0.2

You can use the HTTP GET method with the `queue` resource to request information about queues. You can use the HTTP POST method to create queues, the PATCH method to modify queues, and the DELETE method to delete queues.

V 9.0.4 You can use the administrative REST API gateway with this resource URL.

For more information about the PCF equivalents to the queue REST API parameters and attributes, see REST API and PCF equivalents for queues.

POST: V 9.0.2

Use the HTTP POST method with the `queue` resource to create a queue on a specified queue manager.

This REST API command is similar to the **`Create Queue`** PCF command, and the **`DEFINE queues`** MQSC commands.

- Resource URL
- Optional query parameters
- "Request headers" on page 1695
- Request body format
- "Security requirements" on page 1697
- Response status codes
- "Response headers" on page 1698
- Response body format
- Examples

Resource URL

V 9.0.4 Version 9.0.4 and later:

`https://host:port/ibmmq/rest/v1/admin/qmgr/{qmgrName}/queue`

Version 9.0.3 and earlier:

`https://host:port/ibmmq/rest/v1/qmgr/{qmgrName}/queue`

`qmgrName`
Specifies the name of the queue manager on which to create the queue.

V 9.0.4 You can specify a remote queue manager as the **`qmgrName`**. If you specify a remote queue manager, you must configure a gateway queue manager. For more information, see Remote administration using the REST API.

The queue manager name is case-sensitive.

If the queue manager name includes a forward slash, a period, or a percent sign, these characters must be URL encoded:

- A forward slash (/) must be encoded as `%2F`.
- A percent sign (%) must be encoded as `%25`.

V 9.0.1 You can use HTTP instead of HTTPS if you enable HTTP connections. For more information about enabling HTTP, see Configuring HTTP and HTTPS ports.

Optional query parameters

`commandScope=scope`

z/OS This parameter is only available on z/OS.

Specifies how the command is run when the queue manager is a member of a queue-sharing group.

You cannot specify this parameter if the queue manager is not a member of a queue-sharing group.

scope can be one of the following values:

The name of a queue manager
Specifies that the command is run on the queue manager that is named. The queue manager must be active within the same queue-sharing group as the queue manager that is specified in the resource URL.

You cannot specify the queue manager name that is the queue manager that is specified in the resource URL.

If the queue manager name includes a percent sign, %, this character must be URL encoded as `%25`.

`*` Specifies that the command is run on the local queue manager and also passed to every active queue manager in the queue-sharing group.

If this option is used, an `ibm-mq-qmgrs` response header is returned with a comma-separated list of the queue managers that generated a response. For example, the header might look like the following header:

```
ibm-mq-qmgrs: MQ21, MQ22
```

like=*qName*
Specifies an existing queue definition to copy.

z/OS On z/OS, the way that a queue is copied depends on the value that is specified for the **disposition** parameter in the request body:

- If `copy` is specified, the **like** parameter is ignored. The queue to copy is a queue with the name that is specified by the **name** parameter in the request body and with a disposition of `group`.
- If `copy` is not specified, the queue to copy is a queue with the name that is specified by the **like** parameter and a disposition of `qmgr`, `copy`, or `shared`.

noReplace
Specifies that the queue is not replaced if it exists. If this flag is not specified, the queue is replaced.

If a queue is replaced, any messages that are on the existing queue are retained.

The queue is not replaced in the following scenarios:

- The queue is a local queue. **allowedSharedInput** is changed to `false`, and more than one application has the local queue open for input.
- The queue is a local queue. The value of **isTransmissionQueue** is changed, and one or more applications has the local queue open, or if one or more messages are on the queue.
- The queue is a remote queue. The value of **transmissionQueueName** is changed, and an application has a remote queue open that would be affected by this change.
- The queue is a remote queue. The value of **queueName**, **qmgrName**, or **transmissionQueueName** is changed, and one or more applications has a queue open that resolved through this definition as a queue manager alias.

Request headers

The following headers must be sent with the request:

Content-Type
This header must be sent with a value of `application/json;charset=utf-8`.

ibm-mq-rest-csrf-token
This header must be sent with a value that is the content of the `csrfToken` cookie. The content of the `csrfToken` cookie is used to confirm that the credentials that are being used to authenticate the request are being used by the owner of the credentials. That is, the token is used to prevent cross-site request forgery attacks.

The csrfToken cookie is returned after a request is made with an HTTP GET method. You cannot use a cached version of the content of the cookie because the content of the cookie can change. You must use the latest value of the cookie for each request.

V 9.0.5 The preceding information applies to releases up to and including IBM MQ Version 9.0.4. From IBM MQ Version 9.0.5, this header must be set, but the value can be anything, including being blank.
The csrfToken cookie is no longer sent in responses from the REST API in Version 9.0.5 and later.

Authorization
This header must be sent if you are using basic authentication. For more information, see Using HTTP basic authentication with the REST API.

V 9.0.4 The following headers can optionally be sent with the request:

ibm-mq-rest-gateway-qmgr
This header specifies the queue manager that is to be used as the gateway queue manager. The gateway queue manager is used to connect to a remote queue manager. For more information, see Remote administration using the REST API.

Request body format

The request body must be in JSON format in UTF-8 encoding. Within the request body attributes are defined, and named JSON objects are created to specify extra attributes. Any attributes that are not specified use the default value. These default values are as specified for the SYSTEM.DEFAULT queues on the queue manager. For example, a local queue inherits the values that are defined in SYSTEM.DEFAULT.LOCAL.QUEUE.

For example, the following JSON contains some attributes, and then the named JSON objects, events and storage. These named JSON objects define the extra attributes to create a local queue with queue depth high events enabled, and a maximum queue depth of 1000:

```
{
   "name": "queue1",
   "type": "local",
   "events" : {
      "depth" : {
          "highEnabled" : true,
          "highPercentage" : 75
       }
    },
   "storage" : {
      "maximumDepth" : 1000
   }
}
```

For more examples, see examples.

The following attributes can be included in the request body:

name
Required.

String.

Specifies the name of the queue to create.

type
String.

Specifies the type of queue.

The value can be one of the following values:

- `local`
- `alias`
- `model`
- `remote`

The default value is `local`.

The following objects can be included in the request body to specify extra attributes:

`remote`
Contains attributes that are related to remote queues. The attributes in this object are supported only for remote queues.

`alias`
Contains attributes that are related to alias queues. The attributes in this object are supported only for alias queues.

`model`
Contains attributes that are related to model queues. The attributes in this object are supported only for model queues.

`cluster`
Contains attributes that are related to clusters.

`trigger`
Contains attributes that are related to triggering.

`events`
Contains two objects, one for queue depth and one for queue service interval events. Each object contains attributes that are related to the event type.

`applicationDefaults`
Contains attributes that are related to default behavior such as message persistence, message priority, shared input settings, and read ahead settings.

`queueSharingGroup`
Contains attributes that are related to queue-sharing groups on z/OS.

`dataCollection`
Contains attributes that are related to data collection, monitoring, and statistics.

`storage`
Contains attributes that are related to message storage, such as the maximum depth of the queue, and the maximum length of messages that are allowed on the queue.

`general`
Contains attributes that are related to general queue properties, such as whether get or put operations are inhibited, the description of the queue, and transmission queue settings.

`extended`
Contains attributes that are related to extended queue properties, such as backout queue settings, and shared input settings.

For more information, see “Request body attributes for queues” on page 1701.

Security requirements

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. For more information about security for the administrative REST API, see IBM MQ Console and REST API security.

The security principal of the caller must be granted the ability to issue the following PCF commands for the specified queue manager:

- If the **like** optional query parameter is not specified:
  - For the queue that is specified by the **name** attribute in the request body, authority to issue the **MQCMD_CREATE_Q** PCF command must be granted.
  - For the relevant SYSTEM.DEFAULT.*.QUEUE, authority to issue the **MQCMD_INQUIRE_Q** PCF command must be granted.
- If the **like** optional query parameter is specified:
  - For the queue that is specified by the **name** attribute in the request body, authority to issue the **MQCMD_COPY_Q** PCF command must be granted.
  - For the queue that is specified by the **like** optional query parameter, authority to issue the **MQCMD_INQUIRE_Q** PCF command must be granted.

ULW On UNIX, Linux, and Windows, you can grant authority to security principals to use IBM MQ resources by using the **mqsetaut** command. For more information, see mqsetaut.

z/OS On z/OS, see Setting up security on z/OS.

Response status codes

201 Queue created successfully.

400 Invalid data provided.

For example, invalid queue data is specified.

401 Not authenticated.

The caller must be authenticated to the mqweb server and must be a member of one or more of the MQWebAdmin, MQWebAdminRO, or MQWebUser roles. The ibm-mq-rest-csrf-token header must also be specified. For more information, see "Security requirements" on page 1697.

403 Not authorized.

The caller is authenticated to the mqweb server and is associated with a valid principal. However, the principal does not have access to all, or a subset of the required IBM MQ resources. For more information about the access that is required, see "Security requirements" on page 1697.

500 Server issue or error code from IBM MQ.

503 Queue manager not running.

Response headers

The following headers are returned with the response:

location
If the request was successful, this header specifies the URL for the new queue.

If the optional query parameter commandScope=* is used, the URL that is returned is the URL for the local copy of the queue. If the optional query parameter commandScope=*qmgrName* is used, the URL that is returned is a partial URL that does not include information about the host and port.

z/OS **ibm-mq-qmgrs**
On z/OS, if the optional query parameter commandScope=* is used, this header is returned with a comma-separated list of the queue managers that generated a response. For example, the header might look like the following header:

```
ibm-mq-qmgrs: MQ21, MQ22
```

If an error occurs before the command is issued to the queue managers, the response header does not contain the list of queue managers. For example, a request that generates a 200 or 201 status code has the header because the command was successful. A request that generates a 401 (not authenticated) status code does not have the header because the request was rejected. A request that generates a 403 (not authorized) status code has the header because individual queue managers decide whether the command is authorized.

V 9.0.4 **ibm-mq-rest-gateway-qmgr**

This header is returned if a remote queue manager is specified in the resource URL. The value of this header is the name of the queue manager that is used as the gateway queue manager.

Response body format

The response body is empty if the queue is created successfully. If an error occurs, the response body contains an error message. For more information, see REST API error handling.

Examples

- The following example creates a local queue called `localQueue`. The following URL is used with the HTTP POST method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue/
  ```

  Version 9.0.3 and earlier:

  ```
  https://localhost:9443/ibmmq/rest/v1/qmgr/QM1/queue/
  ```

  The following JSON payload is sent:

  ```
  {
     "name": "localQueue"
  }
  ```

- The following example creates a remote queue called `remoteQueue`. The following URL is used with the HTTP POST method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue/
  ```

  Version 9.0.3 and earlier:

  ```
  https://localhost:9443/ibmmq/rest/v1/qmgr/QM1/queue/
  ```

  The following JSON payload is sent:

  ```
  {
     "name": "remoteQueue",
     "type": "remote",
     "remote" : {
        "queueName": "localQueue",
        "qmgrName": "QM2"
      }
  }
  ```

- The following example creates an alias queue called `aliasQueue`. The following URL is used with the HTTP POST method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue/
  ```

  Version 9.0.3 and earlier:

  ```
  https://localhost:9443/ibmmq/rest/v1/qmgr/QM1/queue/
  ```

  The following JSON payload is sent:

```
{
   "name": "aliasQueue",
   "type": "alias",
   "alias" : {
      "targetName": "localQueue"
   }
}
```

- The following example creates a model queue called `modelQueue`. The following URL is used with the HTTP POST method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue/
  ```

  Version 9.0.3 and earlier:

  ```
  https://localhost:9443/ibmmq/rest/v1/qmgr/QM1/queue/
  ```

  The following JSON payload is sent:

  ```
  {
     "name": "modelQueue",
     "type": "model",
     "model": {
         "type": "permanentDynamic"
      }
   }
  ```

- The following example creates a clustered remote queue that is called `remoteQueue1`. The following URL is used with the HTTP POST method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue/
  ```

  Version 9.0.3 and earlier:

  ```
  https://localhost:9443/ibmmq/rest/v1/qmgr/QM1/queue/
  ```

  The following JSON payload is sent:

  ```
  {
     "name": "remoteQueue1",
     "type": "remote",
     "remote" : {
        "queueName": "aLocalQueue1",
        "qmgrName" : "QM2",
        "transmissionQueueName": "MY.XMITQ"
     },
     "general" : {
           "description" : "My clustered remote queue"
     },
     "cluster" : {
        "name": "Cluster1",
        "workloadPriority": 9
     }
   }
  ```

- The following example creates a clustered remote queue, `remoteQueue2`, based on another queue, `remoteQueue1`. All the attributes from `remoteQueue1` are used, except for the queue name and the remote queue name. The following URL is used with the HTTP POST method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue/?like=remoteQueue1
  ```

  Version 9.0.3 and earlier:

  ```
  https://localhost:9443/ibmmq/rest/v1/qmgr/QM1/queue/?like=remoteQueue1
  ```

  The following JSON payload is sent:

```
{
   "name": "remoteQueue2",
   "type": "remote",
   "remote": {
      "queueName": "aLocalQueue2"
   }
}
```

Request body attributes for queues: V 9.0.2

When you create the request body for creating or modifying a queue with the administrative REST API, you can specify attributes for the queue within named JSON objects. A number of objects and attributes are available.

The following objects are available:

- "remote"
- "alias" on page 1702
- "model" on page 1702
- "cluster" on page 1703
- "trigger" on page 1704
- "events" on page 1705
- "applicationDefaults" on page 1706
- "queueSharingGroup" on page 1708
- "dataCollection" on page 1710
- "storage" on page 1711
- "general" on page 1712
- "extended" on page 1713

For more information about the PCF equivalents to the queue REST API parameters and attributes, see REST API and PCF equivalents for queues.

remote

Note: The `remote` object and the `qmgrName` attribute are required when you create a remote queue by using the HTTP POST method. You cannot use the `remote` object unless you are creating a remote queue, or updating a remote queue.

The `remote` object can contain the following attributes that relate to remote queues:

`queueName`
String.

Specifies the name of the queue as it is known on the remote queue manager.

If this attribute is omitted, a queue manager alias or reply-to queue alias is created.

`qmgrName`
String.

Specifies the name of the remote queue manager.

Required when you create a queue by using the HTTP POST method, unless you use the **`like`** optional query parameter.

If this remote queue is used as a queue manager alias, this attribute is the name of the queue manager. The value can be the name of the queue manager in the resource URL.

If this remote queue is used as a reply-to queue alias, this attribute is the name of the queue manager that is to be the reply-to queue manager.

transmissionQueueName
String.

Specifies the name of the transmission queue that is to be used for messages that are destined for either a remote queue or for a queue manager alias definition.

This attribute is ignored in the following cases:

- The remote queue is used as a queue manager alias and **qmgrName** attribute is the name of the queue manager in the resource URL.
- The remote queue is used as a reply-to queue alias.

If this attribute is omitted, a local queue with the name that is specified by the **qmgrName** attribute must exist. This queue is used as the transmission queue.

alias

Note: The `alias` object and the `targetName` attribute are required when you create an alias queue by using the HTTP POST method. You cannot use the `alias` object unless you are creating an alias queue, or updating an alias queue.

The `alias` object can contain the following attributes that relate to alias queues:

targetName
String.

Specifies the name of the queue or topic that the alias resolves to.

Required when you create a queue by using the HTTP POST method, unless you use the **like** optional query parameter.

targetType
String.

Specifies the type of object that the alias resolves to.

The value must be one of the following values:

queue
Specifies that the object is a queue.

topic
Specifies that the object is a topic.

The default value is `queue`.

model

Note: The `model` object and the `type` attribute are required when you create a model queue by using the HTTP POST method. You cannot use the `model` object unless you are creating a model queue, or updating a model queue.

The `model` object can contain the following attributes that relate to model queues:

type
String.

Specifies the model queue definition type.

The value must be one of the following values:

permanentDynamic
Specifies that the queue is a dynamically defined permanent queue.

sharedDynamic

z/OS This attribute is only available on z/OS.

Specifies that the queue is a dynamically defined shared queue.

temporaryDynamic
Specifies that the queue is a dynamically defined temporary queue.

The default value is `temporaryDynamic`.

cluster

The `cluster` object can contain the following attributes that relate to clusters:

name
String.

Specifies the name of the cluster that the queue belongs to.

Specify either the **name** or **namelist** cluster attributes. You cannot specify both attributes.

namelist
String.

Specifies the namelist that lists the clusters that the queue belongs to.

Specify either the **name** or **namelist** cluster attributes. You cannot specify both attributes.

transmissionQueueForChannelName
String.

Specifies the generic name of the cluster-sender channels that use the queue as a transmission queue. The attribute specifies which cluster-sender channels send messages to a cluster-receiver channel from the cluster transmission queue.

You can also set this attribute to a cluster-sender channel manually. Messages that are destined for the queue manager that is connected by the cluster-sender channel are stored in the transmission queue that identifies the cluster-sender channel. The messages are not stored in the default cluster transmission queue.

If you set the **transmissionQueueForChannelName** attribute to blanks, the channel switches to the default cluster transmission queue when the channel restarts. The default cluster transmission queue is `SYSTEM.CLUSTER.TRANSMIT.QUEUE` if the queue manager **DefClusterXmitQueueType** attribute is set to `SCTQ`. A specific cluster transmission queue, `SYSTEM.CLUSTER.TRANSMIT.`*`ChannelName`*, is used for each cluster-sender channel if the queue manager **DefClusterXmitQueueType** attribute is set to `CHANNEL`.

By specifying asterisks, *, in **transmissionQueueForChannelName**, you can associate a transmission queue with a set of cluster-send channels. The asterisks can be at the beginning, end, or any number of places in the middle of the channel name string.

workloadPriority
Integer.

Specifies the priority of the queue in cluster workload management.

The value must be in the range 0 - 9, where 0 is the lowest priority and 9 is the highest.

workloadRank
Integer.

Specifies the rank of the queue in cluster workload management.

The value must be in the range 0 - 9, where 0 is the lowest priority and 9 is the highest.

workloadQueueUse
String.

Specifies whether remote and local instances of the clustered queues are to be used in cluster workload distribution.

The value must be one of the following values:

asQmgr
Use the value that is defined on the queue manager.

any
Use remote and local instances of the queues.

local
Use only local instances of the queues.

trigger

The `trigger` object can contain the following attributes that relate to triggering:

data
String.

Specifies the user data that is included in the trigger message. This data is made available to the monitoring application that processes the initiation queue and to the application that is started by the monitor.

depth
Integer.

Specifies the number of messages that initiates a trigger message to the initiation queue.

The value must be in the range 1 - 999,999,999.

This attribute is required when **type** is set to `depth`.

enabled
Boolean.

Specifies whether trigger messages are written to the initiation queue.

If the value is set to `true`, trigger messages are written to the initiation queue.

initiationQueueName
String.

Specifies the local queue for trigger messages that relate to the queue. The queues must be on the same queue manager.

messagePriority
Integer.

Specifies the minimum priority that a message must have before it can cause, or be counted for, a trigger event.

The value must be in the range 0 - 9.

processName
String.

Specifies the local name of the IBM MQ process that identifies the application to be started when a trigger event occurs.

If the queue is a transmission queue, the process definition contains the name of the channel to be started.

type
String.

Specifies the condition that initiates a trigger event. When the condition is true, a trigger message is sent to the initiation queue.

The value must be one of the following values:

none
Send no trigger messages.

every
Send a trigger message for every message that arrives on the queue.

first
Send a trigger message when the queue depth goes from 0 to 1.

depth
Send a trigger message when the queue depth exceeds the value of the **depth**attribute.

events

The `events` object can contain the following objects and attributes that relate to queue depth and queue service interval events:

depth
JSON object.

A JSON object that can contain the following attributes that related to queue depth events:

fullEnabled
Boolean.

Specifies whether queue full events are generated.

A queue full event indicates that no more messages can be put on a queue because the queue is full. That is, the queue depth reached the maximum queue depth, as specified by the **maximumDepth** attribute in the `storage` object.

If the value is set to `true`, queue full events are enabled.

highEnabled
Boolean.

Specifies whether queue depth high events are generated.

A queue depth high event indicates that the number of messages on the queue is greater than or equal to the queue depth high limit, **highPercentage**.

If the value is set to `true`, queue depth high events are enabled.

highPercentage
Integer.

Specifies the threshold against which the queue depth is compared to generate a queue depth high event.

This value is expressed as a percentage of the maximum queue depth, as specified by the **maximumDepth** attribute in the `storage` object. The value must be a value in the range 0 - 100.

lowEnabled
Boolean.

Specifies whether queue depth low events are generated.

A queue depth low event indicates that the number of messages on the queue is less than or equal to the queue depth low limit, **lowPercentage**.

If the value is set to `true`, queue depth low events are enabled.

lowPercentage
Integer.

Specifies the threshold against which the queue depth is compared to generate a queue depth low event.

This value is expressed as a percentage of the maximum queue depth, as specified by the **maximumDepth** attribute in the `storage` object. The value must be a value in the range 0 - 100.

serviceInterval
JSON object.

A JSON object that can contain the following attributes that are related to queue service interval events:

duration
Integer.

Specifies the service interval duration that is used for comparison to generate queue service interval high and queue service interval OK events.

The value must be a value in the range 0 - 999,999,999 milliseconds.

highEnabled
Boolean.

Specifies whether queue service interval high events are generated.

A queue service interval high event is generated when a check indicates that no messages were put to, or retrieved from, the queue for at least the amount of time specified by the **duration** attribute.

If the value is set to `true`, queue service interval high events are enabled.

If you set the **highEnabled** attribute to `false`, you must also specify a value for the **okEnabled** attribute. You cannot set both the **highEnabled** attribute and the **okEnabled** attribute to true at the same time.

okEnabled
Boolean.

Specifies whether queue service interval OK events are generated.

A queue service interval OK event is generated when a check indicates that a message was retrieved from the queue within the amount of time that is specified by the **duration** attribute.

If the value is set to `true`, queue service interval OK events are enabled.

If you set the **okEnabled** attribute to `false`, you must also specify a value for **highEnabled**. You cannot set both the **highEnabled** attribute and the **okEnabled** attribute to `true` at the same time.

applicationDefaults

The `applicationDefaults` object can contain the following attributes that relate to default behavior such as message persistence:

clusterBind
String.

Specifies the binding to be used when `MQOO_BIND_AS_Q_DEF` is specified on the MQOPEN call.

The value must be one of the following values:

onOpen
Specifies that the binding is fixed by the MQOPEN call.

notFixed
Specifies that the binding is not fixed.

onGroup
Specifies that the application can request that a group of messages is allocated to the same destination instance.

messagePersistence
String.

Specifies the default for message persistence on the queue. Message persistence determines whether messages are preserved across restarts of the queue manager.

The value must be one of the following values:

persistent
Specifies that the messages on the queue are persistent, and are preserved when the queue manager restarts.

nonPersistent
Specifies that the messages on the queue are not persistent, and are lost when the queue manager restarts.

messagePriority
Integer.

Specifies the default priority of messages that are put on the queue.

The value must be in the range 0 - 9, where 0 represents the lowest priority, and 9 represents the highest priority.

messagePropertyControl
String.

Specifies how message properties are handled when messages are retrieved from queues when MQGMO_PROPERTIES_AS_Q_DEF is specified on the MQGET call.

This attribute is applicable to local, alias, and model queues.

The value must be one of the following values:

all
Specifies that all properties of the message are included when the message is sent to the remote queue manager. The properties, except those properties in the message descriptor or extension, are place in one of more MQRFH2 headers in the message data.

compatible
Specifies that if the message contains a property with the prefix mcd., jms., usr., or mqext., all message properties are delivered to the application in an MQRFH2 header. Otherwise, all properties, except those properties in the message descriptor or extension, are discarded and are no longer accessible.

force
Specifies that properties are always returned in the message data in an MQRFH2 header regardless of whether the application specifies a message handle. A valid message handle that is included in the MsgHandle field of the MQGMO structure on the MQGET call is ignored. Properties of the message are not accessible by using the message handle.

none
Specifies that all properties of the message are removed from the message before the message is sent to the remote queue manager. Properties in the message descriptor, or extension, are not removed.

version6Compatible
Any application MQRFH2 header is received as it was sent. Any properties set by using MQSETMP

must be retrieved by using MQINQMP. They are not added to the MQRFH2 created by the application. Properties that were set in the MQRFH2 header by the sending application cannot be retrieved by using MQINQMP.

putResponse
String.

Specifies the type of response that is to be used for put operations to the queue when an application specifies MQPMO_RESPONSE_AS_Q_DEF.

The value must be one of the following values:

synchronous
The put operation is run synchronously, returning a response.

asynchronous
The put operation is run asynchronously, returning a subset of MQMD fields.

readAhead
String.

Specifies the default read-ahead behavior for non-persistent messages that are delivered to the client.

The value must be one of the following values:

no Specifies that non-persistent messages are not read ahead unless the client application is configured to request read ahead.

yes
Specifies that non-persistent messages are sent ahead to the client before an application requests them. Non-persistent messages can be lost if the client ends abnormally or if the client does not consume all the messages that it is sent.

disabled
Specifies that non-persistent messages are not read ahead, regardless of whether read ahead is requested by the client application.

sharedInput
Boolean.

Specifies the default share option for applications that open this queue for input.

If the value is set to true, queues are enabled to get messages with shared access.

queueSharingGroup

z/OS The queueSharingGroup object can contain the following attributes that relate to queue-sharing groups:

z/OS

disposition
String.

z/OS This attribute is only available on z/OS.

Specifies where the queue is defined and how it behaves. That is, it specifies the disposition of the queue.

The value must be one of the following values:

copy
Specifies that the queue definition exists on the page set of the queue manager that runs the command. The group object of the same name as the **name** attribute is used to create the queue.

For local queues, messages are stored on the page sets of each queue manager and are available only through that queue manager.

group
Specifies that the queue definition exists in the shared repository.

This value is allowed only in a shared queue manager environment.

If the creation is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group. The command attempts to make or refresh local copies on page set zero:

```
DEFINE queue(q-name) REPLACE QSGDISP(COPY)
```

The creation of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

qmgr
Specifies that the queue definition exists on the page set of the queue manager that runs the command.

For local queues, messages are stored on the page sets of each queue manager and are available only through that queue manager.

shared
This value is only valid for local queues.

Specifies that the queue exists in the shared repository.

Messages are stored in the coupling facility and are available to any queue manager in the queue-sharing group. You can specify `shared` only if the following things are true:

- The value of **structureName** is not blank.
- The value of **indexType** is not `messageToken`.
- The queue is not `SYSTEM.CHANNEL.INITQ` or `SYSTEM.COMMAND.INPUT`.

The default value is `qmgr`.

structureName
String.

z/OS This attribute is only available on z/OS.

Specifies the name of the coupling facility structure where you want to store messages when you used shared queues.

The value cannot have more than 12 characters, it must start with an uppercase letter (A - Z), and can include only the characters A - Z and 0 - 9.

The name of the queue-sharing group to which the queue manager is connected is prefixed to the name you supply. The name of the queue-sharing group is always 4 characters, padded with the at sign, @, if necessary. For example, if you use a queue-sharing group that is named `NY03` and you supply the name `PRODUCT7`, the resultant coupling facility structure name is `NY03PRODUCT7`. Note the administrative structure for the queue-sharing group (in this case `NY03CSQ_ADMIN`) cannot be used for storing messages.

For local and model queues, the following rules apply. The rules apply if you create a queue without specifying the **noReplace** optional query parameter, or if you change the queue:

- On a local queue with a **disposition** value of `shared`, **structureName** cannot change. If you need to change the **structureName** or the **disposition**, you must delete and redefine the queue. To preserve any of the messages on the queue, you must offload the messages before you delete the queue. Reload the messages after you redefine the queue, or move the messages to another queue.
- On a model queue with a **definitionType** value of `sharedDynamic`, the **structureName** cannot be blank.

For local and model queues, when you create a queue with the **noReplace** optional query parameter, the following rules apply:

- On a local queue with a **disposition** value of shared, or a model queue with a **definitionType** value of sharedDynamic, the **structureName** cannot be blank.

dataCollection

The dataCollection object can contain the following attributes that relate to the collection of data, monitoring, and statistics:

accounting
String.

Specifies whether accounting data is collected for the queue.

The value must be one of the following values:

asQmgr
Specifies that the queue inherits the value from the queue manager MQSC parameter ACCTQ.

off
Specifies that accounting data is not collected for the queue.

on Specifies that accounting data is collected for the queue if the ACCTQ MQSC parameter on the queue manager is not set to none.

monitoring
String.

Specifies whether online monitoring data is to be collected, and if so, the rate at which the data is collected.

The value must be one of the following values:

off
Specifies that online monitoring data is not collected for the queue.

asQmgr
Specifies that the queue inherits the value from the queue manager MQSC parameter MONQ.

low
Specifies that online monitoring data is collected for the queue if the MONQ MQSC parameter on the queue manager is not set to none. The rate of data collection is low.

medium
Specifies that online monitoring data is collected for the queue if the MONQ MQSC parameter on the queue manager is not set to none. The rate of data collection is moderate.

high
Specifies that online monitoring data is collected for the queue if the MONQ MQSC parameter on the queue manager is not set to none. The rate of data collection is high.

statistics
ULW IBM MQ Appliance This attribute is only available on the IBM MQ Appliance, UNIX, Linux, and Windows.

String.

Specifies whether statistics data is to be collected for the queue.

The value must be one of the following values:

asQmgr
Specifies that the queue inherits the value from the queue manager STATQ MQSC parameter.

off
Specifies that statistics data is not collected for the queue.

on Specifies that statistics data is collected for the queue if the STATQ MQSC parameter on the queue manager is not set to none.

storage

The storage object can contain the following attributes that relate to message storage:

indexType

z/OS This attribute is only available on z/OS.

String.

Specifies the type of index that is maintained by the queue manager to expedite MQGET operations on the queue. For shared queues, the type of index determines what type of MQGET calls can be used.

The value must be one of the following values:

none
Specifies that there is no index. Messages are retrieved sequentially.

correlationId
Specifies that the queue is indexed by using correlation identifiers.

groupId
Specifies that the queue is indexed by using group identifiers.

messageId
Specifies that the queue is indexed by using message identifiers.

messageToken
Specifies that the queue is indexed by using message tokens.

The default value is none.

maximumDepth
Integer.

Specifies the maximum number of messages that are allowed on the queue.

The value must be in the range 0 - 999,999,999.

maximumMessageLength
Integer.

Specifies the maximum message length that is allowed for messages on the queue.

Do not set a value that is greater than the **maximumMessageLength** attribute for the queue manager.

The value must be in the range 0 - 104,857,600 bytes.

messageDeliverySequence
String.

Specifies whether messages are delivered in priority order or by sequence.

The value must be one of the following values:

priority
Specifies that messages are returned in priority order.

fifo
Specifies that messages are returned in first in, first out order.

nonPersistentMessageClass

ULW IBM MQ Appliance This attribute is only available on the IBM MQ Appliance, UNIX, Linux, and Windows.

String.

This attribute is valid only on local and model queues.

Specifies the level of reliability to be assigned to non-persistent messages that are put to the queue.

The value must be one of the following values:

normal
Specifies that non-persistent messages persist for the lifetime of the queue manager session. They are discarded if the queue manager restarts.

high
Specifies that the queue manager attempts to retain non-persistent messages for the lifetime of the queue. Non-persistent messages might still be lost if a failure occurs.

storageClass

z/OS This attribute is only available on z/OS.

String.

Specifies the name of the storage class.

general

The `general` object can contain the following attributes that relate to general queue properties:

description
String.

Specifies a description for the queue.

The characters in the description field are converted from UTF-8 into the CCSID of the queue manager. Ensure that you use only the characters that can be converted. Certain characters must be escaped:

- Double quotation marks, ", must be escaped as `\"`
- A backslash, `\`, must be escaped as `\\`
- A forward slash, `/`, must be escaped as `\/`

inhibitGet
Boolean.

Specifies whether get operations are allowed on the queue.

If the value is set to `true`, get operations are not allowed on the queue.

inhibitPut
Boolean.

Specifies whether put operations are allowed on the queue.

If the value is set to `true`, put operations are not allowed on the queue.

isTransmissionQueue
String.

Specifies whether the queue is for normal usage or for transmitting messages to a remote queue manager.

If the value is set to `true`, the queue is a transmission queue for transmitting messages to a remote queue manager.

The `isTransmissionQueue` attribute must not normally be changed while messages are on the queue. The format of messages changes when they are put on a transmission queue.

extended

The `extended` object can contain the following attributes that relate to extended queue properties:

allowSharedInput
Boolean.

Specifies whether multiple instances of applications can open the queue for input.

If the value is set to `true`, multiple instances of applications can open the queue for input.

backoutRequeueQueueName
String.

Specifies the name of the queue to which a message is transferred if it is backed out more times than the value of **backoutThreshold**.

The backout queue does not need to exist when the queue is created, but it must exist when the **backoutThreshold** value is exceeded.

backoutThreshold
Integer.

Specifies the number of times that a message can be backed out before it is transferred to the backout queue that is specified by the **backoutRequeueQueueName** attribute.

If the **backoutThreshold** value is later reduced, messages that are already on the queue that were backed out at least as many times as the new value remain on the queue. Those messages are transferred if they are backed out again.

The value must be a value in the range 0 - 999,999,999.

custom
String.

Specifies custom attributes for new features.

This attribute contains the values of attributes, as pairs of attribute name and value, which are separated by at least one space. The attribute name-value pairs have the form `NAME(VALUE)`. Single quotation marks, ', must be escaped with another single quotation mark.

V 9.0.4 **enableMediaImageOperations**

ULW IBM MQ Appliance This attribute is available only on the IBM MQ Appliance, UNIX, Linux, and Windows.

Specifies whether a local or permanent dynamic queue object is recoverable from a media image, if linear logging is being used.

String.

The value must be one of the following values:

yes
Specifies that this queue object is recoverable.

no The `rcdmqimg` and `rcrmqobj` commands are not permitted for these objects. If automatic media images are enabled, the media images are not written for these objects.

asQmgr
Specifies that the queue inherits the value from the queue manager `ImageRecoverQueue` attribute.

This is the default value for this attribute.

hardenGetBackout

z/OS This attribute is only available on z/OS.

Boolean.

Specifies whether the count of backed out messages is saved across restarts of the queue manager.

If the value is set to `true`, the backout count is saved across restarts.

supportDistributionLists

ULW IBM MQ Appliance This attribute is only available on the IBM MQ Appliance, UNIX, Linux, and Windows.

Boolean.

Specifies whether distribution-list messages can be placed on the queue.

If the value is set to `true`, distribution lists can be placed on the queue.

PATCH: V 9.0.2

Use the HTTP PATCH method with the `queue` resource to modify a queue on a specified queue manager.

This REST API command is similar to the **Change Queue** PCF command, and the **ALTER queues** MQSC commands.

- Resource URL
- Optional query parameters
- "Request headers" on page 1716
- Request body format
- "Security requirements" on page 1718
- Response status codes
- "Response headers" on page 1718
- Response body format
- Examples

Resource URL

V 9.0.4 Version 9.0.4 and later:

`https://host:port/ibmmq/rest/v1/admin/qmgr/{qmgrName}/queue/{queueName}`

Version 9.0.3 and earlier:

`https://host:port/ibmmq/rest/v1/qmgr/{qmgrName}/queue/{queueName}`

qmgrName

Specifies the name of the queue manager on which the queue to modify exists.

The queue manager name is case sensitive.

If the queue manager name includes a forward slash, a period, or a percent sign, these characters must be URL encoded:

- A forward slash (/) must be encoded as `%2F`.
- A period (.) must be encoded as `%2E`.
- A percent sign (%) must be encoded as `%25`.

queueName

Specifies the name of the queue to modify.

V 9.0.4 You can specify a remote queue manager as the **qmgrName**. If you specify a remote queue manager, you must configure a gateway queue manager. For more information, see Remote administration using the REST API.

The queue manager name is case-sensitive.

If the queue manager name includes a forward slash, a period, or a percent sign, these characters must be URL encoded:

- A forward slash (/) must be encoded as `%2F`.
- A percent sign (%) must be encoded as `%25`.

V 9.0.1 You can use HTTP instead of HTTPS if you enable HTTP connections. For more information about enabling HTTP, see Configuring HTTP and HTTPS ports.

Optional query parameters

commandScope=*scope*

z/OS This parameter is only available on z/OS.

Specifies how the command is run when the queue manager is a member of a queue-sharing group.

You cannot specify this parameter if the queue manager is not a member of a queue-sharing group.

scope can be one of the following values:

The name of a queue manager

Specifies that the command is run on the queue manager that is named. The queue manager must be active within the same queue-sharing group as the queue manager that is specified in the resource URL.

You cannot specify the queue manager name that is the queue manager that is specified in the resource URL.

If the queue manager name includes a percent sign, %, this character must be URL encoded as `%25`.

***** Specifies that the command is run on the local queue manager and also passed to every active queue manager in the queue-sharing group.

If this option is used, an `ibm-mq-qmgrs` response header is returned with a comma-separated list of the queue managers that generated a response. For example, the header might look like the following header:

```
ibm-mq-qmgrs: MQ21, MQ22
```

force

Specifies that the command is forced to complete, regardless of whether completing affects an open queue.

This parameter is not valid for model queues.

An open queue is affected in the following cases:

- The queue is an alias queue. The **targetName** is modified, and an application has the alias queue open.
- The queue is a local queue. The **allowedSharedInput** attribute is modified, and more than one application has the queue open for input.
- The queue is a local queue. The **isTransmissionQueue** attribute is modified, and messages are on the queue, or applications have the queue open.

- The queue is a remote queue. The **transmissionQueueName** attribute is modified, and an application has a remote queue open that would be affected by this change.
- The queue is remote queue. The **queueName**, **qmgrName**, or **transmissionQueueName** attributes are modified, and one or more applications has a queue open that resolved through this definition as a queue manager alias.

Request headers

The following headers must be sent with the request:

Content-Type
This header must be sent with a value of `application/json;charset=utf-8`.

ibm-mq-rest-csrf-token
This header must be sent with a value that is the content of the `csrfToken` cookie. The content of the `csrfToken` cookie is used to confirm that the credentials that are being used to authenticate the request are being used by the owner of the credentials. That is, the token is used to prevent cross-site request forgery attacks.

The `csrfToken` cookie is returned after a request is made with an HTTP GET method. You cannot use a cached version of the content of the cookie because the content of the cookie can change. You must use the latest value of the cookie for each request.

V 9.0.5 The preceding information applies to releases up to and including IBM MQ Version 9.0.4. From IBM MQ Version 9.0.5, this header must be set, but the value can be anything, including being blank.
The `csrfToken` cookie is no longer sent in responses from the REST API in Version 9.0.5 and later.

Authorization
This header must be sent if you are using basic authentication. For more information, see Using HTTP basic authentication with the REST API.

V 9.0.4 The following headers can optionally be sent with the request:

ibm-mq-rest-gateway-qmgr
This header specifies the queue manager that is to be used as the gateway queue manager. The gateway queue manager is used to connect to a remote queue manager. For more information, see Remote administration using the REST API.

Request body format

The request body must be in JSON format in UTF-8 encoding. Within the request body attributes are specified, and named JSON objects are created to specify extra attributes to modify. Any attributes that are not specified are not changed.

For example, the following JSON contains the attribute **type**, and then the named JSON objects, `events` and `storage`. The named JSON objects define the additional attributes to modify the queue to disable queue depth high events, and change the maximum queue depth to 2000:

```
{
   "type": "local",
   "events" : {
      "serviceInterval" : {
          "highEnabled" : false,
          "okEnabled" : false
       }
    },
   "storage" : {
      "maximumDepth" : 2000
   }
}
```

For more examples, see examples.

The following attributes can be included in the request body:

type
String.

Specifies the type of queue.

The value can be one of the following values:

- `local`
- `alias`
- `model`
- `remote`

The default value is `local`.

The following objects can be included in the request body to specify extra attributes:

remote
Contains attributes that are related to remote queues. The attributes in this object are supported only for remote queues.

alias
Contains attributes that are related to alias queues. The attributes in this object are supported only for alias queues.

model
Contains attributes that are related to model queues. The attributes in this object are supported only for model queues.

cluster
Contains attributes that are related to clusters.

trigger
Contains attributes that are related to triggering.

events
Contains two objects, one for queue depth and one for queue service interval events. Each object contains attributes that are related to the event type.

applicationDefaults
Contains attributes that are related to default behavior such as message persistence, message priority, shared input settings, and read ahead settings.

queueSharingGroup
Contains attributes that are related to queue-sharing groups on z/OS.

dataCollection
Contains attributes that are related to data collection, monitoring, and statistics.

storage
Contains attributes that are related to message storage, such as the maximum depth of the queue, and the maximum length of messages that are allowed on the queue.

general
Contains attributes that are related to general queue properties, such as whether get or put operations are inhibited, the description of the queue, and transmission queue settings.

extended
Contains attributes that are related to extended queue properties, such as backout queue settings, and shared input settings.

For more information, see “Request body attributes for queues” on page 1701.

Security requirements

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. For more information about security for the administrative REST API, see IBM MQ Console and REST API security.

The security principal of the caller must be granted the ability to issue the following PCF commands for the specified queue manager:

- For the queue that is specified by the *{queueName}* portion of the resource URL, authority to issue the **MQCMD_CHANGE_Q** PCF command must be granted.

ULW On UNIX, Linux, and Windows, you can grant authority to security principals to use IBM MQ resources by using the **mqsetaut** command. For more information, see mqsetaut.

z/OS On z/OS, see Setting up security on z/OS.

Response status codes

204 Queue modified successfully.

400 Invalid data provided.

For example, invalid queue data is specified.

401 Not authenticated.

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. The `ibm-mq-rest-csrf-token` header must also be specified.. For more information, see “Security requirements.”

403 Not authorized.

The caller is authenticated to the mqweb server and is associated with a valid principal. However, the principal does not have access to all, or a subset of the required IBM MQ resources. For more information about the access that is required, see “Security requirements.”

404 Queue does not exist.

500 Server issue or error code from IBM MQ.

503 Queue manager not running.

Response headers

The following headers are returned with the response:

z/OS **ibm-mq-qmgrs**

On z/OS, if the optional query parameter commandScope=* is used, this header is returned with a comma-separated list of the queue managers that generated a response. For example, the header might look like the following header:

```
ibm-mq-qmgrs: MQ21, MQ22
```

If an error occurs before the command is issued to the queue managers, the response header does not contain the list of queue managers. For example, a request that generates a 200 or 201 status code has the header because the command was successful. A request that generates a 401 (not authenticated) status code does not have the header because the request was rejected. A request that generates a 403 (not authorized) status code has the header because individual queue managers decide whether the command is authorized.

V 9.0.4 **ibm-mq-rest-gateway-qmgr**
This header is returned if a remote queue manager is specified in the resource URL. The value of this header is the name of the queue manager that is used as the gateway queue manager.

Response body format

The response body is empty if the queue is modified successfully. If an error occurs, the response body contains an error message. For more information, see REST API error handling.

Examples

- The following example modifies an alias queue called `aliasQueue`. The following URL is used with the HTTP PATCH method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue/aliasQueue
  ```

  Version 9.0.3 and earlier:

  ```
  https://localhost:9443/ibmmq/rest/v1/qmgr/QM1/queue/aliasQueue
  ```

  The following JSON payload is sent:

  ```
  {
     "type": "alias",
     "alias": {
        "targetName": "aDifferentLocalQueue"
    }
  }
  ```

GET: V 9.0.2

Use the HTTP GET method with the `queue` resource to request information about queues.

The information that is returned is similar to the information returned by the **Inquire Queue**, and **Inquire Queue Status** PCF commands, and the **DISPLAY QUEUE** and **DISPLAY QSTATUS** MQSC commands.

Note: z/OS On z/OS, the channel initiator must be running before you use the `queue` resource with the HTTP GET method in either of the following situations:

- The **type** optional query parameter is not specified.
- The **type** optional query parameter is specified as either `all` or `cluster`.

Resource URL

V 9.0.4 Version 9.0.4 and later:

`https://`*host*`:`*port*`/ibmmq/rest/v1/admin/qmgr/{`*qmgrName*`}/queue/{`*queueName*`}`

Version 9.0.3 and earlier:

`https://host:port/ibmmq/rest/v1/qmgr/{qmgrName}/queue/{queueName}`

qmgrName
Specifies the name of the queue manager on which to query the queues.

V 9.0.4 You can specify a remote queue manager as the **qmgrName**. If you specify a remote queue manager, you must configure a gateway queue manager. For more information, see Remote administration using the REST API.

The queue manager name is case-sensitive.

If the queue manager name includes a forward slash, a period, or a percent sign, these characters must be URL encoded:

- A forward slash (/) must be encoded as `%2F`.
- A percent sign (%) must be encoded as `%25`.

queueName
Optionally specifies the name of a queue that exists on the queue manager specified.

The queue name is case-sensitive.

If the queue name includes a forward slash or a percent sign, these characters must be URL encoded:

- A forward slash, /, must be encoded as `%2F`.
- A percent sign, %, must be encoded as `%25`.

V 9.0.1 You can use HTTP instead of HTTPS if you enable HTTP connections. For more information about enabling HTTP, see Configuring HTTP and HTTPS ports.

Optional query parameters

attributes={*object*,...|*|*object.attributeName*,...}

object,...
Specifies a comma-separated list of JSON objects that contain related queue configuration attributes to return.

For example, to return all queue configuration attributes that are related to time stamps, specify `timestamps`. To return all queue configuration attributes that are related to storage and to data collection, specify `storage,dataCollection`.

The `status` and `applicationHandle` objects cannot be specified with this query parameter. Use the **status** and **applicationHandle** query parameters to return these attributes.

You cannot specify the same object more than once. If you request objects that are not valid for a particular queue, the attributes are not returned for that queue. However, if you specify a value for the **type** parameter that is not `all`, and request objects that are not valid for that queue type, an error is returned.

For a full list of objects and associated attributes, see Attributes for queues.

***** Specifies all attributes.

object.attributeName,...
Specifies a comma-separated list of queue configuration attributes to return.

Each attribute must specify the JSON object that contains the attribute, in the form `object.attributeName`. For example, to return the `maximumDepth` attribute, which is contained in the storage object, specify `storage.maximumDepth`.

Attributes from the `status` and `applicationHandle` objects cannot be specified with this query parameter. Use the **status** and **applicationHandle** query parameters to return these attributes.

You cannot specify the same attribute more than once. If you request attributes that are not valid for a particular queue, the attributes are not returned for that queue. However, if you specify the **type** parameter and request attributes that are not valid for that queue type, an error is returned.

For a full list of attributes and associated objects, see Attributes for queues.

status={status|*|status.*attributeName*,...}

status
Specifies that all status attributes are returned.

***** Specifies all attributes. This parameter is equivalent to **status**.

status.*attributeName*,...
Specifies a comma-separated list of status attributes to return.

For example, to return the `currentDepth` attribute, specify `status.currentDepth`.

For a full list of status attributes, see Status attributes for queues.

If you specify the **status** optional query parameter, you can specify the **type** parameter only with the `all` or `local` values. You cannot specify the **queueSharingGroupDisposition** parameter with the `group` value.

applicationHandle={applicationHandle|*|applicationHandle.*attributeName*,...}

applicationHandle
Specifies that all application handle attributes are returned.

***** Specifies all attributes. This parameter is equivalent to **applicationHandle**.

applicationHandle.*attributeName*,...
Specifies a comma-separated list of application handle attributes to return.

For example, to return the `handleState` attribute, specify `applicationHandle.handleState`.

For a full list of application handle attributes, see Application handle attributes for queues.

If you specify the **applicationHandle** optional query parameter, you can specify the **type** parameter only with the `all` or `local` values. You cannot specify the **queueSharingGroupDisposition** parameter with the `group` value.

commandScope=*scope*

z/OS This parameter is only available on z/OS.

Specifies how the command is run when the queue manager is a member of a queue-sharing group.

You cannot specify this parameter if the queue manager is not a member of a queue-sharing group.

scope can be one of the following values:

The name of a queue manager
Specifies that the command is run on the queue manager that is named. The queue manager must be active within the same queue-sharing group as the queue manager that is specified in the resource URL.

You cannot specify the queue manager name that is the queue manager that is specified in the resource URL.

If the queue manager name includes a percent sign, %, this character must be URL encoded as `%25`.

***** Specifies that the command is run on the local queue manager and also passed to every active queue manager in the queue-sharing group.

If this option is used, an `ibm-mq-qmgrs` response header is returned with a comma-separated list of the queue managers that generated a response. For example, the header might look like the following header:

```
ibm-mq-qmgrs: MQ21, MQ22
```

filter=*filterValue*

This query parameter cannot be used if you specify a queue name in the resource URL.

Specifies a filter for the queue definitions that are returned.

You can specify only one filter. If you filter on an application handle attribute, you must specify the **applicationHandle** query parameter. If you filter on a status attribute, you must specify the **status** query parameter.

filterValue has the following format:

attribute:*operator*:*value*

where:

attribute

Specifies one of the applicable attributes. For a full list of attributes, see Attributes for queues. The following attributes cannot be specified:

- `status.onQueueTime`
- `status.tpipeName`
- `applicationHandle.qmgrTransactionId`
- `applicationHandle.unitOfWorkId`
- `applicationHandle.openOptions`

To filter on any attributes that are time stamps, the filter can specify any portion of the time stamp, with a trailing asterisk, *. The format of a time stamp is, `YYYY-MM-DDThh:mm:ss`. For example, you can specify `2001-11-1*` to filter on dates in the range `2001-11-10` to `2001-11-19`, or `2001-11-12T14:*` to filter any minute in the specified hour of the specified day.

Valid values for the `YYYY` section of the date are in the range 1900 - 9999.

The time stamp is a string. Therefore, only the `equalTo` and `notEqualTo` operators can be used with the time stamp.

Note: z/OS If either the **filter** query parameter, or the **name** query parameter with a wildcard, are used with the **commandScope=*** query parameter, and there are no matching queues on at least one of the active queue managers in the queue-sharing group, then an error message is returned.

operator

Specifies one of the following operators:

lessThan

Use this operator only with integer attributes.

greaterThan

Use this operator only with integer attributes.

equalTo

Use this operator with any attribute.

notEqualTo

Use this operator with any attribute.

lessThanOrEqualTo

Use this operator only with integer attributes.

greaterThanOrEqualTo
Use this operator only with integer attributes.

value
Specifies the constant value to test against the attribute.

The value type is determined by the attribute type.

For string and boolean attributes, you can omit the value field after the colon. For string attributes, omit the value to return queues with no value for the specified attribute. For boolean attributes, omit the value to return any queues that have the specified attribute set to false. For example, the following filter returns all queues where the description attribute is not specified:

```
filter=general.description:equalTo:
```

You can use a single asterisk, *, at the end of the value as a wildcard. You cannot use only an asterisk.

If the value includes a space, a forward slash, a percent sign, or an asterisk that is not a wildcard, these characters must be URL encoded:

- A space must be encoded as `%20`
- A forward slash, `/`, must be encoded as `%2F`.
- A percent sign, %, must be encoded as `%25`.
- An asterisk, *, must be encoded as `%2A`.

z/OS If the filter query parameter is used with the **commandScope=*** query parameter, and there are no matching values on at least one of the active queue managers in the queue-sharing group, an error message is returned.

name=*name*
This query parameter cannot be used if you specify a queue name in the resource URL.

Specifies a wildcard queue name to filter on.

The *name* specified must include an asterisk, *, as a wildcard. You can specify one of the following combinations:

***** Specifies that all queues are returned.

prefix*
Specifies that all queues with the specified prefix in the queue name are returned.

***suffix**
Specified that all queues with the specified suffix in the queue name are returned.

prefix*suffix
Specifies that all queues with the specified prefix and the specified suffix in the queue name are returned.

z/OS If the name query parameter is used with a wildcard, the **commandScope=*** query parameter is specified, and there are no matching values on at least one of the active queue managers in the queue-sharing group, an error message is returned.

queueSharingGroupDisposition=*disposition*
z/OS This parameter is only available on z/OS.

Specifies where the queue for which information is to be returned is defined and how it behaves. That is, it specifies the disposition of the queue for which information is to be returned.

You cannot specify the **queueSharingGroupDisposition** parameter if you specify `type=cluster` for the **type** parameter.

The value can be one of the following values:

live
Specifies that the queue is defined as qmgr or copy.

In a shared queue manager environment, **live** also displays information for queues that are defined with shared.

If the **commandScope** optional query parameter is specified with the **live** option, then any queue definitions with a disposition of shared are returned only by the queue manager that received the REST request. Other queue managers in the group do not return these queue definitions.

If you specify **live** with the **attributes** parameter, and specify the **commandScope** parameter with a queue manager name, queue attributes are not returned for shared queues.

all
Specifies that the queue is defined as qmgr or copy.

In a shared queue manager environment, **all** also displays information for queues that are defined with group or shared.

If the **commandScope** optional query parameter is specified with **all**, then any queue definitions with a disposition of group or shared are returned only by the queue manager that received the REST request. Other queue managers in the group do not return these queue definitions.

If you specify **all** with the **attributes** parameter, and specify the **commandScope** parameter with a queue manager name, queue attributes are not returned for shared queues.

If you specify **all** and specify **type=all**, no cluster queues are returned.

copy
Specifies that the queue is defined as copy.

group
Specifies that the queue is defined as group.

If you specify **group**, you cannot specify the **commandScope** optional query parameter.

private
Specifies that the queue is defined as copy or qmgr.

qmgr
Specifies that the queue is defined as qmgr.

shared
Specifies that the queue is defined as shared.

You cannot specify the **commandScope** optional query parameter with this option, unless the **status** or **applicationHandle** optional query parameter is also specified.

You cannot specify this option with the **attributes** parameter if you also specify the **commandScope** parameter with a queue manager name.

If you specify **shared** and specify **type=all**, all shared queues are returned, including cluster queues with a disposition of shared.

The default value is live.

type=*type*
Specifies the type of queue to return information about.

The value can be one of the following values:

all
Specifies that information about all queues, including cluster queues, is returned.

z/OS On z/OS, ensure that the channel initiator is running when you use this option.

local
Specifies that information about local queues is returned.

alias
Specifies that information about alias queues is returned.

remote
Specifies that information about remote queues is returned.

cluster
Specifies that information about cluster queues is returned.

z/OS You cannot specify **type=cluster** if you specify the **queueSharingGroupDisposition** parameter.

z/OS On z/OS, ensure that the channel initiator is running when you use this option.

model
Specifies that information about model queues is returned.

The default value is `all`.

Request headers

The following headers must be sent with the request:

Authorization
This header must be sent if you are using basic authentication. For more information, see Using HTTP basic authentication with the REST API.

Authorization
This header must be sent if you are using basic authentication. For more information, see Using HTTP basic authentication with the REST API.

V 9.0.4 The following headers can optionally be sent with the request:

ibm-mq-rest-gateway-qmgr
This header specifies the queue manager that is to be used as the gateway queue manager. The gateway queue manager is used to connect to a remote queue manager. For more information, see Remote administration using the REST API.

Request body format

None.

Security requirements

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. For more information about security for the administrative REST API, see IBM MQ Console and REST API security.

The security principal of the caller must be granted the ability to issue the following PCF commands for the specified queue manager:

- If the **status** or **applicationHandle** query parameters are not specified:
  - For the queue that is specified by the *{queueName}* portion of the resource URL, or for queues that match the specified query parameters, authority to issue the **MQCMD_INQUIRE_Q** PCF command must be granted.
- If the **status** or **applicationHandle** query parameters are specified:

- For the queue that is specified by the *{queueName}* portion of the resource URL, or for queues that match the specified query parameters, authority to issue the **MQCMD_INQUIRE_Q** PCF command must be granted.
- For the queue that is specified by the *{queueName}* portion of the resource URL, or for queues that match the specified query parameters, authority to issue the **MQCMD_INQUIRE_QSTATUS** PCF command must be granted.

A principal has display authority if the principal can issue one or both of the **MQCMD_INQUIRE_Q** and **MQCMD_INQUIRE_QSTATUS** PCF commands. If the principal has display authority for only some of the queues that are specified by the resource URL and query parameters, then the array of queues that is returned from the REST request is limited to those queues that the principal has authority to display. No information is returned about queues that cannot be displayed. If the principal does not have display authority for any of the queues that are specified by the resource URL and query parameters, an HTTP status code of 403 is returned.

ULW On UNIX, Linux, and Windows, you can grant authority to security principals to use IBM MQ resources by using the **mqsetaut** command. For more information, see mqsetaut.

z/OS On z/OS, see Setting up security on z/OS.

Response status codes

200 Queue information retrieved successfully.

400 Invalid data provided.

For example, invalid queue attributes specified.

401 Not authenticated.

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. For more information, see "Security requirements" on page 1725.

403 Not authorized.

The caller is authenticated to the mqweb server and is associated with a valid principal. However, the principal does not have access to all, or a subset of the required IBM MQ resources. For more information about the access that is required, see "Security requirements" on page 1725.

404 Queue does not exist.

500 Server issue or error code from IBM MQ.

503 Queue manager not running.

Response headers

The following headers are returned with the response:

`Content-Type`
This header is returned with a value of `application/json;charset=utf-8`.

z/OS **`ibm-mq-qmgrs`**
On z/OS, if the optional query parameter commandScope=* is used, this header is returned with a comma-separated list of the queue managers that generated a response. For example, the header might look like the following header:

```
ibm-mq-qmgrs: MQ21, MQ22
```

If an error occurs before the command is issued to the queue managers, the response header does not contain the list of queue managers. For example, a request that generates a 200 or 201 status code has

the header because the command was successful. A request that generates a 401 (not authenticated) status code does not have the header because the request was rejected. A request that generates a 403 (not authorized) status code has the header because individual queue managers decide whether the command is authorized.

V 9.0.4 **ibm-mq-rest-gateway-qmgr**
This header is returned if a remote queue manager is specified in the resource URL. The value of this header is the name of the queue manager that is used as the gateway queue manager.

Response body format

The response is in JSON format in UTF-8 encoding. The response contains an outer JSON object that contains a single JSON array called `queue`. Each element in the array is a JSON object that represents information about a queue. Each of these JSON objects contains the following attributes:

name
String.

Specifies the name of the queue.

This attribute is always returned.

type
String.

Specifies the type of queue.

The value is one of the following values:

- `local`
- `alias`
- `remote`
- `cluster`
- `model`

This attribute is always returned.

The following objects can be included in the JSON object that represents information about a queue. Which objects and attributes are returned depends on the URL that was specified for the request:

remote
Contains attributes that are related to remote queues.

alias
Contains attributes that are related to alias queues.

dynamic
Contains attributes that are related to dynamic queues.

model
Contains attributes that are related to model queues.

cluster
Contains attributes that are related to clusters.

trigger
Contains attributes that are related to triggering.

events
Contains two objects, one for queue depth and one for queue service interval events. Each object contains attributes that are related to the event type.

applicationDefaults
Contains attributes that are related to default behavior such as message persistence, message priority, shared input settings, and read ahead settings.

queueSharingGroup
Contains attributes that are related to queue-sharing groups on z/OS.

dataCollection
Contains attributes that are related to data collection, monitoring, and statistics.

storage
Contains attributes that are related to message storage, such as the maximum depth of the queue, and the maximum length of messages that are allowed on the queue.

general
Contains attributes that are related to general queue properties, such as whether get or put operations are inhibited, the description of the queue, and transmission queue settings.

extended
Contains attributes that are related to extended queue properties, such as backout queue settings, and shared input settings.

timestamps
Contains attributes that are related to date and time information, such as the time stamp of when a queue was created.

status
Contains attributes that are related to queue status information.

applicationHandle
Contains attributes that are related to application handle information.

If a queue has no application handles, but information about application handles is requested, an empty object is returned.

For more information, see "Response body attributes for queues" on page 1732.

If a damaged object is found, and the REST request did not specify a queue, an extra JSON array that is called `damaged` is returned. This JSON array contains a list of the objects that are damaged, specifying the object names. If the REST request specifies a queue name within the resource URL, but the object is damaged, an error is returned.

If an error occurs, the response body contains an error message. For more information, see REST API error handling.

Examples

Note: Information about the `SYSTEM.*` queues is returned. It is expected that all queues are returned. However, for brevity, the results shown in the following examples do not include all the expected results.

- The following example lists all queues on the queue manager `QM1`. The following URL is used with the HTTP GET method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue
  ```

  Version 9.0.3 and earlier:

  ```
  https://localhost:9443/ibmmq/rest/v1/qmgr/QM1/queue
  ```

  The following JSON response is returned:

```
{
    "queue":
    [{
        "name": "localQueue",
        "type": "local"
    } , {
        "name": "remoteQueue",
        "type": "remote",
        "remote": {
            "queueName": "queueOnQM1",
            "qmgrName": "QM1"
        }
    } , {
        "name": "aliasQueue",
        "type": "alias",
        "alias": {
            "targetName": "localQueue"
        }
    }, {
        "name": "modelQueue",
        "type": "model",
        "model": {
            "type": "permanentDynamic"
        }
    }, {
        "name": "permanentDynamicQueue",
        "type": "local",
        "dynamic": {
            "type": "permanentDynamic"
        }
    },{
        "name": "aliasQueue2",
        "type": "cluster",
        "cluster": {
            "name": "CLUSTER1",
            "qmgrName" : "QM2",
            "queueType": "alias"
        }
    }]
}
```

- The following example lists all local queues on the queue manager QM1, showing whether they are get or put enabled. The following URL is used with the HTTP GET method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QMGR2/queue?type=local&attributes=general.inhibitPut,general.inhibitGe
  ```

  Version 9.0.3 and earlier:

  ```
  https://localhost:9443/ibmmq/rest/v1/qmgr/QMGR2/queue?type=local&attributes=general.inhibitPut,general.inhibitGet
  ```

  The following JSON response is returned:

  ```
  {
      "queue":
      [{
          "name": "localQueue",
          "type": "local",
          "general": {
              "inhibitPut": true,
              "inhibitGet": false,
          }
      }, {
          "name": "permanentDynamicQueue",
          "type": "local",
          "dynamic": {
              "type": "permanentDynamic"
          },
  ```

```
        "general": {
            "inhibitPut": false,
            "inhibitGet": false,
        }
    }]
}
```

- The following example lists the status attributes for the queue Q1, on queue manager QM1. The following URL is used with the HTTP GET method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue/Q1?status=*
  ```

  Version 9.0.3 and earlier:

  ```
  https://localhost:9443/ibmmq/rest/v1/qmgr/QM1/queue/Q1?status=*
  ```

  The following JSON response is returned:

  ```
  {
   "queue":
   [{
    "name": "Q1",
    "status": {
     "currentDepth": 0,
     "lastGet": "2016-12-05T15:56:28.000Z",
     "lastPut": "2016-12-05T15:56:28.000Z",
     "mediaRecoveryLogExtent": "",
     "oldestMessageAge": 42,
     "onQueueTime": {
      "longSamplePeriod": 3275,
      "shortSamplePeriod": 3275
     },
     "openInputCount": 1,
     "openOutputCount": 1,
     "uncommittedMessages": 2
    },
    "type": "local"
   }]
  }
  ```

- The following example lists the application handle attributes for a queue Q1, on queue manager QM1. The following URL is used with the HTTP GET method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue/Q1?applicationHandle=*
  ```

  Version 9.0.3 and earlier:

  ```
  https://localhost:9443/ibmmq/rest/v1/qmgr/QM1/queue/Q1?applicationHandle=*
  ```

  The following JSON response is returned:

  ```
  {
   "queue":
   [{
    "applicationHandle":
    [{
     "asynchronousState": "none",
     "channelName": "",
     "connectionName": "",
     "description": "",
     "state": "inactive",
     "openOptions": [
                          "MQOO_INPUT_SHARED",
                          "MQOO_BROWSE",
                          "MQOO_INQUIRE",
                          "MQOO_SAVE_ALL_CONTEXT",
                          "MQOO_FAIL_IF_QUIESCING"
                    ],
  ```

```
   "processID": 9388,
   "qmgrTransactionID": "AAAAAAhAAAA=",
   "recoveryID": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA==",
   "tag": "IBM\\Java70\\jre\\bin\\javaw.exe",
   "threadID": 0,
   "transactionType": "qmgr",
   "type": "userApplication",
   "userID": "myID"
  },
  {
   "asynchronousState": "none",
   "channelName": "",
   "connectionName": "",
   "description": "",
   "state": "inactive",
   "openOptions": [
                        "output ",
                        "failIfQuiescing"
                  ],
                  "processID": 9388,
   "qmgrTransactionID": "AAAAAAhAAAA=",
   "recoveryID": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA==",
   "tag": "IBM\\Java70\\jre\\bin\\javaw.exe",
   "threadID": 0,
   "transactionType": "qmgr",
   "type": "userApplication",
   "userID": "myID"
  }],
  "name": "Q1",
  "type": "local"
 }]
}
```


- The following example shows how to get all information, including status and application handles, for the queue `Q2` on queue manager `QM1`. The following URL is used with the HTTP GET method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue/Q2?attributes=*&status=*&applicationHandle=*
  ```

  Version 9.0.3 and earlier:

  ```
  https://localhost:9443/ibmmq/rest/v1/qmgr/QM1/queue/Q2?attributes=*&status=*&applicationHandle=*
  ```

- The following example shows how to get all queue configuration and status information for queues with an **openInputCount** greater than three, for the queue manager `QM1`. The following URL is used with the HTTP GET method:

  V 9.0.4 Version 9.0.4 and later:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue?attributes=*&status=*&filter=status.openInputCount:greaterTh
  ```

  Version 9.0.3 and earlier:

  ```
  https://localhost:9443/ibmmq/rest/v1/qmgr/QM1/queue?attributes=*&status=*&filter=status.openInputCount:greaterThan:3
  ```

Response body attributes for queues: V 9.0.2

When you use the HTTP GET method with the `queue` object to request information about queues, the following attributes are returned within named JSON objects.

The following objects are available:

For more information about the PCF equivalents to the queue REST API parameters and attributes, see REST API and PCF equivalents for queues.

remote

The `remote` object contains information about remote queues and is returned only for remote queues:

qmgrName
String.

Specifies the name of the remote queue manager.

If this remote queue is used as a queue manager alias, this attribute is the name of the queue manager.

If this remote queue is used as a reply-to queue alias, this attribute is the name of the queue manager that is to be the reply-to queue manager.

This attribute is always returned.

queueName
String.

Specifies the name of the queue as it is known on the remote queue manager.

This attribute is always returned.

transmissionQueueName
String.

Specifies the name of the transmission queue that is used for messages that are destined for either a remote queue or for a queue manager alias definition.

alias

The `alias` object contains information about alias queues and is returned only for alias queues:

`targetName`
String.

Specifies the name of the queue or topic that the alias resolves to.

This attribute is always returned.

`targetType`
String.

Specifies the type of object that the alias resolves to.

The value is one of the following values:

`queue`
Specifies that the object is a queue.

`topic`
Specifies that the object is a topic.

dynamic

The `dynamic` object contains information about dynamic queues and is returned only for local queues that are programmatically created from a model queue:

`type`
String.

Specifies the type of dynamic queue.

This attribute is always returned.

The value is one of the following values:

`permanentDynamic`
Specifies that the queue is a dynamically defined permanent queue.

`sharedDynamic`

z/OS This attribute is only available on z/OS.

Specifies that the queue is a dynamically defined shared queue.

`temporaryDynamic`
Specifies that the queue is a dynamically defined temporary queue.

model

The `model` object contains information about model queues and is returned only for model queues:

`type`
String.

Specifies the model queue definition type.

This attribute is always returned.

The value is one of the following values:

`permanentDynamic`
Specifies that the queue is a dynamically defined permanent queue.

sharedDynamic

z/OS This attribute is only available on z/OS.

Specifies that the queue is a dynamically defined shared queue.

temporaryDynamic
Specifies that the queue is a dynamically defined temporary queue.

cluster

The `cluster` object contains information about queues that are part of one or more clusters. The object is returned only for queues when `type=cluster` is specified, or if requested by the attributes query parameter:

name
String.

Specifies the name of the cluster that the queue belongs to.

This attribute, or the **namelist** attribute, is always returned.

namelist
String.

Specifies the namelist that lists the clusters that the queue belongs to.

This attribute, or the **name** attribute, is always returned.

qmgrId
String.

Specifies the unique identifier of the queue manager.

This attribute is returned only when `type=cluster` is specified.

qmgrName
String.

Specifies the name of the local queue manager.

This attribute is returned only when `type=cluster` is specified.

queueType
String.

Specifies the type of queue.

This attribute is returned only when `type=cluster` is specified.

The value is one of the following values:

local
Specifies that the cluster queue represents a local queue.

alias
Specifies that the cluster queue represents an alias queue.

remote
Specifies that the cluster queue represents a remote queue.

qmgrAlias
Specifies that the cluster queue represents a queue manager alias.

transmissionQueueForChannelName
String.

Specifies the generic name of the cluster-sender channels that use the queue as a transmission queue. The attribute specifies which cluster-sender channels send messages to a cluster-receiver channel from the cluster transmission queue.

workloadPriority
Integer.

Specifies the priority of the queue in cluster workload management.

A value of 0 specifies the lowest priority and 9 specifies the highest priority.

workloadQueueUse
String.

Specifies whether remote and local instances of the clustered queues are used in cluster workload distribution.

The value is one of the following values:

asQmgr
Use the value that is defined on the queue manager.

any
Use remote and local instances of the queues.

local
Use only local instances of the queues.

workloadRank
Integer.

Specifies the rank of the queue in cluster workload management.

A value of 0 specifies the lowest priority and 9 specifies the highest priority.

trigger

The `trigger` object contains information about triggering:

enabled
Boolean.

Specifies whether trigger messages are written to the initiation queue.

data
String.

Specifies the user data that is included in the trigger message.

depth
Integer.

Specifies the number of messages that initiates a trigger message to the initiation queue.

initiationQueueName
String.

Specifies the local queue for trigger messages that relate to the queue.

messagePriority
Integer.

Specifies the minimum priority that a message must have before it can cause, or be counted for, a trigger event.

processName
String.

Specifies the local name of the IBM MQ process that identifies the application to be started when a trigger event occurs.

If the queue is a transmission queue, the process definition contains the name of the channel to be started.

type
String.

Specifies the condition that initiates a trigger event. When the condition is true, a trigger message is sent to the initiation queue.

The value is one of the following values:

none
Send no trigger messages.

every
Send a trigger message for every message that arrives on the queue.

first
Send a trigger message when the queue depth goes from 0 to 1.

depth
Send a trigger message when the queue depth exceeds the value of the **depth**attribute.

events

The `events` object contains two objects, one for queue depth and one for queue service interval events. Each object contains attributes that are related to the event type:

depth
JSON object.

A JSON object that can contain the following attributes that related to queue depth events:

highEnabled
Boolean.

Specifies whether queue depth high events are generated.

A queue depth high event indicates that the number of messages on the queue is greater than or equal to the queue depth high limit, **highPercentage**.

highPercentage
Integer.

Specifies the threshold against which the queue depth is compared to generate a queue depth high event.

This value is expressed as a percentage of the maximum queue depth.

lowEnabled
Boolean.

Specifies whether queue depth low events are generated.

A queue depth low event indicates that the number of messages on the queue is less than or equal to the queue depth low limit, **lowPercentage**.

lowPercentage
Integer.

Specifies the threshold against which the queue depth is compared to generate a queue depth low event.

This value is expressed as a percentage of the maximum queue depth.

fullEnabled
Boolean.

Specifies whether queue full events are generated.

A queue full event indicates that no more messages can be put on a queue because the queue is full. That is, the queue depth reached the maximum queue depth.

serviceInterval
JSON object.

A JSON object that can contain the following attributes that are related to queue service interval events:

highEnabled
Boolean.

Specifies whether queue service interval high events are generated.

A queue service interval high event is generated when no messages were put to, or retrieved from, the queue for at least the amount of time specified by the **duration** attribute.

okEnabled
Boolean.

Specifies whether queue service interval OK events are generated.

A queue service interval OK event is generated when a message was retrieved from the queue within the amount of time that is specified by the **duration** attribute.

duration
Integer.

Specifies the service interval duration, in milliseconds, that is used to generate queue service interval high and queue service interval OK events.

applicationDefaults

The `applicationDefaults` object contains attributes that are related to default behavior such as message persistence, message priority, shared input settings, and read ahead settings:

clusterBind
String.

Specifies the binding to be used when `MQOO_BIND_AS_Q_DEF` is specified on the MQOPEN call.

The value is one of the following values:

onOpen
Specifies that the binding is fixed by the MQOPEN call.

notFixed
Specifies that the binding is not fixed.

onGroup
Specifies that the application can request that a group of messages is allocated to the same destination instance.

messagePropertyControl
String.

Specifies how message properties are handled when messages are retrieved from queues when `MQGMO_PROPERTIES_AS_Q_DEF` is specified on the MQGET call.

This attribute is applicable to local, alias, and model queues.

The value is one of the following values:

all
Specifies that all properties of the message are included when the message is sent to the remote queue manager. The properties, except those properties in the message descriptor or extension, are place in one of more MQRFH2 headers in the message data.

compatible
Specifies that if the message contains a property with the prefix mcd., jms., usr., or mqext., all message properties are delivered to the application in an MQRFH2 header. Otherwise, all properties, except those properties in the message descriptor or extension, are discarded and are no longer accessible.

force
Specifies that properties are always returned in the message data in an MQRFH2 header regardless of whether the application specifies a message handle. A valid message handle that is included in the MsgHandle field of the MQGMO structure on the MQGET call is ignored. Properties of the message are not accessible by using the message handle.

none
Specifies that all properties of the message are removed from the message before the message is sent to the remote queue manager. Properties in the message descriptor, or extension, are not removed.

version6Compatible
Any application MQRFH2 header is received as it was sent. Any properties set by using MQSETMP must be retrieved by using MQINQMP. They are not added to the MQRFH2 created by the application. Properties that were set in the MQRFH2 header by the sending application cannot be retrieved by using MQINQMP.

messagePersistence
String.

Specifies the default for message persistence on the queue. Message persistence determines whether messages are preserved across restarts of the queue manager.

The value is one of the following values:

persistent
Specifies that the messages on the queue are persistent, and are preserved when the queue manager restarts.

nonPersistent
Specifies that the messages on the queue are not persistent, and are lost when the queue manager restarts.

messagePriority
Integer.

Specifies the default priority of messages that are put on the queue.

putResponse
String.

Specifies the type of response that is used for put operations to the queue when an application specifies MQPMO_RESPONSE_AS_Q_DEF.

The value is one of the following values:

synchronous
The put operation is run synchronously, returning a response.

asynchronous
The put operation is run asynchronously, returning a subset of MQMD fields.

readAhead
String.

Specifies the default read-ahead behavior for non-persistent messages that are delivered to the client.

The value is one of the following values:

no Specifies that non-persistent messages are not read ahead unless the client application is configured to request read ahead.

yes
Specifies that non-persistent messages are sent ahead to the client before an application requests them. Non-persistent messages can be lost if the client ends abnormally or if the client does not consume all the messages that it is sent.

disabled
Specifies that non-persistent messages are not read ahead, regardless of whether read ahead is requested by the client application.

sharedInput
Boolean.

Specifies the default share option for applications that open this queue for input.

If the value is set to `true`, queues are enabled to get messages with shared access.

queueSharingGroup

The `queueSharingGroup` object contains attributes that are related to queue-sharing groups on z/OS:

disposition
String.

z/OS This attribute is only available on z/OS.

Specifies where the queue is defined and how it behaves. That is, it specifies the disposition of the queue.

This value is always returned if the queue manager is a member of the queue-sharing group.

The value is one of the following values:

copy
Specifies that the queue definition exists on the page set of the queue manager that runs the command. For local queues, messages are stored on the page sets of each queue manager and are available only through that queue manager.

group
Specifies that the queue definition exists in the shared repository.

qmgr
Specifies that the queue definition exists on the page set of the queue manager that runs the command. For local queues, messages are stored on the page sets of each queue manager and are available only through that queue manager.

shared
This value is only valid for local queues.

Specifies that the queue exists in the shared repository. Messages are stored in the coupling facility and are available to any queue manager in the queue-sharing group.

qmgrName
String.

z/OS This attribute is only available on z/OS.

Specifies the name of the queue manager that generates the response to the REST request.

This attribute is only returned if the queue manager to which the REST request is made is part of a queue-sharing group, and the **commandScope** optional query parameter is specified.

structureName
String.

z/OS This attribute is only available on z/OS.

Specifies the name of the coupling facility structure where messages are stored when you used shared queues.

dataCollection

The `dataCollection` object contains attributes that are related to data collection, monitoring, and statistics:

accounting
String.

Specifies whether accounting data is collected for the queue.

The value is one of the following values:

asQmgr
Specifies that the queue inherits the value from the queue manager MQSC parameter `ACCTQ`.

off
Specifies that accounting data is not collected for the queue.

on Specifies that accounting data is collected for the queue if the `ACCTQ` MQSC parameter on the queue manager is not set to none.

monitoring
String.

Specifies whether online monitoring data is collected, and if so, the rate at which the data is collected.

The value is one of the following values:

off
Specifies that online monitoring data is not collected for the queue.

asQmgr
Specifies that the queue inherits the value from the queue manager `MONQ` MQSC parameter.

low
Specifies that online monitoring data is collected for the queue if the `MONQ` MQSC parameter on the queue manager is not set to none. The rate of data collection is low.

medium
Specifies that online monitoring data is collected for the queue if the `MONQ` MQSC parameter on the queue manager is not set to none. The rate of data collection is moderate.

high
Specifies that online monitoring data is collected for the queue if the `MONQ` MQSC parameter on the queue manager is not set to none. The rate of data collection is high.

statistics
ULW IBM MQ Appliance This attribute is only available on the IBM MQ Appliance, UNIX, Linux, and Windows.

String.

Specifies whether statistics data is collected for the queue.

The value is one of the following values:

asQmgr
Specifies that the queue inherits the value from the queue manager STATQ MQSC parameter.

off
Specifies that statistics data is not collected for the queue.

on Specifies that statistics data is collected for the queue if the STATQ MQSC parameter on the queue manager is not set to none.

storage

The storage object contains attributes that are related to message storage, such as the maximum depth of the queue, and the maximum length of messages that are allowed on the queue:

indexType

z/OS This attribute is only available on z/OS.

String.

Specifies the type of index that is maintained by the queue manager to expedite MQGET operations on the queue. For shared queues, the type of index determines what type of MQGET calls can be used.

The value is one of the following values:

none
Specifies that there is no index. Messages are retrieved sequentially.

correlationId
Specifies that the queue is indexed by using correlation identifiers.

groupId
Specifies that the queue is indexed by using group identifiers.

messageId
Specifies that the queue is indexed by using message identifiers.

messageToken
Specifies that the queue is indexed by using message tokens.

maximumMessageLength
Integer.

Specifies the maximum message length that is allowed, in bytes, for messages on the queue.

maximumDepth
Integer.

Specifies the maximum number of messages that are allowed on the queue.

messageDeliverySequence
String.

Specifies whether messages are delivered in priority order or by sequence.

The value is one of the following values:

priority
Specifies that messages are returned in priority order.

fifo
Specifies that messages are returned in first in, first out order.

nonPersistentMessageClass

ULW IBM MQ Appliance This attribute is only available on the IBM MQ Appliance, UNIX, Linux, and Windows.

String.

This attribute is valid only on local and model queues.

Specifies the level of reliability that is assigned to non-persistent messages that are put to the queue.

The value is one of the following values:

normal
Specifies that non-persistent messages persist for the lifetime of the queue manager session. They are discarded if the queue manager restarts.

high
Specifies that the queue manager attempts to retain non-persistent messages for the lifetime of the queue. Non-persistent messages might still be lost if a failure occurs.

pageSet

z/OS This attribute is only available on z/OS.

Integer.

Specifies the ID of the page set.

storageClass

z/OS This attribute is only available on z/OS.

String.

Specifies the name of the storage class.

general

The `general` object contains attributes that are related to general queue properties, such as whether get or put operations are inhibited, the description of the queue, and transmission queue settings:

description
String.

Specifies the description of the queue.

inhibitGet
Boolean.

Specifies whether get operations are allowed on the queue.

If the value is set to `true`, get operations are not allowed on the queue.

inhibitPut
Boolean.

Specifies whether put operations are allowed on the queue.

If the value is set to `true`, put operations are not allowed on the queue.

isTransmissionQueue
String.

Specifies whether the queue is for normal usage or for transmitting messages to a remote queue manager.

If the value is set to `true`, the queue is a transmission queue for transmitting messages to a remote queue manager.

extended

The `extended` object contains attributes that are related to extended queue properties, such as backout queue settings, and shared input settings:

`allowSharedInput`
Boolean.

Specifies whether multiple instances of applications can open the queue for input.

If the value is set to `true`, multiple instances of applications can open the queue for input.

`backoutRequeueQueueName`
String.

Specifies the name of the queue to which a message is transferred if it is backed out more times than the value of **`backoutThreshold`**.

`backoutThreshold`
Integer.

Specifies the number of times that a message can be backed out before it is transferred to the backout queue that is specified by the **`backoutRequeueQueueName`** attribute.

`custom`
String.

Specifies custom attributes for new features.

V 9.0.4 **`enableMediaImageOperations`**

ULW IBM MQ Appliance This attribute is available only on the IBM MQ Appliance, UNIX, Linux, and Windows.

Specifies whether a local or permanent dynamic queue object is recoverable from a media image, if linear logging is being used.

String.

The value is one of the following values:

`yes`
Specifies that this queue object is recoverable.

`no` The `rcdmqimg` and `rcrmqobj` commands are not permitted for these objects. If automatic media images are enabled, the media images are not written for these objects.

`asQmgr`
Specifies that the queue inherits the value from the queue manager `ImageRecoverQueue` attribute.

This is the default value for this attribute.

`hardenGetBackout`

z/OS This attribute is only available on z/OS.

Boolean.

Specifies whether the count of backed out messages is saved across restarts of the queue manager.

If the value is set to `true`, the backout count is saved across restarts.

`supportDistributionLists`

ULW IBM MQ Appliance This attribute is only available on the IBM MQ Appliance, UNIX, Linux, and Windows.

Boolean.

Specifies whether distribution-list messages can be placed on the queue.

If the value is set to `true`, distribution lists can be placed on the queue.

timestamps

The `timestamps` object contains attributes that are related to date and time information.

`altered`
String.

Specifies the date and time at which the queue was last altered.

For more information about the time stamp format that is used to return the date and time, see REST API time stamps.

`clustered`
String.

Specifies the date and time at which the information became available to the local queue manager.

For more information about the time stamp format that is used to return the date and time, see REST API time stamps.

`created`
String.

Specifies the date and time at which the queue was created.

For more information about the time stamp format that is used to return the date and time, see REST API time stamps.

status

The `status` object contains attributes that are related to queue status information:

`currentDepth`
Integer.

Specifies the current queue depth.

`lastGet`
String.

Specifies the date and time at which the last message was destructively read from the queue.

For more information about the time stamp format that is used to return the date and time, see REST API time stamps.

This attribute cannot be used to filter results.

`lastPut`
String.

Specifies the date and time at which the last message was successfully put to the queue.

For more information about the time stamp format that is used to return the date and time, see REST API time stamps.

This attribute cannot be used to filter results.

`mediaRecoveryLogExtent`
ULW IBM MQ Appliance This attribute is only available on the IBM MQ Appliance, UNIX, Linux, and Windows.

String.

Specifies the name of the oldest log extent that is required to perform media recovery of the queue.

The name that is returned is of the form `Snnnnnnn.LOG` and is not a fully qualified path name.

oldestMessageAge
Integer.

Specifies the age, in seconds, of the oldest message on the queue.

If the queue is empty, 0 is returned. If the value is greater than 999 999 999, it is returned as 999 999 999. If no data is available, -1 is returned.

onQueueTime
JSON object.

A JSON object that can contain the following attributes that related to the amount of time that a message remains on the queue:

longSamplePeriod
Integer.

Specifies an indication of the time, in microseconds, that a message remains on the queue based on activity over a long period.

shortSamplePeriod
Integer.

Specifies an indication of the time, in microseconds, that a message remains on the queue based on activity over a short period.

This attribute cannot be used to filter results.

openInputCount
Integer.

Specifies the number of handles that are currently valid for removing messages from the queue by using the MQGET call.

openOutputCount
Integer.

Specifies the number of handles that are currently valid for putting messages to the queue by using the MQPUT call.

monitoringRate
String.

Specifies the rate at which monitoring data is collected for the queue.

The value is one of the following values:

off
Specifies that no data is collected.

low
Specifies a low rate of data collection.

medium
Specifies a medium rate of data collection.

high
Specifies a high rate of data collection.

tpipeName
z/OS This attribute is only available on z/OS.

Array.

Specifies the TPIPE names that are used for communication with OTMA by using the IBM MQ IMS bridge, if the bridge is active.

uncommittedMessages
Integer.

Specifies the number of uncommitted changes that are pending for the queue.

On z/OS, the value can be only either 0 or 1. A value of 1 indicates that there is at least one uncommitted message on the queue.

applicationHandle

The `applicationHandle` object contains attributes that are related to application handle information:

description
String.

Specifies a description for the application.

tag
z/OS This attribute is only available on z/OS.

String.

Specifies the tag of the open application.

type
String.

Specifies the type of application.

This value is one of the following values:

queueManagerProcess
Specifies that the open application is a queue manager process.

channelInitiator
Specifies that the open application is a channel initiator.

userApplication
Specifies that the open application is a user application.

batchConnection
z/OS This attribute is only available on z/OS.

Specifies that the open application is using a batch connection.

rrsBatchConnection
z/OS This attribute is only available on z/OS.

Specifies that the open application is an RRS-coordinated application that uses a batch connection.

cicsTransaction
z/OS This attribute is only available on z/OS.

Specifies that the open application is a CICS transaction.

imsTransaction
z/OS This attribute is only available on z/OS.

Specifies that the open application is an IMS transaction.

systemExtension
Specifies that the open application is an application that performs an extension of function that is provided by the queue manager.

asynchronousConsumerState
String.

Specifies the state of the asynchronous consumer on the queue.

The value is one of the following values:

active
Specifies that an MQCB call set up a function to call back to process messages asynchronously, and the connection handle has started so that asynchronous message consumption can proceed.

inactive
Specifies that an MQCB call set up a function to call back to process messages asynchronously, but the connection handle is not started, or is stopped or suspended.

suspended
Specifies that the asynchronous consumption callback is suspended so that asynchronous message consumption cannot proceed on the handle.

This situation can be either because an MQCB or MQCTL call with *Operation* MQOP_SUSPEND was issued against this object handle by the application, or because it was suspended by the system. If it was suspended by the system, as part of the process of suspending asynchronous message consumption the callback function is called with the reason code that describes the problem that resulted in suspension. This situation is reported in the reason field in the MQCBC structure passed to the callback. In order for asynchronous message consumption to proceed, the application must issue an MQCB or MQCTL call with *Operation* MQOP_RESUME.

suspendedTemporarily
Specifies that the asynchronous consumption callback is temporarily suspended by the system so that asynchronous message consumption cannot proceed on this handle.

As part of the process of suspending asynchronous message consumption the callback function is called with the reason code that describes the problem that resulted in suspension. This situation is reported in the reason field in the MQCBC structure passed to the callback. The callback function is called again when asynchronous message consumption is resumed by the system after the temporary condition is resolved.

none
Specifies that an MQCB call was not issued against this handle, so asynchronous message consumption is not configured on the handle.

addressSpaceId
z/OS This attribute is only available on z/OS.

String.

Specifies a four character address space identifier for the application.

channelName
String.

Specifies the channel name.

connectionName
String.

Specifies the connection name.

state
String.

Specifies the state of the handle.

This value is one of the following values:

active
Specifies that an API call from a connection is in progress for the queue. This state can occur when an MQGET WAIT call is in progress.

inactive
Specifies that no API call from a connection is in progress for the queue. This state can occur when no MQGET WAIT call is in progress.

openOptions
JSON array.

Specifies the open options that are in force for the queue.

Any of the valid MQOO options can be present in the array. For more information about the MQOO_* options, see MQOO_* (Open Options).

processId

ULW IBM MQ Appliance This attribute is only available on the IBM MQ Appliance, UNIX, Linux, and Windows.

Integer.

Specifies the process ID of the open application.

processSpecificationBlockName

z/OS This attribute is only available on z/OS.

String.

Specifies the eight character name of the program specification block that is associated with the running IMS transaction.

processSpecificationTableId

z/OS This attribute is only available on z/OS.

String.

Specifies the four character identifier of the program specification table region identifier for the connected IMS region.

qmgrTransactionId
String.

Specifies the unit of recovery that is assigned by the queue manager.

V 9.0.4 This identifier is represented as 2 hexadecimal digits for each byte of the recovery identifier.

This attribute cannot be used to filter results.

cicsTaskNumber

z/OS This attribute is only available on z/OS.

Integer.

Specifies a seven digit CICS task number.

threadId

ULW IBM MQ Appliance This attribute is only available on the IBM MQ Appliance, UNIX, Linux, and Windows.

Integer.

Specifies the thread ID of the open application.

A value of 0 indicates that the handle was opened by a shared connection. A handle that is created by a shared connection is logically open to all threads.

cicsTransactionId

z/OS This attribute is only available on z/OS.

String.

Specifies a four character CICS transaction ID.

unitOfWorkId

String.

Specifies the recovery identifier for the unit of recovery. The format of this value is determined by the value of **unitOfWorkType**.

V 9.0.4 This identifier is represented as 2 hexadecimal digits for each byte of the recovery identifier.

This attribute cannot be used to filter results.

unitOfWorkType

String.

Specifies the type of external unit of recovery identifier as perceived by the queue manager.

The value is one of the following values:

qmgr

cics

z/OS This value is only available on z/OS.

ims

z/OS This value is only available on z/OS.

rrs

z/OS This value is only available on z/OS.

xa

userId

String.

Specifies the user identifier of the open application.

DELETE: V 9.0.2

Use the HTTP DELETE method with the `queue` resource to delete a specified queue on a specified queue manager.

This REST API command is similar to the **Delete Queue** PCF command, and the **DELETE queues** MQSC commands.

- Resource URL
- Optional query parameters
- "Request headers" on page 1752
- Request body format
- "Security requirements" on page 1753
- Response status codes
- "Response headers" on page 1753
- Response body format
- Examples

Resource URL

V 9.0.4 Version 9.0.4 and later:

`https://host:port/ibmmq/rest/v1/admin/qmgr/{qmgrName}/queue/{queueName}`

Version 9.0.3 and earlier:

`https://host:port/ibmmq/rest/v1/qmgr/{qmgrName}/queue/{queueName}`

qmgrName
Specifies the name of the queue manager on which the queue to delete exists.

V 9.0.4 You can specify a remote queue manager as the **qmgrName**. If you specify a remote queue manager, you must configure a gateway queue manager. For more information, see Remote administration using the REST API.

The queue manager name is case-sensitive.

If the queue manager name includes a forward slash, a period, or a percent sign, these characters must be URL encoded:

- A forward slash (/) must be encoded as `%2F`.
- A percent sign (%) must be encoded as `%25`.

queueName
Specifies the name of the queue to delete.

The queue name is case-sensitive.

If the queue name includes a forward slash or a percent sign, these characters must be URL encoded:

- A forward slash, /, must be encoded as `%2F`.
- A percent sign, %, must be encoded as `%25`.

V 9.0.1 You can use HTTP instead of HTTPS if you enable HTTP connections. For more information about enabling HTTP, see Configuring HTTP and HTTPS ports.

Optional query parameters

keepAuthorityRecords

ULW IBM MQ Appliance This parameter is only available on the IBM MQ Appliance, UNIX, Linux, and Windows.

Specifies that the associated authority records are not deleted.

commandScope=*scope*

z/OS This parameter is only available on z/OS.

Specifies how the command is run when the queue manager is a member of a queue-sharing group.

You cannot specify this parameter if the queue manager is not a member of a queue-sharing group.

scope can be one of the following values:

The name of a queue manager
Specifies that the command is run on the queue manager that is named. The queue manager must be active within the same queue-sharing group as the queue manager that is specified in the resource URL.

You cannot specify the queue manager name that is the queue manager that is specified in the resource URL.

If the queue manager name includes a percent sign, %, this character must be URL encoded as `%25`.

***** Specifies that the command is run on the local queue manager and also passed to every active queue manager in the queue-sharing group.

If this option is used, an `ibm-mq-qmgrs` response header is returned with a comma-separated list of the queue managers that generated a response. For example, the header might look like the following header:

```
ibm-mq-qmgrs: MQ21, MQ22
```

purge
Specifies that all messages are purged from the queue.

If messages are on the queue, you must specify **purge**, or the queue cannot be deleted.

queueSharingGroupDisposition=*disposition*

z/OS This parameter is only available on z/OS.

Specifies where the queue is defined and how it behaves. That is, it specifies the disposition of the queue.

disposition can be one of the following values:

copy
Specifies that the queue definition exists on the page set of the queue manager that runs the command. The queue was defined by a command that used the **MQQSGD_COPY** PCF parameter, or the **copy** REST API parameter.

Any queue in the shared repository, or any queue that is defined by using the **MQQSGD_Q_MGR** PCF parameter, or **qmgr** REST API parameter, is not affected by this command.

group
Specifies that the queue definition exists in the shared repository. The queue was defined by a command that used the **MQQSGD_GROUP** PCF parameter, or the **group** REST API parameter.

Any queue that exists on the page set of the queue manager that runs the command, except for a local copy of the queue, is not affected by this command.

If the deletion is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE queue(q-name) QSGDISP(COPY)
```

or for a local queue only:

```
DELETE QLOCAL(q-name) NOPURGE QSGDISP(COPY)
```

The deletion of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

Note: You always get the NOPURGE option even if you specify the **purge** flag. To delete messages on local copies of the queues you must explicitly run, for each copy, a command to delete the queue with the **purge** flag, and a **queueSharingGroupDisposition** value of `copy`.

qmgr
Specifies that the queue definition exists on the page set of the queue manager that runs the command. The object was defined by a command that used the **MQQSGD_Q_MGR** PCF parameter or the **qmgr** REST API parameter.

Any queue that exists in the shared repository, or any local copy of such a queue, is not affected by this command.

shared
This value is only valid for local queues.

Specifies that the queue exists in the shared repository. The object was defined by a command that used the **MQQSGD_SHARED** PCF parameter or the **shared** REST API parameter.

Any queue that exists on the page set of the queue manager that runs the command, or any queue that is defined by a command that uses the parameter **MQQSGD_GROUP** is not affected by this command.

The default value is `qmgr`.

Request headers

The following headers must be sent with the request:

ibm-mq-rest-csrf-token
This header must be sent with a value that is the content of the `csrfToken` cookie. The content of the `csrfToken` cookie is used to confirm that the credentials that are being used to authenticate the request are being used by the owner of the credentials. That is, the token is used to prevent cross-site request forgery attacks.

The `csrfToken` cookie is returned after a request is made with an HTTP GET method. You cannot use a cached version of the content of the cookie because the content of the cookie can change. You must use the latest value of the cookie for each request.

V 9.0.5 The preceding information applies to releases up to and including IBM MQ Version 9.0.4. From IBM MQ Version 9.0.5, this header must be set, but the value can be anything, including being blank.
The `csrfToken` cookie is no longer sent in responses from the REST API in Version 9.0.5 and later.

Authorization
This header must be sent if you are using basic authentication. For more information, see Using HTTP basic authentication with the REST API.

V 9.0.4 The following headers can optionally be sent with the request:

ibm-mq-rest-gateway-qmgr
This header specifies the queue manager that is to be used as the gateway queue manager. The gateway queue manager is used to connect to a remote queue manager. For more information, see Remote administration using the REST API.

Request body format

None.

Security requirements

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. For more information about security for the administrative REST API, see IBM MQ Console and REST API security.

The security principal of the caller must be granted the ability to issue the following PCF commands for the specified queue manager:

- For the queue that is specified by the *{queueName}* portion of the resource URL, authority to issue the **MQCMD_DELETE_Q** PCF command must be granted.

ULW On UNIX, Linux, and Windows, you can grant authority to security principals to use IBM MQ resources by using the **mqsetaut** command. For more information, see mqsetaut.

z/OS On z/OS, see Setting up security on z/OS.

Response status codes

204 Queue deleted successfully.

400 Invalid data provided.

For example, invalid queue data is specified, or the queue is not empty.

401 Not authenticated.

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. The `ibm-mq-rest-csrf-token` header must also be specified. For more information, see "Security requirements."

403 Not authorized.

The caller is authenticated to the mqweb server and is associated with a valid principal. However, the principal does not have access to all, or a subset of the required IBM MQ resources. For more information about the access that is required, see "Security requirements."

404 Queue does not exist.

500 Server issue or error code from IBM MQ.

503 Queue manager not running.

Response headers

The following headers are returned with the response:

z/OS **ibm-mq-qmgrs**
On z/OS, if the optional query parameter commandScope=* is used, this header is returned with a comma-separated list of the queue managers that generated a response. For example, the header might look like the following header:

```
ibm-mq-qmgrs: MQ21, MQ22
```

If an error occurs before the command is issued to the queue managers, the response header does not contain the list of queue managers. For example, a request that generates a 200 or 201 status code has the header because the command was successful. A request that generates a 401 (not authenticated) status code does not have the header because the request was rejected. A request that generates a 403 (not authorized) status code has the header because individual queue managers decide whether the command is authorized.

V 9.0.4 **ibm-mq-rest-gateway-qmgr**
This header is returned if a remote queue manager is specified in the resource URL. The value of this header is the name of the queue manager that is used as the gateway queue manager.

Response body format

The response body is empty if the queue is deleted successfully. If an error occurs, the response body contains an error message. For more information, see REST API error handling.

Examples

The following example deletes the queue `Q1` from the queue manager `QM1`, and purges all messages from the queue when used with the HTTP DELETE method:

V 9.0.4 Version 9.0.4 and later:

```
https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue/Q1?purge
```

Version 9.0.3 and earlier:

```
https://localhost:9443/ibmmq/rest/v1/qmgr/QM1/queue/Q1?purge
```

/admin/qmgr/{qmgrName}/subscription: V 9.0.4

You can use the HTTP GET method with the `subscription` resource to request information about subscriptions.

You can use the administrative REST API gateway with this resource URL.

For more information about the PCF equivalents to the subscription REST API parameters and attributes, see “REST API and PCF equivalents for subscriptions” on page 1802.

GET: V 9.0.4

Use the HTTP GET method with the `subscription` resource to request information about subscriptions.

The information that is returned is similar to the information returned by the **Inquire Subscription** PCF command, and the **DISPLAY SUB** MQSC command.

Resource URL

`https://host:port/ibmmq/rest/v1/admin/qmgr/{qmgrName}/subscription/{subscriptionName}`

qmgrName
Specifies the name of the queue manager on which to query the subscriptions.

You can specify a remote queue manager as the **qmgrName**. If you specify a remote queue manager, you must configure a gateway queue manager. For more information, see Remote administration using the REST API.

The queue manager name is case-sensitive.

If the queue manager name includes a forward slash, a period, or a percent sign, these characters must be URL encoded:

- A forward slash (/) must be encoded as `%2F`.
- A percent sign (%) must be encoded as `%25`.

subscriptionName
Optionally specifies the name of a subscription that exists on the specified queue manager.

The subscription name is case-sensitive.

If the subscription name includes any non-alphanumeric characters, they must be URL encoded.

You can use HTTP instead of HTTPS if you enable HTTP connections. For more information about enabling HTTP, see Configuring HTTP and HTTPS ports.

Optional query parameters

attributes={*object*,...|*|*object.attributeName*,...}

object,...
Specifies a comma-separated list of JSON objects that contain related subscription attributes to return.

For example, to return all subscription attributes that are related to time stamps, specify `timestamps`. To return all subscription attributes that are related to the destination and user, specify `destination,user`.

You cannot specify the same object more than once.

For a full list of objects and associated attributes, see Attributes for subscriptions.

***** Specifies all attributes.

object.attributeName,...
Specifies a comma-separated list of queue configuration attributes to return.

Each attribute must specify the JSON object that contains the attribute, in the form `object.attributeName`. For example, to return the `correlationId` attribute, which is contained in the destination object, specify `destination.correlationId`.

You cannot specify the same attribute more than once.

For a full list of attributes and associated objects, see Attributes for subscriptions.

filter=*filterValue*
Specifies a filter for the subscription definitions that are returned.

This query parameter cannot be used if you specify a subscription name in the resource URL or if you use the ID query parameter.

You can specify only one filter.

filterValue has the following format:

attribute:*operator*:*value*

where:

attribute
Specifies one of the applicable attributes. For a full list of attributes, see Attributes for subscriptions. The following attributes cannot be specified:

- `name`
- `id`

To filter on any attributes that are time stamps, the filter can specify any portion of the time stamp, with a trailing asterisk, *. The format of a time stamp is, `YYYY-MM-DDThh:mm:ss`. For example, you can specify `2001-11-1*` to filter on dates in the range `2001-11-10` to `2001-11-19`, or `2001-11-12T14:*` to filter any minute in the specified hour of the specified day.

Valid values for the `YYYY` section of the date are in the range 1900 - 9999.

The time stamp is a string. Therefore, only the `equalTo` and `notEqualTo` operators can be used with the time stamp.

operator
Specifies one of the following operators:

lessThan
Use this operator only with integer attributes.

greaterThan
Use this operator only with integer attributes.

equalTo
Use this operator with any attribute.

notEqualTo
Use this operator with any attribute.

lessThanOrEqualTo
Use this operator only with integer attributes.

greaterThanOrEqualTo
Use this operator only with integer attributes.

value
Specifies the constant value to test against the attribute.

The value type is determined by the attribute type.

For string and boolean attributes, you can omit the value field after the colon. For string attributes, omit the value to return subscriptions with no value for the specified attribute. For boolean attributes, omit the value to return any subscriptions that have the specified attribute set to false. For example, the following filter returns all subscriptions where the topic name attribute is not specified:

```
filter=topic.name:equalTo:
```

A single asterisk, *, can be used for string attributes specified at the end of the value as a wildcard.

If the value includes non-alphanumeric characters, then they must be URL encoded. If the value contains a percent character or any asterisk that is not intended to be a wildcard, then the value must be URL encoded a second time. That is, a percent character must be encoded as `%2525`. An asterisk must be encoded as `%252A`.

id=*id*
Specifies the ID of a subscription that exists on the queue manager specified.

This query parameter cannot be used if you specify a subscription name in the resource URL or the name query parameter.

The ID is a string that contains a hexadecimal number. It can be comprised of a mixture of uppercase and lowercase characters.

name=*name*
Specifies a wildcard subscription name to filter on.

This query parameter cannot be used if you specify a subscription name in the resource URL or the `id` query parameter.

The *name* specified must either be blank or include an asterisk, *, as a wildcard. You can specify one of the following combinations:

Specifies that subscriptions that do have a blank name attribute are returned.

***** Specifies that all subscriptions are returned.

prefix*
Specifies that all subscriptions with the specified prefix in the subscription name are returned.

***suffix**
Specified that all subscriptions with the specified suffix in the subscription name are returned.

prefix*suffix
Specifies that all subscriptions with the specified prefix and the specified suffix in the subscription name are returned.

Request headers

The following headers must be sent with the request:

Authorization
This header must be sent if you are using basic authentication. For more information, see Using HTTP basic authentication with the REST API.

The following headers can optionally be sent with the request:

ibm-mq-rest-gateway-qmgr
This header specifies the queue manager that is to be used as the gateway queue manager. The gateway queue manager is used to connect to a remote queue manager. For more information, see Remote administration using the REST API.

Request body format

None.

Security requirements

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. For more information about security for the administrative REST API, see IBM MQ Console and REST API security.

The security principal of the caller must be granted the ability to issue the following PCF commands for the specified queue manager:

- For the subscription that is specified by the *{subscriptionName}* portion of the resource URL, the *id* query parameter, or for subscriptions that match the specified query parameters, authority to issue the **MQCMD_INQUIRE_SUBSCRIPTION** PCF command must be granted.

A principal has display authority if the principal can issue the **MQCMD_INQUIRE_SUBSCRIPTION** PCF command. If the principal has display authority for only some of the subscriptions that are specified by

the resource URL and query parameters, then the array of subscriptions that is returned from the REST request is limited to those subscriptions that the principal has authority to display. No information is returned about subscriptions that cannot be displayed. If the principal does not have display authority for any of the subscriptions that are specified by the resource URL and query parameters, an HTTP status code of 403 is returned.

ULW On UNIX, Linux, and Windows, you can grant authority to security principals to use IBM MQ resources by using the **mqsetaut** command. For more information, see mqsetaut.

z/OS On z/OS, see Setting up security on z/OS.

Response status codes

200 Subscriptions retrieved successfully.

400 Invalid data provided.

For example, invalid subscription attributes specified.

401 Not authenticated.

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. For more information, see "Security requirements" on page 1757.

403 Not authorized.

The caller is authenticated to the mqweb server and is associated with a valid principal. However, the principal does not have access to all, or a subset of the required IBM MQ resources. For more information about the access that is required, see "Security requirements" on page 1757.

404 Subscription does not exist.

500 Server issue or error code from IBM MQ.

503 Queue manager not running.

Response headers

The following headers are returned with the response:

`Content-Type`
This header is returned with a value of `application/json;charset=utf-8`.

V 9.0.4 **`ibm-mq-rest-gateway-qmgr`**
This header is returned if a remote queue manager is specified in the resource URL. The value of this header is the name of the queue manager that is used as the gateway queue manager.

Response body format

The response is in JSON format in UTF-8 encoding. The response contains an outer JSON object that contains a single JSON array called `subscription`. Each element in the array is a JSON object that represents information about a subscription. Each of these JSON objects contains the following attributes:

`id` Hexadecimal string

Specifies the unique key that identifies the subscription.

This attribute is always returned.

`name`
String

Specifies the name of the subscription.

This attribute is always returned.

resolvedTopicString
String

Specifies the fully resolved topic string using the combined values from the topic name and defined string when the subscription was created.

This attribute is always returned.

The following objects can be included in the JSON object that represents information about a subscription. Which objects and attributes are returned depends on the URL that was specified for the request:

topic
Contains attributes that are related to a defined topic.

selector
Contains attributes that are related to the message selector.

destination
Contains attributes that are related to the destination queue / queue manager.

user
Contains attributes that are related to user, such as the accounting token, the user ID that owns the subscription and the user data.

general
Contains attributes that are related to general subscription properties, such whether the subscription is durable, how the subscription was created and whether wildcards should be interpreted in the topic string.

extended
Contains attributes that are related to extended subscription properties, such as the expiry time, the message priority, and the network scope.

timestamps
Contains attributes that are related to date and time information, such as the time stamp of when the subscription was created.

For more information, see “Response body attributes for subscriptions” on page 1761.

If an error occurs, the response body contains an error message. For more information, see REST API error handling.

Examples

- The following example lists all subscriptions on the queue manager QM1. The following URL is used with the HTTP GET method:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/subscription
  ```

  The following JSON response is returned:

  ```
  {
      "subscription":
      [{
          "id": "414D5120514D33202020202020202020202020A878195911AFD206",
          "name": "SYSTEM.DEFAULT.SUB",
          "resolvedTopicString": ""
      },
      {
          "id": "414D5120514D33202020202020202020202020202C0740592162214A",
  ```

```
        "name": "MySubscription",
        "resolvedTopicString": "sports/golf"
    },
    {
        "id": "414D5120514D33202020202020202020202C07405921621307",
        "name": "QM1 SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS 414D515901010000000000000000000000000000000000000000 SYSTEM.BR
        "resolvedTopicString": "SYSTEM.BROKER.ADMIN.STREAM/MQ/QM1 /StreamSupport"
    }]
}
```

- The following example lists all subscriptions on the queue manager QM1, showing their topic properties. The following URL is used with the HTTP GET method:

  ```
  https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/subscription?attributes=topic
  ```

  The following JSON response is returned:

  ```
  {
      "subscription":
      [{
          "id": "414D5120514D33202020202020202020A878195911AFD206",
          "name": "SYSTEM.DEFAULT.SUB",
          "resolvedTopicString": "",
          "topic": {
              "definedString": "",
              "name": ""
          }
      },
      {
          "id": "414D5120514D33202020202020202020202C07405921621307",
          "name": "MySubscription",
          "resolvedTopicString": "sports/snooker",
          "topic": {
              "definedString": "sports/snooker",
              "name": ""
          }
      },
      {
          "id": "414D5120514D33202020202020202020202C07405921621307",
          "name": "QM1 SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS 414D515901010000000000000000000000000000000000000000 SYSTEM.BR
          "resolvedTopicString": "SYSTEM.BROKER.ADMIN.STREAM/MQ/QM1 /StreamSupport",
          "topic": {
              "definedString": "MQ/QM1 /StreamSupport",
              "name": "SYSTEM.BROKER.ADMIN.STREAM"
          }
      }]
  }
  ```

Response body attributes for subscriptions: V 9.0.4

When you use the HTTP GET method with the `subscription` object to request information about subscriptions, the following attributes are returned within named JSON objects.

The following objects are available:

- "topic"
- "selector"
- "destination" on page 1762
- "user" on page 1762
- "general" on page 1763
- "extended" on page 1763
- "timestamps" on page 1765

For more information about the PCF equivalents to the subscription REST API parameters and attributes, see REST API and PCF equivalents for subscriptions.

topic

The `topic` object contains attributes that are related to a defined topic.

`name`
String.

Specifies the name of a previously defined topic object from which the topic string prefix is obtained for the subscription.

`definedString`
String.

Specifies the topic string that contains the application part of the topic string only.

selector

The `selector` object contains attributes that are related to the message selector.

`value`
String.

Specifies the selector applied to messages published to the topic.

Only those messages that satisfy the selection criteria are put to the destination specified by this subscription.

`type`
String.

Specifies type of selector.

The value is one of the following values:

`none`
Specifies that no selector is present.

`standard`
Specifies that the selector references only the properties of the message, not its content, using the standard IBM MQ selector syntax. Selectors of this type are to be handled internally by the queue manager.

extended
Specifies that the selector uses extended selector syntax, typically referencing the content of the message. Selectors of this type cannot be handled internally by the queue manager; extended selectors can be handled only by another program such as IBM Integration Bus.

destination

The `destination` object contains attributes that are related to the destination queue / queue manager.

isManaged
Boolean.

Specifies whether the destination is managed.

qmgrName
String.

Specifies the name of the destination queue manager, either local or remote, to which messages for the subscription are forwarded.

name
String.

Specifies the name of the alias, local, remote, or cluster queue to which messages for this subscription are put.

correlationId
Hexadecimal.

Specifies the correlation identifier that is placed in the CorrelId field of the message descriptor for all the messages sent to this subscription.

user

The `user` object contains attributes that are related to user that created the subscription, such as the accounting token, the user ID that owns the subscription and the user data.

accountingToken
Hexadecimal.

Specifies the accounting token used in the AccountingToken field of the message descriptor.

applicationIdentityData
String.

Specifies the application identity data used in the ApplIdentityData field of the message descriptor.

data
String.

Specifies the user data associated with the subscription.

name
String.

Specifies the userid that 'owns' this subscription. This parameter is either the userid associated with the creator of the subscription, or, if subscription takeover is permitted, the userid which last took over the subscription.

isVariable
Boolean.

Specifies whether any user other than the one who created the subscription can take over ownership.

general

The `general` object contains attributes that are related to general subscription properties, such as whether the subscription is durable, how the subscription was created and whether wildcards should be interpreted in the topic string.

`isDurable`
Boolean.

Specifies whether this subscription is a durable subscription.

If the subscription is durable, the subscription persists, even if the creating application disconnects from the queue manager or issues an MQCLOSE call for the subscription. The queue manager reinstates the subscription during restart.

If the subscription is non-durable, the queue manager removes the subscription when the creating application disconnects from the queue manager or issues an MQCLOSE call for the subscription. If the subscription has a **`destination.class`** of `managed`, the queue manager removes any messages not yet consumed when it closes the subscription.

`type`
String.

Specifies how the subscription was created.

The value is one of the following values:

`administrative`
Created using DEF SUB MQSC, REST or PCF command. It also indicates that a subscription has been modified using an administrative command.

`api`
Created using an MQSUB API request.

`proxy`
Created internally and used for routing publications through a queue manager.

`usesCharacterWildcard`
Boolean.

Specifies the schema to be used when any wildcard characters that are contained in the topic string are interpreted.

If the value is set to `true`, wildcard characters represent portions of strings; this is for compatibility with IBM MQ V6.0 brokers.

If the value is set to `false`, wildcard characters represent portions of the topic hierarchy; this value is for compatibility with IBM Integration Bus brokers.

extended

The `extended` object contains attributes that are related to extended subscription properties, such as the expiry time, the message priority and the network scope.

`expiry`
Integer.

Specifies the time, in tenths of seconds, at which a subscription expires after its creation date.

A value of `-1` can be used to represent unlimited.

`level`
Integer.

Specifies the level within the subscription interception hierarchy at which this subscription is made.

messagePriority
String.

Specifies the priority of messages sent to this subscription. It has the range 0-9.

Additionally, the value can be one of the following values:

asPublished
The priority of messages sent to this subscription is taken from that priority supplied to the published message.

asQueue
The priority of messages sent to this subscription is determined by the default priority of the queue defined as a destination.

messagePropertyControl
String.

Specifies how publish/subscribe related message properties are added to messages sent to this subscription.

The value is one of the following values:

none
Specifies that publish/subscribe properties are not added to the messages.

compatible
Specifies that if the original publication is a PCF message, then the publish/subscribe properties are added as PCF attributes. Otherwise, publish/subscribe properties are added within an MQRFH version 1 header. This method is compatible with applications coded for use with previous versions of IBM MQ.

pcf
Specifies that publish/subscribe properties are added as PCF attributes.

rfh2
Specifies that publish/subscribe properties are added within an MQRFH version 2 header. This method is compatible with applications coded for use with IBM Integration Bus brokers.

deliverOnRequest
Boolean.

Specifies whether the subscriber polls for updates using the MQSUBRQ API call, or whether all publications are delivered to this subscription.

If the value is set to `true`, publications are only delivered to this subscription in response to an MQSUBRQ API call.

If the value is set to `false`, all publications on the topic are delivered to this subscription.

networkScope
String.

Specifies whether this subscription is passed to other queue managers in the network.

The value is one of the following values:

all
Specifies that the subscription is forwarded to all queue managers directly connected through a publish/subscribe collective or hierarchy.

qmgr
Specifies that the subcription forwards only messages that are published on the topic within this queue manager.

timestamps

The `timestamps` object contains attributes that are related to date and time information.

`altered`
String.

Specifies the date and time at which the subscription was last altered.

For more information about the time stamp format that is used to return the date and time, see REST API time stamps.

`created`
String.

Specifies the date and time at which the subscription was created.

For more information about the time stamp format that is used to return the date and time, see REST API time stamps.

MFT REST API resources: V 9.0.5

This collection of topics provides reference information for each of the MFT REST API resources.

/admin/mft/agent: V 9.0.5

You can use the HTTP GET method with the agent resource, to request information about the status of agents, and other attribute details.

- Resource URL
- Optional query parameters
- Request body format
- "Security requirements" on page 1766
- Response status codes
- "Response headers" on page 1766
- Response body format
- Examples

Resource URL

```
https://{hostName}/ibmmq/rest/{version}/admin/mft/transfer/{Transferid}
```

Note that `Transferid` is required for this format, and is case insensitive; for example, `14d512050524d465444454d4f312020c5c6705924cf9e02`

Optional query parameters

attributes
A comma separated list of attributes to be returned.

See "JSON response body attributes" on page 1790 for a list of the attributes.

Notes:

1. If you do not specify **`attributes`**, a subset is returned
2. Requesting the same attribute multiple times is an error
3. You can make a request specifying attributes that are not valid for some of the transfers, or if **`attributes`** is set to *

4. If you request information on specific attributes not valid for that transfer, an error results

Request body format

None.

Security requirements

The following token must be provided with the request to authenticate:

- The LTPA token that is used to authenticate the user must be provided as a cookie.

Response status codes

200 Query successful. Response body contains information all the transfers

400 Invalid data provided.

401 Not authenticated

403 Not authorized

404 Resource was not found.

500 Server issue or error code from IBM MQ.

Response headers

`ibm-mq-rest-mft-total-transfers` contains the total number of transfers present in the internal storage.

The internal storage can contain a maximum of 25,000 files. Therefore, if each managed transfer contains one transfer item, the internal data storage can hold 25,000 managed transfers. Similarly, if each managed transfer contains 5000 transfer items, the internal data storage holds information about five managed transfers.

If the connection to the coordination queue manager is broken, the transfers are deleted from the internal data storage.

Note that the header is returned, only in the case of HTTP 200 and HTTP 404 (cases where `transferId` provided does not exist) response codes.

Response body format

The response body in the case of a 200 status code, represented in a JSON object.

Examples

Example 1

```
GET https://{hostName}/ibmmq/rest/{version}/admin/mft/transfer/{Transferid}
```

The preceding request returns the transfer details with the default schema of the response and the default attributes.

Example 2

```
https://{hostName}/ibmmq/rest/{version}/admin/mft/transfer/{Transferid}?attributes=*
```

The preceding request returns the entire schema for the transfer details.

Example 3

```
GET https://{hostName}/ibmmq/rest/{version}/admin/mft/transfer/{Transferid}?
attributes=transferSet.item.source,transferSet.item.destination
```

The preceding request returns the default schema, along with the comma separated attributes **transferSet.item.source** and **transferSet.item.destination**, mentioned in the URL.

Related reference:

"/admin/mft/transfer"
You can use the HTTP GET method with the transfer resource, to request information about transfers, and other status details.

"JSON response body attributes" on page 1790
A description of the attributes available to the MFT REST API in the JSON response body.

Related information:

Required configuration for the MFT REST API

/admin/mft/transfer: V 9.0.5

You can use the HTTP GET method with the transfer resource, to request information about transfers, and other status details.

- Resource URL
- Optional query parameters
- Request body format
- "Security requirements" on page 1768
- Response status codes
- "Response headers" on page 1768
- Response body format
- Examples

Resource URL

```
https://{hostName}/ibmmq/rest/{version}/admin/mft/transfer
```

Optional query parameters

limit The maximum number of transfers to be retrieved.

Parameter type
Integer

Default value, if limit not specified
100 transfers

The 100 transfers are the latest 100 transfers.

Note: The transfers are lost once the `mqweb` server is shutdown, and will not be available for any future references using the REST API.

after `transferId` of the transfer from where the transfer list has to fetched. All the transfers that are initiated after that particular transfer are fetched.

Parameter type
String

Default value
None

before `transferId` of the transfer from where the transfer list has to fetched. All the transfers that are initiated before that particular transfer are fetched.

Parameter type
String

Default value
None

Request body format

None.

Security requirements

The following token must be provided with the request to authenticate:

- The LTPA token that is used to authenticate the user must be provided as a cookie.

Response status codes

200 Query successful. Response body contains information all the transfers

400 Invalid data provided.

401 Not authenticated

403 Not authorized

404 Resource was not found.

500 Server issue or error code from IBM MQ.

Response headers

`ibm-mq-rest-mft-total-transfers` contains the total number of transfers present in the internal data storage.

The internal data storage can contain a maximum of 25,000 files. Therefore, if each managed transfer contains one transfer item, the internal data storage can hold 25,000 managed transfers. Similarly, if each managed transfer contains 5000 transfer items, the internal data storage holds information about five managed transfers.

If the connection to the coordination queue manager is broken, the transfers are deleted from the internal data storage.

Note that the header is returned, only in the case of HTTP 200 and HTTP 404 (cases where `transferId` provided does not exist) response codes.

Response body format

The response body in the case of a 200 status code.

The transfer detail is represented as a JSON object. These objects contain all the required information as described in "JSON response body attributes" on page 1790.

Examples

Example 1

```
GET https://{hostName}/ibmmq/rest/{version}/admin/mft/transfer?after={transferId}&limit=10
```

The preceding request is for a list of 10 transfers after a particular transfer where:

transferid
Is the identifier of the transfer used to go through the transfers

after Is the attribute needed in the URI to move forwards.

Similarly, if you wanted a list of transfers before a particular transfer, use the `back` attribute in place of the `after` attribute.

Example 2

If you used the basic format of the command, as shown in "Resource URL" on page 1767, you received a list of 100 transfers, starting with the most recent.

If you initiated few more transfers, and want to see the list of transfers which have been initiated after the previous request, use a REST APIrequest of the following form:

```
GET https://{hostName}/ibmmq/rest/{version}/admin/mft/transfer?after=transferId100&limit=10
```

where:

transferid100
Is the identifier of transfer number 100

limit=10
The order of transfers is transfer110 to transfer101

Similarly, if you want to get a list of the older transfers, change the value of `transferId`. For example, if you want a list from transfer 49 to transfer 40, set the value of `transferId` to 50.

Example 3

```
GET https://{hostName}/ibmmq/rest/{version}/admin/mft/transfer?limit=100
```

The preceding request gives you, at most, the latest 100 transfers, with the latest transfer being at the top of the list.

Notes:

1. To get the latest transfers you need only provide *limit* as an attribute in the URI.
2. If the number of transfers present in the internal storage is less than the *limit* attribute you have set, you do not receive a warning message. However many transfers that are present in the internal storage are retrieved.

You can also specify:

```
GET https://{hostName}/ibmmq/rest/{version}/admin/mft/transfer?after={transferId}
```

or

```
GET https://{hostName}/ibmmq/rest/{version}/admin/mft/transfer?before={transferId}
```

If you specify either the *after* or *before* attribute, without specifying *limit*, then *limit* is set to the default value of 100.

Related reference:

"/admin/mft/agent" on page 1765
You can use the HTTP GET method with the agent resource, to request information about the status of agents, and other attribute details.

Related information:

Required configuration for the MFT REST API

MFT REST API configuration parameters: V 9.0.5

A description of the configuration parameters used in the REST API.

Attention: If you set a value that is not valid, for any of the following properties, an error message is logged in the `message.log` file, and the default value of that property is used.

`mqRestMftEnabled`

The MFT REST API is disabled by default. To enable MFT REST API services, you must set the **`mqRestMftEnabled`** property in the `mqwebuser.xml` file to *true*.

Parameter type
Boolean

Default value
false

For example:

```
<variable name="mqRestMftEnabled" value= "false"/>
```

If you change the value to *true*, you must restart the `mqweb` server.

`mqRestMftCoordinationQmgr`

The name of the coordination queue manager from which the transfer details are to be retrieved. You must define the coordination queue manager in `mqwebuser.xml` to retrieve the transfers.

Notes:

1. The coordination queue manager must be on the machine where the `mqweb` server is running.
2. Only the transfers that are initiated after starting the `mqweb` server are retrieved.
3. The `mqweb` server establishes a bindings mode connection to the queue manager.

Parameter type
String

Default value
Empty string ""

If the you do not set a value, and invoke the REST API, HTTP 400 is returned

For example:

```
<variable name="mqRestMftCoordinationQmgr" value= "CORDQMGR"/>
```

If you change the value, you must restart the `mqweb` server.

mqRestMftReconnectTimeoutInMinutes

The time in minutes until MFT stops attempting to connect to the coordination queue manager, when the queue manager has failed.

The first attempt to re-establish the connection is made immediately after the connection to the coordination queue manager is broken. If this fails, there is an interval of five minutes between every reconnection attempt.

Messages are logged to the `message.log` file when:

- The connection is broken
- Every reconnection attempt is made
- The connection is successful
- The reconnection times out

After reconnection times out, the next attempt to reconnect is made when the REST API resource:

```
"/admin/mft/transfer"
```

or

```
"/admin/mft/agent"
```

is invoked. If this reconnection attempt fails, MFT will again attempt to reconnect every five minutes until the reconnect timeout has passed.

If you invoke a REST API when the coordination queue manager has not started, HTTP 503 is returned.

Parameter type

Integer

Default value

30

The value is in minutes, and configured in the `mqwebuser.xml` file.

A value of `-1` causes MFT to attempt to reconnect, until the connection is successful.

A value between `0-5` means that MFT tries to reconnect to the coordination queue manager once only, and if the connection fails, MFT will not try again to re-establish the connection, until the REST API is invoked.

For example:

```
<variable name=" mqRestMftReconnectTimeoutInMinutes" value="30"/>
```

If you change the value, you must restart the `mqweb` server.

Related information:
Required configuration for the MFT REST API

MFT REST API error values: V 9.0.5

The MFT REST API uses the following IBM MQ REST API standards to throw an exception whenever an error is encountered.

See REST API error handling for a description of the error response format.

MFT examples

If MFT REST API services are not enabled, and you invoke the MFT REST API, you receive the following exception:

```
{"error": [{
  "action": "Enable the Managed File Transfer REST API and resubmit the request.",
  "completionCode": 0,
  "explanation": "Managed File Transfer REST calls are not permitted as the service is disabled.",
  "message": "MQWB0400E: Managed File Transfer REST API is not enabled.",
  "msgId": "MQWB0400E",
  "reasonCode": 0,
  "type": "rest"
}]}
```

If MFT REST API services are enabled and the coordination queue manager is not set in the `mqwebuser.xml` file, you receive the following exception:

```
{"error": [{
  "action": "Set the coordination queue manager name and restart the mqweb server.",
  "completionCode": 0,
  "explanation": "Coordination queue manager name must be set before using Managed File Transfer REST services.",
  "message": "MQWB0402E: Coordination queue manager name is not set.",
  "msgId": "MQWB0402E",
  "reasonCode": 0,
  "type": "rest"
}]}
```

Related information:
Required configuration for the MFT REST API

GET - agent status: V 9.0.5

Use the HTTP GET method with the `agent` resource to request information about agents.

The information that is returned is similar to the information returned by the fteListAgents and fteShowAgentDetails commands.

- Resource URL
- Optional query parameters
- "Request headers" on page 1775
- Request body format
- "Security requirements" on page 1775
- Response status codes
- "Response headers" on page 1775
- Response body format
- Examples

Resource URL

`https://host:port/ibmmq/rest/v1/admin/mft/agent`

`https://host:port/ibmmq/rest/v1/admin/mft/agent/ {agentname}`

agentName
Optional parameter which gives details of a agent; equivalent to fteShowAgentDetails

The agent name is not case-sensitive, but the agent name value, received as a response from the REST service is always in upper case.

- Agent names can be a maximum of 28 characters long and are not case-sensitive
- Agent names entered in lowercase or mixed case are converted to uppercase
- The name of the agent must conform to the IBM MQ rules for naming objects , and must be unique to its coordination queue manager.
- In addition to the IBM MQ object naming conventions, the percent (%) character cannot be used in agent names.
- The names of properties in the properties files are case-sensitive.

You can use HTTP instead of HTTPS if you enable HTTP connections. For more information about enabling HTTP, see Configuring the HTTP and HTTPS ports.

For more information about configuring the MFT REST service, see "MFT REST API configuration parameters" on page 1770

Optional query parameters

attributes={*object*,...|*|*object.attributeName*,...}

object
Specifies a comma-separated list of JSON objects that are added to a JSON object, which is a subsection of the complete details.

For example to return:

- All general details of all agents or a particular agent, specify *general*.
- All queue manager connection details of all agents or aparticular agent specify *qmgrConnection*.
- Details of connect direct bridge agent, specify *connectDirectBridge.* (applicable only for agent of type "connect direct bridge")
- Details of protocol agent, specify *protocolBridge*. (applicable only for agents of type "protocal bridge)

For a full list of attributes see "Response body attributes for agents" on page 1778

***** Specifies all attributes.

object.attributeName.,...
Specifies a comma-separated list of queue configuration attributes to return.

Each attribute must specify the JSON object that contains the attribute, in the form `object.attributeName`. For example, to return the `statusAge` attribute, which is contained in the general object, specify `general.statusAge`.

Attributes from the `status` and `applicationHandle` objects cannot be specified with this query parameter. Use the **status** and **applicationHandle** query parameters to return these attributes.

You cannot specify the same attribute more than once. If you request attributes that are not valid for a particular queue, the attributes are not returned for that queue.

You can query details for a set of agents for a particular type, state or name having a specified patterns:

```
name=validPattern
state=valid atent State
type=validType
```

name=*name*
This query parameter cannot be used if you specify a queue name in the resource URL. Specifies a wildcard agent name to filter on.

The name specified must include an asterisk, `*`, as a wildcard. You can specify one of the following combinations:

***** Specifies that all agents are returned

prefix*
Specifies that all agents with the specified prefix in the agent name are returned.

***suffix**
Specifies that all agents with the specified suffix in the agent name are returned.

prefix*suffix
Specifies that all agents with the specified prefix and the specified suffix in the agent name are returned.

type=validAgentType
Specifies the type of the agent to return information about. The value can be one of the following values:

all
Specifies that information about all agents, that include `standard`, `connectDirectBridge` and `protocolBridge`, is returned.

This is the default value.

standard
Specifies that information about agent of type standard is returned.

connectDirectBridge
Specifies that information about agents of type connect direct bridge is returned.

protocolBridge
Specifies that information about agents of type protocol bridge is returned.

state=validAgentState
Specifies the state of the agent to return information about. The value can be one of the following values:

all
Specifies that information about all agents, which includes all the valid states listed in the following text.

This is the default value.

active
Specifies that information about agents that are in an active state is returned.

ready
Specifies that information about agents that are in a ready state is returned.

starting
Specifies that information about agents that are in a starting state is returned.

unreachable
Specifies that information about agents that are in an unreachable state is returned.

stopped
Specifies that information about agents that are in a stopped state is returned.

endedUnexpectedly
Specifies that information about agents that are in an endedUnexpectedly state is returned.

noInformation
Specifies that information about agents that are in a noInformation state is returned.

unknown
Specifies that information about agents that are in an unknown state is returned.

problem
Specifies that information about agents that are in a problem state is returned.

Request headers

The following header must be sent with the request:

Authorization
This header must be sent if you are using basic authentication. For more information, see Using HTTP basic authentication with the REST API.

Request body format

None.

Security requirements

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MFTWebAdmin`, or `MFTWebAdminRO` roles. For more information about security for the administrative REST API, see IBM MQ Console and REST API security.

Response status codes

200 Queue information retrieved successfully.

400 Invalid data provided.

For example, invalid queue attributes specified.

401 Not authenticated.

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MFTWebAdmin` or `MFTWebAdminRO` roles. For more information, see Security Requirements.

403 Not authorized.

The caller is authenticated to the mqweb server and is associated with a valid principal. However, the principal does not have access to all, or a subset of the required IBM MQ resources. For more information about the access that is required, see Security Requirements.

404 Queue does not exist.

500 Server issue or error code from IBM MQ.

503 Queue manager not running.

Response headers

Content-Type
This header is returned with a value of `application/json;charset=utf-8`.

Response body format

The response is in JSON format in UTF-8 encoding. The response contains an outer JSON object that contains a single JSON array called `agent`. Each element in the array is a JSON object that represents information about an agent. Each of these JSON objects contains the following attributes:

`name`
String.

Specifies the name of the agent.

This attribute is always returned.

`type`
String.

Specifies the type of agent.

The value is one of the following values:

- `standard`
- `connectDirectBridge`
- `protocolBridge`

`state`
Specifies the state of the agent. The value can be one of the following values:

- active
- ready
- starting
- unreachable
- stopped

For more information, see "JSON response body attributes" on page 1790.

If an error occurs, see REST API error handling.

Examples

The following example returns the basic details of all agents, that is, only the following information is displayed:

- agent name
- agent type
- agent state

The following URL is used with the HTTP GET method:

```
https://localhost:9443/ibmmq/rest/v1/admin/mft/agent/
```

The following JSON response is returned:

```
{
     "agent":[  {"name": "AGENT1",
                       "state" : "ready",
                       "type":"standard"},
                      {"name": "AGENT2",
                       "state" : "ready",
                       "type":"standard"},
                     { "name": "BRIDGE_AGENT3",
                       "type" : "protocolBridge",
                       "state" : "ready"},
```

```
                {"name": "CD_AGENT",
                  "type":"connectDirectBridge",
                  "state" : "ready "}]
}
```

The following example lists all the agent of type **standard**, along with the **general** object. The following URL is used with the HTTP GET method:

```
https://localhost:9443/ibmmq/rest/v1/admin/mft/agent?attributes=general?type=standard
```

The following JSON response is returned:

```
{
     "agent":[
            {"name": "AGENT1",
              "state" : "ready",
              "type":"standard",
              "general": { "description" : "Standard connected to the qmgr in client mode",
                                "statusAge" : "06:31:00",
                                "version" : "9.0.3.0",
                                "level" : "p903-L170513",
                "statusPublicationRate" : 300,
                               "statusPublishTime" : "2017-10-31T06:57:07.000Z",
                               "maximumQueuedTransfers" : 1000,
                               "maximumDestinationTransfers":25,
                              "maximumSourceTransfers":25,
                               "operatingSystem" : "Windows7" }
         },
        {"name": "AGENT2",
          "state" : "ready",
           "type":"standard"
            "general" : { "description" : "Standard connected to qmgr in Binding mode",
                                "statusAge ":"05:00:00",
                                "version" : "9.0.3.0",
                              "level" : "p903-L170513",
                              "statusPublicationRate" : 300,
                              "statusPublishTime" : "2017-09-13T09:10:09.000Z",
                             "maximumQueuedTransfers" : 1000,
                             "maximumDestinationTransfers":25,
                            "maximumSourceTransfers":25,
                            "operatingSystem" : "Windows7" }
            }
   ]
}
```

The following example lists all the agents starting with the name AGENT, in a **ready** state, and of type **standard**, along with the **general** object of *statusAge*. The following URL is used with the HTTP GET method:

```
https://localhost:9443//ibmmq/rest/v1/admin/mft/agent?name=AGENT*?state=ready&type=standard&attributes=general.statusAge
```

The following JSON response is returned:

```
{"agent":[
              {   "name": "AGENT1",
                   "state" : "ready",
                   "type":"standard",
                   "general":
                      { "statusAge" : "05:00:00" }
                 },
                { "name": "AGENT2",
                   "state" : "ready",
                   "type":"standard"
                   "general" :
                     { " statusAge": "03:00:00"}
                },
              { "name": "AGENT3",
```

```
            "state" : "ready",
            "type" : "standard" ,
            "general":
                {  "statusAge " : "05:00:00"}
           }
      ]
```

Related reference:

"Response body attributes for agents"
When you use the HTTP GET method with the agent object to request information about agents, the following attributes are returned within named JSON objects.

Response body attributes for agents: V 9.0.5

When you use the HTTP GET method with the agent object to request information about agents, the following attributes are returned within named JSON objects.

The following objects are the default attributes found in the response, and are always returned:

name
String

Specifies the name of the agent, registered under the coordination queue manager.

type
String

Specifies the type of the agent, see "GET - agent status" on page 1772 for more information.

state
String

Specifies the state of the agent, see "GET - agent status" on page 1772 for more information.

The following objects are available:

- "general"
- "qmgrConnection" on page 1780
- "connectDirectBridge" on page 1782
- "protocolBridge" on page 1783

general

description
String

Specifies the description of the agent, if you have set a description during agent creation

There is no default value for this attribute

This attribute is returned only when the any of the following is set during the query:

- attributes=**general**
- attributes=*
- attributes=**general**.*description*

statusAge
String

Specifies the age of the agent. The age is calculated as the difference in time between the system time of the machine where the coordination queue manager is running, and the time the last status was published by an agent.

There is no default value for this attribute

This attribute is returned only when the any of the following is set during the query:

- attributes=**general**
- attributes=*
- attributes=**general**.*statusAge*

version
String

Specifies the version of the queue manager

There is no default value for this attribute

This attribute is returned only when the any of the following is set during the query:

- attributes=**general**
- attributes=*
- attributes=**general**.*version*

level
String

Specifies the build level on which the queue manager is running

There is no default value for this attribute

This attribute is returned only when the any of the following is set during the query:

- attributes=**general**
- attributes=*
- attributes=**general**.*level*

statusPublicationRate
Integer

Specifies the rate in seconds that the agent publishes its status.

The default value for this attribute is 300 seconds

This attribute is returned only when the any of the following is set during the query:

- attributes=**general**
- attributes=*
- attributes=**general**.*statusPublicationRate*

statusPublishTime
String

Specifies the time at which the agent published its status, in Universal Time Constant format

There is no default value for this attribute

This attribute is returned only when the any of the following is set during the query:

- attributes=**general**
- attributes=*
- attributes=**general**.*statusPublishTime*

maximumQueuedTransfers
Integer

Specifies the maximum number of pending transfers that can be queued by an agent until the agent rejects a new transfer request.

The default value for this attribute is 1000

This attribute is returned only when the any of the following is set during the query:

- attributes=**general**
- attributes=*
- attributes=**general**.*maximumQueuedTransfers*

maximumQueuedTransfers
Integer

Specifies the maximum number of pending transfers that can be queued by an agent until the agent rejects a new transfer request.

The default value for this attribute is 1000

This attribute is returned only when the any of the following is set during the query:

- attributes=**general**
- attributes=*
- attributes=**general**.*maximumQueuedTransfers*

maximumDestinationTransfers
Integer

Specifies the maximum number of concurrent transfers that the destination agent processes at any given point in time.

The default value for this attribute is 25

This attribute is returned only when the any of the following is set during the query:

- attributes=**general**
- attributes=*
- attributes=**general**.*maximumDestinationTransfers*

maximumSourceTransfers
Integer

Specifies the maximum number of concurrent transfers that the source agent processes at any given point in time.

The default value for this attribute is 25

This attribute is returned only when the any of the following is set during the query:

- attributes=**general**
- attributes=*
- attributes=**general**.*maximumSourceTransfers*

operatingSystem
String

Specifies the operating system where the agent queue manager is created

There is no default value for this attribute

This attribute is returned only when the any of the following is set during the query:

- attributes=**general**
- attributes=*
- attributes=**general**.*operatingSystem*

qmgrConnection

This object provides information about the queue manager connections.

qmgrName
String

Specifies the name of the agent queue manager

There is no default value for this attribute

This attribute is returned only when the any of the following is set during the query:

- attributes=**qmgrConnection**
- attributes=*
- attributes=**qmgrConnection**.*qmgrName*

transportType
String

Specifies the transport type in which the agent is connecting with the queue manager; the transport type can be client or bindings

The default value is *bindings*

This attribute is returned only when the any of the following is set during the query:

- attributes=**qmgrConnection**
- attributes=*
- attributes=**qmgrConnection**.*transportType*

host
String

Specifies the agent queue manager host name; applicable only if **transportType** is client

There is no default value for this attribute

This attribute is returned only when the any of the following is set during the query:

- attributes=**qmgrConnection**
- attributes=*
- attributes=**qmgrConnection**.*host*

port
integer

Specifies the agent queue manager channel communication port; applicable only if **transportType** is client

There is no default value for this attribute

This attribute is returned only when the any of the following is set during the query:

- attributes=**qmgrConnection**
- attributes=*
- attributes=**qmgrConnection**.*port*

channelName
String

Specifies the agent queue manager channel; applicable only if **transportType** is client

The default value for this attribute is set to SYSTEM.DEF.SVRCONN

This attribute is returned only when the any of the following is set during the query:

- attributes=**qmgrConnection**
- attributes=*
- attributes=**qmgrConnection**.*channelName*

standbyHost
String

Specifies the host name used by client connections, to connect to the standby instance of a multi-instance agent queue manager

There is no default value for this attribute

This attribute is returned only when the any of the following is set during the query:

- attributes=**qmgrConnection**
- attributes=*
- attributes=**qmgrConnection**.*standbyHost*

standbyPort
Integer

Specifies the port number through which a client can connect to the standby instance of a multi-instance agent queue manager

The default value for this attribute is set to -1

This attribute is returned only when the any of the following is set during the query:

- attributes=**qmgrConnection**
- attributes=*
- attributes=**qmgrConnection**.*standbyPort*

connectDirectBridge

This object provides information about to connect direct bridge type agent. For other type of agents this object is not added.

nodeName
String

Specifies the name of the Connect:Direct node to use, to transfer messages from this agent to the destination Connect:Direct nodes.

There is no default value for this attribute

This attribute is returned only when the any of the following is set during the query:

- attributes=**connectDirectBridge**
- attributes=*
- attributes=**connectDirectBridge**.*nodeName*

host
String

Specifies the host name or IP address of the system where the Connect:Direct node, specified by the **-cdNode** parameter, is located.

If you do not specify the **-cdNodeHost** parameter, a default of the host name or IP address of the local system is used.

The default value for this attribute are the details of the host where it is configured, for example, `localhost`

This attribute is returned only when the any of the following is set during the query:

- attributes=**connectDirectBridge**
- attributes=*
- attributes=**connectDirectBridge**.*host*

port
Integer

Specifies the port number of the Connect:Direct node that client applications use to communicate with the node.

The default value for this attribute is 1363

This attribute is returned only when the any of the following is set during the query:

- attributes=**connectDirectBridge**
- attributes=*
- attributes=**connectDirectBridge**.*port*

protocolBridge

This object provides information about protocol bridge type agent. For other type of agents this object is not added.

endpoints
String

Specifies the number of endpoints the bridge can support.

Note: If you have not set the default protocol server, the **defaultServer** field is not available.

The default value for this attribute is *multiple* from Version 7.0.1

This attribute is returned only when the any of the following is set during the query:

- attributes=**protocolBridge**
- attributes=*
- attributes=**protocolBridge**.*endpoints*

defaultServer
String

Specifies the host name or IP address of the default protocol server if it is set. If the default protocol field is not set, this value is blank.

The value is a complete string containing the protocol type, server, and port, in the following format:

```
<protocolType>://<serverName or IP address>:<port>
```

For example:

```
"ftp://localhost:21"
```

There are no default values for this attribute

This attribute is returned only when the any of the following is set during the query:

- attributes=**protocolBridge**
- attributes=*
- attributes=**protocolBridge**.*defaultServer*

Related reference:

"GET - agent status" on page 1772
Use the HTTP GET method with the `agent` resource to request information about agents.

Related information:

Required configuration for the MFT REST API

GET - list of transfers: V 9.0.5

Use the HTTP GET method with the `transfer` resource to request information about the list of transfers.

The information that is returned is similar to the information returned by the fteListScheduledTransfers command.

- Resource URL
- Optional query parameters
- "Request headers"
- Request body format
- "Security requirements" on page 1785
- Response status codes
- "Response headers" on page 1785
- Response body format
- Examples

Resource URL

`https://host:port/ibmmq/rest/v1/admin/mft/transfer`

`https://host:port/ibmmq/rest/v1/admin/mft/?before={transferId}&limit=10&attributes=*`

You can use HTTP instead of HTTPS if you enable HTTP connections. For more information about enabling HTTP, see Configuring the HTTP and HTTPS ports.

For more information about configuring the MFT REST service, see "MFT REST API configuration parameters" on page 1770

Optional query parameters

`limit`
The maximum number of transfers to be retrieved. For example, if the limit=200, the REST API returns a maximum of 200 transfers.

`after`
Specifies the `transferId` of the transfer from where the transfer list has to fetched. All the transfers that are initiated after that particular transfer are fetched.

`before`
Specifies the `transferId` of the transfer from where the transfer list has to fetched. All the transfers that are initiated before that particular transfer are fetched.

Request headers

The following header must be sent with the request:

`Authorization`
This header must be sent if you are using basic authentication. For more information, see Using HTTP basic authentication with the REST API.

Request body format

None.

Security requirements

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MFTWebAdmin`, or `MFTWebAdminRO` roles. For more information about security for the administrative REST API, see IBM MQ Console and REST API security.

Response status codes

200 Transfer information retrieved successfully.

400 Invalid data provided.

For example, invalid queue attributes specified.

401 Not authenticated.

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MFTWebAdmin` or `MFTWebAdminRO` roles. For more information, see Security Requirements.

403 Not authorized.

The caller is authenticated to the mqweb server and is associated with a valid principal. However, the principal does not have access to all, or a subset of the required IBM MQ resources. For more information about the access that is required, see Security Requirements.

404 A transfer with the specified ID does not exist.

500 Server issue or error code from IBM MQ.

503 Queue manager not running.

Response headers

`Content-Type`
This header is returned with a value of `application/json;charset=utf-8`.

Response body format

The response is in JSON format in UTF-8 encoding. The response contains an outer JSON object that contains a single JSON array called `transfer`. Each element in the array is a JSON object that represents information about a transfer.

For more information, see "JSON response body attributes" on page 1790.

If an error occurs, see REST API error handling.

Examples

The following example returns a default set of data in the response

The following URL is used with the HTTP GET method:

```
https://localhost:9443/ibmmq/ibmmq/rest/v1/admin/mft/transfer/414d512050524d465444454d4f312020f5189c5921f22302
```

The following JSON response is returned:

```
{
 "transfer": [{
                "id": "414D512050524D465444454D4F312020F5189C5921F22302",
           "destinationAgent": {"name": "AGENT.TRI.BANK"},
```

```
            "originator": {
                      "host": "192.168.99.1",
                      "userId": "johndoe"
                          },
            "sourceAgent": {"name": "TESTAGENT"},
            "statistics": {
                           "endTime": "2018-01-08T16:22:15.569Z",
                           "numberOfFileFailures": 0,
                           "numberOfFileSuccesses": 2,
                           "numberOfFileWarnings": 0,
                           "numberOfFiles": 2,
                           "startTime": "2018-01-08T16:22:15.242Z"
                          },
            "status": {
                           "state": "successful"
                      }


 }]
}
```

The following example lists all the attributes for the specified transfer ID, on the coordination queue manager. The following URL is used with the HTTP GET method:

```
https://localhost:9443//ibmmq/rest/v1/admin/mft/transfer/414d512050524d465444454d4f312020c5c6705924cf9e02?attributes=*
```

The following JSON response is returned:

```
{
 "transfer": [{
                "id": "414D512050524D465444454D4F312020C5C6705924CF9E02",
  "sourceAgent": {
   "qmgrName": "PRMFTDEMO1",
   "name": "AGENT2"
  },
  "destinationAgent": {
   "qmgrName": "PRMFTDEMO1",
   "name": "AGENT1"
  },
  "originator": {
   "host": "192.168.56.1",
   "userId": "johndoe",
   "mqmdUserId": "johndoe"
  },
  "transferSet": {
   "item": [
                                                        {
    "source": {
     "file": {
      "lastModified": "2017-07-13T11:25:20.780Z",
      "size":179367055 ,
      "path": "D:/ProgramFiles/WASlibertyprofile.zip"
     },
     "checksum": {
      "method": "md5",
      "value": "5F0ED36FBD3C0E1F4083B12B34A318D3"
     },
     "disposition": "leave",
     "type": "file"
    },
    "destination": {
     "file": {
      "lastModified": "2017-07-28T08:00:12.065Z",
      "size": 179367055,
      "path": "C:/Users/IBMADMIN/Desktop/demo.zip"
     },
     "checksum": {
```

```
      "method": "md5",
      "value": "5F0ED36FBD3C0E1F4083B12B34A318D3"
     },
     "actionIfExists": "overwrite",
     "type": "file"
    },
                                                                  "status": {
                                                                                  "description": "BFGRP0032I: The file tra
                                                                                  "state":"successful"
                                                                               }
    "mode": "binary"
   }],
   "bytesSent": 0,
   "startTime": "2017-07-28T08:00:10.599Z"
    },
                               "job": {
                                          "name":"job1"
                                       },
                               "userProperties":{
},
  "status": {
                                            "lastStatusUpdate": "2017-07-28T08:00:10.599Z",
                                            "description": "BFGRP0032I: The file transfer request has successfully co

  },
  "statistics": {
   "startTime": "2017-07-28T08:00:09.897Z",
   "retryCount": 0,
                                            "endTime": "2017-07-28T08:00:10.599Z",
                                       "numberOfFilesSuccesses":1,
   "numberOfFileFailures": 0,
   "numberOfFileWarnings": 0,
                                            "numberOfFiles":1

  }
 }]
}
```

Related reference:

"JSON response body attributes" on page 1790
A description of the attributes available to the MFT REST API in the JSON response body.

GET - transfer status: V 9.0.5

Use the HTTP GET method with the `transfer` resource to request information about the status of transfers.

The information that is returned is similar to the information returned by the fteListScheduledTransfers command.

- Resource URL
- Optional query parameters
- "Request headers" on page 1788
- Request body format
- "Security requirements" on page 1788
- Response status codes
- "Response headers" on page 1789
- Response body format
- Examples

Resource URL

```
https://host:port/ibmmq/rest/v1/admin/mft/transfer/
{Transferid}
```

```
https://host:port/ibmmq/rest/v1/admin/mft/transfer/{TransferId}?&attributes=*
```

Important: `TransferId` is a required option for obtaining the status of a transfer, and is case insensitive.

For example,

```
https://host:port/ibmmq/rest/v1/admin/mft/transfer/
14d512050524d465444454d4f312020c5c6705924cf9e0
```

You can use HTTP instead of HTTPS if you enable HTTP connections. For more information about enabling HTTP, see Configuring the HTTP and HTTPS ports.

For more information about configuring the MFT REST service, see "MFT REST API configuration parameters" on page 1770

Optional query parameters

`attributes`
A comma separated list of attributes to be returned.

If you do not specify **`attributes`**, the default set of attributes is returned. See "JSON response body attributes" on page 1790 for a list of the available attributes.

Requesting the same attribute multiple times is an error.

If you make a request specifying attributes that are not valid for some of the transfers, or you set **`attributes`** to * this is permitted. However, if you request information on specific attributes which are not valid for that transfer, an error results.

Request headers

The following header must be sent with the request:

`Authorization`
This header must be sent if you are using basic authentication. For more information, see Using HTTP basic authentication with the REST API.

Request body format

None.

Security requirements

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MFTWebAdmin`, or `MFTWebAdminRO` roles. For more information about security for the administrative REST API, see IBM MQ Console and REST API security.

Response status codes

200 Transfer information retrieved successfully.

400 Invalid data provided.

For example, invalid queue attributes specified.

401 Not authenticated.

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MFTWebAdmin` or `MFTWebAdminRO` roles. For more information, see Security Requirements.

403 Not authorized.

The caller is authenticated to the mqweb server and is associated with a valid principal. However, the principal does not have access to all, or a subset of the required IBM MQ resources. For more information about the access that is required, see Security Requirements.

404 A transfer with the specified ID does not exist.

500 Server issue or error code from IBM MQ.

503 Queue manager not running.

Response headers

`Content-Type`
This header is returned with a value of `application/json;charset=utf-8`.

Response body format

The response is in JSON format in UTF-8 encoding. The response contains an outer JSON object that contains a single JSON array called `transfer`. Each element in the array is a JSON object that represents information about a transfer.

For more information, see "JSON response body attributes" on page 1790.

If an error occurs, see REST API error handling.

Examples

The following example returns a default set of data in the response

The following URL is used with the HTTP GET method:

```
https://localhost:9443/ibmmq/ibmmq/rest/v1/admin/mft/transfer/414d512050524d465444454d4f312020f5189c5921f22302
```

The following JSON response is returned:

```
{
 "transfer": [{
                "id": "414D512050524D465444454D4F312020F5189C5921F22302",
            "destinationAgent": {"name": "AGENT.TRI.BANK"},
            "originator": {
                      "host": "192.168.99.1",
                      "userId": "johndoe"
                          },
            "sourceAgent": {"name": "TESTAGENT"},
            "statistics": {
                           "endTime": "2018-01-08T16:22:15.569Z",
                           "numberOfFileFailures": 0,
                           "numberOfFileSuccesses": 2,
                           "numberOfFileWarnings": 0,
                           "numberOfFiles": 2,
                           "startTime": "2018-01-08T16:22:15.242Z"
                          },
            "status": {
                           "state": "successful"
                      }
```

```
 }]
}
```

Related reference:

"JSON response body attributes"
A description of the attributes available to the MFT REST API in the JSON response body.

JSON response body attributes: V 9.0.5

A description of the attributes available to the MFT REST API in the JSON response body.

Outer object attributes

id Specifies the unique transfer or transaction ID. The ID can be a maximum of 48 alphanumeric characters

job Job Name for the transfer (if specified)

sourceAgent
Specifies the name of the agent (along with other details) on the system where the source file is located

destinationAgent
Specifies the name of the agent (along with other details) on the system to which the file has been transferred

originator
Group element that contains the elements specifying the originator of the request

transferSet
Contains an array of items, that consists of all the information about the transfers, such as source filename, destination filename, along with each of their path location, size of file, and so on.

userProperties
Contains some additional meta data information about the transfer (if you provided this information before starting the transfer). For example: `"userProperties":{"key1":"value1"}`

status The state and description messages for the status of the transfer

statistics
Group element for statistical information for the transfer (when available)

Inner object attributes

Attention: The attributes marked `default` are always returned, and are part of the default JSON response. All other attributes are returned only if queried.

`sourceAgent`

qmgrName
The name of the queue manager on the source system

name (default)
The name of the agent on the source system.

`destinationAgent`

qmgrName
The name of the queue manager on the destination system

name (default)
The name of the agent on the destination system.

originator

Group element that contains the elements specifying the originator of the request.

host (default)
The host name of the system where the source file is located.

userID (default)
The user ID that originated the file transfer

mqmdUserId
The IBM MQ user ID that was supplied in the message descriptor (MQMD)

transferSet

Specifies a group of file transfers you want to perform together. During transmission, **transferSet** is a group element containing an array of item objects.

item Group element that contains elements specifying the source and destination file names and locations

bytesSent
Total bytes sent

startTime
Records the time that the set of transfers started, expressed in UTC format

item

source Group element that contains the **file** element or the **queue** element, and the **checksum** element for the file on the source system

destination
Group element that contains the **file** element or the **queue** element, and the **checksum** element for the file on the destination system

Only one of **file** and **queue** is present as a child element of destination

status The state for a transfer; that is for a particular item object inside the **transferSet**

mode Specifies the transfer mode as either binary or text

source

recursive
Specifies that files are transferred recursively in sub-directories, when the source element is a directory or contains wildcard characters.

disposition
Specifies the action that is taken on the source element when source has successfully been transferred to its destination. The valid options are as follows:

leave The source files are left unchanged

delete The source files are deleted from the source system after the source file is successfully transferred

file Specifies the absolute path of the file that was transferred. The fully-qualified path is in the format consistent with your operating system, for example `C:/from/here.txt`. Note that a file URI is not used.

The valid options are as follows:

lastModified
Last modified date and time for the file (UTC format)

size File size

path Path location for the file

encoding
The encoding for a text file transfer

endOfLine
Specifies the end of line marker. The permitted values are:

- LF - line feed character only
- CRLF - carriage return and line feed character sequence

checksum
checksum does not appear if a checksum was not performed.

Specifies the type of hash algorithm that generated the message digest to create the digital signature. Managed File Transfer supports Message Digest algorithm 5 (md5) only. The checksum provides a way for you to confirm the integrity of transferred files is intact.

The valid options are:

method
Method used for generating **checksum**

value Checksum value generated

type Specifies the type of source. The valid options are as follows:

queue Specifies an IBM MQ queue as the source

file Specifies a file as the source, if the source is a file or directory

dataset
Specifies a z/OS dataset as the source

dataset
Specifies a z/OS dataset. The valid options are as follows:

attributes
Attributes related to the dataset

size File size

name Name of the dataset

queue when used with the **source** element

Specifies the name of the queue that the transferred messages were read from, which is located on the source agent queue manager.

messageCount
The number of messages that were read from the queue

name Name of the queue along with the queue manager name, as shown
queueName@queueManagerName

setMqProperties
A boolean operator specifying whether IBM MQ message properties are set on the first message in a file, and any messages written to the queue when an error occurs.

destination

actionIfExists
Specifies the action that is taken if a destination file exists on the destination system. The valid options are as follows:

error Reports an error and the file is not transferred

overwrite
Overwrites the existing destination file

file Specifies the absolute path of the file that was transferred. The fully-qualified path is in the format consistent with your operating system, for example `C:/from/here.txt`. Note that a file URI is not used.

The valid options are as follows:

lastModified
Last modified date and time for the file (UTC format)

size File size

path Path location for the file

checksum
`checksum` does not appear if a checksum was not performed.

Specifies the type of hash algorithm that generated the message digest to create the digital signature. Managed File Transfer supports Message Digest algorithm 5 (md5) only. The checksum provides a way for you to confirm the integrity of transferred files is intact.

The valid options are:

method
Method used for generating **`checksum`**

value Checksum value generated

type Specifies the type of source. The valid options are as follows:

queue Specifies an IBM MQ queue as the source

file Specifies a file as the source, if the source is a file or directory

dataset
Specifies a z/OS dataset as the source

dataset
Specifies a z/OS dataset. The valid options are as follows:

attributes
Attributes related to the dataset

size File size

name Name of the dataset

`queue` when used with the **`destination`** element

Specifies the name of the queue that was transferred to, which is located on any queue manager that is connected to the destination agent queue manager.

messageCount
The number of messages that were written to the queue

messageLength
The length of the message written to the queue

name Name of the queue along with the queue manager name, as shown

queueName@queueManagerName

messageOrGroupId
If the transfer request did not specify that the file is split into multiple messages, the value of this attribute is the IBM MQ message ID of the message written to the queue.

If the transfer request specified that the file is split into multiple messages, the value of this attribute is the IBM MQ group ID of the messages written to the queue.

delimiter
If **delimiterType**.*size*, for example 1K

If **delimiterType**.*binary*, for example 12

If **delimiter** is an empty string, that is, "", the field is not set while initiating the transfer

delimiterType
Type of delimiter being used to split the messages. The valid values are:

size Split by size

binary Split by delimiter bytes

If **delimiterType** is an empty string, that is, "", the field is not set while initiating the transfer

includeDelimiterInMessage
Valid only for **delimiterType**.*binary*.

This option can be *true* or *false*. For example:

```
"includeDelimiterInMessage" : true
```

delimiterPosition
Valid only for **delimiterType**.*binary*. The valid values are:

"prefix"
Before each message

"postfix"
After each message

If **delimiterPosition** is an empty string, that is, "", the field is not set while initiating the transfer

Note, that if **delimiterType** is *size*, both **includeDelimiterInMesage**, and **delimiterPosition** are not included in the JSON.

status

Group element for status information for the transfer.

state (default)
The state of the transfer. The value can be one of the following:

- started
- inProgress
- successful
- failed
- partiallySuccessful
- cancelled
- malformed - indicates that the file transfer request message content can not be interpreted
- notAuthorized
- deleted
- inProgressWithFailures

- inProgressWithWarnings

lastStatusUpdate
Most recent time when the transfer status was captured, expressed in UTC format

description
More detailed information about the status completion. Whether it is:

- Partially successful
- Successful
- Failed, or
- Any other relevant information

`statistics`

Group element for statistical information for the transfer (when available).

startTime (default)
The time when the transfer was submitted (UTC format)

retryCount
The number of times that the transfer went into the recovery state and was retried by the agent.

A transfer can go into a recovery state because the source and destination agents lose communication, either because of an IBM MQ network error, or because the agents are not receiving data or acknowledgment messages for a period.

This period is determined by the agent properties: **`transferAckTimeout`** and **`transferAckTimeoutRetries`**.

numberOfFilefailures (default)
The number of files in the **`transferSet`** that failed to transfer successfully

numberOfFileWarnings (default)
The number of files in the **`transferSet`** that generated warnings while being transferred, but otherwise transferred successfully

numberOfFiles (default)
This number denotes the total number of files included in the current transfer request. This number includes all the files considered for the transfer operation

endTime (default)
The time when the transfer was completed. This field gets updated only when the transfer is complete.

If the transfer is in any other state, then **`"endTime"`** will be an empty string

numberOfFileSuccesses (default)
The number of files successfully transferred

Related reference:

"/admin/mft/agent" on page 1765
You can use the HTTP GET method with the agent resource, to request information about the status of agents, and other attribute details.

Related information:

Required configuration for the MFT REST API

REST API and PCF equivalents

V 9.0.2

For most REST API optional query parameters and attributes, an equivalent PCF parameter or attribute exists. Use these topics to understand these equivalents.

REST API and PCF equivalents for queue managers: V 9.0.3

For most REST API optional query parameters and attributes for queue managers, an equivalent PCF parameter or attribute exists. Use the tables that are provided to understand these equivalents.

- "Queue manager attribute equivalents"
- "Unsupported PCF attributes"

Queue manager attribute equivalents

Table 138. Queue manager attributes for the REST API and equivalent PCF attributes.

REST API attribute	PCF attribute	Related values (REST API)	Related values (PCF)
`name`	**MQCA_Q_MGR_NAME**		
`state`	**MQIACF_Q_MGR_STATUS**		
`status.started`	**MQCACF_Q_MGR_START_DATE** **MQCACF_Q_MGR_START_TIME**		
`status.channelInitiatorState`	**MQIACF_CHINIT_STATUS**	`MQSVC_STATUS_STOPPED` `MQSVC_STATUS_STARTING` `MQSVC_STATUS_RUNNING` `MQSVC_STATUS_STOPPING`	`stopped` `starting` `running` `stopping`
`status.ldapConnectionState`	**MQIACF_LDAP_CONNECTION_STATUS**	`MQLDAPC_CONNECTED` `MQLDAPC_ERROR` `MQLDAPC_INACTIVE`	`connected` `error` `disconnected`
`status.connectionCount`	**MQIACF_CONNECTION_COUNT**		

Unsupported PCF attributes

The following queue manager PCF attributes are not supported by the administrative REST API `qmgr` resource:

- **MQCA_INSTALLATION_DESC**
- **MQCA_INSTALLATION_NAME**
- **MQCA_INSTALLATION_PATH**
- **MQCACF_CURRENT_LOG_EXTENT_NAME**
- **MQCACF_LOG_PATH**
- **MQCACF_MEDIA_LOG_EXTENT_NAME**
- **MQCACF_RESTART_LOG_EXTENT_NAME**

REST API and PCF equivalents for queues: V 9.0.2

For most REST API optional query parameters and attributes for queues, an equivalent PCF parameter or attribute exists. Use the tables that are provided to understand these equivalents.

- "Optional query parameter equivalents"
- "Queue attribute equivalents"
- "Unsupported PCF attributes" on page 1801

Optional query parameter equivalents

Table 139. Queue optional query parameters for the REST API and equivalent PCF parameters.

REST API optional query parameter	PCF parameter	Related values (REST API)	Related values (PCF)
commandScope=*scope*	**MQCACF_COMMAND_SCOPE**	None.	None.
filter=*filterValue*	**MQCFT_INTEGER_FILTER** **MQCFT_STRING_FILTER**	lessThan greaterThan lessThanOrEqualTo greaterThanOrEqualTo equalTo notEqualTo	MQCFOP_LESS MQCFOP_GREATERMQCFOP_NOT_GREATER MQCFOP_NOT_LESS MQCFOP_EQUAL MQCFOP_LIKE MQCFOP_NOT_EQUAL MQCFOP_NOT_LIKE
force	**MQIACF_FORCE**		
keepAuthorityRecords	**MQIACF_REMOVE_AUTHREC**		
like=*queueName*	**MQCACF_FROM_Q_NAME**		
noReplace	**MQIACF_REPLACE**		
purge	**MQIACF_PURGE**		
queueSharingGroupDisposition=*qsgDisposition*	**MQIA_QSG_DISP**	live all copy group private qmgr shared	MQQSGD_LIVE MQQSGD_ALL MQQSGD_COPY MQQSGD_GROUP MQQSGD_PRIVATE MQQSGD_Q_MGR MQQSGD_SHARED
type=*type*	**MQIA_Q_TYPE**	all local alias remote cluster model	None. MQQT_LOCAL MQQT_ALIAS MQQT_REMOTE MQQT_CLUSTER MQQT_MODEL

Queue attribute equivalents

Table 140. Queue attributes for the REST API and equivalent PCF attributes.

REST API attribute	PCF attribute	Related values (REST API)	Related values (PCF)
name	**MQCA_Q_NAME**		
type	**MQIA_Q_TYPE**	local alias remote cluster model	MQQT_LOCAL MQQT_ALIAS MQQT_REMOTE MQQT_CLUSTER MQQT_MODEL
remote.qmgrName	**MQCA_REMOTE_Q_MGR_NAME**		

Table 140. Queue attributes for the REST API and equivalent PCF attributes. (continued)

REST API attribute	PCF attribute	Related values (REST API)	Related values (PCF)
remote.queueName	**MQCA_REMOTE_Q_NAME**		
remote.transmissionQueueName	**MQCA_XMIT_Q_NAME**		
alias.targetName	**MQCA_BASE_OBJECT_NAME**		
alias.targetType	**MQIA_BASE_TYPE**	queue topic	MQOT_Q MQOT_TOPIC
dynamic.type	**MQIA_DEFINITION_TYPE**	permanentDynamic sharedDynamic temporaryDynamic	MQQDT_PERMANENT_DYNAMIC MQQDT_SHARED_DYNAMIC MQQDT_TEMPORARY_DYNAMIC
model.type	**MQIA_DEFINITION_TYPE**	permanentDynamic sharedDynamic temporaryDynamic	MQQDT_PERMANENT_DYNAMIC MQQDT_SHARED_DYNAMIC MQQDT_TEMPORARY_DYNAMIC
cluster.name	**MQCA_CLUSTER_NAME**		
cluster.namelist	**MQCA_CLUSTER_NAMELIST**		
cluster.qmgrId	**QMgrIdentifier**		
cluster.qmgrName	**QMgrName**		
cluster.queueType	**ClusterQType**	local alias remote qmgrAlias	MQCQT_LOCAL_Q MQCQT_ALIAS_Q MQCQT_REMOTE_Q MQCQT_Q_MGR_ALIAS
cluster.transmissionQueueForChannelName	**ClusterChannelName**		
cluster.workloadPriority	**MQIA_CLWL_Q_PRIORITY**		
cluster.workloadQueueUse	**MQIA_CLWL_USEQ**	true false	MQTC_ON MQTC_OFF
cluster.workloadRank	**MQIA_CLWL_Q_RANK**		
trigger.enabled	**MQIA_TRIGGER_CONTROL**	true false	MQTC_ON MQTC_OFF
trigger.data	**MQCA_TRIGGER_DATA**		
trigger.depth	**MQIA_TRIGGER_DEPTH**		
trigger.initiationQueueName	**MQCA_INITIATION_Q_NAME**		
trigger.messagePriority	**MQIA_TRIGGER_MSG_PRIORITY**		
trigger.processName	**MQCA_PROCESS_NAME**		
trigger.type	**MQIA_TRIGGER_TYPE**	none every first depth	MQTT_NONE MQTT_EVERY MQTT_FIRST MQTT_DEPTH
events.depth.highEnabled	**MQIA_Q_DEPTH_HIGH_EVENT**	true false	MQEVR_ENABLED MQEVR_DISABLED
events.depth.highPercentage	**MQIA_Q_DEPTH_HIGH_LIMIT**		
events.depth.lowEnabled	**MQIA_Q_DEPTH_LOW_EVENT**	true false	MQEVR_ENABLED MQEVR_DISABLED
events.depth.lowPercentage	**MQIA_Q_DEPTH_LOW_LIMIT**		
events.depth.fullEnabled	**MQIA_Q_DEPTH_MAX_EVENT**	true false	MQEVR_ENABLED MQEVR_DISABLED
events.serviceInterval.highEnabled	**MQIA_Q_SERVICE_INTERVAL_EVENT**	true false	MQSIE_HIGH MQSIE_NONE (Equivalent only when okEnabled

Table 140. Queue attributes for the REST API and equivalent PCF attributes. (continued)

REST API attribute	PCF attribute	Related values (REST API)	Related values (PCF)
events.serviceInterval.okEnabled	**MQIA_Q_SERVICE_INTERVAL_EVENT**	true false	MQSIE_OK MQSIE_NONE (Equivalent only when highEn
events.serviceInterval.duration	**MQIA_Q_SERVICE_INTERVAL**		
applicationDefaults.clusterBind	**MQIA_DEF_BIND**	onOpen notFixed onGroup	MQBND_BIND_ON_OPEN MQBND_BIND_NOT_FIXED MQBND_BIND_ON_GROUP
applicationDefaults.messageBodyControl	**MQIA_PROPERTY_CONTROL**	all compatible force none version6Compatible	MQPROP_ALL MQPROP_COMPATIBILITY MQPROP_FORCE_MQRFH2 MQPROP_NONE MQPROP_V6COMPAT
applicationDefaults.messagePersistence	**MQIA_DEF_PERSISTENCE**	persistent nonPersistent	MQPER_PERSISTENT MQPER_NOT_PERSISTENT
applicationDefaults.messagePriority	**MQIA_DEF_PRIORITY**		
applicationDefaults.putResponse	**MQIA_DEF_PUT_RESPONSE_TYPE**	synchronous asynchronous	MQPRT_SYNC_RESPONSE MQPRT_ASYNC_RESPONSE
applicationDefaults.readAhead	**MQIA_DEF_READ_AHEAD**	no yes disabled	MQREADA_NO MQREADA_YES MQREADA_DISABLED
applicationDefaults.sharedInput	**MQIA_DEF_INPUT_OPEN_OPTION**	true false	MQOO_INPUT_SHARED MQOO_INPUT_EXCLUSIVE
queueSharingGroup.disposition	**MQIA_QSG_DISP**	copy group qmgr shared	MQQSGD_COPY MQQSGD_GROUP MQQSGD_Q_MGR MQQSGD_SHARED
queueSharingGroup.qmgrName	No equivalent.		
queueSharingGroup.structureName	**MQCA_CF_STRUC_NAME**		
dataCollection.accounting	**MQIA_ACCOUNTING_Q**	asQmgr off on	MQMON_Q_MGR MQMON_OFF MQMON_ON
dataCollection.monitoring	**MQIA_MONITORING_Q**	off asQmgr low medium high	MQMON_OFF MQMON_Q_MGR MQMON_LOW MQMON_MEDIUM MQMON_HIGH
dataCollection.statistics	**MQIA_STATISTICS_Q**	asQmgr off on	MQMON_Q_MGR MQMON_OFF MQMON_ON
storage.indexType	**MQIA_INDEX_TYPE**	none correlationId groupId messageId messageToken	MQIT_NONE MQIT_CORREL_ID MQIT_GROUP_ID MQIT_MSG_ID MQIT_MSG_TOKEN
storage.maximumMessageLength	**MQIA_MAX_MSG_LENGTH**		
storage.maximumDepth	**MQIA_MAX_Q_DEPTH**		
storage.messageDeliverySequence	**MQIA_MSG_DELIVERY_SEQUENCE**	priority fifo	MQMDS_PRIORITY MQMDS_FIFO
storage.nonPersistentMessageClass	**MQIA_NPM_CLASS**	normal high	MQNPM_CLASS_NORMAL MQNPM_CLASS_HIGH

Table 140. Queue attributes for the REST API and equivalent PCF attributes. (continued)

REST API attribute	PCF attribute	Related values (REST API)	Related values (PCF)
storage.pageSet	**PageSetID**		
storage.storageClass	**MQCA_STORAGE_CLASS**		
general.description	**MQCA_Q_DESC**		
general.inhibitGet	**MQIA_INHIBIT_GET**	true false	MQQA_GET_INHIBITED MQQA_GET_ALLOWED
general.inhibitPut	**MQIA_INHIBIT_PUT**	true false	MQQA_PUT_INHIBITED MQQA_PUT_ALLOWED
general.isTransmissionQueue	**MQIA_USAGE**	true false	MQUS_TRANSMISSION MQUS_NORMAL
extended.allowSharedInput	**MQIA_SHAREABILITY**	true false	MQQA_SHAREABLE MQQA_NOT_SHAREABLE
extended.backoutRequeueQueueName	**MQCA_BACKOUT_REQ_Q_NAME**		
extended.backoutThreshold	**MQIA_BACKOUT_THRESHOLD**		
extended.custom	**MQCA_CUSTOM**		
extended.supportDistributionLists	**MQIA_DIST_LISTS**	true false	MQDL_SUPPORTED MQDL_NOT_SUPPORTED
extended.hardenGetBackout	**MQIA_HARDEN_GET_BACKOUT**	true false	MQQA_BACKOUT_HARDENED MQQA_BACKOUT_NOT_HARDENED
extended.enableMediaImageOperations	**ImageRecoverQueue**	yes no asQmgr	MQIMGRCOV_YES MQIMGRCOV_NO MQIMGRCOV_AS_QMGR
timestamps.altered	**MQCA_ALTERATION_DATE** **MQCA_ALTERATION_TIME**		
timestamps.clustered	**MQCA_CLUSTER_DATE** **MQCA_CLUSTER_TIME**		
timestamps.created	**MQCA_CREATION_DATE** **MQCA_CREATION_TIME**		
status.currentDepth	**MQIA_CURRENT_Q_DEPTH**		
status.lastGet	**MQCACF_LAST_GET_DATE** **MQCACF_LAST_GET_TIME**		
status.lastPut	**MQCACF_LAST_PUT_DATE** **MQCACF_LAST_PUT_TIME**		
status.mediaRecoveryLogExtent	**MQCACF_MEDIA_LOG_EXTENT_NAME**		
status.oldestMessageAge	**MQIACF_OLDEST_MSG_AGE**		
status.onQueueTime.longSamplePeriod	**MQIACF_Q_TIME_INDICATOR**		
status.onQueueTime.shortSamplePeriod	**MQIACF_Q_TIME_INDICATOR**		
status.openInputCount	**MQIA_OPEN_INPUT_COUNT**		
status.openOutputCount	**MQIA_OPEN_OUTPUT_COUNT**		
status.monitoringRate	**MQIA_MONITORING_Q**	off low medium high	MQMON_OFF MQMON_LOW MQMON_MEDIUM MQMON_HIGH
status.tPipeName	**MQCA_TPIPE_NAME**		
status.uncommittedMessages	**MQIACF_UNCOMMITTED_MSGS**		

Table 140. Queue attributes for the REST API and equivalent PCF attributes. (continued)

REST API attribute	PCF attribute	Related values (REST API)	Related values (PCF)
applicationHandle.description	**MQCACF_APPL_DESC**		
applicationHandle.tag	**MQCACF_APPL_TAG**		
applicationHandle.type	**MQIA_APPL_TYPE**	queueManagerProcess channelInitiator userApplication batchConnection rrsBatchConnection cicsTransaction imsTransaction SystemExtension	MQAT_QMGR MQAT_CHANNEL_INITIATOR MQAT_USER MQAT_BATCH MQAT_RRS_BATCH MQAT_CICS MQAT_IMS MQAT_SYSTEM_EXTENSION
applicationHandle.asynchronousConsumerState	**MQIACF_ASYNC_STATE**	active inactive suspended suspendedTemporarily none	MQAS_ACTIVE MQAS_INACTIVE MQAS_SUSPENDED MQAS_SUSPENDED_TEMPORARY MQAS_NONE
applicationHandle.addressSpaceId	**MQIACF_ASID**		
applicationHandle.channelName	**MQCACH_CHANNEL_NAME**		
applicationHandle.connectionName	**MQCACH_CONNECTION_NAME**		
applicationHandle.state	**MQIACF_HANDLE_STATE**	active inactive	MQHSTATE_ACTIVE MQHSTATE_INACTIVE
applicationHandle.openOptions	**MQIACF_OPEN_OPTIONS**		
applicationHandle.processId	**MQIACF_PROCESS_ID**		
applicationHandle.processSpecificName	**MQCACF_PSB_NAME**		
applicationHandle.processSpecificId	**MQCACF_PST_ID**		
applicationHandle.qmgrTransactionId	**MQBACF_Q_MGR_UOW_ID**		
applicationHandle.cicsTaskNumber	**MQIACF_TASK_NUMBER**		
applicationHandle.threadId	**MQIACF_THREAD_ID**		
applicationHandle.cicsTransactionId	**MQCACF_TRANSACTION_ID**		
applicationHandle.unitOfWorkId	**MQBACF_EXTERNAL_UOW_ID**		
applicationHandle.unitOfWorkType	**MQIACF_UOW_TYPE**	qmgr cics ims rrs xa	MQUOWT_Q_MGR MQUOWT_CICS MQUOWT_IMS MQUOWT_RRS MQUOWT_XA
applicationHandle.UserId	**MQCACF_USER_IDENTIFIER**		

Unsupported PCF attributes

The following queue PCF attributes are not supported by the administrative REST API:

- **MQIA_SCOPE**
- **MQIA_RETENTION_INTERVAL**

REST API and PCF equivalents for subscriptions: V 9.0.4

For most REST API optional query parameters and attributes for subscriptions, an equivalent PCF parameter or attribute exists. Use the tables that are provided to understand these equivalents.

- "Optional query parameter equivalents"
- "Subscription attribute equivalents"
- "Unsupported PCF parameters" on page 1803

Optional query parameter equivalents

Table 141. Subscription optional query parameters for the REST API and equivalent PCF parameters.

REST API optional query parameter	PCF parameter	Related values (REST API)	Related values (PCF)
filter=*filterValue*	**MQCFT_INTEGER_FILTER** **MQCFT_STRING_FILTER**	lessThan greaterThan lessThanOrEqualTo greaterThanOrEqualTo equalTo notEqualTo	MQCFOP_LESS MQCFOP_GREATER MQCFOP_NOT_GREATER MQCFOP_NOT_LESS MQCFOP_EQUAL MQCFOP_LIKE MQCFOP_NOT_EQUAL MQCFOP_NOT_LIKE

Subscription attribute equivalents

Table 142. Subscription attributes for the REST API and equivalent PCF attributes.

REST API attribute	PCF attribute	Related values (REST API)	Related values (PCF)
name	**MQCACF_SUB_NAME**		
id	**MQBACF_SUB_ID**		
resolvedTopicString	**MQCA_TOPIC_STRING**		
topic.name	**MQCA_TOPIC_NAME**		
topic.definedString	**MQCA_TOPIC_STRING**		
selector.value	**MQCACF_SUB_SELECTOR**		
selector.type	**MQIACF_SELECTOR_TYPE**	none standard extended	MQSELTYPE_NONE MQSELTYPE_STANDARD MQSELTYPE_EXTENDED
destination.isManaged	**MQIACF_DESTINATION_CLASS**	true false	MQDC_MANAGED MQDC_PROVIDED
destination.qmgrName	**MQCACF_DESTINATION_Q_MGR**		
destination.name	**MQCACF_DESTINATION**		
destination.correlationId	**MQBACF_DESTINATION_CORREL_ID**		
user.accountingToken	**MQBACF_ACCOUNTING_TOKEN**		
user.applicationIdentityData	**MQCACF_APPL_IDENTITY_DATA**		
user.data	**MQCACF_SUB_USER_DATA**		
user.name	**MQCACF_SUB_USER_ID**		
user.isVariable	**MQIACF_VARIABLE_USER_ID**	true false	MQVU_ANY_USER MQVU_FIXED_USER
general.isDurable	**MQIACF_DURABLE_SUBSCRIPTION**	true false	MQSUB_DURABLE_YES MQSUB_DURABLE_NO

Table 142. Subscription attributes for the REST API and equivalent PCF attributes. (continued)

REST API attribute	PCF attribute	Related values (REST API)	Related values (PCF)
general.type	**MQIACF_SUB_TYPE**	administrative api proxy	MQSUBTYPE_ADMIN MQSUBTYPE_API MQSUBTYPE_PROXY
general.usesCharacterWildcard	**MQIACF_WILDCARD_SCHEMA**	true false	MQWS_CHAR MQWS_TOPIC
extended.expiry	**MQIACF_EXPIRY**		
extended.level	**MQIACF_SUB_LEVEL**		
extended.messagePriority	**MQIACF_PUB_PRIORITY**	asPublished asQueue	MQPRI_PRIORITY_AS_PUBLISHED MQPR_PRIORITY_AS_QDEF
extended.messagePropertyControl	**MQIACF_PUBSUB_PROPERTIES**	none compatible pcf rfh2	MQPSPROP_NONE MQPSPROP_COMPAT MQPSPROP_MSGPROP MQPSPROP_RFH2
extended.deliverOnRequest	**MQIACF_REQUEST_ONLY**	true false	MQRU_PUBLISH_ON_REQUEST MQRU_PUBLISH_ALL
extended.networkScope	**MQIACF_SUBSCRIPTION_SCOPE**	all qmgr	MQTSCOPE_ALL MQTSCOPE_QMGR
timestamps.altered	**MQCA_ALTERATION_DATE** **MQCA_ALTERATION_TIME**		
timestamps.created	**MQCA_CREATION_DATE** **MQCA_CREATION_TIME**		

Unsupported PCF parameters

The following subscription PCF inquire parameters are not supported by the administrative REST API:

- **MQIA_DISPLAY_TYPE**
- **MQIACF_SUB_TYPE**
- **MQIACF_SUB_ATTRS**

REST API and PCF equivalents for channels: V 9.0.4

For most REST API optional query parameters and attributes for channels, an equivalent PCF parameter or attribute exists. Use the tables that are provided to understand these equivalents.

- "Optional query parameter equivalents" on page 1804
- "Channel attribute equivalents" on page 1804
- "Unsupported PCF parameters" on page 1810

Optional query parameter equivalents

Table 143. Channel optional query parameters for the REST API and equivalent PCF parameters.

REST API optional query parameter	PCF parameter	Related values (REST API)	Related values (PCF)
filter=*filterValue*	**MQCFT_INTEGER_FILTER** **MQCFT_STRING_FILTER**	lessThan greaterThan lessThanOrEqualTo greaterThanOrEqualTo equalTo notEqualTo	MQCFOP_LESS MQCFOP_GREATER MQCFOP_NOT_GREATER MQCFOP_NOT_LESS MQCFOP_EQUAL MQCFOP_LIKE MQCFOP_NOT_EQUAL MQCFOP_NOT_LIKE
type=*type*	**MQIACH_CHANNEL_TYPE**	all sender receiver server requester clusterSender clusterReceiver	None. MQCHT_SENDER MQCHT_RECEIVER MQCHT_SERVER MQCHT_REQUESTER MQCHT_CLUSSDR MQCHT_CLUSRCVR
queueSharingGroupDisposition=*qsgDisp*	**MQIA_QSG_DISP**	live all copy group private qmgr	MQQSGD_LIVE MQQSGD_ALL MQQSGD_COPY MQQSGD_GROUP MQQSGD_PRIVATE MQQSGD_Q_MGR

Channel attribute equivalents

Table 144. Channel attributes for the REST API and equivalent PCF attributes.

REST API attribute	PCF attribute	Related values (REST API)	Related values (PCF)
name	**MQIACH_CHANNEL_NAME**		
type	**MQIACH_CHANNEL_TYPE**		
clusterRouting.workloadPriority	**MQIACH_CLWL_CHANNEL_PRIORITY**		
clusterRouting.workloadRank	**MQIACH_CLWL_CHANNEL_RANK**		
clusterRouting.workloadWeight	**MQIACH_CLWL_CHANNEL_WEIGHT**		
clusterRouting.networkPriority	**MQIACH_NETWORK_PRIORITY**		
[type].connection.host [type].connection.port sender.connection.host sender.connection.port server.connection.host server.connection.port requester.connection.host requester.connection.port clusterSender.connection.host clusterSender.connection.port clusterReceiver.connection.host clusterReceiver.connection.port	**MQCACH_CONNECTION_NAME**		

Table 144. Channel attributes for the REST API and equivalent PCF attributes. (continued)

REST API attribute	PCF attribute	Related values (REST API)	Related values (PCF)
[type].transmissionQueueName sender.transmissionQueueName server.transmissionQueueName	**MQCACH_XMIT_Q_NAME**		
clusterSender.clusterName clusterReceiver.clusterName	**MQCA_CLUSTER_NAME**		
clusterSender.clusterNamelist clusterReceiver.clusterNamelist	**MQCA_CLUSTER_NAMELIST**		
connectionManagement.heartbeatInterval	**MQIACH_HB_INTERVAL**		
connectionManagement.disconnectInterval	**MQIACH_DISC_INTERVAL**		
connectionManagement.keepAliveInterval	**MQIACH_KEEP_ALIVE_INTERVAL**		
connectionManagement.localAddress.host connectionManagement.localAddress.port connectionManagement.localAddress.portRange	**MQCACH_LOCAL_ADDRESS**		
connectionManagement.longRetryCount	**MQIACH_LONG_RETRY**		
connectionManagement.longRetryInterval	**MQIACH_LONG_TIMER**		
connectionManagement.shortRetryCount	**MQIACH_SHORT_RETRY**		
connectionManagement.shortRetryInterval	**MQIACH_SHORT_TIMER**		
compression.header	**MQIACH_HDR_COMPRESSION**	none system	MQCOMPRESS_NONE MQCOMPRESS_SYSTEM
compression.message	**MQIACH_MSG_COMPRESSION**	none runLengthEncoding zlibFast zlibHigh any	MQCOMPRESS_NONE MQCOMPRESS_RLE MQCOMPRESS_ZLIBFAST MQCOMPRESS_ZLIBHIGH MQCOMPRESS_ANY
dataCollection.monitoring	**MQIA_MONITORING_CHANNEL**	off asQmgr low medium high	MQMON_OFF MQMON_Q_MGR MQMON_LOW MQMON_MEDIUM MQMON_HIGH
dataCollection.statistics	**MQIA_STATISTICS_CHANNEL**	off asQmgr low medium high	MQMON_OFF MQMON_Q_MGR MQMON_LOW MQMON_MEDIUM MQMON_HIGH
exits.message.name	**MQCACH_MSG_EXIT_NAME**		
exits.message.userData	**MQCACH_MSG_EXIT_USER_DATA**		
exits.messageRetry.name	**MQCACH_MR_EXIT_NAME**		

Table 144. Channel attributes for the REST API and equivalent PCF attributes. (continued)

REST API attribute	PCF attribute	Related values (REST API)	Related values (PCF)
exits.messageRetry.userData	**MQCACH_MR_EXIT_USER_DATA**		
exits.receive.name	**MQCACH_RCV_EXIT_NAME**		
exits.receive.userData	**MQCACH_RCV_EXIT_USER_DATA**		
exits.security.name	**MQCACH_SEC_EXIT_NAME**		
exits.security.userData	**MQCACH_SEC_EXIT_USER_DATA**		
exits.send.name	**MQCACH_SEND_EXIT_NAME**		
exits.send.userData	**MQCACH_SEND_EXIT_USER_DATA**		
extended.channelAgentType	**MQIACH_MCA_TYPE**	process thread	MQMCAT_PROCESS MQMCAT_THREAD
extended.senderDataConversion	**MQIACH_DATA_CONVERSION**	false true	MQCDC_NO_SENDER_CONVERSION MQCDC_SENDER_CONVERSION
extended.messagePropertyControl	**MQIA_PROPERTY_CONTROL**	compatible none all	MQPROP_COMPATIBILITY MQPROP_NONE MQPROP_ALL
extended.sequenceNumberWrap	**MQIACH_SEQUENCE_NUMBER_WRAP**		
failedDelivery.retry.count	**MQIACH_MR_COUNT**		
failedDelivery.retry.interval	**MQIACH_MR_INTERVAL**		
failedDelivery.useDeadLetterQueue	**MQIA_USE_DEAD_LETTER_Q**	true false	MQUSEDLQ_YES MQUSEDLQ_NO
general.description	**MQCACH_DESC**		
general.maximumMessageLength	**MQIACH_MAX_MSG_LENGTH**		
batch.preCommitHeartbeat	**MQIACH_BATCH_HB**		
batch.timeExtend	**MQIACH_BATCH_INTERVAL**		
batch.dataLimit	**MQIACH_BATCH_DATA_LIMIT**		
batch.messageLimit	**MQIACH_BATCH_SIZE**		
batch.nonPersistentMessageSpeed currentStatus.batch.nonPersistentMessageSpeedFast	**MQIACH_NPM_SPEED**	true false	MQNPMS_FAST MQNPMS_NORMAL
queueSharingGroup.dispositionMQIA_QSG_DISP	**MQIA_QSG_DISP**	copy group qmgr	MQQSDG_COPY MQQSDG_GROUP MQQSDG_QMGR
queueSharingGroup.defaultChannelDisposition	**MQIACH_DEF_CHANNEL_DISP**	private fixShared shared	MQCHLD_PRIVATE MQCHLD_FIXSHARED MQCHLD_SHARED
receiverSecurity.channelAgentUserId	**MQCACH_MCA_USER_ID**		

Table 144. Channel attributes for the REST API and equivalent PCF attributes. (continued)

REST API attribute	PCF attribute	Related values (REST API)	Related values (PCF)
receiverSecurity.putAuthority	**MQCACH_MCA_USER_ID**	default context alternateOrChannelAgent onlyChannelAgent	MQPA_DEFAULT MQPA_CONTEXT MQPA_ALTERNATE_OR_MCA MQPA_ONLY_MCA
transmissionSecurity.certificateLabel	**MQCACH_CERT_LABEL**		
transmissionSecurity.cipherSpecification	**MQCACH_SSL_CIPHER_SPEC**		
transmissionSecurity.requireClientCertificate	**MQIACH_SSL_CLIENT_AUTH**	true false	MQSCA_REQUIRED MQSCA_OPTIONAL
transmissionSecurity.certificateDistinguishedName	**MQCACH_SSL_PEER_NAME**		
timestamps.altered	**MQCA_ALTERATION_DATE** **MQCA_ALTERATION_TIME**		
currentStatus.inDoubt savedStatus.inDoubt	**MQIACH_INDOUBT_STATUS**	true false	MQCHIDS_INDOUBT MQCHIDS_NOT_INDOUBT
currentStatus.state	**MQIACH_CHANNEL_STATUS**	binding starting running paused stopping retrying stopped requesting switching initializing	MQCHS_BINDING MQCHS_STARTING MQCHS_RUNNING MQCHS_PAUSED MQCHS_STOPPING MQCHS_RETRYING MQCHS_STOPPED MQCHS_REQUESTING MQCHS_SWITCHING MQCHS_INITIALIZING
currentStatus.agent.jobName	**MQCACH_MCA_JOB_NAME**		
currentStatus.agent.running	**MQIACH_MCA_STATUS**	true false	MQMCAS_RUNNING MQMCAS_STOPPED
currentStatus.agent.state	**MQIACH_CHANNEL_SUBSTATE**	runningChannelAutoDefinitionExit compressingData processingEndOfBatch performingSecurityHandshake heartbeating executingMQGET executingMQI executingMQPUT runningRetryExit runningMessageExit communicatingWithNameServer connectingToNetwork undefined runningReceiveExit receivingFromNetwork resynchingWithPartner runningSecurityExit runningSendExit sendingToNetwork serializingAccessToQmgr	MQCHSSTATE_CHADEXIT MQCHSSTATE_COMPRESSING MQCHSSTATE_END_OF_BATCH MQCHSSTATE_HANDSHAKING MQCHSSTATE_HEARTBEATING MQCHSSTATE_IN_MQGET MQCHSSTATE_IN_MQI_CALL MQCHSSTATE_IN_MQPUT MQCHSSTATE_MREXIT MQCHSSTATE_MSGEXIT MQCHSSTATE_NAME_SERVER MQCHSSTATE_NET_CONNECTING MQCHSSTATE_OTHER MQCHSSTATE_RCVEXIT MQCHSSTATE_RECEIVING MQCHSSTATE_RESYNCHING MQCHSSTATE_SCYEXIT MQCHSSTATE_SENDEXIT MQCHSSTATE_SENDING MQCHSSTATE_SERIALIZING

Table 144. Channel attributes for the REST API and equivalent PCF attributes. (continued)

REST API attribute	PCF attribute	Related values (REST API)	Related values (PCF)
currentStatus.agent.userId	**MQCACH_MCA_USER_ID**		
currentStatus.batch.count	**MQIACH_BATCHES**		
currentStatus.batch.currentMessages savedStatus.batch.currentMessages	**MQIACH_CURRENT_MSGS**		
currentStatus.batch.luwid.current savedStatus.batch.luwid.current	**MQCACH_CURRENT_LUWID**		
currentStatus.batch.luwid.last savedStatus.batch.luwid.last	**MQCACH_LAST_LUWID**		
currentStatus.batch.sequenceNumber.current savedStatus.batch.sequenceNumber.current	**MQIACH_CURRENT_SEQ_NUMBER**		
currentStatus.batch.sequenceNumber.last savedStatus.batch.sequenceNumber.last	**MQIACH_LAST_SEQ_NUMBER**		
currentStatus.batch.size	**MQIACH_BATCH_SIZE**		
currentStatus.compression.header.currentMessage currentStatus.compression.header.lastMessage	**MQIACH_HDR_COMPRESSION**	none system unavailable (applies to lastMessage only)	MQCOMPRESS_NONE MQCOMPRESS_SYSTEM MQCOMPRESS_NOT_AVAILABLE
currentStatus.compression.message.currentMessage currentStatus.compression.message.lastMessage	**MQIACH_MSG_COMPRESSION**	none runLengthEncoding zlibFast zlibHigh unavailable (applies to lastMessage only)	MQCOMPRESS_NONE MQCOMPRESS_RLE MQCOMPRESS_ZLIBFAST MQCOMPRESS_ZLIBHIGH MQCOMPRESS_NOT_AVAILABLE
currentStatus.connectionManagement.heartbeatInterval	**MQIACH_HB_INTERVAL**		
currentStatus.connectionManagement.keepAliveInterval	**MQIACH_KEEP_ALIVE_INTERVAL**		
currentStatus.connectionManagement.localAddress.host currentStatus.connectionManagement.localAddress.port	**MQCACH_LOCAL_ADDRESS**		
currentStatus.connectionManagement.longRetriesLeft	**MQIACH_LONG_RETRIES_LEFT**	long	
currentStatus.connectionManagement.shortRetriesLeft	**MQIACH_SHORT_RETRIES_LEFT**	short	
currentStatus.extended.buffersReceived	**MQIACH_BUFFERS_RCVD**		
currentStatus.extended.buffersSent	**MQIACH_BUFFERS_SENT**		
currentStatus.extended.bytesReceived	**MQIACH_BYTES_RCVD**		
currentStatus.extended.bytesSent	**MQIACH_BYTES_SENT**		
currentStatus.extended.messages	**MQIACH_MSGS**		
currentStatus.general.connectionId currentStatus.general.connection.port savedStatus.general.connection.host	**MQCACH_CONNECTION_NAME**		

Table 144. Channel attributes for the REST API and equivalent PCF attributes. (continued)

REST API attribute	PCF attribute	Related values (REST API)	Related values (PCF)
currentStatus.general.transmissionQueueName savedStatus.general.transmissionQueueName	**MQCACH_XMIT_Q_NAME**		
currentStatus.general.maximumMessageLength	**MQIACH_MAX_MSG_LENGTH**		
currentStatus.general.stopRequested	**MQIACH_STOP_REQUESTED**	true false	MQCHSR_STOP_REQUESTED MQCHSR_STOP_NOT_REQUESTED
currentStatus.general.statistics	**MQIA_STATISTICS_CHANNEL**	disabledByQmgr off low medium high	MQMON_NONE MQMON_OFF MQMON_Q_MGR MQMON_LOW MQMON_MEDIUM MQMON_HIGH
currentStatus.monitoring.messagesInBatch.shortSamplePeriod currentStatus.monitoring.messagesInBatch.longSamplePeriod	**MQIACH_BATCH_SIZE_INDICATOR**	-1	MQMON_NOT_AVAILABLE
currentStatus.monitoring.rate	**MQIA_MONITORING_CHANNEL**	off low medium high	MQMON_OFF MQMON_LOW MQMON_MEDIUM MQMON_HIGH
currentStatus.monitoring.compressionRate.shortSamplePeriod currentStatus.monitoring.messagesInBatch.longSamplePeriod	**MQIACH_COMPRESSION_RATE**	-1	MQMON_NOT_AVAILABLE
currentStatus.monitoring.compressionTime.shortSamplePeriod currentStatus.monitoring.compressionTime.longSamplePeriod	**MQIACH_COMPRESSION_TIME**	-1	MQMON_NOT_AVAILABLE
currentStatus.monitoring.exitTime.shortSamplePeriod currentStatus.monitoring.exitTime.longSamplePeriod	**MQIACH_EXIT_TIME_INDICATOR**	-1	MQMON_NOT_AVAILABLE
currentStatus.monitoring.messagesAvailableOnXmitQ	**MQIACH_XMITQ_MSGS_AVAILABLE**	-1	MQMON_NOT_AVAILABLE
currentStatus.monitoring.networkTime.shortSamplePeriod currentStatus.monitoring.networkTime.longSamplePeriod	**MQIACH_NETWORK_TIME_INDICATOR**	-1	MQMON_NOT_AVAILABLE
currentStatus.monitoring.transmissionQueueTime.shortSamplePeriod currentStatus.monitoring.transmissionQueueTime.longSamplePeriod	**MQIACH_XMITQ_TIME_INDICATOR**	-1	MQMON_NOT_AVAILABLE

Table 144. Channel attributes for the REST API and equivalent PCF attributes. (continued)

REST API attribute	PCF attribute	Related values (REST API)	Related values (PCF)
currentStatus.partner.product	**MQCACH_REMOTE_PRODUCT**	MQMM MQMV MQCC MQNM MQJB MQJM MQJN MQJU MQXC MQXD MQXN MQXM MQXU MQNU	MQMM MQMV MQCC MQNM MQJB MQJM MQJN MQJU MQXC MQXD MQXN MQXM MQXU MQNU
currentStatus.partner.qmgrName	**MQCA_REMOTE_Q_MGR_NAME**		
currentStatus.partner.version	**MQCACH_REMOTE_VERSION**		
currentStatus.queueSharingGroup.channelDisposition savedStatus.queueSharingGroup.channelDisposition	**MQIACH_CHANNEL_DISP**	private shared fixShared	MQCHLD_PRIVATE MQCHLD_SHARED MQCHLD_FIXSHARED
currentStatus.timestamps.started	**MQCACH_CHANNEL_START_DATE** **MQCACH_CHANNEL_START_TIME**		
currentStatus.timestamps.lastMessage	**MQCACH_LAST_MSG_DATE** **MQCACH_LAST_MSG_TIME**		
currentStatus.transmissionSecurity.sslClientCertIssuerName	**MQCACH_SSL_CERT_ISSUER_NAME**		
currentStatus.transmissionSecurity.sslClientUserId	**MQCACH_SSL_CERT_USER_ID**		
currentStatus.transmissionSecurity.sslKeyResetTime	**MQCACH_SSL_KEY_RESET_DATE** **MQCACH_SSL_KEY_RESET_TIME**		
currentStatus.transmissionSecurity.sslKeyResets	**MQIACH_SSL_KEY_RESETS**		
currentStatus.transmissionSecurity.sslProtocol	**MQCACH_SSL_CERT_USER_ID**	none sslV30 tlsV10 tlsV12	MQSECPROT_NONE MQSECPROT_SSLV30 MQSECPROT_TLSV10 MQSECPROT_TLSV12
currentStatus.transmissionSecurity.sslShortPeerName	**MQCACH_SSL_SHORT_PEER_NAME**		

Unsupported PCF parameters

The following parameters are not supported by the administrative REST API:

- **MQIACH_CLIENT_CHANNEL_WEIGHT**
- **MQIACH_CONNECTION_AFFINITY**
- **MQIACH_DEF_RECONNECT**
- **MQIACH_IN_DOUBT_IN**
- **MQIACH_IN_DOUBT_OUT**
- **MQCACH_LAST_MSG_TIME**

- **MQIACH_MAX_INSTANCES**
- **MQIACH_MAX_INSTS_PER_CLIENT**
- **MQCACH_MODE_NAME**
- **MQIACH_MSGS_RECEIVED/MQIACH_MSGS_RCVD**
- **MQIACH_MSGS_SENT**
- **MQCACH_PASSWORD**
- **MQIACH_SHARING_CONVERSATIONS**
- **MQCACH_TP_NAME**
- **MQIACH_XMIT_PROTOCOL_TYPE**
- **MQCACH_USER_ID**

IBM MQ Administration Interface

Use the reference information in this section to accomplish the tasks that address your business needs.

MQAI reference

Reference information for the MQAI.

A list of reference information for the MQAI.

There are two types of selector: *user selector* and *system selector*. These are described in "MQAI Selectors" on page 1889.

There are three types of call:

- Data-bag manipulation calls for configuring data bags:
  - "mqAddBag" on page 1812
  - "mqAddByteString" on page 1814
  - "mqAddByteStringFilter" on page 1815
  - "mqAddInquiry" on page 1817
  - "mqAddInteger" on page 1819
  - "mqAddInteger64" on page 1821
  - "mqAddIntegerFilter" on page 1822
  - "mqAddString" on page 1824
  - "mqAddStringFilter" on page 1826
  - "mqClearBag" on page 1831
  - "mqCountItems" on page 1832
  - "mqCreateBag" on page 1834
  - "mqDeleteBag" on page 1837
  - "mqDeleteItem" on page 1838
  - "mqInquireBag" on page 1846
  - "mqInquireByteString" on page 1848
  - "mqInquireByteStringFilter" on page 1850
  - "mqInquireInteger" on page 1853
  - "mqInquireInteger64" on page 1855
  - "mqInquireIntegerFilter" on page 1857
  - "mqInquireItemInfo" on page 1859
  - "mqInquireString" on page 1861
  - "mqInquireStringFilter" on page 1864

- "mqSetByteString" on page 1870
- "mqSetByteStringFilter" on page 1872
- "mqSetInteger" on page 1875
- "mqSetInteger64" on page 1877
- "mqSetIntegerFilter" on page 1879
- "mqSetString" on page 1881
- "mqSetStringFilter" on page 1884
- "mqTruncateBag" on page 1888

- Command calls for sending and receiving administration commands and PCF messages:
  - "mqBagToBuffer" on page 1828
  - "mqBufferToBag" on page 1830
  - "mqExecute" on page 1840
  - "mqGetBag" on page 1844
  - "mqPutBag" on page 1868
- Utility calls for handling blank-padded and null-terminated strings:
  - "mqPad" on page 1867
  - "mqTrim" on page 1887

These calls are described in alphabetical order in the following sections.

mqAddBag:

The mqAddBag call nests a bag in another bag.

Syntax for mqAddBag

mqAddBag *(Bag, Selector, ItemValue, CompCode, Reason)*

Parameters for mqAddBag

Bag (MQHBAG) - input
Bag handle into which the item is to be added.

The bag must be a user bag. This means that it must have been created using the MQCBO_USER_BAG option on the mqCreateBag call. If the bag was not created in this way, MQRC_WRONG_BAG_TYPE results.

Selector (MQLONG) - input
Selector identifying the item to be nested.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

If the selector is zero or greater (that is, a user selector) and the bag was created with the MQCBO_CHECK_SELECTORS option, the selector must be in the range MQGA_FIRST through MQGA_LAST; if not, again MQRC_SELECTOR_OUT_OF_RANGE results.

If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value of zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the data type of this occurrence must be the same as the data type of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

ItemValue (MQHBAG) - input
The bag which is to be nested.

If the bag is not a group bag, MQRC_BAG_WRONG_TYPE results. If an attempt is made to add a bag to itself, MQRC_HBAG_ERROR results.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicate error conditions that can be returned from the mqAddBag call:

MQRC_BAG_WRONG_TYPE
Wrong type of bag for intended use (either Bag or ItemValue).

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of this occurrence of selector differs from data type of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

Usage notes for mqAddBag

If a bag with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily adjacent to the existing instance.

C language invocation for mqAddBag

```
mqAddBag (Bag, Selector, ItemValue, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHBAG   Bag;       /* Bag handle */
MQLONG   Selector;  /* Selector */
MQHBAG   ItemValue; /* Nested bag handle */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqAddBag

(Supported on Windows only.)

```
mqAddGroup Bag, Selector, ItemValue, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemValue As Long 'Nested bag handle'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

Note: The mqAddBag call can be used with user bags only; you cannot add nested bags to administration or command bags. You can only nest group bags.

mqAddByteString:

The mqAddByteString call adds a byte string identified by a user selector to the end of a specified bag.

Syntax for mqAddByteString

mqAddByteString *(Bag, Selector, BufferLength, Buffer, CompCode, Reason)*

Parameters for mqAddByteString

Bag (MQHBAG) - input
Handle of the bag to be modified.

This value must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the value you specify relates to a system bag.

Selector (MQLONG) - input
Selector identifying the item to be added to the bag.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQBA_FIRST through MQBA_LAST. MQRC_SELECTOR_OUT_OF_RANGE results if it is not in the correct range.

If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the data type of this occurrence must be the same as the data type of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

BufferLength (MQLONG) - input
The length in bytes of the string contained in the **Buffer** parameter. The value must be zero or greater.

Buffer (MQBYTE ⌂ BufferLength) - input
Buffer containing the byte string.

The length is given by the **BufferLength** parameter. If zero is specified for **BufferLength**, the null pointer can be specified for the address of the **Buffer** parameter. In all other cases, a valid (nonnull) address must be specified for the **Buffer** parameter.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqAddByteString call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of this occurrence of selector differs from data type of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqAddByteString

1. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily adjacent to the existing instance.
2. This call cannot be used to add a system selector to a bag.

C language invocation for mqAddByteString

```
mqAddByteString (hBag, Selector, BufferLength, Buffer, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;     /* Selector */
MQLONG   BufferLength; /* Buffer length */
PMQBYTE  Buffer        /* Buffer containing item value */
MQLONG   CompCode;     /* Completion code */
MQLONG   Reason;       /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqAddByteString

(Supported on Windows only.)

```
mqAddByteString Bag, Selector, BufferLength, Buffer, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long 'Bag handle'
Dim Selector     As Long 'Selector'
Dim BufferLength As Long 'Buffer length'
Dim Buffer       As Byte 'Buffer containing item value'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

mqAddByteStringFilter:

The mqAddByteStringFilter call adds a byte string filter identified by a user selector to the end of a specified bag.

Syntax for mqAddByteStringFilter

mqAddByteStringFilter *(Bag, Selector, BufferLength, Buffer, Operator, CompCode, Reason)*

Parameters for mqAddByteStringFilter

Bag (MQHBAG) - input
Handle of the bag to be modified.

This value must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the value you specify relates to a system bag.

Selector (MQLONG) - input
Selector identifying the item to be added to the bag.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQBA_FIRST through MQBA_LAST. MQRC_SELECTOR_OUT_OF_RANGE results if it is not in the correct range.

If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the data type of this occurrence must be the same as the data type of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

BufferLength (MQLONG) - input
The length in bytes of the condition byte string contained in the **Buffer** parameter. The value must be zero or greater.

Buffer (MQBYTE x BufferLength) - input
Buffer containing the condition byte string.

The length is given by the **BufferLength** parameter. If zero is specified for **BufferLength**, the null pointer can be specified for the address of the **Buffer** parameter. In all other cases, a valid (nonnull) address must be specified for the **Buffer** parameter.

Operator (MQLONG) - input
The byte string filter operator to be placed in the bag. Valid operators are of the form MQCFOP_*.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqAddByteStringFilter call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_FILTER_OPERATOR_ERROR
Filter operator not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of this occurrence of selector differs from data type of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqAddByteStringFilter

1. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily adjacent to the existing instance.

2. This call cannot be used to add a system selector to a bag.

C language invocation for mqAddByteStringFilter

```
mqAddByteStringFilter (hBag, Selector, BufferLength, Buffer, Operator,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   hBag;          /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   BufferLength;  /* Buffer length */
PMQBYTE  Buffer         /* Buffer containing item value */
MQLONG   Operator       /* Operator */
PMQLONG  CompCode;      /* Completion code */
PMQLONG  Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqAddByteStringFilter

(Supported on Windows only.)

```
mqAddByteStringFilter Bag, Selector, BufferLength, Buffer, Operator, CompCode,
Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long 'Bag handle'
Dim Selector     As Long 'Selector'
Dim BufferLength As Long 'Buffer length'
Dim Buffer       As String 'Buffer containing item value'
Dim Operator     As Long 'Operator'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

mqAddInquiry:

The mqAddInquiry call can be used with administration bags only; it is specifically for administration purposes.

The mqAddInquiry call adds a selector to an administration bag. The selector refers to an IBM MQ object attribute that is to be returned by a PCF INQUIRE command. The value of the **Selector** parameter specified on this call is added to the end of the bag, as the value of a data item that has the selector value MQIACF_INQUIRY.

Syntax for mqAddInquiry

mqAddInquiry *(Bag, Selector, CompCode, Reason)*

Parameters for mqAddInquiry

Bag (MQHBAG) - input
Bag handle.

The bag must be an administration bag; that is, it must have been created with the MQCBO_ADMIN_BAG option on the mqCreateBag call. If the bag was not created this way, MQRC_BAG_WRONG_TYPE results.

Selector (MQLONG) - input
Selector of the IBM MQ object attribute that is to be returned by the appropriate INQUIRE administration command.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicate error conditions that can be returned from the mqAddInquiry call:

MQRC_BAG_WRONG_TYPE
Wrong type of bag for intended use.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqAddInquiry

1. When the administration message is generated, the MQAI constructs an integer list with the MQIACF_*_ATTRS or MQIACH_*_ATTRS selector that is appropriate to the `Command` value specified on the mqExecute, mqPutBag, or mqBagToBuffer call. It then adds the values of the attribute selectors specified by the mqAddInquiry call.
2. If the `Command` value specified on the mqExecute, mqPutBag, or mqBagToBuffer call is not recognized by the MQAI, MQRC_INQUIRY_COMMAND_ERROR results. Instead of using the mqAddInquiry call, this can be overcome by using the mqAddInteger call with the appropriate MQIACF_*_ATTRS or MQIACH_*_ATTRS selector and the **ItemValue** parameter of the selector being inquired.

C language invocation for mqAddInquiry

```
mqAddInquiry (Bag, Selector, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqAddInquiry

(Supported on Windows only.)

```
mqAddInquiry Bag, Selector, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag      As Long 'Bag handle'
Dim Selector As Long 'Selector'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

Supported INQUIRE command codes

- MQCMD_INQUIRE_AUTH_INFO
- MQCMD_INQUIRE_AUTH_RECS
- MQCMD_INQUIRE_AUTH_SERVICE
- MQCMD_INQUIRE_CF_STRUC
- MQCMD_INQUIRE_CHANNEL

- MQCMD_INQUIRE_CHANNEL_STATUS
- MQCMD_INQUIRE_CLUSTER_Q_MGR
- MQCMD_INQUIRE_CONNECTION
- MQCMD_INQUIRE_LISTENER
- MQCMD_INQUIRE_LISTENER_STATUS
- MQCMD_INQUIRE_NAMELIST
- MQCMD_INQUIRE_PROCESS
- MQCMD_INQUIRE_Q
- MQCMD_INQUIRE_Q_MGR
- MQCMD_INQUIRE_Q_MGR_STATUS
- MQCMD_INQUIRE_Q_STATUS
- MQCMD_INQUIRE_SECURITY

For an example that demonstrates the use of supported INQUIRE command codes, see Inquiring about queues and printing information (amqsailq.c).

mqAddInteger:

The mqAddInteger call adds an integer item identified by a user selector to the end of a specified bag.

Syntax for mqAddInteger

mqAddInteger *(Bag, Selector, ItemValue, CompCode, Reason)*

Parameters for mqAddInteger

Bag (MQHBAG) - input
Handle of the bag to be modified.

This must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the value you specify identifies a system bag.

Selector (MQLONG)
Selector identifying the item to be added to the bag.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

If the selector is zero or greater (that is, a user selector) and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQIA_FIRST through MQIA_LAST; if not, again MQRC_SELECTOR_OUT_OF_RANGE results.

If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value of zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the data type of this occurrence must be the same as the data type of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

ItemValue (MQLONG) - input
The integer value to be placed in the bag.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicate error conditions that can be returned from the mqAddInteger call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of this occurrence of selector differs from data type of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqAddInteger

1. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily next to the existing instance.
2. This call cannot be used to add a system selector to a bag.

C language invocation for mqAddInteger

```
mqAddInteger (Bag, Selector, ItemValue, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHBAG   Bag;       /* Bag handle */
MQLONG   Selector;  /* Selector */
MQLONG   ItemValue; /* Integer value */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqAddInteger

(Supported on Windows only.)

```
mqAddInteger Bag, Selector, ItemValue, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemValue As Long 'Integer value'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqAddInteger64:

The mqAddInteger64 call adds a 64-bit integer item identified by a user selector to the end of a specified bag.

Syntax for mqAddInteger64

mqAddInteger64 *(Bag, Selector, ItemValue, CompCode, Reason)*

Parameters for mqAddInteger64

Bag (MQHBAG) - input
Handle of the bag to be modified.

This must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the value you specify identifies a system bag.

Selector (MQLONG) - input
Selector identifying the item to be added to the bag.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

If the selector is zero or greater (that is, a user selector) and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQIA_FIRST through MQIA_LAST; if not, again MQRC_SELECTOR_OUT_OF_RANGE results.

If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value of zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the data type of this occurrence must be the same as the data type of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

ItemValue (MQINT64) - input
The 64-bit integer value to be placed in the bag.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicate error conditions that can be returned from the mqAddInteger64 call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of this occurrence of selector differs from data type of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqAddInteger64

1. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily adjacent to the existing instance.
2. This call cannot be used to add a system selector to a bag.

C language invocation for mqAddInteger64

```
mqAddInteger64 (Bag, Selector, ItemValue, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHBAG   Bag;       /* Bag handle */
MQLONG   Selector;  /* Selector */
MQINT64  ItemValue; /* Integer value */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqAddInteger64

(Supported on Windows only.)

```
mqAddInteger64 Bag, Selector, ItemValue, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag        As Long 'Bag handle'
Dim Selector   As Long 'Selector'
Dim Item Value As Long 'Integer value'
Dim CompCode   As Long 'Completion code'
Dim Reason     As Long 'Reason code qualifying CompCode'
```

mqAddIntegerFilter:

The mqAddIntegerFilter call adds an integer filter identified by a user selector to the end of a specified bag.

Syntax for mqAddIntegerFilter

mqAddIntegerFilter *(Bag, Selector, ItemValue, Operator, CompCode, Reason)*

Parameters for mqAddIntegerFilter

Bag (MQHBAG) - input
Handle of the bag to be modified.

This must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the value you specify identifies a system bag.

Selector (MQLONG) - input
Selector identifying the item to be added to the bag.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

If the selector is zero or greater (that is, a user selector) and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQIA_FIRST through MQIA_LAST; if not, again MQRC_SELECTOR_OUT_OF_RANGE results.

If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value of zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the data type of this occurrence must be the same as the data type of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

ItemValue (MQLONG) - input
The integer condition value to be placed in the bag.

Operator (MQLONG) - input
The integer filter operator to be placed in the bag. Valid operators take the form MQCFOP_*.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicate error conditions that can be returned from the mqAddIntegerFilter call:

MQRC_FILTER_OPERATOR_ERROR
Filter operator not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of this occurrence of selector differs from data type of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqAddIntegerFilter

1. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily adjacent to the existing instance.
2. This call cannot be used to add a system selector to a bag.

C language invocation for mqAddIntegerFilter

```
mqAddIntegerFilter (Bag, Selector, ItemValue, Operator, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHBAG   Bag;       /* Bag handle */
MQLONG   Selector;  /* Selector */
MQLONG   ItemValue; /* Integer value */
MQLONG   Operator;  /* Item operator */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqAddIntegerFilter

(Supported on Windows only.)

```
mqAddIntegerFilter Bag, Selector, ItemValue, Operator, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemValue As Long 'Integer value'
Dim Operator  As Long 'Item Operator'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqAddString:

The mqAddString call adds a character data item identified by a user selector to the end of a specified bag.

Syntax for mqAddString

mqAddString *(Bag, Selector, BufferLength, Buffer, CompCode, Reason)*

Parameters for mqAddString

Bag (MQHBAG) - input
Handle of the bag to be modified.

This value must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the value you specify relates to a system bag.

Selector (MQLONG) - input
Selector identifying the item to be added to the bag.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQCA_FIRST through MQCA_LAST. MQRC_SELECTOR_OUT_OF_RANGE results if it is not in the correct range.

If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the data type of this occurrence must be the same as the data type of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

BufferLength (MQLONG) - input
The length in bytes of the string contained in the **Buffer** parameter. The value must be zero or greater, or the special value MQBL_NULL_TERMINATED:

- If MQBL_NULL_TERMINATED is specified, the string is delimited by the first null encountered in the string. The null is not added to the bag as part of the string.
- If MQBL_NULL_TERMINATED is not specified, *BufferLength* characters are inserted into the bag, even if null characters are present. Nulls do not delimit the string.

Buffer (MQCHAR x BufferLength) - input
Buffer containing the character string.

The length is given by the **BufferLength** parameter. If zero is specified for **BufferLength**, the null pointer can be specified for the address of the **Buffer** parameter. In all other cases, a valid (nonnull) address must be specified for the **Buffer** parameter.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqAddString call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_CODED_CHAR_SET_ID_ERROR
Bag CCSID is MQCCSI_EMBEDDED.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of this occurrence of selector differs from data type of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqAddString

1. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily adjacent to the existing instance.
2. This call cannot be used to add a system selector to a bag.
3. The Coded Character Set ID associated with this string is copied from the current CCSID of the bag.

C language invocation for mqAddString

```
mqAddString (hBag, Selector, BufferLength, Buffer, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   hBag;           /* Bag handle */
MQLONG   Selector;       /* Selector */
MQLONG   BufferLength;   /* Buffer length */
PMQCHAR  Buffer          /* Buffer containing item value */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqAddString

(Supported on Windows only.)

```
mqAddString Bag, Selector, BufferLength, Buffer, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long 'Bag handle'
Dim Selector     As Long 'Selector'
Dim BufferLength As Long 'Buffer length'
Dim Buffer       As String 'Buffer containing item value'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

mqAddStringFilter:

The mqAddStringFilter call adds a string filter identified by a user selector to the end of a specified bag.

Syntax for mqAddStringFilter

mqAddStringFilter *(Bag, Selector, BufferLength, Buffer, Operator, CompCode, Reason)*

Parameters for mqAddStringFilter

Bag (MQHBAG) - input
Handle of the bag to be modified.

This value must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the value you specify relates to a system bag.

Selector (MQLONG) - input
Selector identifying the item to be added to the bag.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQCA_FIRST through MQCA_LAST. MQRC_SELECTOR_OUT_OF_RANGE results if it is not in the correct range.

If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the data type of this occurrence must be the same as the data type of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

BufferLength (MQLONG) - input
The length in bytes of the character condition string contained in the **Buffer** parameter. The value must be zero or greater, or the special value MQBL_NULL_TERMINATED:

- If MQBL_NULL_TERMINATED is specified, the string is delimited by the first null encountered in the string. The null is not added to the bag as part of the string.
- If MQBL_NULL_TERMINATED is not specified, *BufferLength* characters are inserted into the bag, even if null characters are present. Nulls do not delimit the string.

Buffer (MQCHAR x BufferLength) - input
Buffer containing the character condition string.

The length is given by the **BufferLength** parameter. If zero is specified for **BufferLength**, the null pointer can be specified for the address of the **Buffer** parameter. In all other cases, a valid (nonnull) address must be specified for the **Buffer** parameter.

Operator (MQLONG) - input
The string filter operator to be placed in the bag. Valid operators are of the form MQCFOP_*.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqAddStringFilter call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_CODED_CHAR_SET_ID_ERROR
Bag CCSID is MQCCSI_EMBEDDED.

MQRC_FILTER_OPERATOR_ERROR
Filter operator not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of this occurrence of selector differs from data type of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqAddStringFilter

1. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily adjacent to the existing instance.
2. This call cannot be used to add a system selector to a bag.
3. The Coded Character Set ID associated with this string is copied from the current CCSID of the bag.

C language invocation for mqAddStringFilter

```
mqAddStringFilter (hBag, Selector, BufferLength, Buffer, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   hBag;          /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   BufferLength;  /* Buffer length */
PMQCHAR  Buffer         /* Buffer containing item value */
MQLONG   Operator       /* Operator */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqAddStringFilter

(Supported on Windows only.)

```
mqAddStringFilter Bag, Selector, BufferLength, Buffer, Operator, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long 'Bag handle'
Dim Selector     As Long 'Selector'
Dim BufferLength As Long 'Buffer length'
Dim Buffer       As String 'Buffer containing item value'
Dim Operator     As Long 'Item operator'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

mqBagToBuffer:

The mqBagToBuffer call converts the bag into a PCF message in the supplied buffer.

Syntax for mqBagToBuffer

mqBagToBuffer *(OptionsBag, DataBag, BufferLength, Buffer, DataLength, CompCode, Reason)*

Parameters for mqBagToBuffer

OptionsBag (MQHBAG) - input
Handle of the bag containing options that control the processing of the call. This is a reserved parameter; the value must be MQHB_NONE.

DataBag (MQHBAG) - input
The handle of the bag to convert.

If the bag contains an administration message and mqAddInquiry was used to insert values into the bag, the value of the MQIASY_COMMAND data item must be an INQUIRE command that is recognized by the MQAI; MQRC_INQUIRY_COMMAND_ERROR results if it is not.

If the bag contains nested system bags, MQRC_NESTED_BAG_NOT_SUPPORTED results.

BufferLength (MQLONG) - input
Length in bytes of the buffer supplied.

If the buffer is too small to accommodate the message generated, MQRC_BUFFER_LENGTH_ERROR results.

Buffer (MQBYTE x BufferLength) - output
The buffer to hold the message.

DataLength (MQLONG) - output
The length in bytes of the buffer required to hold the entire bag. If the buffer is not long enough, the contents of the buffer are undefined but the `DataLength` is returned.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqBagToBuffer call:

MQRC_BAG_WRONG_TYPE
Input data bag is a group bag.

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid or buffer too small. (Required length returned in *DataLength*.)

MQRC_DATA_LENGTH_ERROR
DataLength parameter not valid (invalid parameter address).

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INQUIRY_COMMAND_ERROR
mqAddInquiry used with a command code that is not recognized as an INQUIRE command.

MQRC_NESTED_BAG_NOT_SUPPORTED
Input data bag contains one or more nested system bags.

MQRC_OPTIONS_ERROR
Options bag contains unsupported data items or a supported option has an invalid value.

MQRC_PARAMETER_MISSING
An administration message requires a parameter that is not present in the bag.

Note: This reason code occurs for bags created with the MQCBO_ADMIN_BAG or MQCBO_REORDER_AS_REQUIRED options only.

MQRC_SELECTOR_WRONG_TYPE
mqAddString or mqSetString was used to add the MQIACF_INQUIRY selector to the bag.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

Usage notes for mqBagToBuffer

1. The PCF message is generated with an encoding of MQENC_NATIVE for the numeric data.
2. The buffer that holds the message can be null if the `BufferLength` is zero. This is useful if you use the mqBagToBuffer call to calculate the size of buffer necessary to convert your bag.

C language invocation for mqBagToBuffer

```
mqBagToBuffer (OptionsBag, DataBag, BufferLength, Buffer, &DataLength,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG  OptionsBag;     /* Options bag handle */
MQHBAG  DataBag;        /* Data bag handle */
MQLONG  BufferLength;   /* Buffer length */
MQBYTE  Buffer[n];      /* Buffer to contain PCF */
MQLONG  DataLength;     /* Length of PCF returned in buffer */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqBagToBuffer

(Supported on Windows only.)

```
mqBagToBuffer OptionsBag, DataBag, BufferLength, Buffer, DataLength,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim OptionsBag   As Long 'Options bag handle'
Dim DataBag      As Long 'Data bag handle'
Dim BufferLength As Long 'Buffer length'
Dim Buffer       As Long 'Buffer to contain PCF'
Dim DataLength   As Long 'Length of PCF returned in buffer'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

mqBufferToBag:

The mqBufferToBag call converts the supplied buffer into bag form.

Syntax for mqBufferToBag

mqBufferToBag *(OptionsBag, BufferLength, Buffer, DataBag, CompCode, Reason)*

Parameters for mqBufferToBag

OptionsBag (MQHBAG) - input
Handle of the bag containing options that control the processing of the call. This is a reserved parameter; the value must be MQHB_NONE.

BufferLength (MQLONG) - input
Length in bytes of the buffer.

Buffer (MQBYTE x *BufferLength*) - input
Pointer to the buffer containing the message to be converted.

Databag (MQHBAG) - input/output
Handle of the bag to receive the message. The MQAI performs an mqClearBag call on the bag before placing the message in the bag.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqBufferToBag call:

MQRC_BAG_CONVERSION_ERROR
Data could not be converted into a bag. This indicates a problem with the format of the data to be converted into a bag (for example, the message is not a valid PCF).

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of second occurrence of selector differs from data type of first occurrence.

MQRC_OPTIONS_ERROR
Options bag contains unsupported data items, or a supported option has a value that is not valid.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqBufferToBag

The buffer must contain a valid PCF message. The encoding of numeric data in the buffer must be MQENC_NATIVE.

The Coded Character Set ID of the bag is unchanged by this call.

C language invocation for mqBufferToBag

```
mqBufferToBag (OptionsBag, BufferLength, Buffer, DataBag,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   OptionsBag;    /* Options bag handle */
MQLONG   BufferLength;  /* Buffer length  */
MQBYTE   Buffer[n];     /* Buffer containing PCF */
MQHBAG   DataBag;       /* Data bag handle */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqBufferToBag

(Supported on Windows only.)

```
mqBufferToBag OptionsBag, BufferLength, Buffer, DataBag,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim OptionsBag   As Long 'Options bag handle'
Dim BufferLength As Long 'Buffer length'
Dim Buffer       As Long 'Buffer containing PCF'
Dim DataBag      As Long 'Data bag handle'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

mqClearBag:

The mqClearBag call deletes all user items from the bag, and resets system items to their initial values.

Syntax for mqClearBag

mqClearBag *(Bag, CompCode, Reason)*

Parameters for mqClearBag

Bag (MQHBAG) - input
Handle of the bag to be cleared. This must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if you specify the handle of a system bag.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqClearBag call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqClearBag

1. If the bag contains system bags, they are also deleted.
2. The call cannot be used to clear system bags.

C language invocation for mqClearBag

```
mqClearBag (Bag, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqClearBag

(Supported on Windows only.)

```
mqClearBag Bag, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag      As Long 'Bag handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

mqCountItems:

The mqCountItems call returns the number of occurrences of user items, system items, or both, that are stored in a bag with the same specific selector.

Syntax for mqCountItems

mqCountItems *(Bag, Selector, ItemCount, CompCode, Reason)*

Parameters for mqCountItems

Bag (MQHBAG) - input
Handle of the bag with items that are to be counted. This can be a user bag or a system bag.

Selector (MQLONG) - input
Selector of the data items to count.

If the selector is less than zero (a system selector), the selector must be one that is supported by the MQAI. MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the specified selector is not present in the bag, the call succeeds and zero is returned for *ItemCount*.

The following special values can be specified for *Selector*:

MQSEL_ALL_SELECTORS
All user and system items are to be counted.

MQSEL_ALL_USER_SELECTORS
All user items are to be counted; system items are excluded from the count.

MQSEL_ALL_SYSTEM_SELECTORS
All system items are to be counted; user items are excluded from the count.

ItemCount (MQLONG) - output
Number of items of the specified type in the bag (can be zero).

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqCountItems call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_ITEM_COUNT_ERROR
ItemCount parameter not valid (invalid parameter address).

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

Usage notes for mqCountItems

This call counts the number of data items, not the number of unique selectors in the bag. A selector can occur multiple times, so there might be fewer unique selectors in the bag than data items.

C language invocation for mqCountItems

```
mqCountItems (Bag, Selector, &ItemCount, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG  Bag;            /* Bag handle */
MQLONG  Selector;       /* Selector */
MQLONG  ItemCount;      /* Number of items */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqCountItems

(Supported on Windows only.)

```
mqCountItems Bag, Selector, ItemCount, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag;      As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemCount As Long 'Number of items'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqCreateBag:

The mqCreateBag call creates a new bag.

Syntax for mqCreateBag

mqCreateBag *(Options, Bag, CompCode, Reason)*

Parameters for mqCreateBag

Options (MQLONG) - input
Options for creation of the bag.

The following values are valid:

MQCBO_ADMIN_BAG
Specifies that the bag is for administering IBM MQ objects. MQCBO_ADMIN_BAG automatically implies the MQCBO_LIST_FORM_ALLOWED, MQCBO_REORDER_AS_REQUIRED, and MQCBO_CHECK_SELECTORS options.

Administration bags are created with the MQIASY_TYPE system item set to MQCFT_COMMAND.

MQCBO_COMMAND_BAG
Specifies that the bag is a command bag. MQCBO_COMMAND_BAG is an alternative to the administration bag (MQCBO_ADMIN_BAG) and MQRC_OPTIONS_ERROR results if both are specified.

A command bag is processed in the same way as a user bag except that the value of the MQIASY_TYPE system item is set to MQCFT_COMMAND when the bag is created.

The command bag is also created for administering objects but they are not used to send administration messages to a command server as an administration bag is. The bag options assume the following default values:

- MQCBO_LIST_FORM_INHIBITIED
- MQCBO_DO_NOT_REORDER
- MQCBO_DO_NOT_CHECK_SELECTORS

Therefore, the MQAI does not change the order of data items or create lists within a message as with administration bags.

MQCBO_GROUP_BAG
Specifies that the bag is a group bag. This means that the bag is used to hold a set of grouped items. Group bags cannot be used for the administration of IBM MQ objects. The bag options assume the following default values:

- MQCBO_LIST_FORM_ALLOWED
- MQCBO_REORDER_AS_REQUIRED
- MQCBO_DO_NOT_CHECK_SELECTORS

Therefore, the MQAI can change the order of data items or create lists within a bag of grouped items.

Group bags are created with two system selectors: MQIASY_BAG_OPTIONS and MQIASY_CODED_CHAR_SET_ID.

If a group bag is nested in a bag in which MQCBO_CHECK_SELECTORS was specified, the group bag to be nested has its selectors checked at that point whether MQCBO_CHECK_SELECTORS was specified when the group bag was created.

MQCBO_USER_BAG
Specifies that the bag is a user bag. MQCBO_USER_BAG is the default bag-type option. User

bags can also be used for the administration of IBM MQ objects, but the MQCBO_LIST_FORM_ALLOWED and MQCBO_REORDER_AS_REQUIRED options must be specified to ensure correct generation of the administration messages.

User bags are created with the MQIASY_TYPE system item set to MQCFT_USER.

For user bags, one or more of the following options can be specified:

MQCBO_LIST_FORM_ALLOWED
Specifies that the MQAI can use the more compact list form in the message sent whenever there are two or more adjacent occurrences of the same selector in the bag. However, the items cannot be reordered if this option is used. Therefore, if the occurrences of the selector are not adjacent in the bag, and MQCBO_REORDER_AS_REQUIRED is not specified, the MQAI cannot use the list form for that particular selector.

If the data items are character strings, these strings must have the same Character Set ID and the same selector, in order to be compacted into list form. If the list form is used, the shorter strings are padded with blanks to the length of the longest string.

This option must be specified if the message to be sent is an administration message but MQCBO_ADMIN_BAG is not specified.

Note: MQCBO_LIST_FORM_ALLOWED does not imply that the MQAI definitely uses the list form. The MQAI considers various factors in deciding whether to use the list form.

MQCBO_LIST_FORM_INHIBITED
Specifies that the MQAI cannot use the list form in the message sent, even if there are adjacent occurrences of the same selector in the bag. MQCBO_LIST_FORM_INHIBITED is the default list-form option.

MQCBO_REORDER_AS_REQUIRED
Specifies that the MQAI can change the order of the data items in the message sent. This option does not affect the order of the items in the sending bag.

This option means that you can insert items into a data bag in any order. That is, the items do not need to be inserted in the way that they must be in the PCF message, because the MQAI can reorder these items as required.

If the message is a user message, the order of the items in the receiving bag is the same as the order of the items in the message. This order can be different from the order of the items in the sending bag.

If the message is an administration message, the order of the items in the receiving bag is determined by the message received.

This option must be specified if the message to be sent is an administration message but MQCBO_ADMIN is not specified.

MQCBO_DO_NOT_REORDER
Specifies that the MQAI cannot change the order of data items in the message sent. Both the message sent and the receiving bag contain the items in the same order as they occur in the sending bag. This option is the default ordering option.

MQCBO_CHECK_SELECTORS
Specifies that user selectors (selectors that are zero or greater) must be checked to ensure that the selector is consistent with the data type implied by the mqAddInteger, mqAddInteger64, mqAddIntegerFilter, mqAddString, mqAddStringFilter, mqAddByteString, mqAddByteStringFilter, mqSetInteger, mqSetInteger64, mqSetIntegerFilter, mqSetString, mqSetStringFilter, mqSetByteString, or mqSetByteStringFilter call:

- For the integer, 64-bit integer, and integer filter calls, the selector must be in the range MQIA_FIRST through MQIA_LAST.
- For the string and string filter calls, the selector must be in the range MQCA_FIRST through MQCA_LAST.
- For byte string and byte string filter calls, the selector must be in the range MQBA_FIRST through MQBA_LAST
- For group bag calls, the selector must be in the range MQGA_FIRST through MQGA_LAST
- For the handle calls, the selector must be in the range MQHA_FIRST through MQHA_LAST.

The call fails if the selector is outside the valid range. System selectors (selectors less than zero) are always checked, and if a system selector is specified, it must be one that is supported by the MQAI.

MQCBO_DO_NOT_CHECK_SELECTORS
Specifies that user selectors (selectors that are zero or greater) are not checked. Any selector that is zero or positive can be used with any call. This option is the default selectors option. System selectors (selectors less than zero) are always checked.

MQCBO_NONE
Specifies that all options must have their default values. This option is provided to aid program documentation, and must not be specified with any of the options that have a nonzero value.

The following list summarizes the default option values:

- MQCBO_USER_BAG
  - MQCBO_LIST_FORM_INHIBITIED
  - MQCBO_DO_NOT_REORDER
  - MQCBO_DO_NOT_CHECK_SELECTORS

Bag (MQHBAG) - output
The handle of the bag created by the call.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqCreateBag call:

MQRC_HBAG_ERROR
Bag handle not valid (invalid parameter address or the parameter location is read-only).

MQRC_OPTIONS_ERROR
Options not valid or not consistent.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

Usage notes for mqCreateBag

Any options used for creating your bag are contained in a system item within the bag when it is created.

C language invocation for mqCreateBag

```
mqCreateBag (Options, &Bag, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQLONG  Options;        /* Bag options */
MQHBAG  Bag;            /* Bag handle */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqCreateBag

(Supported on Windows only.)

```
mqCreateBag Options, Bag, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Options  As Long 'Bag options'
Dim Bag      As Long 'Bag handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

mqDeleteBag:

The mqDeleteBag call deletes the specified bag.

Syntax for mqDeleteBag

mqDeleteBag *(Bag, CompCode, Reason)*

Parameters for mqDeleteBag

Bag (MQHBAG) - input/output
The handle of the bag to be deleted. This must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_DELETABLE results if you specify the handle of a system bag. The handle is reset to MQHB_UNUSABLE_HBAG.

If the bag contains system-generated bags, they are also deleted.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqDeleteBag call:

MQRC_HBAG_ERROR
Bag handle not valid, or invalid parameter address, or parameter location is read only.

MQRC_SYSTEM_BAG_NOT_DELETABLE
System bag cannot be deleted.

Usage notes for mqDeleteBag

1. Delete any bags created with mqCreateBag.
2. Nested bags are deleted automatically when the containing bag is deleted.

C language invocation for mqDeleteBag

```
mqDeleteBag (&Bag, CompCode, Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqDeleteBag

(Supported on Windows only.)

```
mqDeleteBag Bag, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag;     As Long 'Bag handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

mqDeleteItem:

The mqDeleteItem call removes one or more user items from a bag.

Syntax for mqDeleteItem

mqDeleteItem *(Bag, Selector, ItemIndex, CompCode, Reason)*

Parameters for mqDeleteItem

Hbag (MQHBAG) - input
Handle of the bag to be modified.

This must be the handle of a bag created by the user, and not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if it is a system bag.

Selector (MQLONG) - input
Selector identifying the user item to be deleted.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

The following special values are valid:

MQSEL_ANY_SELECTOR
The item to be deleted is a user item identified by the **ItemIndex** parameter, the index relative to the set of items that contains both user and system items.

MQSEL_ANY_USER_SELECTOR
The item to be deleted is a user item identified by the **ItemIndex** parameter, the index relative to the set of user items.

If an explicit selector value is specified, but the selector is not present in the bag, the call succeeds if MQIND_ALL is specified for `ItemIndex`, and fails with reason code MQRC_SELECTOR_NOT_PRESENT if MQIND_ALL is not specified.

ItemIndex (MQLONG) - input
Index of the data item to be deleted.

The value must be zero or greater, or one of the following special values:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results. If MQIND_NONE is specified with one of the MQSEL_XXX_SELECTOR values, MQRC_INDEX_ERROR results.

MQIND_ALL
This specifies that all occurrences of the selector in the bag are to be deleted. If MQIND_ALL is specified with one of the MQSEL_XXX_SELECTOR values, MQRC_INDEX_ERROR results. If MQIND_ALL is specified when the selector is not present within the bag, the call succeeds.

If MQSEL_ANY_SELECTOR is specified for the **Selector** parameter, the **ItemIndex** parameter is the index relative to the set of items that contains both user items and system items, and

must be zero or greater. If ItemIndex identifies a system selector MQRC_SYSTEM_ITEM_NOT_DELETABLE results. If MQSEL_ANY_USER_SELECTOR is specified for the **Selector** parameter, the **ItemIndex** parameter is the index relative to the set of user items, and must be zero or greater.

If an explicit selector value is specified, ItemIndex is the index relative to the set of items that have that selector value, and can be MQIND_NONE, MQIND_ALL, zero, or greater.

If an explicit index is specified (that is, not MQIND_NONE or MQIND_ALL) and the item is not present in the bag, MQRC_INDEX_NOT_PRESENT results.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqDeleteItem call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
MQIND_NONE or MQIND_ALL specified with one of the MQSEL_ANY_XXX_SELECTOR values.

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag is read only and cannot be altered.

MQRC_SYSTEM_ITEM_NOT_DELETABLE
System item is read only and cannot be deleted.

Usage notes for mqDeleteItem

1. Either a single occurrence of the specified selector can be removed, or all occurrences of the specified selector.
2. The call cannot remove system items from the bag, or remove items from a system bag. However, the call can remove the handle of a system bag from a user bag. This way, a system bag can be deleted.

C language invocation for mqDeleteItem

```
mqDeleteItem (Bag, Selector, ItemIndex, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHBAG   Hbag;          /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Index of the data item */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqDeleteItem

(Supported on Windows only.)

```
mqDeleteItem Bag, Selector, ItemIndex, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Index of the data item'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqExecute:

The mqExecute call sends an administration command message and waits for the reply (if expected).

Syntax for mqExecute

mqExecute *(Hconn, Command, OptionsBag, AdminBag, ResponseBag, AdminQ, ResponseQ, CompCode, Reason)*

Parameters for mqExecute

Hconn (MQHCONN) - input
MQI Connection handle.

This is returned by a preceding MQCONN call issued by the application.

Command (MQLONG) - input
The command to be executed.

This should be one of the MQCMD_* values. If it is a value that is not recognized by the MQAI servicing the mqExecute call, the value is still accepted. However, if mqAddInquiry was used to insert values in the bag, the **Command** parameter must be an INQUIRE command recognized by the MQAI; MQRC_INQUIRY_COMMAND_ERROR results if it is not.

OptionsBag (MQHBAG) - input
Handle of a bag containing options that affect the operation of the call.

This must be the handle returned by a preceding mqCreateBag call or the following special value:

MQHB_NONE
No options bag; all options assume their default values.

Only the options listed in this topic can be present in the options bag (MQRC_OPTIONS_ERROR results if other data items are present).

The appropriate default value is used for each option that is not present in the bag. The following option can be specified:

MQIACF_WAIT_INTERVAL
This data item specifies the maximum time in milliseconds that the MQAI should wait for each reply message. The time interval must be zero or greater, or the special value MQWI_UNLIMITED; the default is thirty seconds. The mqExecute call completes either when all of the reply messages are received or when the specified wait interval expires without the expected reply message having been received.

Note: The time interval is an approximate quantity.

If the MQIACF_WAIT_INTERVAL data item has the wrong data type, or there is more than one occurrence of that selector in the options bag, or the value of the data item is not valid, MQRC_WAIT_INTERVAL_ERROR results.

AdminBag (MQHBAG) - input
Handle of the bag containing details of the administration command to be issued.

All user items placed in the bag are inserted into the administration message that is sent. It is the application's responsibility to ensure that only valid parameters for the command are placed in the bag.

If the value of the MQIASY_TYPE data item in the command bag is not MQCFT_COMMAND, MQRC_COMMAND_TYPE_ERROR results. If the bag contains nested system bags, MQRC_NESTED_BAG_NOT_SUPPORTED results.

ResponseBag (MQHBAG) - input
Handle of the bag where reply messages are placed.

The MQAI performs an mqClearBag call on the bag before placing reply messages in the bag. To retrieve the reply messages, the selector, MQIACF_CONVERT_RESPONSE, can be specified.

Each reply message is placed into a separate system bag, with a handle that is then placed in the response bag. Use the mqInquireBag call with selector MQHA_BAG_HANDLE to determine the handles of the system bags within the reply bag, and those bags can then be inquired to determine their contents.

If some but not all of the expected reply messages are received, MQCC_WARNING with MQRC_NO_MSG_AVAILABLE results. If none of the expected reply messages is received, MQCC_FAILED with MQRC_NO_MSG_AVAILABLE results.

Group bags cannot be used as response bags.

AdminQ (MQHOBJ) - input
Object handle of the queue on which the administration message is to be placed.

This handle was returned by a preceding MQOPEN call issued by the application. The queue must be open for output.

The following special value can be specified:

MQHO_NONE
This indicates that the administration message should be placed on the SYSTEM.ADMIN.COMMAND.QUEUE belonging to the currently connected queue manager. If MQHO_NONE is specified, the application need not use MQOPEN to open the queue.

ResponseQ
Object handle of the queue on which reply messages are placed.

This handle was returned by a preceding MQOPEN call issued by the application. The queue must be open for input and for inquiry.

The following special value can be specified:

MQHO_NONE
This indicates that the reply messages should be placed on a dynamic queue created automatically by the MQAI. The queue is created by opening SYSTEM.DEFAULT.MODEL.QUEUE, that must therefore have suitable characteristics. The queue created exists for the duration of the call only, and is deleted by the MQAI on exit from the mqExecute call.

CompCode
Completion code.

Reason
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqExecute call:

MQRC_*
Anything from the MQINQ, MQPUT, MQGET, or MQOPEN calls.

MQRC_BAG_WRONG_TYPE
Input data bag is a group bag.

MQRC_CMD_SERVER_NOT_AVAILABLE
The command server that processes administration commands is not available.

MQRC_COMMAND_TYPE_ERROR
The value of the MQIASY_TYPE data item in the request bag is not MQCFT_COMMAND.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INQUIRY_COMMAND_ERROR
mqAddInteger call used with a command code that is not a recognized INQUIRE command.

MQRC_NESTED_BAG_NOT_SUPPORTED
Input data bag contains one or more nested system bags.

MQRC_NO_MSG_AVAILABLE
Some reply messages received, but not all. Reply bag contains system-generated bags for messages that were received.

MQRC_NO_MSG_AVAILABLE
No reply messages received during the specified wait interval.

MQRC_OPTIONS_ERROR
Options bag contains unsupported data items, or a supported option has a value which is not valid.

MQRC_PARAMETER_MISSING
Administration message requires a parameter which is not present in the bag. This reason code occurs for bags created with the MQCBO_ADMIN_BAG or MQCBO_REORDER_AS_REQUIRED options only.

MQRC_SELECTOR_NOT_UNIQUE
Two or more instances of a selector exist within the bag for a mandatory parameter that permits one instance only.

MQRC_SELECTOR_WRONG_TYPE
mqAddString or mqSetString was used to add the MQIACF_INQUIRY selector to the bag.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRCCF_COMMAND_FAILED
Command failed; details of failure are contained in system-generated bags within the reply bag.

Usage notes for mqExecute

1. If no *AdminQ* is specified, the MQAI checks to see if the command server is active before sending the administration command message. However, if the command server is not active, the MQAI does not start it. If you are sending many administration command messages, you are recommended to open the SYSTEM.ADMIN.COMMAND.QUEUE yourself and pass the handle of the administration queue on each administration request.
2. Specifying the MQHO_NONE value in the **ResponseQ** parameter simplifies the use of the mqExecute call, but if mqExecute is issued repeatedly by the application (for example, from within a loop), the response queue will be created and deleted repeatedly. In this situation, it is better for the application itself to open the response queue before any mqExecute call, and close it after all mqExecute calls have been issued.

3. If the administration command results in a message being sent with a message type of MQMT_REQUEST, the call waits for the time given by the MQIACF_WAIT_INTERVAL data item in the options bag.
4. If an error occurs during the processing of the call, the response bag might contain some data from the reply message, but the data will typically be incomplete.

C language invocation for mqExecute

```
mqExecute (Hconn, Command, OptionsBag, AdminBag, ResponseBag,
AdminQ, ResponseQ, CompCode, Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;         /* MQI connection handle */
MQLONG   Command;       /* Command to be executed */
MQHBAG   OptionsBag;    /* Handle of a bag containing options */
MQHBAG   AdminBag;      /* Handle of administration bag containing
                        /* details of administration command */
MQHBAG   ResponseBag;   /* Handle of bag for response messages */
MQHOBJ   AdminQ         /* Handle of administration queue for
                           administration messages */
MQHOBJ   ResponseQ;     /* Handle of response queue for response
                           messages */
MQLONG   pCompCode;     /* Completion code */
MQLONG   pReason;       /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqExecute

(Supported on Windows only.)

```
mqExecute (Hconn, Command, OptionsBag, AdminBag, ResponseBag,
AdminQ, ResponseQ, CompCode, Reason);
```

Declare the parameters as follows:

```
Dim HConn       As Long 'MQI connection handle'
Dim Command     As Long 'Command to be executed'
Dim OptionsBag  As Long 'Handle of a bag containing options'
Dim AdminBag    As Long 'Handle of command bag containing details of
                         administration command'
Dim ResponseBag As Long 'Handle of bag for reply messages'
Dim AdminQ      As Long 'Handle of command queue for
                         administration messages'
Dim ResponseQ   As Long 'Handle of response queue for reply messages'
Dim CompCode    As Long 'Completion code'
Dim Reason      As Long 'Reason code qualifying CompCode'
```

mqGetBag:

The mqGetBag call removes a message from the specified queue and converts the message data into a data bag.

Syntax for mqGetBag

mqGetBag *(Hconn, Hobj, MsgDesc, GetMsgOpts, Bag, CompCode, Reason)*

Parameters for mqGetBag

Hconn (MQHCONN) - input
MQI connection handle.

Hobj (MQHOBJ) - input
Object handle of the queue from which the message is to be retrieved. This handle was returned by a preceding MQOPEN call issued by the application. The queue must be open for input.

MsgDesc (MQMD) - input/output
Message descriptor (for more information, see MQMD - Message descriptor).

If the *Format* field in the message has a value other than MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF, MQRC_FORMAT_NOT_SUPPORTED results.

If, on entry to the call, the *Encoding* field in the application's MQMD has a value other than MQENC_NATIVE and MQGMO_CONVERT is specified, MQRC_ENCODING_NOT_SUPPORTED results. Also, if MQGMO_CONVERT is not specified, the value of the **Encoding** parameter must be the retrieving application's MQENC_NATIVE; if not, again MQRC_ENCODING_NOT_SUPPORTED results.

GetMsgOpts (MQGMO) - input/output
Get-message options (for more information, see MQGMO - Get-message options).

MQGMO_ACCEPT_TRUNCATED_MSG cannot be specified; MQRC_OPTIONS_ERROR results if it is. MQGMO_LOCK and MQGMO_UNLOCK are not supported in a 16-bit or 32-bit Window environment. MQGMO_SET_SIGNAL is supported in a 32-bit Window environment only.

Bag (MQHBAG) - input/output
Handle of a bag into which the retrieved message is placed. The MQAI performs an mqClearBag call on the bag before placing the message in the bag.

MQHB_NONE
Gets the retrieved message. This provides a means of deleting messages from the queue.

If an option of MQGMO_BROWSE_* is specified, this value sets the browse cursor to the selected message; it is not deleted in this case.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating warning and error conditions can be returned from the mqGetBag call:

MQRC_*
Anything from the MQGET call or bag manipulation.

MQRC_BAG_CONVERSION_ERROR
Data could not be converted into a bag.

This indicates a problem with the format of the data to be converted into a bag (for example, the message is not a valid PCF).

If the message was retrieved destructively from the queue (that is, not browsing the queue), this reason code indicates that it has been discarded.

MQRC_BAG_WRONG_TYPE
Input data bag is a group bag.

MQRC_ENCODING_NOT_SUPPORTED
Encoding not supported; the value in the *Encoding* field of the MQMD must be MQENC_NATIVE.

MQRC_FORMAT_NOT_SUPPORTED
Format not supported; the *Format* name in the message is not MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF. If the message was retrieved destructively from the queue (that is, not browsing the queue), this reason code indicates that it has been discarded.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of second occurrence of selector differs from data type of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqGetBag

1. Only messages that have a supported format can be returned by this call. If the message has a format that is not supported, the message is discarded, and the call completes with an appropriate reason code.
2. If the message is retrieved within a unit of work (that is, with the MQGMO_SYNCPOINT option), and the message has an unsupported format, the unit of work can be backed out, reinstating the message on the queue. This allows the message to be retrieved by using the MQGET call in place of the mqGetBag call.

C language invocation for mqGetBag

```
mqGetBag (hConn, hObj, &MsgDesc, &GetMsgOpts, hBag, CompCode, Reason);
```

Declare the parameters as follows:

```
MQHCONN  hConn;         /* MQI connection handle */
MQHOBJ   hObj;          /* Object handle */
MQMD     MsgDesc;       /* Message descriptor */
MQGMO    GetMsgOpts;    /* Get-message options */
MQHBAG   hBag;          /* Bag handle */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqGetBag

(Supported on Windows only.)

```
mqGetBag (HConn, HObj, MsgDesc, GetMsgOpts, Bag, CompCode, Reason);
```

Declare the parameters as follows:

```
Dim HConn      As Long 'MQI connection handle'
Dim HObj       As Long 'Object handle'
Dim MsgDesc    As Long 'Message descriptor'
```

```
Dim GetMsgOpts As Long 'Get-message options'
Dim Bag        As Long 'Bag handle'
Dim CompCode   As Long 'Completion code'
Dim Reason     As Long 'Reason code qualifying CompCode'
```

mqInquireBag:

The mqInquireBag call inquires the value of a bag handle that is present in the bag. The data item can be a user item or a system item.

Syntax for mqInquireBag

mqInquireBag *(Bag, Selector, ItemIndex, ItemValue, CompCode, Reason)*

Parameters for mqInquireBag

Bag (MQHBAG) - input
Bag handle to be inquired. The bag can be a user bag or a system bag.

Selector (MQLONG) - input
Selector identifying the item to be inquired.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The data type of the item must agree with the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for **Selector**:

MQSEL_ANY_SELECTOR
The item to be inquired is a user or system item identified by the **ItemIndex** parameter.

MQSEL_ANY_USER_SELECTOR
The item to be inquired is a user item identified by the **ItemIndex** parameter.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired is a system item identified by the **ItemIndex** parameter.

ItemIndex (MQLONG) - input
Index of the data item to be inquired.

The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results.

The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for the **Selector** parameter, the **ItemIndex** parameter is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for the **Selector** parameter, the **ItemIndex** parameter is the index relative to the set of system items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for the **Selector** parameter, the **ItemIndex** parameter is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, the **ItemIndex** parameter is the index relative to the set of items that have that selector value and can be MQIND_NONE, zero, or greater.

ItemValue (MQHBAG) - output
Value of the item in the bag.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqInquireBag call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_ITEM_VALUE_ERROR
The **ItemValue** parameter is not valid (invalid parameter address).

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present within the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation for mqInquireBag

```
mqInquireBag (Bag, Selector, ItemIndex, &ItemValue, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Index of the data item to be inquired */
MQHBAG   ItemValue;     /* Value of item in the bag */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqInquireBag

(Supported on Windows only.)

```
mqInquireBag (Bag, Selector, ItemIndex, ItemValue, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Index of the data item to be inquired'
Dim ItemValue As Long 'Value of item in the bag'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqInquireByteString:

The mqInquireByteString call requests the value of a byte string data item that is present in the bag. The data item can be a user item or a system item.

Syntax for mqInquireByteString

mqInquireByteString *(Bag, Selector, ItemIndex, Bufferlength, Buffer, ByteStringLength, CompCode, Reason)*

Parameters for mqInquireByteString

Bag (MQHBAG) - input
Handle of the bag to which the inquiry relates. The bag can be a user bag or a system bag.

Selector (MQLONG) - input
Selector of the item to which the inquiry relates.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The data type of the item must be the same as the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for *Selector*:

MQSEL_ANY_SELECTOR
The item to be inquired about is a user or system item identified by *ItemIndex*.

MQSEL_ANY_USER_SELECTOR
The item to be inquired about is a user item identified by *ItemIndex*.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired about is a system item identified by *ItemIndex*.

ItemIndex (MQLONG) - input
Index of the data item to which the inquiry relates. The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results. The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for the **Selector** parameter, **ItemIndex** is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for the **Selector** parameter, **ItemIndex** is the index relative to the set of user items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for **Selector**, **ItemIndex** is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, **ItemIndex** is the index relative to the set of items that have that selector value, and can be MQIND_NONE, zero, or greater.

BufferLength (MQLONG) - input
Length in bytes of the buffer to receive the byte string. Zero is a valid value.

Buffer (MQBYTE x BufferLength) - output
Buffer to receive the byte string. The length is given by the **BufferLength** parameter. If zero is specified for **BufferLength**, the null pointer can be specified for the address of the **Buffer** parameter; in all other cases, a valid (non-null) address must be specified for the **Buffer** parameter.

The string is padded with nulls to the length of the buffer. If the string is longer than the buffer, the string is truncated to fit; in this case *ByteStringLength* indicates the size of the buffer needed to accommodate the string without truncation.

ByteStringLength (MQLONG) - output
The length in bytes of the string contained in the bag. If the **Buffer** parameter is too small, the length of the string returned is less than *ByteStringLength*.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqInquireByteString call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_STRING_LENGTH_ERROR
ByteStringLength parameter not valid (invalid parameter address).

MQRC_STRING_TRUNCATED
Data too long for output buffer and has been truncated.

C language invocation for mqInquireByteString

```
mqInquireByteString (Bag, Selector, ItemIndex,
BufferLength, Buffer, &StringLength, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   BufferLength;  /* Buffer length */
PMQBYTE  Buffer;        /* Buffer to contain string */
MQLONG   ByteStringLength;  /* Length of byte string returned */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqInquireByteString

(Supported on Windows only.)

```
mqInquireByteString Bag, Selector, ItemIndex,
BufferLength, Buffer, StringLength, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag              As Long   'Bag handle'
Dim Selector         As Long   'Selector'
Dim ItemIndex        As Long   'Item index'
Dim BufferLength     As Long   'Buffer length'
Dim Buffer           As Byte   'Buffer to contain string'
Dim ByteStringLength As Long   'Length of byte string returned'
Dim CompCode         As Long   'Completion code'
Dim Reason           As Long   'Reason code qualifying CompCode'
```

mqInquireByteStringFilter:

The mqInquireByteStringFilter call requests the value and operator of a byte string filter item that is present in the bag. The data item can be a user item or a system item.

Syntax for mqInquireByteStringFilter

mqInquireByteStringFilter *(Bag, Selector, ItemIndex, Bufferlength, Buffer, ByteStringLength, Operator, CompCode, Reason)*

Parameters for mqInquireByteStringFilter

Bag (MQHBAG) - input
Handle of the bag to which the inquiry relates. The bag can be a user bag or a system bag.

Selector (MQLONG) - input
Selector of the item to which the inquiry relates.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The data type of the item must be the same as the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for *Selector*:

MQSEL_ANY_SELECTOR
The item to be inquired about is a user or system item identified by *ItemIndex*.

MQSEL_ANY_USER_SELECTOR
The item to be inquired about is a user item identified by *ItemIndex*.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired about is a system item identified by *ItemIndex*.

ItemIndex (MQLONG) - input
Index of the data item to which the inquiry relates. The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results. The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for the **Selector** parameter, **ItemIndex** is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for the **Selector** parameter, **ItemIndex** is the index relative to the set of user items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for **Selector**, **ItemIndex** is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, **ItemIndex** is the index relative to the set of items that have that selector value, and can be MQIND_NONE, zero, or greater.

BufferLength (MQLONG) - input
Length in bytes of the buffer to receive the condition byte string. Zero is a valid value.

Buffer (MQBYTE x BufferLength) - output
Buffer to receive the condition byte string. The length is given by the **BufferLength** parameter. If zero is specified for **BufferLength**, the null pointer can be specified for the address of the **Buffer** parameter; in all other cases, a valid (non-null) address must be specified for the **Buffer** parameter.

The string is padded with blanks to the length of the buffer; the string is not null-terminated. If the string is longer than the buffer, the string is truncated to fit; in this case **ByteStringLength** indicates the size of the buffer needed to accommodate the string without truncation.

ByteStringLength (MQLONG) - output
The length in bytes of the condition string contained in the bag. If the **Buffer** parameter is too small, the length of the string returned is less than **StringLength**.

Operator (MQLONG) - output
Byte string filter operator in the bag.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqInquireByteStringFilter call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_FILTER_OPERATOR_ERROR
Filter operator not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_STRING_LENGTH_ERROR
ByteStringLength parameter not valid (invalid parameter address).

MQRC_STRING_TRUNCATED
Data too long for output buffer and has been truncated.

C language invocation for mqInquireByteStringFilter

```
mqInquireByteStringFilter (Bag, Selector, ItemIndex,
BufferLength, Buffer, &ByteStringLength, &Operator, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;              /* Bag handle */
MQLONG   Selector;         /* Selector */
MQLONG   ItemIndex;        /* Item index */
MQLONG   BufferLength;     /* Buffer length */
PMQBYTE  Buffer;           /* Buffer to contain string */
MQLONG   ByteStringLength; /* Length of string returned */
MQLONG   Operator          /* Item operator */
PMQLONG  CompCode;         /* Completion code */
PMQLONG  Reason;           /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqInquireByteStringFilter

(Supported on Windows only.)

```
mqInquireByteStringFilter Bag, Selector, ItemIndex,
BufferLength, Buffer, ByteStringLength,
Operator, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag             As Long   'Bag handle'
Dim Selector        As Long   'Selector'
Dim ItemIndex       As Long   'Item index'
Dim BufferLength    As Long   'Buffer length'
Dim Buffer          As String 'Buffer to contain string'
Dim ByteStringLength  As Long   'Length of byte string returned'
Dim Operator        As Long   'Operator'
Dim CompCode        As Long   'Completion code'
Dim Reason          As Long   'Reason code qualifying CompCode'
```

mqInquireInteger:

The mqInquireInteger call requests the value of an integer data item that is present in the bag. The data item can be a user item or a system item.

Syntax for mqInquireInteger

mqInquireInteger *(Bag, Selector, ItemIndex, ItemValue, CompCode, Reason)*

Parameters for mqInquireInteger

Bag (MQHBAG) - input
Handle of the bag to which the inquiry relates. The bag can be a user bag or a system bag.

Selector (MQLONG) - input
Selector identifying the item to which the inquiry relates.

If the selector is less than zero (a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The data type of the item must agree with the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for *Selector*:

MQSEL_ANY_SELECTOR
The item to be inquired about is a user or system item identified by *ItemIndex*.

MQSEL_ANY_USER_SELECTOR
The item to be inquired about is a user item identified by *ItemIndex*.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired about is a system item identified by *ItemIndex*.

ItemIndex (MQLONG) - input
Index of the data item to which the inquiry relates. The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and is not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results. The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of user items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, *ItemIndex* is the index relative to the set of items that have that selector value, and can be MQIND_NONE, zero, or greater.

ItemValue (MQLONG) - output
The value of the item in the bag.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqInquireInteger call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_ITEM_VALUE_ERROR
ItemValue parameter not valid (invalid parameter address).

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation for mqInquireInteger

```
mqInquireInteger (Bag, Selector, ItemIndex, &ItemValue,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   ItemValue;     /* Item value */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqInquireInteger

(Supported on Windows only.)

```
mqInquireInteger Bag, Selector, ItemIndex, ItemValue,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Item index'
Dim ItemValue As Long 'Item value'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqInquireInteger64:

The mqInquireInteger64 call requests the value of a 64-bit integer data item that is present in the bag. The data item can be a user item or a system item.

Syntax for mqInquireInteger64

mqInquireInteger64 *(Bag, Selector, ItemIndex, ItemValue, CompCode, Reason)*

Parameters for mqInquireInteger64

Bag (MQHBAG) - input
Handle of the bag to which the inquiry relates. The bag can be a user bag or a system bag.

Selector (MQLONG) - input
Selector identifying the item to which the inquiry relates.

If the selector is less than zero (a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The data type of the item must agree with the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for *Selector*:

MQSEL_ANY_SELECTOR
The item to be inquired about is a user or system item identified by *ItemIndex*.

MQSEL_ANY_USER_SELECTOR
The item to be inquired about is a user item identified by *ItemIndex*.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired about is a system item identified by *ItemIndex*.

ItemIndex (MQLONG) - input
Index of the data item to which the inquiry relates. The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and is not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results. The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of user items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, *ItemIndex* is the index relative to the set of items that have that selector value, and can be MQIND_NONE, zero, or greater.

ItemValue (MQINT64) - output
The value of the item in the bag.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqInquireInteger64 call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_ITEM_VALUE_ERROR
ItemValue parameter not valid (invalid parameter address).

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation for mqInquireInteger64

```
mqInquireInteger64 (Bag, Selector, ItemIndex, &ItemValue,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQINT64  ItemValue;     /* Item value */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqInquireInteger64

(Supported on Windows only.)

```
mqInquireInteger64 Bag, Selector, ItemIndex, ItemValue,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Item index'
Dim ItemValue As Long 'Item value'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqInquireIntegerFilter:

The mqInquireIntegerFilter call requests the value and operator of an integer filter item that is present in the bag. The data item can be a user item or a system item.

Syntax for mqInquireIntegerFilter

mqInquireIntegerFilter *(Bag, Selector, ItemIndex, ItemValue, Operator, CompCode, Reason)*

Parameters for mqInquireIntegerFilter

Bag (MQHBAG) - input
Handle of the bag to which the inquiry relates. The bag can be a user bag or a system bag.

Selector (MQLONG) - input
Selector identifying the item to which the inquiry relates.

If the selector is less than zero (a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The data type of the item must agree with the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for *Selector*:

MQSEL_ANY_SELECTOR
The item to be inquired about is a user or system item identified by *ItemIndex*.

MQSEL_ANY_USER_SELECTOR
The item to be inquired about is a user item identified by *ItemIndex*.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired about is a system item identified by *ItemIndex*.

ItemIndex (MQLONG) - input
Index of the data item to which the inquiry relates. The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and is not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results. The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of user items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, *ItemIndex* is the index relative to the set of items that have that selector value, and can be MQIND_NONE, zero, or greater.

ItemValue (MQLONG) - output
The condition value.

Operator (MQLONG) - output
Integer filter operator in the bag.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqInquireIntegerFilter call:

MQRC_FILTER_OPERATOR_ERROR
Filter operator not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_ITEM_VALUE_ERROR
ItemValue parameter not valid (invalid parameter address).

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation for mqInquireIntegerFilter

```
mqInquireIntegerFilter (Bag, Selector, ItemIndex, &ItemValue,
&Operator, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   ItemValue;     /* Item value */
MQLONG   Operator;      /* Item operator */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqInquireIntegerFilter

(Supported on Windows only.)

```
mqInquireIntegerFilter Bag, Selector, ItemIndex, ItemValue,
Operator, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Item index'
Dim ItemValue As Long 'Item value'
Dim Operator  As Long 'Item operator'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqInquireItemInfo:

The mqInquireItemInfo call returns information about a specified item in a bag. The data item can be a user item or a system item.

Syntax for mqInquireItemInfo

mqInquireItemInfo *(Bag, Selector, ItemIndex, ItemType, OutSelector, CompCode, Reason)*

Parameters for mqInquireItemInfo

Bag (MQHBAG) - input
Handle of the bag to be inquired.

The bag can be a user bag or a system bag.

Selector (MQLONG) - input
Selector identifying the item to be inquired.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The following special values can be specified for **Selector**:

MQSEL_ANY_SELECTOR
The item to be inquired is a user or system item identified by the **ItemIndex** parameter.

MQSEL_ANY_USER_SELECTOR
The item to be inquired is a user item identified by the **ItemIndex** parameter.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired is a system item identified by the **ItemIndex** parameter.

ItemIndex (MQLONG) - input
Index of the data item to be inquired.

The item must be present within the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The value must be zero or greater, or the following special value:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for the **Selector** parameter, the **ItemIndex** parameter is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for the **Selector** parameter, the **ItemIndex** parameter is the index relative to the set of system items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for the **Selector** parameter, the **ItemIndex** parameter is the index relative to the set of system items, and must be zero or greater. If an explicit selector value is specified, the **ItemIndex** parameter is the index relative to the set of items that have that selector value and can be MQIND_NONE, zero, or greater.

ItemType (MQLONG) - output
The data type of the specified data item.

The following can be returned:

MQITEM_BAG
Bag handle item.

MQITEM_BYTE_STRING
Byte string.

MQITEM_INTEGER
Integer item.

MQITEM_INTEGER_FILTER
Integer filter.

MQITEM_INTEGER64
64-bit integer item.

MQITEM_STRING
Character-string item.

MQITEM_STRING_FILTER
String filter.

OutSelector (MQLONG) - output
Selector of the specified data item.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqInquireItemInfo call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
MQIND_NONE specified with one of the MQSEL_ANY_XXX_SELECTOR values.

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_ITEM_TYPE_ERROR
ItemType parameter not valid (invalid parameter address).

MQRC_OUT_SELECTOR_ERROR
OutSelector parameter not valid (invalid parameter address).

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation for mqInquireItemInfo

```
mqInquireItemInfo (Bag, Selector, ItemIndex, &OutSelector, &ItemType,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector identifying item */
MQLONG   ItemIndex;     /* Index of data item */
MQLONG   OutSelector;   /* Selector of specified data item */
MQLONG   ItemType;      /* Data type of data item */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqInquireItemInfo

(Supported on Windows only.)

```
mqInquireItemInfo Bag, Selector, ItemIndex, OutSelector, ItemType,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag         As Long 'Bag handle'
Dim Selector    As Long 'Selector identifying item'
Dim ItemIndex   As Long 'Index of data item'
Dim OutSelector As Long 'Selector of specified data item'
Dim ItemType    As Long 'Data type of data item'
Dim CompCode    As Long 'Completion code'
Dim Reason      As Long 'Reason code qualifying CompCode'
```

mqInquireString:

The mqInquireString call requests the value of a character data item that is present in the bag. The data item can be a user item or a system item.

Syntax for mqInquireString

mqInquireString *(Bag, Selector, ItemIndex, Bufferlength, Buffer, StringLength, CodedCharSetId, CompCode, Reason)*

Parameters for mqInquireString

Bag (MQHBAG) - input
Handle of the bag to which the inquiry relates. The bag can be a user bag or a system bag.

Selector (MQLONG) - input
Selector of the item to which the inquiry relates.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The data type of the item must be the same as the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for *Selector*:

MQSEL_ANY_SELECTOR
The item to be inquired about is a user or system item identified by *ItemIndex*.

MQSEL_ANY_USER_SELECTOR
The item to be inquired about is a user item identified by *ItemIndex*.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired about is a system item identified by *ItemIndex*.

ItemIndex (MQLONG) - input
Index of the data item to which the inquiry relates. The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results. The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for the **Selector** parameter, **ItemIndex** is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for the **Selector** parameter, **ItemIndex** is the index relative to the set of user items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for **Selector**, **ItemIndex** is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, **ItemIndex** is the index relative to the set of items that have that selector value, and can be MQIND_NONE, zero, or greater.

BufferLength (MQLONG) - input
Length in bytes of the buffer to receive the string. Zero is a valid value.

Buffer (MQCHAR x *BufferLength*) - output
Buffer to receive the character string. The length is given by the **BufferLength** parameter. If zero is specified for **BufferLength**, the null pointer can be specified for the address of the **Buffer** parameter; in all other cases, a valid (non-null) address must be specified for the **Buffer** parameter.

The string is padded with blanks to the length of the buffer; the string is not null-terminated. If the string is longer than the buffer, the string is truncated to fit; in this case **StringLength** indicates the size of the buffer needed to accommodate the string without truncation.

StringLength (MQLONG) - output
The length in bytes of the string contained in the bag. If the **Buffer** parameter is too small, the length of the string returned is less than *StringLength*.

CodedCharSetId (MQLONG) - output
The coded character set identifier for the character data in the string. This parameter can be set to a null pointer if not required.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqInquireString call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_STRING_LENGTH_ERROR
StringLength parameter not valid (invalid parameter address).

MQRC_STRING_TRUNCATED
Data too long for output buffer and has been truncated.

C language invocation for mqInquireString

```
mqInquireString (Bag, Selector, ItemIndex,
BufferLength, Buffer, &StringLength, &CodedCharSetId,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   BufferLength;  /* Buffer length */
PMQCHAR  Buffer;        /* Buffer to contain string */
MQLONG   StringLength;  /* Length of string returned */
MQLONG   CodedCharSetId /* Coded Character Set ID */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqInquireString

(Supported on Windows only.)

```
mqInquireString Bag, Selector, ItemIndex,
BufferLength, Buffer, StringLength, CodedCharSetId,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag             As Long   'Bag handle'
Dim Selector        As Long   'Selector'
Dim ItemIndex       As Long   'Item index'
Dim BufferLength    As Long   'Buffer length'
Dim Buffer          As String 'Buffer to contain string'
Dim StringLength    As Long   'Length of string returned'
Dim CodedCharSetId  As Long   'Coded Character Set ID'
Dim CompCode        As Long   'Completion code'
Dim Reason          As Long   'Reason code qualifying CompCode'
```

mqInquireStringFilter:

The mqInquireStringFilter call requests the value and operator of a string filter item that is present in the bag. The data item can be a user item or a system item.

Syntax for mqInquireStringFilter

mqInquireStringFilter *(Bag, Selector, ItemIndex, Bufferlength, Buffer, StringLength, CodedCharSetId, Operator, CompCode, Reason)*

Parameters for mqInquireStringFilter

Bag (MQHBAG) - input
Handle of the bag to which the inquiry relates. The bag can be a user bag or a system bag.

Selector (MQLONG) - input
Selector of the item to which the inquiry relates.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The data type of the item must be the same as the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for *Selector*:

MQSEL_ANY_SELECTOR
The item to be inquired about is a user or system item identified by *ItemIndex*.

MQSEL_ANY_USER_SELECTOR
The item to be inquired about is a user item identified by *ItemIndex*.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired about is a system item identified by *ItemIndex*.

ItemIndex (MQLONG) - input
Index of the data item to which the inquiry relates. The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results. The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for the **Selector** parameter, *ItemIndex* is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for the **Selector** parameter, *ItemIndex* is the index relative to the set of user items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, *ItemIndex* is the index relative to the set of items that have that selector value, and can be MQIND_NONE, zero, or greater.

BufferLength (MQLONG) - input
Length in bytes of the buffer to receive the condition string. Zero is a valid value.

Buffer (MQCHAR x *BufferLength*) - output
Buffer to receive the character condition string. The length is given by the **BufferLength** parameter. If zero is specified for **BufferLength**, the null pointer can be specified for the address of the **Buffer** parameter; in all other cases, a valid (nonnull) address must be specified for the **Buffer** parameter.

The string is padded with blanks to the length of the buffer; the string is not null-terminated. If the string is longer than the buffer, the string is truncated to fit; in this case *StringLength* indicates the size of the buffer needed to accommodate the string without truncation.

StringLength (MQLONG) - output
The length in bytes of the condition string contained in the bag. If the **Buffer** parameter is too small, the length of the string returned is less than *StringLength*.

CodedCharSetId (MQLONG) - output
The coded character set identifier for the character data in the string. This parameter can be set to a null pointer if not required.

Operator (MQLONG) - output
String filter operator in the bag.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqInquireStringFilter call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_FILTER_OPERATOR_ERROR
Filter operator not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_STRING_LENGTH_ERROR
StringLength parameter not valid (invalid parameter address).

MQRC_STRING_TRUNCATED
Data too long for output buffer and has been truncated.

C language invocation for mqInquireStringFilter

```
mqInquireStringFilter (Bag, Selector, ItemIndex,
BufferLength, Buffer, &StringLength, &CodedCharSetId,
&Operator, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   BufferLength;  /* Buffer length */
PMQCHAR  Buffer;        /* Buffer to contain string */
MQLONG   StringLength;  /* Length of string returned */
MQLONG   CodedCharSetId /* Coded Character Set ID */
MQLONG   Operator       /* Item operator */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqInquireStringFilter

(Supported on Windows only.)

```
mqInquireStringFilter Bag, Selector, ItemIndex,
BufferLength, Buffer, StringLength, CodedCharSetId,
Operator, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag             As Long   'Bag handle'
Dim Selector        As Long   'Selector'
Dim ItemIndex       As Long   'Item index'
Dim BufferLength    As Long   'Buffer length'
Dim Buffer          As String 'Buffer to contain string'
Dim StringLength    As Long   'Length of string returned'
Dim CodedCharSetId  As Long   'Coded Character Set ID'
Dim Operator        As Long   'Item operator'
Dim CompCode        As Long   'Completion code'
Dim Reason          As Long   'Reason code qualifying CompCode'
```

mqPad:

The mqPad call pads a null-terminated string with blanks.

Syntax for mqPad

mqPad *(String, BufferLength, Buffer, CompCode, Reason)*

Parameters for mqPad

String (PMQCHAR) - input
Null-terminated string. The null pointer is valid for the address of the **String** parameter, and denotes a string of zero length.

BufferLength (MQLONG) - input
Length in bytes of the buffer to receive the string padded with blanks. Must be zero or greater.

Buffer (MQCHAR x *BufferLength*) - output
Buffer to receive the blank-padded string. The length is given by the **BufferLength** parameter. If zero is specified for **BufferLength**, the null pointer can be specified for the address of the **Buffer** parameter; in all other cases, a valid (nonnull) address must be specified for the **Buffer** parameter.

If the number of characters preceding the first null in the **String** parameter is greater than the **BufferLength** parameter, the excess characters are omitted and MQRC_DATA_TRUNCATED results.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqPad call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_STRING_ERROR
String parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_STRING_TRUNCATED
Data too long for output buffer and has been truncated.

Usage notes for mqPad

1. If the buffer pointers are the same, the padding is done in place. If not, at most *BufferLength* characters are copied into the second buffer; any space remaining, including the null-termination character, is overwritten with spaces.
2. If the *String* and **Buffer** parameters partially overlap, the result is undefined.

C language invocation for mqPad

```
mqPad (String, BufferLength, Buffer, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR   String;          /* String to be padded */
MQLONG   BufferLength;    /* Buffer length */
PMQCHAR  Buffer           /* Buffer to contain padded string */
MQLONG   CompCode;        /* Completion code */
MQLONG   Reason;          /* Reason code qualifying CompCode */
```

Note: This call is not supported in Visual Basic.

mqPutBag:

The mqPutBag call converts the contents of the specified bag into a PCF message and sends the message to the specified queue. The contents of the bag are unchanged after the call.

Syntax for mqPutBag

mqPutBag *(Hconn, Hobj, MsgDesc, PutMsgOpts, Bag, CompCode, Reason)*

Parameters for mqPutBag

Hconn (MQHCONN) - input
MQI connection handle.

Hobj (MQHOBJ) - input
Object handle of the queue on which the message is to be placed. This handle was returned by a preceding MQOPEN call issued by the application. The queue must be open for output.

MsgDesc (MQMD) - input/output
Message descriptor. (For more information, see MQMD - Message descriptor.)

If the *Format* field has a value other than MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF, MQRC_FORMAT_NOT_SUPPORTED results.

If the *Encoding* field has a value other than MQENC_NATIVE, MQRC_ENCODING_NOT_SUPPORTED results.

PutMsgOpts (MQPMO) - input/output
Put-message options. (For more information, see MQPMO - Put-message options.)

Bag (MQHBAG) - input
Handle of the data bag to be converted to a message.

If the bag contains an administration message, and mqAddInquiry was used to insert values into the bag, the value of the MQIASY_COMMAND data item must be an INQUIRE command recognized by the MQAI; MQRC_INQUIRY_COMMAND_ERROR results if it is not.

If the bag contains nested system bags, MQRC_NESTED_BAG_NOT_SUPPORTED results.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*. The following reason codes indicating error and warning conditions can be returned from the mqPutBag call:

MQRC_*
Anything from the MQPUT call or bag manipulation.

MQRC_BAG_WRONG_TYPE
Input data bag is a group bag.

MQRC_ENCODING_NOT_SUPPORTED
Encoding not supported (value in *Encoding* field in MQMD must be MQENC_NATIVE).

MQRC_FORMAT_NOT_SUPPORTED
Format not supported (name in *Format* field in MQMD must be MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF).

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INQUIRY_COMMAND_ERROR
mqAddInquiry call used with a command code that is not a recognized INQUIRE command.

MQRC_NESTED_BAG_NOT_SUPPORTED
Input data bag contains one or more nested system bags.

MQRC_PARAMETER_MISSING
Administration message requires a parameter that is not present in the bag. This reason code occurs for bags created with the MQCBO_ADMIN_BAG or MQCBO_REORDER_AS_REQUIRED options only.

MQRC_SELECTOR_WRONG_TYPE
mqAddString or mqSetString was used to add the MQIACF_INQUIRY selector to the bag.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation for mqPutBag

```
mqPutBag (HConn, HObj, &MsgDesc, &PutMsgOpts, Bag,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  HConn;         /* MQI connection handle */
MQHOBJ   HObj;          /* Object handle */
MQMD     MsgDesc;       /* Message descriptor */
MQPMO    PutMsgOpts;    /* Put-message options */
MQHBAG   Bag;           /* Bag handle */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqPutBag

(Supported on Windows only.)

```
mqPutBag (HConn, HObj, MsgDesc, PutMsgOpts, Bag,
CompCode, Reason);
```

Declare the parameters as follows:

```
Dim HConn      As Long  'MQI connection handle'
Dim HObj       As Long  'Object handle'
Dim MsgDesc    As MQMD  'Message descriptor'
Dim PutMsgOpts As MQPMO 'Put-message options'
Dim Bag        As Long  'Bag handle'
Dim CompCode   As Long  'Completion code'
Dim Reason     As Long  'Reason code qualifying CompCode'
```

mqSetByteString:

The mqSetByteString call either modifies a byte string data item that is already present in the bag, or deletes all existing occurrences of the specified selector and adds a new occurrence at the end of the bag. The data item is usually a user item, but certain system-data items can also be modified.

Syntax for mqSetByteString

mqSetByteString *(Bag, Selector, ItemIndex, Bufferlength, Buffer, CompCode, Reason)*

Parameters for mqSetByteString

Bag (MQHBAG) - input
Handle of the bag to be set. This must be the handle of a bag created by the user, not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if you specify the handle of a system bag.

Selector (MQLONG) - input
Selector of the item to be modified.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the selector is a supported system selector, but is one that is read only, MQRC_SYSTEM_ITEM_NOT_ALTERABLE results.

If the selector is an alterable system selector, but is always a single-instance selector and the application attempts to create a second instance in the bag, MQRC_MULTIPLE_INSTANCE_ERROR results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQBA_FIRST through MQBA_LAST; MQRC_SELECTOR_OUT_OF_RANGE results if it is not. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If MQIND_ALL is not specified for the **ItemIndex** parameter, the specified selector must already be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

If MQIND_ALL is not specified for the **ItemIndex** parameter, the data type of the item must be the same as the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

ItemIndex (MQLONG) - input
This identifies which occurrence of the item with the specified selector is to be modified. The value must be zero or greater, or one of the special values described in this topic; if it is none of these, MQRC_INDEX_ERROR results.

Zero or greater
The item with the specified index must already be present in the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The index is counted relative to the items in the bag that have the specified selector. For example, if there are five items in the bag with the specified selector, the valid values for *ItemIndex* are 0 through 4.

MQIND_NONE
This specifies that there must be only one occurrence of the specified selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

MQIND_ALL
This specifies that all existing occurrences of the specified selector (if any) are to be deleted from the bag, and a new occurrence of the selector created at the end of the bag.

BufferLength (MQLONG) - input
The length in bytes of the byte string contained in the **Buffer** parameter. The value must be zero or greater.

Buffer (MQBYTE x *BufferLength*) - input
Buffer containing the byte string. The length is given by the **BufferLength** parameter. If zero is specified for **BufferLength**, the null pointer can be specified for the address of the **Buffer** parameter; in all other cases, a valid (nonnull) address must be specified for the **Buffer** parameter.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqSetByteString call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE or MQIND_ALL).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_MULTIPLE_INSTANCE_ERROR
Multiple instances of system selector not valid.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
System item is read-only and cannot be altered.

C language invocation for mqSetByteString

```
mqSetByteString (Bag, Selector, ItemIndex, BufferLength, Buffer,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   BufferLength;  /* Buffer length */
PMQBYTE  Buffer;        /* Buffer containing string */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqSetByteString

(Supported on Windows only.)

```
mqSetByteString Bag, Selector, ItemIndex, BufferLength, Buffer,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long   'Bag handle'
Dim Selector     As Long   'Selector'
Dim ItemIndex    As Long   'Item index'
Dim BufferLength As Long   'Buffer length'
Dim Buffer       As Byte   'Buffer containing string'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

mqSetByteStringFilter:

The mqSetByteStringFilter call either modifies a byte string filter item that is already present in the bag, or deletes all existing occurrences of the specified selector and adds a new occurrence at the end of the bag. The data item is usually a user item, but certain system-data items can also be modified.

Syntax for mqSetByteStringFilter

mqSetByteStringFilter *(Bag, Selector, ItemIndex, Bufferlength, Buffer, Operator, CompCode, Reason)*

Parameters for mqSetByteStringFilter

Bag (MQHBAG) - input
Handle of the bag to be set. This must be the handle of a bag created by the user, not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if you specify the handle of a system bag.

Selector (MQLONG) - input
Selector of the item to be modified.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the selector is a supported system selector, but is one that is read only, MQRC_SYSTEM_ITEM_NOT_ALTERABLE results.

If the selector is an alterable system selector, but is always a single-instance selector and the application attempts to create a second instance in the bag, MQRC_MULTIPLE_INSTANCE_ERROR results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQBA_FIRST through MQBA_LAST; MQRC_SELECTOR_OUT_OF_RANGE results if it is not. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If MQIND_ALL is not specified for the **ItemIndex** parameter, the specified selector must already be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

If MQIND_ALL is not specified for the **ItemIndex** parameter, the data type of the item must be the same as the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

ItemIndex (MQLONG) - input
This identifies which occurrence of the item with the specified selector is to be modified. The value must be zero or greater, or one of the special values described in this topic; if it is none of these, MQRC_INDEX_ERROR results.

Zero or greater
The item with the specified index must already be present in the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The index is counted relative to the items in the bag that have the specified selector. For example, if there are five items in the bag with the specified selector, the valid values for *ItemIndex* are 0 through 4.

MQIND_NONE
This specifies that there must be only one occurrence of the specified selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

MQIND_ALL
This specifies that all existing occurrences of the specified selector (if any) are to be deleted from the bag, and a new occurrence of the selector created at the end of the bag.

BufferLength (MQLONG) - input
The length in bytes of the condition byte string contained in the **Buffer** parameter. The value must be zero or greater.

Buffer (MQBYTE x *BufferLength*) - input
Buffer containing the condition byte string. The length is given by the **BufferLength** parameter. If zero is specified for **BufferLength**, the null pointer can be specified for the address of the **Buffer** parameter; in all other cases, a valid (nonnull) address must be specified for the **Buffer** parameter.

Operator (MQLONG x *Operator*) - input
Byte string filter operator to be placed in the bag. Valid operators are of the form MQCFOP_*.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqSetByteStringFilter call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_FILTER_OPERATOR_ERROR
Bag handle not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE or MQIND_ALL).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_MULTIPLE_INSTANCE_ERROR
Multiple instances of system selector not valid.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
System item is read-only and cannot be altered.

C language invocation for mqSetByteStringFilter

```
mqSetByteStringFilter (Bag, Selector, ItemIndex, BufferLength, Buffer,
Operator, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;     /* Selector */
MQLONG   ItemIndex;    /* Item index */
MQLONG   BufferLength; /* Buffer length */
PMQBYTE  Buffer;       /* Buffer containing string */
MQLONG   Operator;     /* Operator */
PMQLONG  CompCode;     /* Completion code */
PMQLONG  Reason;       /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqSetByteStringFilter

(Supported on Windows only.)

```
mqSetByteStringFilter Bag, Selector, ItemIndex, BufferLength, Buffer,
Operator, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long   'Bag handle'
Dim Selector     As Long   'Selector'
Dim ItemIndex    As Long   'Item index'
Dim BufferLength As Long   'Buffer length'
Dim Buffer       As String 'Buffer containing string'
Dim Operator     As Long   'Item operator'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

mqSetInteger:

The mqSetInteger call either modifies an integer item that is already present in the bag, or deletes all existing occurrences of the specified selector and adds a new occurrence at the end of the bag. The data item is usually a user item, but specific system-data items can also be modified.

Syntax for mqSetInteger

mqSetInteger *(Bag, Selector, ItemIndex, ItemValue, CompCode, Reason)*

Parameters for mqSetInteger

Bag (MQHBAG) - input
Handle of the bag to be set. This must be the handle of a bag created by the user, and not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the handle you specify refers to a system bag.

Selector (MQLONG) - input
Selector of the item to be modified. If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the selector is a supported system selector, but is one that is read-only, MQRC_SYSTEM_ITEM_NOT_ALTERABLE results.

If the selector is an alterable system selector, but is always a single-instance selector and the application attempts to create a second instance in the bag, MQRC_MULTIPLE_INSTANCE_ERROR results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQIA_FIRST through MQIA_LAST; MQRC_SELECTOR_OUT_OF_RANGE results if it is not. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If MQIND_ALL is not specified for the **ItemIndex** parameter, the specified selector must already be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

If MQIND_ALL is not specified for the **ItemIndex** parameter, the data type of the item must agree with the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

ItemIndex (MQLONG) - input
This value identifies the occurrence of the item with the specified selector that is to be modified. The value must be zero or greater, or one of the special values described in this topic; if it is none of these, MQRC_INDEX_ERROR results.

Zero or greater
The item with the specified index must already be present in the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The index is counted relative to the items in the bag that have the specified selector. For example, if there are five items in the bag with the specified selector, the valid values for *ItemIndex* are 0 through 4.

MQIND_NONE
This specifies that there must be one occurrence only of the specified selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

MQIND_ALL
This specifies that all existing occurrences of the specified selector (if any) are to be deleted from the bag, and a new occurrence of the selector created at the end of the bag.

Note: For system selectors, the order is not changed.

ItemValue (MQLONG) - input
The integer value to be placed in the bag.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqSetInteger call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE or MQIND_ALL).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_MULTIPLE_INSTANCE_ERROR
Multiple instances of system selector not valid.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not in valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
System item is read only and cannot be altered.

C language invocation for mqSetInteger

```
mqSetInteger (Bag, Selector, ItemIndex, ItemValue, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   ItemValue;     /* Integer value */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqSetInteger

(Supported on Windows only.)

```
mqSetInteger Bag, Selector, ItemIndex, ItemValue, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Item index'
Dim ItemValue As Long 'Integer value'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqSetInteger64:

The mqSetInteger64 call either modifies a 64-bit integer item that is already present in the bag, or deletes all existing occurrences of the specified selector and adds a new occurrence at the end of the bag. The data item is usually a user item, but specific system-data items can also be modified.

Syntax for mqSetInteger64

mqSetInteger64 *(Bag, Selector, ItemIndex, ItemValue, CompCode, Reason)*

Parameters for mqSetInteger64

Bag (MQHBAG) - input
Handle of the bag to be set. This must be the handle of a bag created by the user, and not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the handle you specify refers to a system bag.

Selector (MQLONG) - input
Selector of the item to be modified. If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the selector is a supported system selector, but is one that is read-only, MQRC_SYSTEM_ITEM_NOT_ALTERABLE results.

If the selector is an alterable system selector, but is always a single-instance selector and the application attempts to create a second instance in the bag, MQRC_MULTIPLE_INSTANCE_ERROR results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQIA_FIRST through MQIA_LAST; MQRC_SELECTOR_OUT_OF_RANGE results if it is not. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If MQIND_ALL is not specified for the **ItemIndex** parameter, the specified selector must already be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

If MQIND_ALL is not specified for the **ItemIndex** parameter, the data type of the item must agree with the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

ItemIndex (MQLONG) - input
This value identifies the occurrence of the item with the specified selector that is to be modified. The value must be zero or greater, or one of the special values described in this topic; if it is none of these, MQRC_INDEX_ERROR results.

Zero or greater
The item with the specified index must already be present in the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The index is counted relative to the items in the bag that have the specified selector. For example, if there are five items in the bag with the specified selector, the valid values for *ItemIndex* are 0 through 4.

MQIND_NONE
This specifies that there must be one occurrence only of the specified selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

MQIND_ALL
This specifies that all existing occurrences of the specified selector (if any) are to be deleted from the bag, and a new occurrence of the selector created at the end of the bag.

Note: For system selectors, the order is not changed.

ItemValue (MQINT64) - input
The integer value to be placed in the bag.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqSetInteger64 call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE or MQIND_ALL).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_MULTIPLE_INSTANCE_ERROR
Multiple instances of system selector not valid.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not in valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
System item is read only and cannot be altered.

C language invocation for mqSetInteger64

```
mqSetInteger64 (Bag, Selector, ItemIndex, ItemValue, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQINT64  ItemValue;     /* Integer value */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqSetInteger64

(Supported on Windows only.)

```
mqSetInteger64 Bag, Selector, ItemIndex, ItemValue, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Item index'
Dim ItemValue As Long 'Integer value'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqSetIntegerFilter:

The mqSetIntegerFilter call either modifies an integer filter item that is already present in the bag, or deletes all existing occurrences of the specified selector and adds a new occurrence at the end of the bag. The data item is usually a user item, but specific system-data items can also be modified.

Syntax for mqSetIntegerFilter

mqSetIntegerFilter *(Bag, Selector, ItemIndex, ItemValue, Operator, CompCode, Reason)*

Parameters for mqSetIntegerFilter

Bag (MQHBAG) - input
Handle of the bag to be set. This must be the handle of a bag created by the user, and not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the handle you specify refers to a system bag.

Selector (MQLONG) - input
Selector of the item to be modified. If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the selector is a supported system selector, but is one that is read-only, MQRC_SYSTEM_ITEM_NOT_ALTERABLE results.

If the selector is an alterable system selector, but is always a single-instance selector and the application attempts to create a second instance in the bag, MQRC_MULTIPLE_INSTANCE_ERROR results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQIA_FIRST through MQIA_LAST; MQRC_SELECTOR_OUT_OF_RANGE results if it is not. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If MQIND_ALL is not specified for the **ItemIndex** parameter, the specified selector must already be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

If MQIND_ALL is not specified for the **ItemIndex** parameter, the data type of the item must agree with the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

ItemIndex (MQLONG) - input
This value identifies the occurrence of the item with the specified selector that is to be modified. The value must be zero or greater, or one of the special values described in this topic; if it is none of these, MQRC_INDEX_ERROR results.

Zero or greater
The item with the specified index must already be present in the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The index is counted relative to the items in the bag that have the specified selector. For example, if there are five items in the bag with the specified selector, the valid values for *ItemIndex* are 0 through 4.

MQIND_NONE
This specifies that there must be one occurrence only of the specified selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

MQIND_ALL
This specifies that all existing occurrences of the specified selector (if any) are to be deleted from the bag, and a new occurrence of the selector created at the end of the bag.

Note: For system selectors, the order is not changed.

ItemValue (MQLONG) - input
The integer condition value to be placed in the bag.

Operator (MQLONG) - input
The integer filter operator to be placed in the bag. Valid operators are of the form MQCFOP_*.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqSetIntegerFilter call:

MQRC_FILTER_OPERATOR_ERROR
Filter operator not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE or MQIND_ALL).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_MULTIPLE_INSTANCE_ERROR
Multiple instances of system selector not valid.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not in valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
System item is read only and cannot be altered.

C language invocation for mqSetIntegerFilter

```
mqSetIntegerFilter (Bag, Selector, ItemIndex, ItemValue, Operator,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   ItemValue;     /* Integer value */
MQLONG   Operator;      /* Item operator */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqSetIntegerFilter

(Supported on Windows only.)

```
mqSetIntegerFilter Bag, Selector, ItemIndex, ItemValue, Operator,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Item index'
Dim ItemValue As Long 'Integer value'
Dim Operator  As Long 'Item operator'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqSetString:

The mqSetString call either modifies a character data item that is already present in the bag, or deletes all existing occurrences of the specified selector and adds a new occurrence at the end of the bag. The data item is usually a user item, but certain system-data items can also be modified.

Syntax for mqSetString

mqSetString *(Bag, Selector, ItemIndex, Bufferlength, Buffer, CompCode, Reason)*

Parameters for mqSetString

Bag (MQHBAG) - input
Handle of the bag to be set. This must be the handle of a bag created by the user, not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if you specify the handle of a system bag.

Selector (MQLONG) - input
Selector of the item to be modified.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the selector is a supported system selector, but is one that is read only, MQRC_SYSTEM_ITEM_NOT_ALTERABLE results.

If the selector is an alterable system selector, but is always a single-instance selector and the application attempts to create a second instance in the bag, MQRC_MULTIPLE_INSTANCE_ERROR results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQCA_FIRST through MQCA_LAST; MQRC_SELECTOR_OUT_OF_RANGE results if it is not. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If MQIND_ALL is not specified for the **ItemIndex** parameter, the specified selector must already be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

If MQIND_ALL is not specified for the **ItemIndex** parameter, the data type of the item must be the same as the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

ItemIndex (MQLONG) - input
This identifies which occurrence of the item with the specified selector is to be modified. The value must be zero or greater, or one of the special values described in this topic; if it is none of these, MQRC_INDEX_ERROR results.

Zero or greater
The item with the specified index must already be present in the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The index is counted relative to the items in the bag that have the specified selector. For example, if there are five items in the bag with the specified selector, the valid values for *ItemIndex* are 0 through 4.

MQIND_NONE
This specifies that there must be only one occurrence of the specified selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

MQIND_ALL
This specifies that all existing occurrences of the specified selector (if any) are to be deleted from the bag, and a new occurrence of the selector created at the end of the bag.

BufferLength (MQLONG) - input
The length in bytes of the string contained in the **Buffer** parameter. The value must be zero or greater, or the special value MQBL_NULL_TERMINATED.

If MQBL_NULL_TERMINATED is specified, the string is delimited by the first null encountered in the string.

If MQBL_NULL_TERMINATED is not specified, *BufferLength* characters are inserted into the bag, even if null characters are present; the nulls do not delimit the string.

Buffer (MQCHAR x BufferLength) - input
Buffer containing the character string. The length is given by the **BufferLength** parameter. If zero is specified for **BufferLength**, the null pointer can be specified for the address of the **Buffer** parameter; in all other cases, a valid (nonnull) address must be specified for the **Buffer** parameter.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqSetString call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE or MQIND_ALL).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_MULTIPLE_INSTANCE_ERROR
Multiple instances of system selector not valid.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
System item is read-only and cannot be altered.

Usage notes for mqSetString

The Coded Character Set ID (CCSID) associated with this string is copied from the current CCSID of the bag.

C language invocation for mqSetString

```
mqSetString (Bag, Selector, ItemIndex, BufferLength, Buffer,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;     /* Selector */
MQLONG   ItemIndex;    /* Item index */
MQLONG   BufferLength; /* Buffer length */
PMQCHAR  Buffer;       /* Buffer containing string */
MQLONG   CompCode;     /* Completion code */
MQLONG   Reason;       /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqSetString

(Supported on Windows only.)

```
mqSetString Bag, Selector, ItemIndex, BufferLength, Buffer,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long   'Bag handle'
Dim Selector     As Long   'Selector'
Dim ItemIndex    As Long   'Item index'
Dim BufferLength As Long   'Buffer length'
Dim Buffer       As String 'Buffer containing string'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

mqSetStringFilter:

The mqSetStringFilter call either modifies a string filter item that is already present in the bag, or deletes all existing occurrences of the specified selector and adds a new occurrence at the end of the bag. The data item is usually a user item, but certain system-data items can also be modified.

Syntax for mqSetStringFilter

mqSetStringFilter *(Bag, Selector, ItemIndex, Bufferlength, Buffer, Operator, CompCode, Reason)*

Parameters for mqSetStringFilter

Bag (MQHBAG) - input
Handle of the bag to be set. This must be the handle of a bag created by the user, not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if you specify the handle of a system bag.

Selector (MQLONG) - input
Selector of the item to be modified.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the selector is a supported system selector, but is one that is read only, MQRC_SYSTEM_ITEM_NOT_ALTERABLE results.

If the selector is an alterable system selector, but is always a single-instance selector and the application attempts to create a second instance in the bag, MQRC_MULTIPLE_INSTANCE_ERROR results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQCA_FIRST through MQCA_LAST; MQRC_SELECTOR_OUT_OF_RANGE results if it is not. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If MQIND_ALL is not specified for the **ItemIndex** parameter, the specified selector must already be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

If MQIND_ALL is not specified for the **ItemIndex** parameter, the data type of the item must be the same as the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

ItemIndex (MQLONG) - input
This identifies which occurrence of the item with the specified selector is to be modified. The value must be zero or greater, or one of the special values described in this topic; if it is none of these, MQRC_INDEX_ERROR results.

Zero or greater
The item with the specified index must already be present in the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The index is counted relative to the items in the bag that have the specified selector. For example, if there are five items in the bag with the specified selector, the valid values for *ItemIndex* are 0 through 4.

MQIND_NONE
This specifies that there must be only one occurrence of the specified selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

MQIND_ALL
This specifies that all existing occurrences of the specified selector (if any) are to be deleted from the bag, and a new occurrence of the selector created at the end of the bag.

BufferLength (MQLONG) - input
The length in bytes of the condition string contained in the **Buffer** parameter. The value must be zero or greater, or the special value MQBL_NULL_TERMINATED.

If MQBL_NULL_TERMINATED is specified, the string is delimited by the first null encountered in the string.

If MQBL_NULL_TERMINATED is not specified, *BufferLength* characters are inserted into the bag, even if null characters are present; the nulls do not delimit the string.

Buffer (MQCHAR x BufferLength) - input
Buffer containing the character condition string. The length is given by the **BufferLength** parameter. If zero is specified for **BufferLength**, the null pointer can be specified for the address of the **Buffer** parameter; in all other cases, a valid (nonnull) address must be specified for the **Buffer** parameter.

Operator (MQLONG x *Operator*) - input
String filter operator to be placed in the bag. Valid operators are of the form MQCFOP_*.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqSetStringFilter call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_FILTER_OPERATOR_ERROR
Bag handle not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE or MQIND_ALL).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_MULTIPLE_INSTANCE_ERROR
Multiple instances of system selector not valid.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
System item is read-only and cannot be altered.

Usage notes for mqSetStringFilter

The Coded Character Set ID (CCSID) associated with this string is copied from the current CCSID of the bag.

C language invocation for mqSetStringFilter

```
mqSetStringFilter (Bag, Selector, ItemIndex, BufferLength, Buffer,
Operator, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;     /* Selector */
MQLONG   ItemIndex;    /* Item index */
MQLONG   BufferLength; /* Buffer length */
PMQCHAR  Buffer;       /* Buffer containing string */
MQLONG   Operator;     /* Item operator */
MQLONG   CompCode;     /* Completion code */
MQLONG   Reason;       /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqSetStringFilter

(Supported on Windows only.)

```
mqSetStringFilter Bag, Selector, ItemIndex, BufferLength, Buffer,
Operator, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long   'Bag handle'
Dim Selector     As Long   'Selector'
Dim ItemIndex    As Long   'Item index'
Dim BufferLength As Long   'Buffer length'
Dim Buffer       As String 'Buffer containing string'
Dim Operator     As Long   'Item operator'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

mqTrim:

The mqTrim call trims the blanks from a blank-padded string, then terminates it with a null.

Syntax for mqTrim

mqTrim *(BufferLength, Buffer, String, CompCode, Reason)*

Parameters for mqTrim

BufferLength (MQLONG) - input
Length in bytes of the buffer containing the string padded with blanks. Must be zero or greater.

Buffer (MQCHAR × *BufferLength*) - input
Buffer containing the blank-padded string. The length is given by the **BufferLength** parameter. If zero is specified for **BufferLength**, the null pointer can be specified for the address of the **Buffer** parameter; in all other cases, a valid (nonnull) address must be specified for the **Buffer** parameter.

String (MQCHAR × (*BufferLength* +1)) - output
Buffer to receive the null-terminated string. The length of this buffer must be at least one byte greater than the value of the **BufferLength** parameter.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqTrim call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_STRING_ERROR
String parameter not valid (invalid parameter address or buffer not completely accessible).

Usage notes for mqTrim

1. If the two buffer pointers are the same, the trimming is done in place. If they are not the same, the blank-padded string is copied into the null-terminated string buffer. After copying, the buffer is scanned backwards from the end until a nonspace character is found. The byte following the nonspace character is then overwritten with a null character.
2. If *String* and *Buffer* partially overlap, the result is undefined.

C language invocation for mqTrim

```
mqTrim (BufferLength, Buffer, String, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQLONG   BufferLength;     /* Buffer length */
PMQCHAR  Buffer;           /* Buffer containing blank-padded string */
MQCHAR   String[n+1];      /* String with blanks discarded */
MQLONG   CompCode;         /* Completion code */
MQLONG   Reason;           /* Reason code qualifying CompCode */
```

Note: This call is not supported in Visual Basic.

mqTruncateBag:

The mqTruncateBag call reduces the number of user items in a user bag to the specified value, by deleting user items from the end of the bag.

Syntax for mqTruncateBag

mqTruncateBag *(Bag, ItemCount, CompCode, Reason)*

Parameters for mqTruncateBag

Bag (MQHBAG) - input
Handle of the bag to be truncated. This must be the handle of a bag created by the user, not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if you specify the handle of a system bag.

ItemCount (MQLONG) - input
The number of user items to remain in the bag after truncation. Zero is a valid value.

Note: The **ItemCount** parameter is the number of data items, not the number of unique selectors. (If there are one or more selectors that occur multiple times in the bag, there will be fewer selectors than data items before truncation.) Data items are deleted from the end of the bag, in the opposite order to which they were added to the bag.

If the number specified exceeds the number of user items currently in the bag, MQRC_ITEM_COUNT_ERROR results.

CompCode (MQLONG) - output
Completion code.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqTruncateBag call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_ITEM_COUNT_ERROR
ItemCount parameter not valid (value exceeds the number of user data items in the bag).

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqTruncateBag

1. System items in a bag are not affected by mqTruncateBag; the call cannot be used to truncate system bags.
2. mqTruncateBag with an *ItemCount* of zero is not the same as the mqClearBag call. The former deletes all of the user items but leaves the system items intact, and the latter deletes all of the user items and resets the system items to their initial values.

C language invocation for mqTruncateBag

```
mqTruncateBag (Bag, ItemCount, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   hBag;          /* Bag handle */
MQLONG   ItemCount;     /* Number of items to remain in bag */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqTruncateBag

(Supported on Windows only.)

```
mqTruncateBag Bag, ItemCount, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim ItemCount As Long 'Number of items to remain in bag'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

MQAI Selectors:

Items in bags are identified by a *selector* that acts as an identifier for the item. There are two types of selector, *user selector* and *system selector*.

User selectors

User selectors have values that are zero or positive. For the administration of MQSeries objects, valid user selectors are already defined by the following constants:

- MQCA_* and MQIA_* (object attributes)
- MQCACF_* and MQIACF_* (items relating specifically to PCF)
- MQCACH_* and MQIACH_* (channel attributes)

For user messages, the meaning of a user selector is defined by the application.

The following additional user selectors are introduced by the MQAI:

MQIACF_INQUIRY
Identifies an IBM MQ object attribute to be returned by an Inquire command.

MQHA_BAG_HANDLE
Identifies a bag handle residing within another bag.

MQHA_FIRST
Lower limit for handle selectors.

MQHA_LAST
Upper limit for handle selectors.

MQHA_LAST_USED
Upper limit for last handle selector allocated.

MQCA_USER_LIST
Default user selector. Supported on Visual Basic only. This selector supports character type and represents the default value used if the **Selector** parameter is omitted on the mqAdd*, mqSet*, or mqInquire* calls.

MQIA_USER_LIST
Default user selector. Supported on Visual Basic only. This selector supports integer type and represents the default value used if the **Selector** parameter is omitted on the mqAdd*, mqSet*, or mqInquire* calls.

System selectors

System selectors have negative values. The following system selectors are included in the bag when it is created:

MQIASY_BAG_OPTIONS
Bag-creation options. A summation of the options used to create the bag. This selector cannot be changed by the user.

MQIASY_CODED_CHAR_SET_ID
Character-set identifier for the character data items in the bag. The initial value is the queue manager's character set.

The value in the bag is used on entry to the mqExecute call and set on exit from the mqExecute call. This also applies when character strings are added to or modified in the bag.

MQIASY_COMMAND
PCF command identifier. Valid values are the MQCMD_* constants. For user messages, the value MQCMD_NONE should be used. The initial value is MQCMD_NONE.

The value in the bag is used on entry to the mqPutBag and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag and mqBufferToBag calls.

MQIASY_COMP_CODE
Completion code. Valid values are the MQCC_* constants. The initial value is MQCC_OK.

The value in the bag is used on entry to the mqExecute, mqPutBag, and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag, and mqBufferToBag calls.

MQIASY_CONTROL
PCF control options. Valid values are the MQCFC_* constants. The initial value is MQCFC_LAST.

The value in the bag is used on entry to the mqExecute, mqPutBag, and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag, and mqBufferToBag calls.

MQIASY_MSG_SEQ_NUMBER
PCF message sequence number. Valid values are 1 or greater. The initial value is 1.

The value in the bag is used on entry to the mqExecute, mqPutBag, and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag, and mqBufferToBag calls.

MQIASY_REASON
Reason code. Valid values are the MQRC_* constants. The initial value is MQRC_NONE.

The value in the bag is used on entry to the mqExecute, mqPutBag, and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag, and mqBufferToBag calls.

MQIASY_TYPE
PCF command type. Valid values are the MQCFT_* constants. For user messages, the value MQCFT_USER should be used. The initial value is MQCFT_USER for bags created as user bags and MQCFT_COMMAND for bags created as administration or command bags.

The value in the bag is used on entry to the mqExecute, mqPutBag, and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag, and mqBufferToBag calls.

MQIASY_VERSION
PCF version. Valid values are the MQCFH_VERSION_* constants. The initial value is MQCFH_VERSION_1.

If the value in the bag is set to a value other than MQCFH_VERSION_1, the value is used on entry to the mqExecute, mqPutBag, and mqBagToBuffer calls. If the value in the bag in MQCFH_VERSION_1, the PCF version is the lowest value required for the parameter structures that are present in the message.

The value in the bag is set on exit from the mqExecute, mqGetBag, and mqBufferToBag calls.

Indexing

Each selector and value within a data item in a bag have three associated index numbers:

- The index relative to other items that have the same selector.
- The index relative to the category of selector (user or system) to which the item belongs.
- The index relative to all the data items in the bag (user and system).

This allows indexing by user selectors, system selectors, or both as shown in Figure 11.

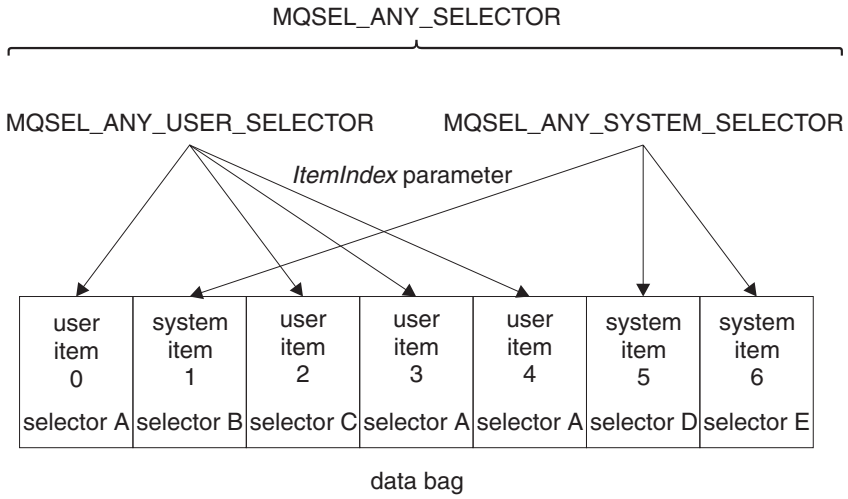


Figure 11. Indexing

In Figure Figure 11, user item 3 (selector A) can be referred to by the following index pairs:

Selector	*ItemIndex*
selector A	1
MQSEL_ANY_USER_SELECTOR	2
MQSEL_ANY_SELECTOR	3

The index is zero-based like an array in C; if there are 'n' occurrences, the index ranges from zero through 'n-1', with no gaps.

Indexes are used when replacing or removing existing data items from a bag. When used in this way, the insertion order is preserved, but indexes of other data items can be affected. For examples of this, see Changing information within a bag and Deleting data items.

The three types of indexing allow easy retrieval of data items. For example, if there are three instances of a particular selector in a bag, the mqCountItems call can count the number of instances of that selector, and the mqInquire* calls can specify both the selector and the index to inquire those values only. This is useful for attributes that can have a list of values such as some of the exits on channels.

Data conversion processing

Like PCF messages, the strings contained in an MQAI data bag can be in a variety of coded character sets. Usually, all of the strings in a PCF message are in the same coded character set; that is, the same set as the queue manager.

Each string item in a data bag contains two values; the string itself and the CCSID. The string that is added to the bag is obtained from the **Buffer** parameter of the mqAddString or mqSetString call. The CCSID is obtained from the system item containing a selector of MQIASY_CODED_CHAR_SET_ID. This is known as the *bag CCSID* and can be changed using the mqSetInteger call.

When you inquire the value of a string contained in a data bag, the CCSID is an output parameter from the call.

Table 145 shows the rules applied when converting data bags into messages and vice versa:

Table 145. CCSID processing

MQAI call	CCSID	Input to call	Output to call
mqBagToBuffer	Bag CCSID (1)	Ignored	Unchanged
mqBagToBuffer	String CCSIDs in bag	Used	Unchanged
mqBagToBuffer	String CCSIDs in buffer	Not applicable	Copied from string CCSIDs in bag
mqBufferToBag	Bag CCSID (1)	Ignored	Unchanged
mqBufferToBag	String CCSIDs in buffer	Used	Unchanged
mqBufferToBag	String CCSIDs in bag	Not applicable	Copied from string CCSIDs in buffer
mqPutBag	MQMD CCSID	Used	Unchanged (2)
mqPutBag	Bag CCSID (1)	Ignored	Unchanged
mqPutBag	String CCSIDs in bag	Used	Unchanged
mqPutBag	String CCSIDs in message sent	Not applicable	Copied from string CCSIDs in bag
mqGetBag	MQMD CCSID	Used for data conversion of message	Set to CCSID of data returned (3)
mqGetBag	Bag CCSID (1)	Ignored	Unchanged
mqGetBag	String CCSIDs in message	Used	Unchanged
mqGetBag	String CCSIDs in bag	Not applicable	Copied from string CCSIDs in message
mqExecute	Request-bag CCSID	Used for MQMD of request message (4)	Unchanged
mqExecute	Reply-bag CCSID	Used for data conversion of reply message (4)	Set to CCSID of data returned (3)
mqExecute	String CCSIDs in request bag	Used for request message	Unchanged
mqExecute	String CCSIDs in reply bag	Not applicable	Copied from string CCSIDs in reply message

Notes:

1. Bag CCSID is the system item with selector MQIASY_CODED_CHAR_SET_ID.
2. MQCCSI_Q_MGR is changed to the actual queue manager CCSID.

3. If data conversion is requested, the CCSID of data returned is the same as the output value. If data conversion is not requested, the CCSID of data returned is the same as the message value. Note that no message is returned if data conversion is requested but fails.
4. If the CCSID is MQCCSI_DEFAULT, the queue manager's CCSID is used.

Related information:

Data conversion

ccsid_part2.tbl file

Use of the message descriptor

The PCF command type is obtained from the system item with selector MQIASY_TYPE. When you create your data bag, the initial value of this item is set depending on the type of bag you create:

Table 146. PCF command type

Type of bag	Initial value of MQIASY_TYPE item
MQCBO_ADMIN_BAG	MQCFT_COMMAND
MQCBO_COMMAND_BAG	MQCFT_COMMAND
MQCBO_*	MQCFT_USER

When the MQAI generates a message descriptor, the values used in the **Format** and **MsgType** parameters depend on the value of the system item with selector MQIASY_TYPE as shown in Table 146.

Table 147. Format and MsgType parameters of the MQMD

PCF command type	Format	MsgType
MQCFT_COMMAND	MQFMT_ADMIN	MQMT_REQUEST
MQCFT_REPORT	MQFMT_ADMIN	MQMT_REPORT
MQCFT_RESPONSE	MQFMT_ADMIN	MQMT_REPLY
MQCFT_TRACE_ROUTE	MQFMT_ADMIN	MQMT_DATAGRAM
MQCFT_EVENT	MQFMT_EVENT	MQMT_DATAGRAM
MQCFT_*	MQFMT_PCF	MQMT_DATAGRAM

Table 147 shows that if you create an administration bag or a command bag, the *Format* of the message descriptor is MQFMT_ADMIN and the *MsgType* is MQMT_REQUEST. This is suitable for a PCF request message sent to the command server when a response is expected back.

Other parameters in the message descriptor take the values shown in Table 148.

Table 148. Message descriptor values

Parameter	Value
StrucId	MQMD_STRUC_ID
Version	MQMD_VERSION_1
Report	MQRO_NONE
MsgType	see Table 147
Expiry	30 seconds (note 1 on page 1894)
Feedback	MQFB_NONE
Encoding	MQENC_NATIVE
CodedCharSetId	depends on the bag CCSID (note 2 on page 1894)

Table 148. Message descriptor values (continued)

Parameter	Value
Format	see Table 147 on page 1893
Priority	MQPRI_PRIORITY_AS_Q_DEF
Persistence	MQPER_NOT_PERSISTENT
MsgId	MQMI_NONE
CorrelId	MQCI_NONE
BackoutCount	0
ReplyToQ	see note 3
ReplyToQMgr	blank

Notes:

1. This value can be overridden on the mqExecute call by using the **OptionsBag** parameter. For information about this, see "mqExecute" on page 1840.
2. See "Data conversion processing" on page 1892.
3. Name of the user-specified reply queue or MQAI-generated temporary dynamic queue for messages of type MQMT_REQUEST. Blank otherwise.

Example code

Here are some example uses of the mqExecute call.

The example shown in figure Figure 12 creates a local queue (with a maximum message length of 100 bytes) on a queue manager:

```
/* Create a bag for the data you want in your PCF message */
mqCreateBag(MQCBO_ADMIN_BAG, &hbagRequest)

/* Create a bag to be filled with the response from the command server */
mqCreateBag(MQCBO_ADMIN_BAG, &hbagResponse)

/* Create a queue  */
/* Supply queue name */
mqAddString(hbagRequest, MQCA_Q_NAME, "QBERT")

/* Supply queue type */
mqAddString(hbagRequest, MQIA_Q_TYPE, MQQT_LOCAL)

/* Maximum message length is an optional parameter */
mqAddString(hbagRequest, MQIA_MAX_MSG_LENGTH, 100)

/* Ask the command server to create the queue */
mqExecute(MQCMD_CREATE_Q, hbagRequest, hbagResponse)

/* Tidy up memory allocated */
mqDeleteBag(hbagRequest)
mqDeleteBag(hbagResponse)
```

Figure 12. Using mqExecute to create a local queue

The example shown in figure Figure 13 on page 1895 inquires about all attributes of a particular queue. The mqAddInquiry call identifies all IBM MQ object attributes of a queue to be returned by the Inquire parameter on mqExecute.

```
/* Create a bag for the data you want in your PCF message */
mqCreateBag(MQCBO_ADMIN_BAG, &hbagRequest)

/* Create a bag to be filled with the response from the command server */
mqCreateBag(MQCBO_ADMIN_BAG, &hbagResponse)

/* Inquire about a queue by supplying its name */
/* (other parameters are optional) */
mqAddString(hbagRequest, MQCA_Q_NAME, "QBERT")

/* Request the command server to inquire about the queue */
mqExecute(MQCMD_INQUIRE_Q, hbagRequest, hbagResponse)

/* If it worked, the attributes of the queue are returned */
/* in a system bag within the response bag */
mqInquireBag(hbagResponse, MQHA_BAG_HANDLE, 0, &hbagAttributes)

/* Inquire the name of the queue and its current depth */
mqInquireString(hbagAttributes, MQCA_Q_NAME, &stringAttribute)
mqInquireString(hbagAttributes, MQIA_CURRENT_Q_DEPTH, &integerAttribute)

/* Tidy up memory allocated */
mqDeleteBag(hbagRequest)
mqDeleteBag(hbagResponse)
```

Figure 13. Using mqExecute to inquire about queue attributes

Using mqExecute is the simplest way of administering IBM MQ, but lower-level calls, mqBagToBuffer and mqBufferToBag, can be used. For more information about the use of these calls, see The IBM MQ Administration Interface (MQAI).

Using the IBM MQ utilities on z/OS

z/OS

Reference information about the syntax, and usage of the various IBM MQ utility programs.

An overview of the IBM MQ utilities for z/OS

z/OS

Use this topic as a reference to the different categories of utilities.

This topic introduces the IBM MQ utility programs that are provided to help you perform various administrative tasks. The utility programs are described in the subsequent sections:

- The IBM MQ CSQUTIL utility program: Managing page sets
- The IBM MQ CSQUTIL utility program: Issuing commands
- The IBM MQ CSQUTIL utility program: Managing queues
- The IBM MQ CSQUTIL utility program: Migrating CSQXPARM
- The IBM MQ CSQJU003 Change log inventory utility
- The remaining IBM MQ utilities summarizes what you can do with these utilities.

Table 149. The IBM MQ CSQUTIL utility program: Managing page sets

Purpose	Function	See topic
Format VSAM data sets as IBM MQ page sets.	FORMAT	"Formatting page sets (FORMAT) on z/OS" on page 1903
Control recovery processing used for IBM MQ page sets.	FORMAT	"Formatting page sets (FORMAT) on z/OS" on page 1903
Extract page set information.	PAGEINFO	"Page set information (PAGEINFO) on z/OS" on page 1906
Copy IBM MQ page sets.	COPYPAGE	"Expanding a page set (COPYPAGE) on z/OS" on page 1907
Copy IBM MQ page sets and reset the log information.	RESETPAGE	"Copying a page set and resetting the log (RESETPAGE) on z/OS" on page 1909

Table 150. The IBM MQ CSQUTIL utility program: Issuing commands

Purpose	Function	See topic
Issue IBM MQ commands.	COMMAND	"Using the COMMAND function of CSQUTIL on z/OS" on page 1911
Produce a set of DEFINE, ALTER or DELETE commands for objects.	COMMAND	Making a list of DEFINE commands
Produce a client channel definition file.	COMMAND	Making a client channel definition file
Produce a set of DEFINE commands for objects (offline).	SDEFS	"Producing a list of IBM MQ define commands (SDEFS) on z/OS" on page 1918

Table 151. The IBM MQ CSQUTIL utility program: Managing queues

Purpose	Function	See topic
Copy contents of a queue to a data set.	COPY	"Copying queues into a data set while the queue manager is running (COPY) on z/OS" on page 1921
Copy contents of a queue to a data set (offline).	SCOPY	"Copying queues into a data set while the queue manager is not running (SCOPY) on z/OS" on page 1923
Delete contents of a queue.	EMPTY	"Emptying a queue of all messages (EMPTY) on z/OS" on page 1927

Table 151. The IBM MQ CSQUTIL utility program: Managing queues (continued)

Purpose	Function	See topic
Restore contents of a queue.	LOAD	"Restoring messages from a data set to a queue (LOAD) on z/OS" on page 1928

Table 152. The IBM MQ CSQUTIL utility program: Migrating CSQXPARM

Purpose	Function	See topic
Produce an ALTER QMGR command from a channel initiator parameter module.	XPARM	"Migrating a channel initiator parameter module (XPARM) on z/OS" on page 1933

Table 153. The IBM MQ CSQJU003 Change log inventory utility

Purpose	Function	See topic
Add active or archive log data sets.	NEWLOG	"Adding information about a data set to the BSDS (NEWLOG) on z/OS" on page 1937
Delete active or archive log data sets.	DELETE	"Deleting information about a data set from the BSDS (DELETE) on z/OS" on page 1940
Supply passwords for archive logs.	ARCHIVE	"Supplying a password for archive log data sets (ARCHIVE) on z/OS" on page 1940
Control the next restart of the queue manager.	CRESTART	"Controlling the next restart (CRESTART) on z/OS" on page 1941
Set checkpoint records.	CHECKPT	"Setting checkpoint records (CHECKPT) on z/OS" on page 1942
Update the highest written log RBA.	HIGHRBA	"Updating the highest written log RBA (HIGHRBA) on z/OS" on page 1943

Table 154. The remaining IBM MQ utilities

Name	Purpose	See topic
CSQJU004 (Print log map utility)	List information about the log.	"The print log map utility (CSQJU004) on z/OS" on page 1944
CSQ1LOGP (Log print utility)	Print the log. Extract log records into sequential files.	"The log print utility (CSQ1LOGP) on z/OS" on page 1945
CSQ5PQSG (IBM MQ table update utility)	Add and remove queue-sharing group and queue manager entries in the IBM MQ tables held in the shared Db2 data-sharing group.	"The queue-sharing group utility (CSQ5PQSG) on z/OS" on page 1955
CSQJUFMT (Active log preformat utility)	Preformat log data sets Preformat Shared Message Data Sets (SMDS)	"The active log preformat utility (CSQJUFMT) on z/OS" on page 1959
CSQUDLQH (Dead-letter queue handler utility)	Process messages on the dead-letter queue.	"The dead-letter queue handler utility (CSQUDLQH) on z/OS" on page 1960
CSQUCVX (Data conversion exit utility)	Generate data conversion exit routines.	Writing a data-conversion exit program for IBM MQ for z/OS
V 9.0.1 **CSQUDSPM** (Display queue manager utility)	Display information about queue managers. The equivalent function on Multiplatforms is **dspmq**.	"Display queue manager information utility (CSQUDSPM)" on page 1978

These utilities are located in the thlqual.SCSQAUTH or thlqual.SCSQLOAD IBM MQ load libraries. Concatenate the appropriate IBM MQ language load library thlqual.SCSQANLx (where x is the language letter) in the STEPLIB with the thlqual.SCSQAUTH and thlqual.SCSQLOAD. The utility control statements are available only in U.S. English. In some cases, the Db2 library db2qual.SDSNLOAD is also needed.

Syntax diagrams

z/OS

The syntax for a command and its options is presented in the form of a syntax diagram called a railroad diagram. Railroad diagrams are a visual format suitable for sighted users. It tells you what options you can supply with the command, how to enter them, indicates relationships between different options, and sometimes different values of an option.

Each railroad diagram begins with a double right arrow and ends with a right and left arrow pair. Lines beginning with a single right arrow are continuation lines. You read a railroad diagram from left to right and from top to bottom, following the direction of the arrows.

Other conventions used in railroad diagrams are:

Table 155. How to read railroad diagrams

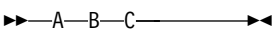
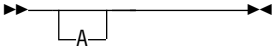
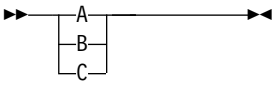
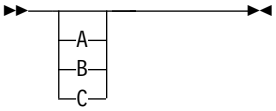
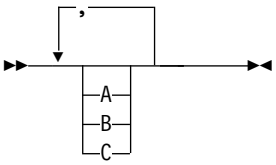
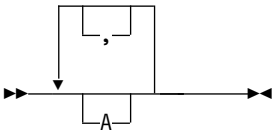
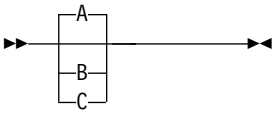
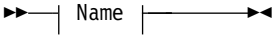
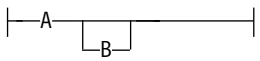
Convention	Meaning
	You must specify values `A`, `B`, and `C`. Required values are shown on the main line of a railroad diagram.
	You may specify value `A`. Optional values are shown below the main line of a railroad diagram.
	Values `A`, `B`, and `C` are alternatives, one of which you must specify.
	Values `A`, `B`, and `C` are alternatives, one of which you might specify.
	You might specify one or more of the values `A`, `B`, and `C`. Any required separator for multiple or repeated values (in this example, the comma (,)) is shown on the arrow.
	You might specify value `A` multiple times. The separator in this example is optional.
	Values `A`, `B`, and `C` are alternatives, one of which you might specify. If you specify none of the values shown, the default `A` (the value shown above the main line) is used.
 **Name:** 	The railroad fragment `Name` is shown separately from the main railroad diagram.

Table 155. How to read railroad diagrams (continued)

Convention	Meaning
Punctuation and uppercase values	Specify exactly as shown.

IBM MQ utility program (CSQUTIL) on z/OS

z/OS

The CSQUTIL utility program is provided with IBM MQ to help you to perform backup, restoration, and reorganization tasks, and to issue IBM MQ commands.

Through this utility program, you can invoke functions in these groups:

Page set management
These functions enable you to manage IBM MQ page sets. You can format data sets as page sets, change the recovery processing performed against page sets, extract page set information, increase the size of page sets and reset the log information contained in a page set. The page set must not belong to a queue manager that is currently running.

Command management
These functions enable you to:

- Issue commands to IBM MQ
- Produce a list of DEFINE, ALTER, or DELETE commands for your IBM MQ objects

Queue management
These functions enable you to back up and restore queues and page sets, copy queues and page sets to another queue manager, reset your queue manager, or to migrate from one queue manager to another.

Specifically, you can:

- Copy messages from a queue to a data set
- Delete messages from a queue
- Restore previously copied messages to their appropriate queues

The scope of these functions can be either:

- A *queue*, in which case the function operates on all messages in the specified queue.
- A *page set*, in which case the function operates on all the messages, in all the queues, on the specified page set.

Use these functions only for your own queues; do not use them for system queues (those with names beginning SYSTEM).

All the page set management functions, and some of the other functions, operate while the queue manager is not running, so you do not need any special authorization other than the appropriate access to the page set data sets. For the functions that operate while the queue manager is running, CSQUTIL runs as an ordinary z/OS batch IBM MQ program, issuing commands through the command server, and using the IBM MQ API to access queues.

You need the necessary authority to use the command server queues (SYSTEM.COMMAND.INPUT, SYSTEM.COMMAND.REPLY.MODEL, and SYSTEM.CSQUTIL.*), to use the IBM MQ DISPLAY commands, and to use the IBM MQ API to access any queues that you want to manage. See the usage notes for each function for more information.

Attention: If you use CSQUTIL to define a channel, and the connection name contains two parts (the host name and port number) you must enclose the host name and port number within single quotation

marks to maintain the limit on the number of permissible parameters. Similarly, if your connection name consists of an IP address and port number, you must enclose these parameters within single quotation marks.

Invoking the IBM MQ utility program on z/OS: z/OS

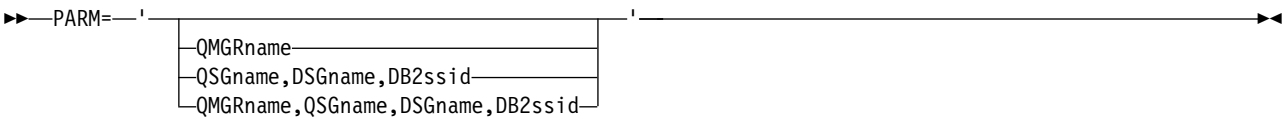
Use this topic to understand how to invoke CSQUTIL, the format of its parameters, and its return codes.

The CSQUTIL utility program runs as a z/OS batch program, below the 16 MB storage line. Specify the resources that the utility is to work with in the PARM parameter of the EXEC statement of the JCL.

```
// EXEC PGM=CSQUTIL,PARM=
```

Figure 14. How to invoke the CSQUTIL utility program

where PARM= expands to:



- PARM parameters
- Return codes

PARM parameters

QMGRname
Specifies the 1- to 4-character name of the queue manager or queue-sharing group to which CSQUTIL is to connect.

If you specify the name of a queue-sharing group, CSQUTIL connects to any queue manager in that group

QSGname
Specifies the 1- to 4-character name of the queue-sharing group from which CSQUTIL is to extract definitions.

DSGname
Specifies the 8-character name of the Db2 data-sharing group from which CSQUTIL is to extract definitions.

db2ssid
Specifies the 4-character name, or group attach name, of the Db2 database subsystem to which CSQUTIL is to attach for stand-alone functions.

Which PARM parameters do you need?

Figure 14 shows that you can specify one of four options on the PARM statement. The option you specify depends on the function you need to implement, as follows:

- Use `PARM=` (or omit it all together) if you are using only offline functions, and not QSGDISP(GROUP) or QSGDISP(SHARED).
- Use `PARM=' QMGRname '` only if you intend to use functions that require the queue manager to be running, such as COPY and COMMAND.
- Use `PARM=' QSGname,DSGname,db2ssid '` if you intend to use the SDEFS function with either QSGDISP(GROUP) or QSGDISP(SHARED) specified. This is because CSQUTIL requires access to Db2 to perform the SDEFS function in this situation.

- Use PARM=' *QMGRname,QSGname,DSGname,db2ssid* ' if you intend to combine the previous two functions in one CSQUTIL job.

If you specify a queue manager name as blanks, CSQUTIL uses the name of the default queue manager specified for z/OS batch programs in CSQBDEFV. The utility then uses this queue manager for the whole job step. When the utility connects to the queue manager, the authorization of the "signed-on user name"? is checked to see which functions the invocation is allowed to use.

You specify the functions required by statements in the SYSIN data set according to these rules:

- The data set must have a record length of 80.
- Only columns 1 through 72 are significant. Columns 73 through 80 are ignored.
- Records with an asterisk (*) in column 1 are interpreted as comments and are ignored.
- Blank records are ignored.
- Each statement must start on a new line.
- A trailing - means continue from column 1 of the next record.
- A trailing + means continue from the first non-blank column of the next record.
- The keywords of statements are not case-sensitive. However, some arguments, such as queue name, are case sensitive.

The utility statements refer to the default or explicitly named DDnames for input and output. Your job can use the COPY and LOAD functions repeatedly and process different page sets or queues during a single run of the utility.

All output messages are sent to the SYSPRINT data set, which must have a record format of VBA and a record length of 125.

While running, CSQUTIL uses temporary dynamic queues with names of the form SYSTEM.CSQUTIL.*

Return codes

When you are using the COMMAND verb to issue MQSC commands, you must use FAILURE(CONTINUE) so any failure in the commands that are issued give a non-zero return code. The default is FAILURE(IGNORE) and the return code from the command is always zero.

When CSQUTIL returns to the operating system, the return code can be:

0 All functions completed successfully.

4 Some functions completed successfully, some did not, or forced a sync point.

8 All the attempted functions failed.

12 No functions attempted; there was a syntax error in the statements or the expected data sets were missing.

In most cases, if a function fails or is forced to take a sync point, no further functions are attempted. In this case, the message CSQU147I replaces the normal completion message CSQU148I.

See the usage notes for each function for more information about success or failure.

Syncpoints

The queue management functions used when the queue manager is running operate within a syncpoint so that, if a function fails, its effects can be backed out. The queue manager attribute, MAXUMSGS, specifies the maximum number of messages that a task can get or put within a single unit of recovery.

MAXUMSGS should normally be set to a low value, both to protect against looping applications, and because there might be a very large processor cost in committing many messages.

The utility forcibly takes sync points as required and issues the warning message CSQU087I. If the function later fails, the changes already committed are not backed out. Do not just rerun the job to correct the problem or you might get duplicate messages on your queues. Instead, use the current depth of the queue to work out, from the utility output, which messages have not been backed out. Then determine the most appropriate course of action. For example, if the function is LOAD, you can empty the queue and start again, or you can choose to accept duplicate messages on the queues.

To avoid such difficulties if the function fails, but at the risk of incurring a large processor cost, set MAXUMSGS to be greater than:

- The number of messages in the queue, if you are working with a single queue.
- The number of messages in the longest queue in the page set, if you are working with an entire page set.

Use the DISPLAY QSTATUS command to find out the value of the CURDEPTH attribute, which is the current depth of the queue. To find out the value of MAXUMSGS, use the DISPLAY QMGR MAXUMSGS command.

Monitoring the progress of the IBM MQ utility program on z/OS: z/OS

You can monitor the progress of the CSQUTIL program by monitoring statements output to SYSPRINT.

To record the progress of CSQUTIL, every SYSIN statement is echoed to SYSPRINT.

The utility first checks the syntax of the statements in the SYSIN. The requested functions are started only if all the statements are syntactically correct.

Messages giving a commentary on the progress of each function are sent to SYSPRINT. When the processing of the utility is complete, statistics are printed with an indication of how the functions completed.

Formatting page sets (FORMAT) on z/OS: z/OS

You can use the CSQUTIL program to format page sets.

Use the FORMAT function to format page sets on all data sets specified by DDnames CSQP0000 through CSQP0099. In this way, you can format up to 100 page sets in a single invocation of the utility program. Use the FORCE keyword to reuse existing data sets.

You can also use the FORMAT function to change the recovery processing that is performed against page sets when the queue manager starts, using the TYPE keyword. This can assist in changing or recovering page sets, or reintroducing page sets that have been offline or suspended.

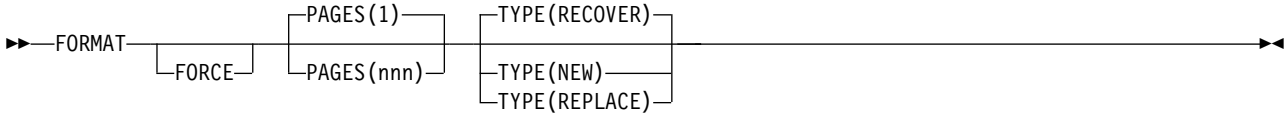
In summary:

- to reinstate a page set with no data, use FORMAT with the TYPE(NEW) option
- to reinstate a page set with old data, use FORMAT with the TYPE(REPLACE) option
- to reinstate a page set with old data made up-to-date, do not use FORMAT but start the queue manager with a backed-up copy of the page set

Page sets have identifiers (PSIDs, in the range 00 through 99) which are established by the DDnames used for the data sets in the queue manager started task procedure; DDname CSQP00nn specifies the

page set with identifier nn. The DDnames you use for the FORMAT function do not have to correspond to those used in the queue manager started task procedure, and do not therefore have any significance regarding page set identifiers.

Page set management (FORMAT)



- Keywords and parameters
- Example
- Usage notes

Keywords and parameters

FORCE
Specifies that existing data sets are to be reused without having to delete and redefine them first. You must define any page sets you want to reuse with the REUSE attribute in the AMS DEFINE CLUSTER statement. For more information about DEFINE CLUSTER, see the *DFSMS/MVS Access Method Services for VSAM* or the *DFSMS/MVS Access Method Services for the Integrated Catalog Facility* manual.

The FORCE keyword is not valid if TYPE(REPLACE) is specified.

PAGES *(nnn)*
Specifies the minimum number of pages to format in each page set. This enables a data set that spans more than one volume to be formatted.

Formatting of the data set is always done in whole space allocations, as specified as primary or secondary quantities when the data set is defined. The number of space allocations formatted is the minimum necessary to provide the requested number of pages; if there is insufficient data set space available, as many extents as can be obtained are formatted. If an existing page set is being reused (with the FORCE keyword), the whole page set is formatted, if that is larger.

The number of pages must be in the range 1 through 16 777 213 (because the maximum page set size is 64 GB (gigabytes)). The default is 1.

The PAGES keyword is not valid if TYPE(REPLACE) is specified.

TYPE
Specifies the type of recovery processing that is performed against queue manager page sets. Values are:

RECOVER

Use RECOVER for a data set that is to be a new page set for a queue manager (that is, to have a PSID which was never been used before).

This is the default.

The data set is formatted, and any messages or other data are erased. If a DDname is added to the queue manager's started task procedure for the new PSID that specifies this data set, it will be recognized as a new page set when the queue manager is restarted.

If such a data set was used as a page set with a PSID that has been used before, on restart the queue manager attempts to recover all queues and their messages that use storage classes that reference the page set from the time the page set was first used. This may make restart a lengthy process, and is unlikely to be what is wanted.

NEW

Use NEW for a data set that is to be a page set with a PSID that has been used before for a queue manager and with data that can be discarded, to restart a failed queue manager quickly or to reintroduce the page set after it has been offline or suspended.

The data set is formatted, and any messages or other data are erased. When the queue manager is restarted, with a DDname for the old PSID that specifies this data set, it does not recover the page set but treats it as if it has been newly added to the queue manager, and any historical information about it is discarded. All queues that use storage classes referencing this page set are cleared of all messages, in a similar fashion to the way that nonpersistent messages are cleared during restart processing. This means that there will be no effect on restart time.

REPLACE

Use REPLACE for a data set with a PSID that has been used before for a queue manager and with data that is known to be consistent and up to date, to reintroduce the page set after being offline or suspended.

The data set is not formatted, and any messages or other data are preserved. When the queue manager is restarted with a DDname for the PSID that specifies this data set, it does not recover the page set but treats it as if it has never been offline, or suspended, and any historical information about it is retained. All queues that use storage classes that reference the page set keep their messages. This means that there will be no effect on restart time.

This option will only be successful if the page set is in a consistent state; that is, on its last use the queue manager was terminated normally by a STOP QMGR MODE(FORCE) or MODE(QUIESCE) command.

Example

Figure 15 illustrates how the FORMAT command is invoked from CSQUTIL. In this example, two page sets, referenced by CSQP0000 and CSQP0003, are formatted by CSQUTIL.

```
//FORMAT   EXEC PGM=CSQUTIL
//STEPLIB  DD   DISP=SHR,DSN=thlqual.SCSQANLE
//         DD   DISP=SHR,DSN=thlqual.SCSQAUTH
//CSQP0000 DD   DISP=OLD,DSN=pageset.dsname0
//CSQP0003 DD   DISP=OLD,DSN=pageset.dsname3
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
FORMAT
/*
```

Figure 15. Sample JCL for the FORMAT function of CSQUTIL

Figure 16 illustrates how the FORMAT command with the TYPE option is invoked from CSQUTIL. In this example, the page set referenced by CSQP0003 is formatted by CSQUTIL.

```
//FORMAT   EXEC PGM=CSQUTIL
//STEPLIB  DD   DISP=SHR,DSN=thlqual.SCSQANLE
//         DD   DISP=SHR,DSN=thlqual.SCSQAUTH
//CSQP0003 DD   DISP=OLD,DSN=page set.dsname3
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
FORMAT TYPE(RECOVER)
/*
```

Figure 16. Sample JCL for the FORMAT function of CSQUTIL with the TYPE option

Usage notes

1. You cannot format page sets that belong to a queue manager that is still running.
2. When you use FORMAT, it is not necessary to specify a queue manager name.
3. If you use TYPE(REPLACE), recovery logs starting from when the page set was first used with the queue manager, or from when the page set was last formatted, must be available.
4. If you use data set names in which the queue manager name is a high-level qualifier, you can more easily identify which page sets are used by which queue manager, if more than one queue manager is defined.
5. Any update to a resource due to the resolution of an incomplete unit of work, where the update relates to a page on a page set that has been formatted with TYPE(REPLACE) or TYPE(NEW), is not honored. The update to the resource is lost.
6. If there is an error when formatting a page set, it does not prevent other page sets from being formatted, although the FORMAT function is considered to have failed.
7. Failure of this function does not prevent other CSQUTIL functions being attempted.

Page set information (PAGEINFO) on z/OS: z/OS

Use the PAGEINFO function to extract page set information from one or more page sets, specified by DDnames in the range CSQP0000 through CSQP0099, for the source data sets from which page set information is required.

Page set management (PAGEINFO)

Keywords and parameters

There are no keywords or parameters.

Example

In Figure 17, page set information is required from two existing page sets.

```
//PAGEINFO EXEC PGM=CSQUTIL
//STEPLIB  DD   DISP=SHR,DSN=thlqual.SCSQANLE
//         DD   DISP=SHR,DSN=thlqual.SCSQAUTH
//CSQP0001 DD   DISP=OLD,DSN=page set.existing.name1
//CSQP0006 DD   DISP=OLD,DSN=page set.existing.name6
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD
* Extract page set information for 2 existing page sets (CSQS0001 and CSQS0006)
 PAGEINFO
/*
```

Figure 17. Sample JCL showing the use of the PAGEINFO function

where:

CSQP0001, CSQP0006
Are the DDnames of the source data sets from which you want to extract page set information.

Information returned from PAGEINFO might include:

- Page set number
- Number of pages in a page set

- Queue manager associated with a page set
- Utility status information
- Page set recovery RBA for each page set
- System recovery RBA for all the page sets reported on by the PAGEINFO function

Usage notes

1. You cannot use PAGEINFO on the page sets of a queue manager that is running.
2. Failure of this function does not prevent other CSQUTIL functions from being attempted.
3. If you attempt to use the PAGEINFO function after the queue manager has terminated abnormally, the page sets might not have been closed properly. If a page set has not been closed properly, you cannot successfully run the PAGEINFO function against it. To avoid this problem, run the AMS VERIFY command before using the PAGEINFO function. The AMS VERIFY command might produce error messages. However, it does close the page sets properly so that the PAGEINFO function can complete successfully.

   For more information about the AMS VERIFY command, see the *DFSMS/MVS™ Access Method Services for VSAM* or the *DFSMS/MVS Access Method Services for the Integrated Catalog Facility* manual.
4. The system recovery RBA relates only to those page sets processed; it does not relate to the whole queue manager unless all the page sets for the queue manager are included. If the page sets are from more than one queue manager, no system recovery RBA can be determined.

Expanding a page set (COPYPAGE) on z/OS: z/OS

Use the COPYPAGE function to copy one or more page sets to a larger page set.

Note: The COPYPAGE function is only used for *expanding* page sets. It is not used for making backup copies of page sets. If you want to do this, use AMS REPRO as described in How to back up and recover page sets. When you have used the COPYPAGE function, the page sets cannot be used by a queue manager with a different name, so do not rename your queue manager.

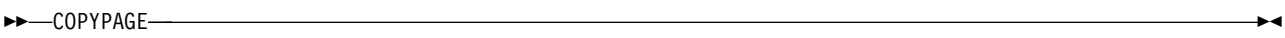
Use the COPYPAGE function to copy one or more page sets to a larger page set. All queues and messages on the page set are copied. If you copy page set zero, all the IBM MQ object definitions are also copied. Each page set is copied to a destination data set that must be formatted as a page set. Copying to a smaller page set is not supported.

If you use this function, you must modify the page set definition in the started task procedure to reflect the change of the name of the data set on which the new page set resides.

To use the COPYPAGE function, define DDnames in the range CSQS0000 through CSQS0099 for the source data sets, and define DDnames for the target data sets from CSQT0000 through CSQT0099.

For more information, see Managing page sets.

Page set management (COPYPAGE)



Keywords and parameters

There are no keywords or parameters.

Example

In Sample JCL showing the use of the COPYPAGE function, two existing page sets are copied onto two new page sets. The procedure for this is:

1. Set up the required DDnames, where:

   CSQP0005, CSQP0006
   Identify the destination data sets. These DDnames are used by the FORMAT function.

   CSQS0005, CSQS0006
   Identify the source data sets containing the two page sets you want to copy.

   CSQT0005, CSQT0006
   Identify the destination data sets (page sets), but this time for the COPYPAGE function.
2. Format the destination data sets, referenced by DDnames CSQP0005 and CSQP0006, as page sets using the FORMAT function.
3. Copy the two existing page sets onto the new page sets using the COPYPAGE function.

```
//JOBLIB  DD  DISP=SHR,DSN=ANTZ.MQ.&VER..&LVL..OUT.SCSQANLE
//        DD  DISP=SHR,DSN=ANTZ.MQ.&VER..&LVL..OUT.SCSQAUTH
//*
//S1    EXEC PGM=IDCAMS
//* Delete any prior attempt, then allocate a new larger pageset
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
DELETE 'VICY.MQ38.PAGE01.NEW' CLUSTER
DEFINE CLUSTER (NAME('VICY.MQ38.PAGE01.NEW') +
MODEL('VICY.MQ38.PAGE01') +
DATACLASS(EXTENDED) +
LINEAR CYLINDERS(100,50))
//*
//MQMUTIL  EXEC PGM=CSQUTIL,PARM='',REGION=4M
//* CSQUTIL
//*  FORMAT acts on DDNAME like CSQPnnnn
//*  optional, FORMAT PAGES(nnn) to force allocation and format of
//*       secondary extents.
//*  COPYPAGE copies from source, CSQSnnnn
//*       to  target, CSQTnnnn
//SYSPRINT DD  SYSOUT=*
//CSQP0001 DD  DISP=SHR,DSN=VICY.MQ38.PAGE01.NEW
//CSQS0001 DD  DISP=SHR,DSN=VICY.MQ38.PAGE01
//CSQT0001 DD  DISP=SHR,DSN=VICY.MQ38.PAGE01.NEW
//SYSIN    DD  * FORMAT COPYPAGE
//*
//RENAME  EXEC PGM=IDCAMS
//* the cluster and data components must be renamed independently
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
ALTER   'VICY.MQ38.PAGE01' NEWNAME('VICY.MQ38.PAGE01.OLD')
ALTER   'VICY.MQ38.PAGE01.DATA' +
NEWNAME('VICY.MQ38.PAGE01.OLD.DATA')
ALTER   'VICY.MQ38.PAGE01.NEW' +
NEWNAME('VICY.MQ38.PAGE01')
ALTER   'VICY.MQ38.PAGE01.NEW.DATA') +
NEWNAME('VICY.MQ38.PAGE01.DATA')
/*
```

Figure 18. Sample JCL showing the use of the COPYPAGE function

Usage notes

1. You cannot use COPYPAGE on page sets of a queue manager that is running.
2. Using COPYPAGE involves stopping the queue manager. This results in the loss of nonpersistent messages.
3. Before you use COPYPAGE, the new data sets must be preformatted as page sets. To do this, use the FORMAT function, as shown in Figure 18.
4. Ensure that the new (destination) data sets are larger than the old (source) data sets.
5. You cannot change the page set identifier (PSID) associated with a page set. For example, you cannot 'make' page set 03 become page set 05.

6. Failure of this function does not prevent other CSQUTIL functions from being attempted.
7. If you attempt to use the COPYPAGE function after the queue manager has terminated abnormally, the page sets might not have been closed properly. If a page set has not been closed properly, you cannot successfully run the COPYPAGE function against it.

   To avoid this problem, run the AMS VERIFY command before using the COPYPAGE function. The AMS VERIFY command might produce error messages. However, it does close the page sets properly, so that the COPYPAGE function can complete successfully.

   For more information about the AMS VERIFY command, see the *DFSMS/MVS Access Method Services for VSAM* or the *DFSMS/MVS Access Method Services for the Integrated Catalog Facility* manual.

Copying a page set and resetting the log (RESETPAGE) on z/OS: z/OS

The RESETPAGE function is like the COPYPAGE function except that it also resets the log information in the new page sets.

RESETPAGE lets you restart the queue manager from a known, valid set of page sets, even if the corresponding log data sets have been corrupted.

The source page sets for RESETPAGE must be in a consistent state. They must be either:

- Page sets that have been through a successful queue manager shutdown using the IBM MQ command STOP QMGR.
- Copies of page sets that have been through a successful stop.

The RESETPAGE function must not be run against copies of page sets made using fuzzy backup (see Method 2: Fuzzy backup), or against page sets that are from a queue manager that has terminated abnormally.

RESETPAGE either:

- Copies page sets on all data sets referenced by DDnames CSQS0000 through CSQS0099 to new data sets referenced by DDnames CSQT0000 through CSQT0099. If you use this function, modify the page set definition in the started task procedure to reflect the change of the name of the data set on which the new page set resides.
- Resets the log information in the page set referenced by DDnames CSQP0000 through CSQP0099.

For more information, see Managing page sets.

Using the RESETPAGE function

You can use the RESETPAGE function to update a set of consistent page sets so that they can be used with a set of new (clean) BSDS and log data sets to start the queue manager. You only have to use the RESETPAGE function if both copies of the log have been lost or damaged; you can restart from backup copies of page sets (and accept the resulting loss of data from the time the copies were made), or from your existing page sets.

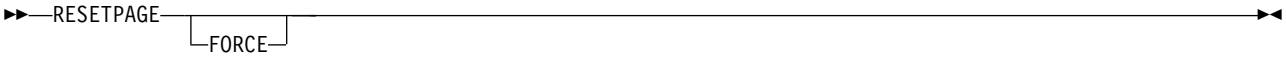
In this situation, use the RESETPAGE function on **all** the page sets of the affected queue manager. You must also create new BSDS and log data sets.

Note: Do not use the RESETPAGE function on a subset of the page sets known to IBM MQ.

If you run the RESETPAGE function against any page sets, but do not provide clean BSDS and log data sets for the queue manager, IBM MQ attempts to recover the logs from RBA zero, and treats the page sets as empty. For example, the following messages are produced if you attempt to use the RESETPAGE function to generate page sets zero, 1, 2, and 3 without providing a clean set of BSDS and log data sets:

```
CSQI021I +CSQ1 CSQIECUR PAGE SET 0 IS EMPTY. MEDIA RECOVERY STARTED
CSQI021I +CSQ1 CSQIECUR PAGE SET 1 IS EMPTY. MEDIA RECOVERY STARTED
CSQI021I +CSQ1 CSQIECUR PAGE SET 2 IS EMPTY. MEDIA RECOVERY STARTED
CSQI021I +CSQ1 CSQIECUR PAGE SET 3 IS EMPTY. MEDIA RECOVERY STARTED
```

Page set management (RESETPAGE)



Keywords and parameters

FORCE
Specifies that the page sets specified by DDnames CSQP0000 through CSQP00nn are to be reset in place.

If FORCE is not specified, the page sets specified by DDnames CSQS0000 through CSQS00nn are copied to new page sets specified by DDnames CSQT0000 through CSQT00nn. This is the default.

Example

An existing page set, referenced by DDname CSQS0007, is copied to a new data set referenced by DDname CSQT0007. The new data set, which is also referenced by DDname CSQP0007, is already formatted as a page set before the RESETPAGE function is called.

```
//RESTPAGE EXEC PGM=CSQUTIL
//STEPLIB  DD   DISP=SHR,DSN=thlqual.SCSQANLE
//         DD   DISP=SHR,DSN=thlqual.SCSQAUTH
//CSQP0007 DD   DISP=OLD,DSN=pageset.newname7
//CSQS0007 DD   DISP=OLD,DSN=pageset.oldname7
//CSQT0007 DD   DISP=OLD,DSN=pageset.newname7
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
* Format new data set, CSQP0007, as page set
 FORMAT
* Copy page set CSQS0007 to CSQT0007 and reset it
 RESETPAGE
/*
```

Figure 19. Sample JCL showing the use of the RESETPAGE function

Usage notes

1. Do not use the RESETPAGE function against page sets after the queue manager has terminated abnormally. Page sets from a queue manager that terminated abnormally will probably contain inconsistent data; using RESETPAGE on page sets in this state leads to data integrity problems.
2. You cannot use RESETPAGE on page sets belonging to a queue manager that is running.
3. Before you use RESETPAGE, the new data sets must be pre-formatted as page sets. To do this, use the FORMAT function, as shown in Figure 19.
4. Ensure that the new (destination) data sets are larger than the old (source) data sets.
5. You cannot change the page set identifier (PSID) associated with a page set. For example, you cannot 'make' page set 03 become page set 05.
6. Failure of this function does not prevent other CSQUTIL functions from being attempted.

Using the COMMAND function of CSQUTIL on z/OS: z/OS

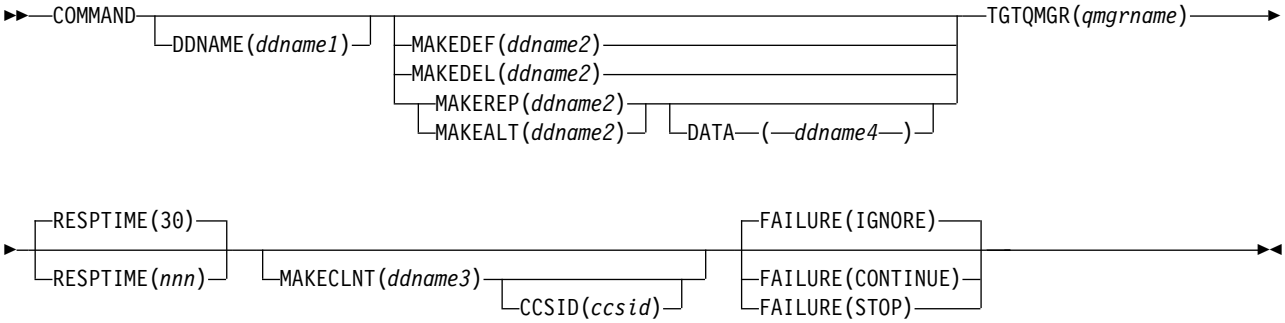
You can use the COMMAND function of CSQUTIL to direct commands to the queue manager.

Use the COMMAND function to:

1. Pass commands from an input data set to the queue manager.
2. Produce a list of DEFINE commands that describe the objects in a queue manager. The commands can be used to keep a record of the object definitions or to regenerate all or part of a queue manager's objects as part of a migration from one queue manager to another.
3. Produce a list of commands to change or delete a set of objects in a queue manager.
4. Make a client channel definition file.

The queue manager specified in the PARM parameter of the EXEC statement must be running.

Command management (COMMAND)



- Keywords and parameters
- Examples
- Usage notes for CSQUTIL COMMAND

If you use **FAILURE** (*IGNORE*) the job step always obtains return code 0.

If you use **FAILURE** (*STOP*) or **FAILURE** (*CONTINUE*) the job step obtains return code 8 if there were any non zero return codes from the statements.

You should use **FAILURE** (*STOP*) or **FAILURE** (*CONTINUE*) to report any errors in the definitions.

Keywords and parameters

DDNAME(*ddname1***)**
Specifies that the commands are to be read from a named input data set. If this keyword is omitted, the default DDname, CSQUCMD, is used.

ddname1 specifies the DDname that identifies the input data set from which commands are to be read.

MAKEDEF(*ddname2***), MAKEDEL(***ddname2***), MAKEREP(***ddname2***), MAKEALT(***ddname2***)**
Specify that commands are to be generated from any DISPLAY object commands in the input data set.

The commands that are generated are:

MAKEDEF
DEFINE NOREPLACE, with all the attributes and values returned by the DISPLAY commands. For the queue manager object, an ALTER command is generated with all the attributes and values. For channel authentication records, a SET command is generated.

Both CSQUTIL SDEFS and the CSQUTIL COMMAND with the MAKEDEF option can be used to produce a set of MQSC commands to re-create the objects currently defined in the queue manager.

The difference between the two is that CSQUTIL COMMAND must be run against an active queue manager and is most appropriate for regular backup of object definitions, whereas CSQUTIL SDEFS can be used to re-create definitions for a queue manager that is not currently running. This makes the CSQUTIL SDEFS option more appropriate for recovery scenarios.

MAKEDEL
DELETE. For local queues, NOPURGE is used. For channel authentication records, a SET command with ACTION(REMOVE) is used

MAKEREP
DEFINE REPLACE, with any keywords and values from the data set specified by the DATA keyword. For channel authentication records, a SET command with ACTION(REPLACE) is used.

MAKEALT
ALTER, with any keywords and values from the data set specified by the DATA keyword. For channel authentication records, a SET command with ACTION(REPLACE) is used.

Only one of these keywords may be specified. If these keywords are omitted, no commands are generated.

ddname2 specifies the DDname that identifies the output data set in which the DEFINE, DELETE or ALTER commands are to be stored. The data set should be RECFM=FB, LRECL=80. This data set can then be used as input for a later invocation of the COMMAND function or it can be incorporated into the initialization data sets CSQINP1 and CSQINP2.

DATA(*ddname4*)
ddname4 specifies a data set from which command keywords and values are to be read, and appended to each command generated for MAKEREP or MAKEALT.

TGTQMGR(*qmgrname*)
Specifies the name of the queue manager where you want the commands to be performed. You can specify a target queue manager that is not the one you connect to. In this case, you would normally specify the name of a remote queue manager object that provides a queue manager alias definition (the name is used as the *ObjectQMgrName* when opening the command input queue). To do this, you must have suitable queues and channels set up to access the remote queue manager.

The default is that commands are performed on the queue manager to which you are connected, as specified in the PARM field of the EXEC statement.

RESPTIME(*nnn*)
Specifies the time in seconds to wait for a response to each command, in the range 5 through 999.

The default is 30 seconds.

MAKECLNT(*ddname3*)

Specifies that a client channel definition file is generated from any DISPLAY CHANNEL commands in the input data set that return information about client-connection channels, and any DISPLAY AUTHINFO commands that return information about authentication information objects for which the LDAPUSER and LDAPPWD attributes are not set.

If this keyword is omitted, no file is generated.

Important: The MAKECLNT utility is now stabilized at the IBM WebSphere MQ Version 7.1 level. You should use the **runmqsc** command using the **-n** option; see "runmqsc (run MQSC commands)" on page 310 for further information.

ddname3 specifies the DDname that identifies the output data set in which the generated file is to be stored; the data set should be RECFM=U, LRECL=6144. The file can then be downloaded as binary data to the client machine by a suitable file transfer program.

CCSID(*ccsid***)**
Specifies the coded character set identifier (CCSID) that is to be used for the data in a client channel definition file. The value must be in the range 1 through 65535; the default is 437. You can only specify CCSID if you also specify MAKECLNT.

Note: IBM MQ assumes that the data is to be in ASCII, and that the encoding for numeric data is to be MQENC_INTEGER_REVERSED.

FAILURE
Specifies what action to take if an IBM MQ command that is issued fails to execute successfully. Values are:

IGNORE
Ignore the failure; continue reading and issuing commands, and treat the COMMAND function as being successful. This is the default.

CONTINUE
Read and issue any remaining commands in the input data set, but treat the COMMAND function as being unsuccessful.

STOP Do not read or issue any more commands, and treat the COMMAND function as being unsuccessful.

Examples

This section gives examples of using the COMMAND function for the following:

- "Issuing commands"
- "Making a list of DEFINE commands" on page 1914
- "Making a list of ALTER commands" on page 1915
- "Making a client channel definition file" on page 1916

Issuing commands

In Figure 20 on page 1914, the data sets referenced by DDnames CSQUCMD and OTHER contain sets of commands. The first COMMAND statement takes commands from the default input data set MY.COMMANDS(COMMAND1) and passes them to the queue manager. The second COMMAND statement takes commands from the input data set MY.COMMANDS(OTHER1), which is referenced by DDname OTHER, and passes them to the queue manager.

```
//COMMAND EXEC PGM=CSQUTIL,PARM='CSQ1'
//STEPLIB  DD   DISP=SHR,DSN=thlqual.SCSQANLE
//         DD   DISP=SHR,DSN=thlqual.SCSQAUTH
//CSQUCMD  DD DSN=MY.COMMANDS(COMMAND1),DISP=SHR
//OTHER    DD DSN=MY.COMMANDS(OTHER1),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
* THE NEXT STATEMENT CAUSES COMMANDS TO BE READ FROM CSQUCMD DDNAME
COMMAND
* THE NEXT SET OF COMMANDS WILL COME FROM 'OTHER' DDNAME
COMMAND DDNAME(OTHER)
* THE NEXT STATEMENT CAUSES COMMANDS TO BE READ FROM CSQUCMD
* DDNAME AND ISSUED ON QUEUE MANAGER CSQ2 WITH A RESPONSE TIME
* OF 10 SECONDS
COMMAND TGTQMGR(CSQ2) RESPTIME(10)
/*
```

Figure 20. Sample JCL for issuing IBM MQ commands using CSQUTIL

Making a list of DEFINE commands

In Figure 21 on page 1915, the data set referenced by DDname CMDINP contains a set of DISPLAY commands. These DISPLAY commands specify generic names for each object type (except the queue manager itself). If you run these commands, a list is produced containing all the IBM MQ objects. In these DISPLAY commands, the ALL keyword is specified to ensure that all the attributes of all the objects are included in the list, and that all queue-sharing group dispositions are included.

The MAKEDEF keyword causes this list to be converted into a corresponding set of DEFINE NOREPLACE commands (ALTER for the queue manager). These commands are put into a data set referenced by the **ddname2** parameter of the MAKEDEF keyword, that is, OUTPUT1. If you run this set of commands, IBM MQ regenerates all the object definitions in the queue manager.

```
//QDEFS   EXEC PGM=CSQUTIL,PARM='CSQ1'
//STEPLIB  DD   DISP=SHR,DSN=thlqual.SCSQANLE
//         DD   DISP=SHR,DSN=thlqual.SCSQAUTH
//OUTPUT1  DD   DISP=OLD,DSN=MY.COMMANDS(DEFS)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
COMMAND DDNAME(CMDINP) MAKEDEF(OUTPUT1)
/*
//CMDINP   DD *
DISPLAY STGCLASS(*) ALL QSGDISP(QMGR)
DISPLAY STGCLASS(*) ALL QSGDISP(GROUP)
DISPLAY CFSTRUCT(*) ALL

DISPLAY QUEUE(*) ALL QSGDISP(QMGR)
DISPLAY QUEUE(*) ALL QSGDISP(GROUP)
DISPLAY QUEUE(*) ALL QSGDISP(SHARED)
DISPLAY TOPIC(*) ALL QSGDISP(QMGR)
DISPLAY TOPIC(*) ALL QSGDISP(GROUP)
DISPLAY NAMELIST(*) ALL QSGDISP(QMGR)
DISPLAY NAMELIST(*) ALL QSGDISP(GROUP)
DISPLAY PROCESS(*) ALL QSGDISP(QMGR)
DISPLAY PROCESS(*) ALL QSGDISP(GROUP)
DISPLAY CHANNEL(*) ALL QSGDISP(QMGR)
DISPLAY CHANNEL(*) ALL QSGDISP(GROUP)
DISPLAY AUTHINFO(*) ALL QSGDISP(QMGR)
DISPLAY AUTHINFO(*) ALL QSGDISP(GROUP)
DISPLAY CHLAUTH('*') ALL
DIS SUB(*) SUBTYPE(ADMIN) ALL DISTYPE(DEFINED)

DISPLAY QMGR ALL

/*
```

Figure 21. Sample JCL for using the MAKEDEF option of the COMMAND function

Making a list of ALTER commands

In Figure 22 on page 1916, the data set referenced by DDname CMDINP contains a DISPLAY command that will produce a list of all local queues with names beginning "ABC".

The MAKEALT keyword causes this list to be converted into a corresponding set of ALTER commands, each of which includes the data from the data set referenced by DDname CMDALT. These commands are put into a data set referenced by the ddname2 parameter of the MAKEALT keyword, that is, OUTPUTA. If you run this set of commands, all the local queues with names beginning "ABC" will be disabled for PUT and GET.

```
//QALTS  EXEC PGM=CSQUTIL,PARM='CSQ1 '
//STEPLIB  DD DISP=SHR,DSN=thlqual.SCSQANLE
//         DD DISP=SHR,DSN=thlqual.SCSQAUTH
//OUTPUTA  DD DISP=OLD,DSN=MY.COMMANDS(ALTS)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
COMMAND DDNAME(CMDINP) MAKEALT(OUTPUTA) DATA(CMDALT)
/*
//CMDINP DD *
DISPLAY QLOCAL(ABC*)
/*
//CMDALT DD *
PUT(DISABLED) +
GET(DISABLED)
/*
```

Figure 22. Sample JCL for using the MAKEALT option of the COMMAND function

Making a client channel definition file

In Figure 23, the data set referenced by DDname CMDCHL contains a DISPLAY CHANNEL command and a DISPLAY AUTHINFO command. The DISPLAY commands specify a generic name and the ALL keyword is specified to ensure that all the attributes are included.

The MAKECLNT keyword converts these attributes into a corresponding set of client channel definitions. These are put into a data set referenced by the *ddname3* parameter of the MAKECLNT keyword, that is, OUTCLNT, which is ready to be downloaded to the client machine.

```
//CLIENT  EXEC PGM=CSQUTIL,PARM='CSQ1'
//STEPLIB  DD   DISP=SHR,DSN=thlqual.SCSQANLE
//         DD   DISP=SHR,DSN=thlqual.SCSQAUTH
//OUTCLNT  DD   DISP=OLD,DSN=MY.CLIENTS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
COMMAND DDNAME(CMDCHL) MAKECLNT(OUTCLNT)
/*
//CMDCHL   DD *
DISPLAY CHANNEL(*) ALL TYPE(CLNTCONN)
DISPLAY AUTHINFO(*) ALL
/*
```

Figure 23. Sample JCL for using the MAKECLNT option of the COMMAND function

Usage notes for CSQUTIL COMMAND

1. The rules for specifying commands in the input data set are the same as for the initialization data sets:
   - The data set must have a record length of 80.
   - Only columns 1 through 72 are significant. Columns 73 through 80 are ignored.
   - Records with an asterisk (*) in column 1 are interpreted as comments and are ignored.
   - Blank records are ignored.
   - Each command must start on a new record.
   - A trailing - means continue from column 1 of the next record.
   - A trailing + means continue from the first non-blank column of the next record.
   - The maximum number of characters permitted in a command is 32 762.

   With the additional rule:
   - A semicolon (;) can be used to terminate a command; the remaining data in the record is ignored.

See "Building command scripts" on page 364 for more information about the rules for building IBM MQ commands.

2. If you specify the MAKEDEF keyword:
   - In the input data set, the DISPLAY commands for objects must contain the ALL parameter so that the complete definition of each object is produced. See Figure 21 on page 1915.
   - To obtain a complete definition, you must DISPLAY the following:
     - queues
     - topic
     - namelists
     - process definitions
     - channels
     - storage classes
     - authentication information objects
     - CF structures
     - channel authentication records
     - queue manager

     Note: DEFINE commands are not generated for any local queues that can be identified as dynamic, or for channels that were defined automatically.
   - Do not specify the same MAKEDEF data set for more than one COMMAND function, unless its DD statement specifies a sequential data set with DISP=MOD.
3. If you specify the MAKEREP, MAKEALT, or MAKEDEL keywords:
   - In the input data set, include DISPLAY commands that select the set of objects for which you want to generate commands.
   - For MAKEREP and MAKEALT, the data (if any) from the data set specified by the DATA keyword is appended to each generated command, exactly as entered. The format of the data set and the rules for specifying command data are the same as for the command input data set. Because the same data is appended to each command, if you want to process several sets of objects, you will need to use several separate COMMAND functions, each with a different DATA data set.
   - Commands are not generated for channels that were defined automatically.
4. If you specify the MAKEDEF, MAKEREP, MAKEALT, or MAKEDEL keywords, commands are generated only for objects reported by the target queue manager (as specified by the TGTQMGR keyword or defaulted), even if CMDSCOPE is used in the DISPLAY commands. To generate commands for several queue managers in a queue-sharing group, use a separate COMMAND function for each.

   In a queue-sharing group, queues, processes, channels, storage classes and authentication information objects should each have two DISPLAY commands, one with QSGDISP(QMGR) and one with QSGDISP(GROUP). Queues should have a third with QSGDISP(SHARED). It is not necessary to specify QSGDISP(COPY) because the required commands will be generated automatically when the commands for objects with QSGDISP(GROUP) are issued.
5. Do not specify the same MAKEDEF, MAKEREP, MAKEALT, or MAKEDEL data set for more than one COMMAND function, unless its DD statement specifies a sequential data set with DISP=MOD.
6. If you specify the MAKECLNT keyword:
   - In the input data set, the display commands for channels and authentication information objects must contain the ALL parameter so that the complete definition of each channel and authentication information object is produced.
   - If the DISPLAY commands return information for a particular channel more than once, only the last set of information is used.
   - Do not specify the same client definition file data set for more than one COMMAND function, unless its DD statement specifies a sequential data set with DISP=MOD.

7. The results of DISPLAY commands used in conjunction with MAKEDEF, MAKEREP, MAKEALT, MAKEDEL or MAKECLNT are also sent to SYSPRINT.
8. If you specify the FAILURE keyword, a command is determined to be a success or failure according to the codes returned in message CSQN205I. If the return code is 00000000 and the reason code is 00000000 or 00000004, it is a success; for all other values it is a failure.
9. The COMMAND function is determined to be a success only if both:
   - All the commands in the input data set are read and issued and get a response from IBM MQ, regardless of whether the response indicates successful execution of the command or not.
   - Every command issued executes successfully, if FAILURE(CONTINUE) or FAILURE(STOP) is specified.

   If COMMAND fails, no further CSQUTIL functions are attempted.
10. You need the necessary authority to use command server queues (SYSTEM.COMMAND.INPUT, SYSTEM.COMMAND.REPLY.MODEL, and SYSTEM.CSQUTIL.*) and to use the IBM MQ commands that you want to issue.

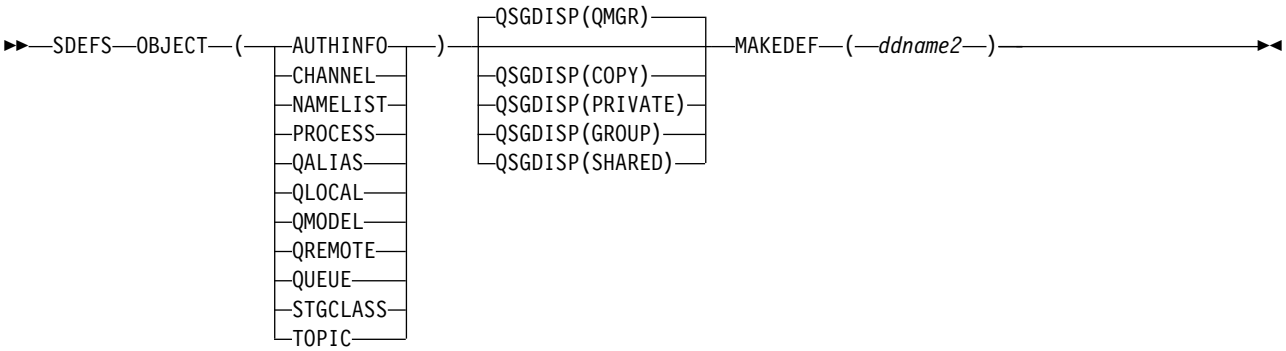
Producing a list of IBM MQ define commands (SDEFS) on z/OS: z/OS

You can use the SDEFS function of CSQUTIL to produce a list of DEFINE commands describing the objects in your queue manager or queue-sharing group.

Both CSQUTIL SDEFS and the CSQUTIL COMMAND with the MAKEDEF option can be used to produce a set of MQSC commands to re-create the objects currently defined in the queue manager.

The difference between the two is that CSQUTIL COMMAND must be run against an active queue manager and is most appropriate for regular backup of object definitions, whereas CSQUTIL SDEFS can be used to re-create definitions for a queue manager that is not currently running. This makes the CSQUTIL SDEFS option more appropriate for recovery scenarios.

Command management (SDEFS)



- Keywords and parameters
- Examples
- Usage notes

Keywords and parameters

OBJECT
Specifies the type of object to be listed.

A value of QUEUE lists queues of all types, as if you had specified QALIAS, QLOCAL, QMODEL and QREMOTE.

QSGDISP

Specifies from where the object definition information is obtained. Depending on how the object has been defined, this information is either:

- On the page set zero referred to by the CSQP0000 DD statement, or
- In a Db2 shared repository.

Permitted values are shown in Table 156.

Table 156. SDEFS QSGDISP parameters and their actions

QSGDISP parameter	What the SDEFS utility does
QMGR	Creates DEFINE statements for the specified object type from definitions held on the page set zero referred to by the CSQP0000 DD statement. (1) Only objects defined with QSGDISP(QMGR) are included.
COPY	Creates DEFINE statements for the specified object type from definitions held on the page set zero referred to by the CSQP0000 DD statement. (1) Only objects defined with QSGDISP(COPY) are included.
PRIVATE	Creates DEFINE statements for the specified object type from definitions held on the page set zero referred to by the CSQP0000 DD statement. (1) Both QSGDISP(QMGR) and QSGDISP(COPY) objects are included.
GROUP	Creates DEFINE statements for the specified object type from definitions held on Db2 resource definition tables for the specified queue-sharing group. Only objects defined with QSGDISP(GROUP) are included. No CSQP0000 DD statement is required; the Db2 subsystem specified at object definition is accessed. The Db2 library db2qual.SDSNLOAD is required.
SHARED	Creates DEFINE statements for all local queues defined with QSGDISP(SHARED) by accessing the Db2 resource definition table for the specified queue-sharing group. This parameter is permitted only with OBJECT(QLOCAL) or OBJECT(QUEUE). No CSQP0000 DD statement is required; the Db2 subsystem specified at object definition is accessed. The Db2 library db2qual.SDSNLOAD is required.

Notes:

1. Because only page set zero is accessed, you must ensure that the queue manager is not running.

MAKEDEF(*ddname2*)

Specifies that define commands generated for the object are to be placed in the output data set identified by the DDname. The data set should be RECFM=FB, LRECL=80. This data set can then be used as input for a later invocation of the COMMAND function or it can be incorporated into the initialization data sets CSQINP1 and CSQINP2.

The commands generated are DEFINE NOREPLACE, with all the attributes and values for the object.

Note: DEFINE commands are not generated for any local queues that can be identified as dynamic, or for channels that were defined automatically.

Examples

```
//SDEFS   EXEC  PGM=CSQUTIL
//STEPLIB  DD   DISP=SHR,DSN=thlqual.SCSQANLE
//         DD   DISP=SHR,DSN=thlqual.SCSQAUTH
//CSQP0000 DD   DISP=OLD,DSN=pageset.dsname0
//OUTPUT1  DD   DISP=OLD,DSN=MY.COMMANDS(DEFS)
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
SDEFS OBJECT(QUEUE) MAKEDEF(OUTPUT1)
/*
```

Figure 24. Sample JCL for the SDEFS function of CSQUTIL

```
//SDEFS   EXEC  PGM=CSQUTIL,PARM='Qsgname,Dsgname,Db2name'
//STEPLIB  DD   DISP=SHR,DSN=thlqual.SCSQANLE
//         DD   DISP=SHR,DSN=thlqual.SCSQAUTH
//         DD   DISP=SHR,DSN=db2qual.SDSNLOAD
//OUTPUT1  DD   DISP=OLD,DSN=MY.COMMANDS(DEFS)
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
SDEFS OBJECT(QLOCAL) QSGDISP(SHARED) MAKEDEF(OUTPUT1)
/*
```

Figure 25. Sample JCL for the SDEFS function of CSQUTIL for objects in the Db2 shared repository

```
//CSQUTIL JOB CLASS=A,MSGCLASS=H,NOTIFY=&SYSUID,REGION=0M
//PS00 EXEC PGM=CSQUTIL
//STEPLIB  DD DISP=SHR,DSN=thlqual.SCSQAUTH
//         DD DISP=SHR,DSN=thlqual.SCSQANLE
//CSQP0000 DD DISP=OLD,DSN=pageset.dsname0
//OUTPUT1  DD DISP=OLD,DSN=MY.COMMANDS(CHANNEL)
//OUTPUT2  DD DISP=OLD,DSN=MY.COMMANDS(AUTHINFO)
//OUTPUT3  DD DISP=OLD,DSN=MY.COMMANDS(NAMELIST)
//OUTPUT4  DD DISP=OLD,DSN=MY.COMMANDS(PROCESS)
//OUTPUT5  DD DISP=OLD,DSN=MY.COMMANDS(QALIAS)
//OUTPUT6  DD DISP=OLD,DSN=MY.COMMANDS(QLOCAL)
//OUTPUT7  DD DISP=OLD,DSN=MY.COMMANDS(QMODEL)
//OUTPUT8  DD DISP=OLD,DSN=MY.COMMANDS(QREMOTE)
//OUTPUT9  DD DISP=OLD,DSN=MY.COMMANDS(QUEUE)
//OUTPUT0  DD DISP=OLD,DSN=MY.COMMANDS(STGCLASS)
//OUTPUTA  DD DISP=OLD,DSN=MY.COMMANDS(TOPIC)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
SDEFS OBJECT(CHANNEL) MAKEDEF(OUTPUT1)
SDEFS OBJECT(AUTHINFO) MAKEDEF(OUTPUT2)
SDEFS OBJECT(NAMELIST) MAKEDEF(OUTPUT3)
SDEFS OBJECT(PROCESS) MAKEDEF(OUTPUT4)
SDEFS OBJECT(QALIAS) MAKEDEF(OUTPUT5)
SDEFS OBJECT(QLOCAL) MAKEDEF(OUTPUT6)
SDEFS OBJECT(QMODEL) MAKEDEF(OUTPUT7)
SDEFS OBJECT(QREMOTE) MAKEDEF(OUTPUT8)
SDEFS OBJECT(QUEUE) MAKEDEF(OUTPUT9)
SDEFS OBJECT(STGCLASS) MAKEDEF(OUTPUT0)
SDEFS OBJECT(TOPIC) MAKEDEF(OUTPUTA)
/*
```

Figure 26. Sample JCL for the SDEFS function of CSQUTIL, when recovering all objects from a valid page set zero

Usage notes

1. For local definitions, do not use SDEFS for a queue manager that is running because results will be unpredictable. You can avoid doing this accidentally by using DISP=OLD in the CSQP0000 DD statement. For shared or group queue definitions, this does not matter because the information is derived from Db2.
2. When you use SDEFS for local queues you do not need to specify a queue manager name. However, for shared and group queue definitions, a queue manager name is required to access Db2.
3. To use the SDEFS function more than once in a job, specify different DDnames and data sets for each invocation of the function, or specify a sequential data set and DISP=MOD in the DD statements.
4. If the SDEFS function fails, no further CSQUTIL functions are attempted.

Related information:

IBM MQ utility program (CSQUTIL)
The CSQUTIL utility program is provided with IBM MQ to help you to perform backup, restoration, and reorganization tasks, and to issue IBM MQ commands.

Copying queues into a data set while the queue manager is running (COPY) on z/OS: z/OS

You can use the COPY function of CSQUTIL to copy queued messages to a sequential data set while the queue manager is running, without destroying any messages in the original queues.

The scope of the COPY function is determined by the keyword that you specify in the first parameter. You can either copy all the messages from a named queue, or all the messages from all the queues on a named page set.

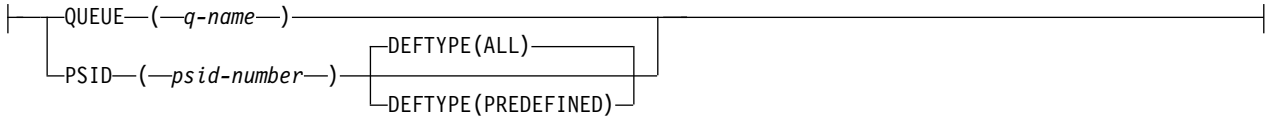
Use the complementary function, LOAD, to restore the messages to their appropriate queues.

Note:

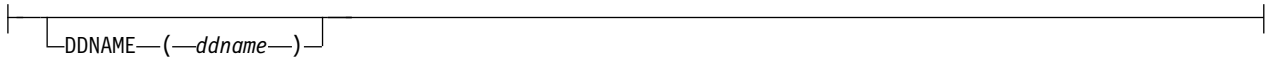
1. If you want to copy the object definitions from the named page set, use COPYPAGE.
2. If you want to copy messages to a data set when the queue manager is stopped, use SCOPY.
3. See Syncpoints for information about how to avoid problems with duplicate messages if this function fails.

Queue management (COPY)

Object Selection



DDname Selection



- Keywords and parameters
- Example
- Usage notes

- Syncpoints

Keywords and parameters

QUEUE(*q-name***)**
Specifies that messages in the named queue are to be copied. The keyword QUEUE can be abbreviated to Q.

q-name specifies the name of the queue to be copied. This name is case-sensitive.

PSID(*psid-number***)**
Specifies that all the messages in all the queues in the specified page set are to be copied.

psid-number is the page set identifier, which specifies the page set to be used. This identifier is a two-digit integer (whole number) representing a single page set.

DEFTYPE
Specifies whether to copy dynamic queues:

ALL Copy all queues; this is the default.

PREDEFINED
Do not include dynamic queues; this is the same set of queues that are selected by the COMMAND and SDEFS functions with the MAKEDEF parameter.

DDNAME(*ddname***)**
Specifies that the messages are to be copied to a named data set. If this keyword is omitted, the default DDname, CSQUOUT, is used. The keyword DDname can be abbreviated to DD.

ddname specifies the DDname of the destination data set, which is used to store the messages. The record format of this data set must be variable block spanned (VBS).

Example

```
//COPY   EXEC PGM=CSQUTIL,PARM='CSQ1',REGION=0M
//STEPLIB  DD   DISP=SHR,DSN=thlqual.SCSQANLE
//         DD   DISP=SHR,DSN=thlqual.SCSQAUTH
//OUTPUTA  DD DSN=SAMPLE.UTILITY.COPYA,DISP=(NEW,CATLG),
//            SPACE=(CYL,(5,1),RLSE),UNIT=SYSDA,
//            DCB=(RECFM=VBS,BLKSIZE=23200)
//CSQUOUT  DD DSN=SAMPLE.UTILITY.COPY3,DISP=(NEW,CATLG),
//            SPACE=(CYL,(5,1),RLSE),UNIT=SYSDA,
//            DCB=(RECFM=VBS,BLKSIZE=23200)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
* COPY WHOLE PAGE SET TO 'CSQUOUT'
COPY PSID(03)
* COPY ONE QUEUE TO 'OUTPUT'
COPY QUEUE(ABC123A) DDNAME(OUTPUTA)
/*
```

Figure 27. Sample JCL for the CSQUTIL COPY functions. The sample shows two instances of the COPY function-one COPY to the default DDNAME, CSQUOUT; the other to DDNAME OUTPUTA, which overrides CSQUOUT.

Usage notes

1. The queues involved must not be in use when the function is started.
2. If you want to operate on a range of page sets, repeat the COPY function for each page set.
3. The function operates only on local queues.
4. A COPY PSID function is considered successful only if it successfully copies all the queues on the page set.
5. If you try to copy an empty queue (either explicitly by COPY QUEUE or because there are one or more empty queues on a page set that you are copying), data indicating this is written to the

sequential data set, and the copy is considered to be a success. However, if you attempt to copy a nonexistent queue, or a page set containing no queues, the COPY function fails, and no data is written to the data set.

6. If COPY fails, no further CSQUTIL functions are attempted.
7. To use the COPY function more than once in the job, specify different DDnames and data sets for each invocation of the function, or specify a sequential data set and DISP=MOD in the DD statements.
8. You need the necessary authority to use the command server queues (SYSTEM.COMMAND.INPUT, SYSTEM.COMMAND.REPLY.MODEL, and SYSTEM.CSQUTIL.*), to use the DISPLAY QUEUE and DISPLAY STGCLASS MQSC commands, and to open the queues that you want to copy with the MQOO_INPUT_EXCLUSIVE and MQOO_BROWSE options.
9. For the **REGION** parameter, a value of 0M means that the job is allowed to have the amount of storage it needs. However, if a job tries to acquire too much storage, it might impact other jobs in the system. You must ideally look to limit the REGION size and specify an absolute maximum value that the job is allowed to acquire.

Copying queues into a data set while the queue manager is not running (SCOPY) on z/OS: z/OS

You can use the SCOPY function of CSQUTIL to copy queued messages to a sequential data set when the queue manager is not running, without destroying any messages in the original queues.

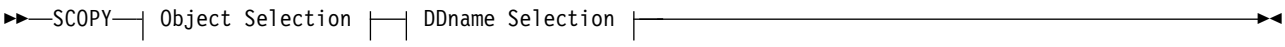
The scope of the SCOPY function is determined by the keyword that you specify in the first parameter. You can either copy all the messages from a named queue, or all the messages from all the queues on a named page set.

Use the complementary function, LOAD, to restore the messages to their queues.

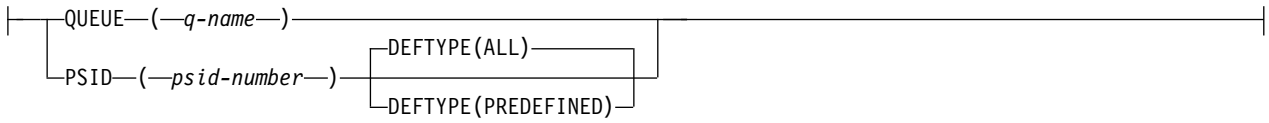
To use the SCOPY function, DDname CSQP0000 must specify the data set with page set zero for the subsystem required.

Note: The SCOPY function does not operate on shared queues.

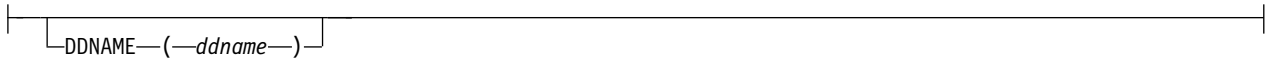
Queue Management (SCOPY)



Object Selection



DDname Selection



- Keywords and parameters
- Example
- Usage notes

Keywords and parameters

QUEUE(*q-name***)**
Specifies that messages in the named queue are to be copied. The keyword QUEUE can be abbreviated to Q.

q-name specifies the name of the queue to be copied. This name is case-sensitive.

DDname CSQP00 *nn* must specify the data set with page set *nn* for the subsystem required, where nn is the number of the page set where the queue resides.

PSID(*psid-number***)**
Specifies that all the messages in all the queues in the specified page set are to be copied.

psid-number is the page set identifier, which specifies the page set to be used. This identifier is a two-digit integer (whole number) representing a single page set.

DDname CSQP00 *psid-number* must specify the data set with the required page set for the subsystem required.

DEFTYPE
Specifies whether to copy dynamic queues:

ALL Copy all queues; this is the default.

PREDEFINED
Do not include dynamic queues; this is the same set of queues that are selected by the COMMAND and SDEFS functions with the MAKEDEF parameter.

This parameter is only valid if you specify PSID.

DDNAME(*ddname***)**
Specifies that the messages are to be copied to a named data set. If this keyword is omitted, the default DDname, CSQUOUT, is used. The keyword DDname can be abbreviated to DD.

ddname specifies the DDname of the destination data set, which is used to store the messages. The record format of this data set must be variable block spanned (VBS).

Do not specify the same DDname on more than one SCOPY statement, unless its DD statement specifies a sequential data set with DISP=MOD.

Example

```
//SCOPY   EXEC PGM=CSQUTIL,REGION=0M
//STEPLIB  DD   DISP=SHR,DSN=thlqual.SCSQANLE
//         DD   DISP=SHR,DSN=thlqual.SCSQAUTH
//OUTPUTA  DD DSN=SAMPLE.UTILITY.COPYA,DISP=(NEW,CATLG),
//            SPACE=(CYL,(5,1),RLSE),UNIT=SYSDA,
//            DCB=(RECFM=VBS,BLKSIZE=23200)
//CSQUOUT  DD DSN=SAMPLE.UTILITY.COPY3,DISP=(NEW,CATLG),
//            SPACE=(CYL,(5,1),RLSE),UNIT=SYSDA,
//            DCB=(RECFM=VBS,BLKSIZE=23200)
//CSQP0000 DD DISP=OLD,DSN=pageset.dsname0
//CSQP0003 DD DISP=OLD,DSN=pageset.dsname3
//CSQP0006 DD DISP=OLD,DSN=pageset.dsname6
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
* COPY WHOLE PAGE SET TO 'CSQUOUT'
SCOPY PSID(03)
* COPY ONE QUEUE TO 'OUTPUT' - QUEUE IS ON PAGE SET 6
SCOPY QUEUE(ABC123A) DDNAME(OUTPUTA)
/*
```

Figure 28. Sample JCL for the CSQUTIL SCOPY functions. The sample shows two instances of the SCOPY function; one SCOPY to the default DDNAME, CSQUOUT; the other to DDNAME OUTPUTA, which overrides CSQUOUT.

Usage notes

1. Do not use SCOPY for a queue manager that is running because results are unpredictable. You can avoid doing this accidentally by using DISP=OLD in the page set DD statement.
2. When you use SCOPY, you do not need to specify a queue manager name.
3. If you want to operate on a range of page sets, repeat the SCOPY function for each page set.
4. The function operates only on local queues and only for persistent messages.
5. A SCOPY PSID function is considered successful only if it successfully copies all the queues on the page set that have messages; empty queues are ignored. If the page set has no queues with messages, the SCOPY function fails, and no data is written to the data set.
6. If you try to copy an empty queue explicitly by SCOPY QUEUE, data indicating this is written to the sequential data set, and the copy is considered to be a success. However, if you attempt to copy a nonexistent queue, the SCOPY function fails, and no data is written to the data set.
7. If the SCOPY function fails, no further CSQUTIL functions are attempted.
8. To use the SCOPY function more than once in the job, specify different DDnames and data sets for each invocation of the function, or specify a sequential data set and DISP=MOD in the DD statements.
9. For the **REGION** parameter, a value of 0M means that the job is allowed to have the amount of storage it needs. However, if a job tries to acquire too much storage, it might impact other jobs in the system. You must ideally look to limit the REGION size and specify an absolute maximum value that the job is allowed to acquire.

Analyzing the queue data copied to a data set by COPY or SCOPY using ANALYZE on z/OS:

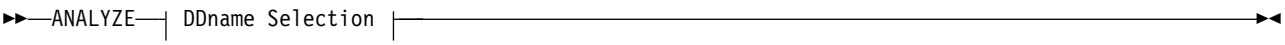
z/OS

Use this topic to understand analyzing the queue data copied to a data set by COPY or SCOPY.

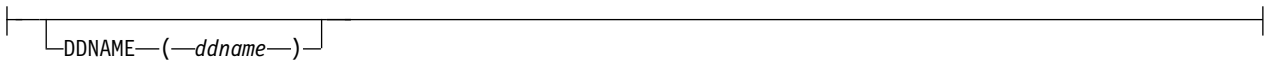
This function reads and analyzes a data set (created using COPY or SCOPY), and for each queue, displays:

- queue name
- number of messages for the queue
- total length of the messages

Data analysis (ANALYZE)



DDname Selection



- "Keywords and parameters"
- "Example"
- "Usage notes"

Keywords and parameters

DDNAME(*ddname***)**

Specifies the data set to be processed. This keyword can be abbreviated to DD.

ddname specifies the DDname that identifies the destination data set of a prior COPY or SCOPY operation. This name is not case sensitive, and can be up to eight characters long.

Example

```
//LOAD EXEC PGM=CSQUTIL
//STEPLIB DD DISP=SHR,DSN=thlqual.SCSQANLE
// DD DISP=SHR,DSN=thlqual.SCSQAUTH
//OUTPUTA DD DSN=MY.UTILITY.OUTPUTA,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
ANALYZE DDNAME(OUTPUTA)
```

Figure 29. Sample JCL for the CSQUTIL ANALYZE function

Usage notes

1. If you omit DDname(ddname) the default DDname, CSQUINP, is used.

Emptying a queue of all messages (EMPTY) on z/OS: z/OS

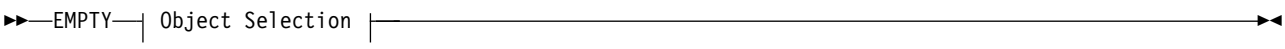
You can use the EMPTY function of CSQUTIL to delete all messages from a named queue or all the queues on a page set.

The queue manager must be running. The scope of the function is determined by the keyword that you specify in the first parameter.

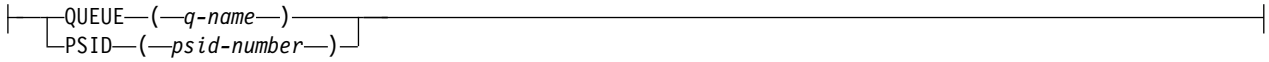
Use this function with care. Only delete messages of which copies have already been made.

Note: See "Syncpoints" on page 1902 for information about how to avoid problems with duplicate messages if this function fails.

Queue management (EMPTY)



Object Selection



- Keywords and parameters
- Example
- Usage notes

Keywords and parameters

You must specify the scope of the EMPTY function. Choose one of these:

QUEUE(*q-name*)
Specifies that messages are to be deleted from a named queue. This keyword can be abbreviated to Q.

q-name specifies the name of the queue from which messages are to be deleted. This name is case sensitive.

PSID(*psid-number*)
Specifies that all the messages are to be deleted from all queues in the named page set.

psid-number specifies the page-set identifier. This identifier is a two-digit integer (whole number) representing a single page set.

Example

```
//EMPTY  EXEC PGM=CSQUTIL,PARM=('CSQ1')
//STEPLIB  DD   DISP=SHR,DSN=thlqual.SCSQANLE
//         DD   DISP=SHR,DSN=thlqual.SCSQAUTH
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
EMPTY QUEUE(SPARE)
EMPTY PSID(66)
/*
```

Figure 30. Sample JCL for the CSQUTIL EMPTY function

Usage notes

1. The queues involved must not be in use when the function is invoked.
2. This function operates only on local queues.
3. If you want to operate on a range of page sets, repeat the EMPTY function for each page set.
4. You cannot empty the system-command input queue (SYSTEM.COMMAND.INPUT).
5. An EMPTY PSID function is considered successful only if it successfully empties all the queues on the page set.
6. If you empty a queue that is already empty (either explicitly by EMPTY QUEUE or because there are one or more empty queues on a page set that you are emptying), the EMPTY function is considered to be a success. However, if you attempt to empty a nonexistent queue, or a page set containing no queues, the EMPTY function fails.
7. If EMPTY fails or is forced to take a syncpoint, no further CSQUTIL functions are attempted.
8. You need the necessary authority to use the command server queues (SYSTEM.COMMAND.INPUT, SYSTEM.COMMAND.REPLY.MODEL, and SYSTEM.CSQUTIL.*), to use the DISPLAY QUEUE and DISPLAY STGCLASS MQSC commands, and to use the IBM MQ API to get messages from the queues that you want to empty.

Related concepts:

"Invoking the IBM MQ utility program on z/OS" on page 1901
Use this topic to understand how to invoke CSQUTIL, the format of its parameters, and its return codes.

Restoring messages from a data set to a queue (LOAD) on z/OS: z/OS

The LOAD function of CSQUTIL is complementary to the COPY or SCOPY function. LOAD restores messages from the destination data set of an earlier COPY or SCOPY operation. The queue manager must be running.

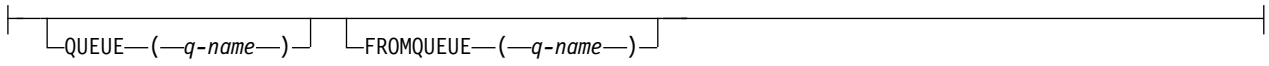
The data set can contain messages from one queue only if it was created by COPY or SCOPY QUEUE, or from a number of queues if it was created by COPY PSID or several successive COPY or SCOPY QUEUE operations. Messages are restored to queues with the same name as those from which they were copied. You can specify that the first or only queue is loaded to a queue with a different name. (This would normally be used with a data set created with a single COPY queue operation to restore the messages to a queue with a different name.)

Note: See "Syncpoints" on page 1902 for information about how to avoid problems with duplicate messages if this function fails.

Messages are restored to queues with the same name as those from which they were copied. You can specify that the first or only queue is loaded to a queue with a different name using the **QUEUE** parameter. (This would normally be used with a data set created with a single COPY queue operation to restore the messages to a queue with a different name.) For a data set containing multiple queues, the first queue to be processed can be specified using the **FROMQUEUE** parameter. Messages are restored to this queue and all subsequent queues in the data set.

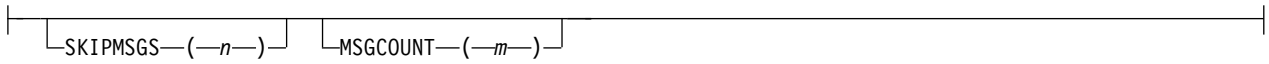
Queue management (LOAD)

Object Selection



DDname Selection

Record Selection



- Keywords and parameters
- Example
- Usage notes

Keywords and parameters

QUEUE(*q-name***)**
This parameter specifies that the messages from the first or only queue on the destination data set of a prior COPY or SCOPY operation are loaded to a named queue. Messages from any subsequent queues are loaded to queues with the same names as those they came from. The keyword QUEUE can be abbreviated to Q.

q-name specifies the name of the queue to which the messages are to be loaded. This name is case sensitive. It must not be a model queue.

FROMQUEUE(*q_name***)**
Specifies the name of the first queue to process on the destination data set of a prior COPY or SCOPY operation. Messages from this queue and any subsequent queues on the data set are loaded to queues with the same names as those that they came from. If this parameter is removed, the LOAD function starts with the first queue on the data set and processes all queues. The keyword FROMQUEUE can be abbreviated to FROMQ.

DDNAME(*ddname***)**
Specifies that messages are loaded from a named data set. This keyword can be abbreviated to DD.

ddname specifies the **DDNAME** that identifies the destination data set of a prior COPY or SCOPY operation, from which the messages are to be loaded. This name is not case sensitive, and can be up to 8 characters long.

If you omit **DDNAME** (*ddname*) the default **DDNAME**, CSQUINP, is used.

SKIPMSGS(*n*)
Specifies that the first *n* messages in the sequential data set are to be skipped before commencing the load of the queue.

If you omit SKIPMSGS(*n*) no messages are skipped; the load starts at the first message.

MSGCOUNT(*m*)

Specifies that only *m* messages are read from the data set and loaded to the queue.

If you omit MSGCOUNT(*m*) the number of messages read is unlimited.

Example

```
//LOAD EXEC PGM=CSQUTIL,PARM=('CSQ1'),REGION=0M
//STEPLIB DD DISP=SHR,DSN=thlqual.SCSQANLE
// DD DISP=SHR,DSN=thlqual.SCSQAUTH
//OUTPUTA DD DSN=MY.UTILITY.OUTPUTA,DISP=SHR
//CSQUINP DD DSN=MY.UTILITY.COPYA,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
LOAD QUEUE(ABC123) DDNAME(OUTPUTA)
LOAD QUEUE(TOQ) FROMQUEUE(QUEUEA) SKIPMSGS(55)
/*
```

Figure 31. Sample JCL for the CSQUTIL LOAD function

Note:

REGION - A value of `0M` means that the job is allowed to have the amount of storage it needs. However, if a job tries to acquire too much storage, it might impact other jobs in the system. You must ideally look to limit the REGION size and specify an absolute maximum value that the job is allowed to acquire.

LOAD QUEUE(ABC123) DDNAME(OUTPUTA) - Reloads all queues from the input data set MY.UTILITY.OUTPUTA. The names of the queues loaded are the same as the queue names from which the data was copied, apart from the first queue on the data set which is reloaded to queue ABC123.

LOAD QUEUE(TOQ) FROMQUEUE(QUEUEA) SKIPMSGS(55) - Reloads all queues from the input data set MY.UTILITY.COPYA, starting from queue QUEUEA. The names of the queues loaded are the same as the queue names from which the data was copied, apart from the first queue QUEUEA, which is reloaded to queue TOQ. In processing the messages in QUEUEA, the first 55 messages are ignored, and loading starts from the 56th message.

Usage notes

1. To use the LOAD function, the queues or page sets involved must not be in use when the function is invoked.
2. If the data set contains multiple queues, the LOAD function is considered successful only if it successfully loads all the queues on the data set. (or all those following the starting queue specified with FROMQUEUE, if this is set).
3. If LOAD fails, or is forced to take a syncpoint, no further CSQUTIL functions are attempted.
4. CSQUTIL uses MQPMO_SET_ALL_CONTEXT to ensure that the message descriptor fields remain the same as the original copy. It therefore needs an access of CONTROL in the CONTEXT profile of the queue. For full details, see Profiles for context security.

Restoring messages from a data set to a queue (SLOAD) on z/OS: z/OS

The SLOAD function of CSQUTIL is complementary to the COPY or SCOPY function. SLOAD restores messages from the destination data set of an earlier COPY or SCOPY operation. SLOAD processes a single queue.

To use SLOAD the queue manager must be running.

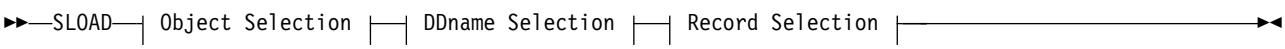
If the data set was created by COPY or SCOPY QUEUE it contains messages from one queue only. If the data set was created by COPY PSID or several successive COPY or SCOPY QUEUE operations, it might contain messages from a number of queues.

By default, SLOAD processes the first queue on the data set. You can specify a particular queue to process using the **FROMQUEUE** parameter.

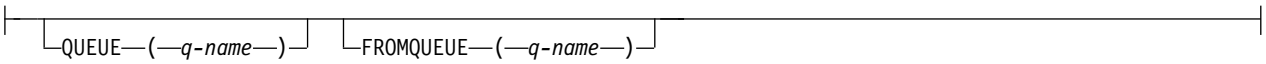
By default, messages are restored to a queue with the same name as the one from which it was copied. You can specify that the queue is loaded to a queue with a different name using the **QUEUE** parameter.

Note: See "Syncpoints" on page 1902 for information about how to avoid problems with duplicate messages if this function fails.

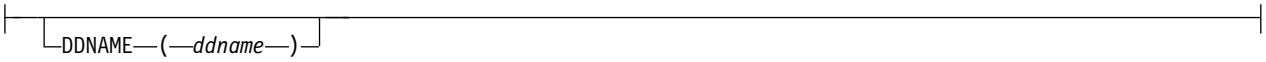
Queue management (SLOAD)



Object Selection



DDname Selection



Record Selection

- "Keywords and parameters"
- "Example" on page 1932
- "Usage notes" on page 1932

Keywords and parameters

QUEUE(*q-name*)
This parameter specifies that the messages from the first or only queue on the destination data set of a prior COPY or SCOPY operation are to be loaded to a named queue. The keyword QUEUE can be abbreviated to Q.

q-name specifies the name of the queue to which the messages are to be loaded. This name is case sensitive. It must not be a model queue.

FROMQUEUE(*q-name***)**
Specifies the name of the queue to process. If this parameter is omitted, the first queue is processed.

The keyword FROMQUEUE can be abbreviated to FROMQ.
`q-name` specifies the name of the queue to be processed. This name is case sensitive.

DDNAME(*ddname***)**
Specifies that messages are to be loaded from a named data set. This keyword can be abbreviated to DD.

ddname specifies the **DDNAME** that identifies the destination data set of a prior COPY or SCOPY operation, from which the messages are to be loaded. This name is not case sensitive, and can be up to 8 characters long.

If you omit **DDNAME** (*ddname*) the default **DDNAME**, CSQUINP, is used.

SKIPMSGS(*n*)
Specifies that the first *n* messages in the sequential data set are to be skipped before commencing the load of the queue.

If you omit SKIPMSGS(*n*) no messages are skipped; the load starts at the first message.

MSGCOUNT(*m*)
Specifies that only *m* messages are to be read from the data set and loaded to the queue.

If you omit MSGCOUNT(*m*) the number of messages read is unlimited.

Example

```
//SLOAD EXEC PGM=CSQUTIL,PARM=('CSQ1'),REGION=0M
//STEPLIB DD DISP=SHR,DSN=thlqual.SCSQANLE
// DD DISP=SHR,DSN=thlqual.SCSQAUTH
//OUTPUTA DD DSN=MY.UTILITY.OUTPUTA,DISP=SHR
//CSQUINP DD DSN=MY.UTILITY.COPYA,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
SLOAD DDNAME(OUTPUTA)
SLOAD QUEUE(TOQ) FROMQUEUE(QUEUEA) SKIPMSGS(55)
/*
```

Figure 32. Sample JCL for the CSQUTIL SLOAD function

Note:

- REGION - A value of `0M` means that the job is allowed to have the amount of storage it needs. However, if a job tries to acquire too much storage, it might impact other jobs in the system. You must ideally look to limit the REGION size and specify an absolute maximum value that the job is allowed to acquire.
- SLOAD DDNAME(OUTPUTA) - Reloads the first queue from the input data set MY.UTILITY.OUTPUTA. The name of the queue loaded is the same as the queue name from which the data was copied.
- SLOAD QUEUE(TOQ) FROMQUEUE(QUEUEA) SKIPMSGS(55) - Reloads the messages that were copied from the queue QUEUEA (from the input data set MY.UTILITY.COPYA). The messages are reloaded to the queue called TOQ. In processing the messages in QUEUEA, the first 55 messages are ignored, and loading starts from the 56th message.

Usage notes

1. To use the SLOAD function, the queues or page sets involved must not be in use when the function is invoked.
2. If SLOAD fails, or is forced to take a syncpoint, no further CSQUTIL functions are attempted.

3. CSQUTIL uses MQPMO_SET_ALL_CONTEXT to ensure that the message descriptor fields remain the same as the original copy. It therefore needs an access of CONTROL in the CONTEXT profile of the queue. For full details, see Profiles for context security.

Migrating a channel initiator parameter module (XPARM) on z/OS: z/OS

You can use the XPARM function of CSQUTIL to generate ALTER QMGR command that can be used to migrate to Version 7.0.

In versions of IBM MQ for z/OS before Version 6.0, you could tailor the channel initiator by creating a channel initiator parameter load module. In Version 7.0, you do it by setting queue manager attributes. To make it easier to migrate to Version 7.0, this command generates an ALTER QMGR command from a pre-Version 6.0 channel initiator parameter module.

Migration (XPARM)

```
►►—XPARM—DDNAME(ddname)—MEMBER(membername)—MAKEALT(ddname2)————————————————————►◄
```

Keywords and parameters

DDNAME(*ddname***)**
Specifies that an ALTER QMGR command is to be generated from a channel initiator parameter module in this data set.

MEMBER(*membername***)**
Specifies the name of the channel initiator parameter module in the data set specified by DDNAME(ddname2).

MAKEALT(*ddname2***)**
Specifies the DDname that identifies the output data set in which the ALTER command is to be stored. The data set should be RECFM=FB, LRECL=80. This data set can then be used as input for a later invocation of the COMMAND function or it can be incorporated into the CSQINP2 initialization input data sets.

Example

```
//MIGRATE1 EXEC PGM=CSQUTIL
//STEPLIB  DD DISP=SHR,DSN=thlqual.SCSQANLE
//         DD DISP=SHR,DSN=thlqual.SCSQAUTH
//CSQXPARM DD DISP=SHR,DSN=user.loadlib
//SYSPRINT DD SYSOUT=*
//ALTQMGR  DD DISP=OLD,DSN=user.commands(ALTQMGR)
//SYSIN DD *
XPARM DDNAME(CSQXPARM) MEMBER(MQ3AXPRM) MAKEALT(ALTQMGR)
/*
```

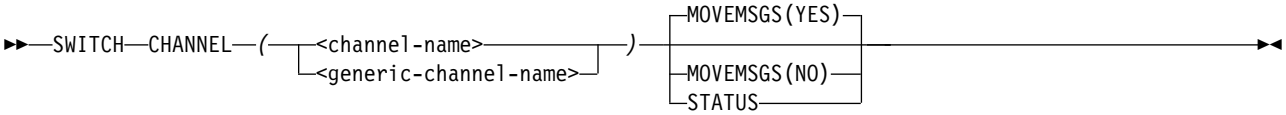
Figure 33. Sample JCL for the CSQUTIL XPARM function

Switch the transmission queue associated with cluster-sender channels (SWITCH): z/OS

You can use the SWITCH function of CSQUTIL to switch or query the transmission queue associated with cluster-sender channels.

To use the SWITCH function the queue manager must be running.

Switch transmission queue (SWITCH)



- Keywords and parameters
- Example
- Usage notes

Keywords and parameters

CHANNEL ***(channel name)***
Specifies the name of a cluster-sender channel, or a generic channel name.

If a generic channel name is specified each cluster-sender channel that matches the generic name is processed.

If a single asterisk is specified all cluster-sender channels are processed.

MOVEMSGS
Specifies whether messages queued for the channel should be moved from the old transmission queue to the new transmission queue during the switching process. Values are:

YES

Messages are moved from the old transmission queue to the new transmission queue. This is the default.

NO

Messages are not moved from the old transmission queue to the new transmission queue. If this option is selected it is the responsibility of the system programmer to resolve any messages for the channel on the old transmission queue after the switch has completed.

STATUS
Display the switching status for matching cluster-sender channels. If this keyword is not specified the command switches the transmission queue for stopped or inactive cluster-sender channels that require switching.

Examples

Figure 1 illustrates how the SWITCH function can be used to query the switching status of all cluster-sender channels whose names match the generic name CLUSTER.*.

```
//SWITCH  EXEC PGM=CSQUTIL,PARM=('CSQ1')
//STEPLIB  DD DISP=SHR,DSN=thlqual.SCSQANLE
//         DD DISP=SHR,DSN=thlqual.SCSQAUTH
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
SWITCH CHANNEL(CLUSTER.*) STATUS
/*
```

Figure 34. Sample JCL for querying the switching status of cluster-sender channels using the CSQUTIL SWITCH function

Figure 2 illustrates how the SWITCH function can be used to switch the transmission queue for the cluster-sender channel CLUSTER.TO.QM1.

```
//SWITCH  EXEC PGM=CSQUTIL,PARM=('CSQ1')
//STEPLIB  DD DISP=SHR,DSN=thlqual.SCSQANLE
//         DD DISP=SHR,DSN=thlqual.SCSQAUTH
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
SWITCH CHANNEL(CLUSTER.TO.QM1)
/*
```

Figure 35. Sample JCL for switching the transmission queue associated with a cluster-sender channel using the CSQUTIL SWITCH function

Usage notes

1. The channel initiator must be running to initiate a switch of transmission queue for cluster-sender channels.
2. The transmission queue associated with a cluster-sender channel can only be switched if the channel is STOPPED or INACTIVE.
3. You need the necessary authority to use the command server queues (SYSTEM.COMMAND.INPUT, SYSTEM.COMMAND.REPLY.MODEL, and SYSTEM.CSQUTIL.*)
4. You need the necessary authority to issue the START CHANNEL command.
5. To initiate a switch of transmission queue for a cluster-sender channel, you also need command resource authority for the channel.
6. You cannot use the SWITCH function to query the status of, or switch the transmission queue for, a cluster-sender channel that has not been started since version 8 new function has been enabled using the OPMODE parameter in the CSQ6SYSP macro.

Related information:
Clustering: Switching cluster transmission queues

The change log inventory utility (CSQJU003) on z/OS

z/OS

The IBM MQ change log inventory utility runs as a z/OS batch job to change the bootstrap data set (BSDS).

Through this utility, you can invoke these functions:

NEWLOG
Add active or archive log data sets.

DELETE
Delete active or archive log data sets.

ARCHIVE
Supply passwords for archive logs.

CRESTART
Control the next restart of IBM MQ.

CHECKPT
Set checkpoint records.

HIGHRBA
Update the highest written log RBA.

Only run this utility when IBM MQ is stopped. This is because the active log data sets named in the BSDS are dynamically added for exclusive use to IBM MQ and remain allocated exclusively to IBM MQ until it terminates. You can add new active log data sets to an active queue manager with the "DEFINE LOG on z/OS" on page 645 command.

The DEFINE LOG command can be used to update a BSDS of any version. However, you must use the CSQJUCNV utility to convert the BSDS from version 1 to version 2. A version 1 BSDS has space for up to 31 active log data sets in each log copy ring, whereas a version 2, or higher, BSDS has space for up to 310 active log data sets in each log copy ring.

Invoking the CSQJU003 utility on z/OS: z/OS

Use this topic to understand how to invoke the CSQJU003 utility.

The utility runs as a z/OS batch program. Figure 36 gives an example of the JCL required.

```
//JU003   EXEC  PGM=CSQJU003
//STEPLIB  DD   DISP=SHR,DSN=thlqual.SCSQANLE
//         DD   DISP=SHR,DSN=thlqual.SCSQAUTH
//SYSPRINT DD  SYSOUT=*,DCB=BLKSIZE=629
//SYSUT1   DD  DISP=SHR,DSN=bsds.dsname
//SYSIN    DD  *
  NEWLOG DSNAME=CSQREPAL.A0001187,COPY1VOL=CSQV04,UNIT=SYSDA,
  STARTRBA=3A190000,ENDRBA=3A1F0FFF,CATALOG=YES,PASSWORD=PASSWRD
/*
```

Figure 36. Sample JCL to invoke the CSQJU003 utility

Data definition (DD) statements

CSQJU003 requires DD statements with these DDnames:

SYSUT1
This statement is required; it names the BSDS.

SYSUT2
This statement is required if you use dual BSDSs; it names the second copy of the BSDS.

Dual BSDSs and CSQJU003
Each time you run the CSQJU003 utility, the BSDS time stamp field is updated with the current system time. If you run CSQJU003 separately for each copy of a dual copy BSDS, the time stamp fields are not synchronized, so the queue manager fails at startup, issuing error message CSQJ120E. Therefore, if CSQJU003 is used to update dual copy BSDSs, both BSDSs must be updated within a single run of CSQJU003.

SYSPRINT
This statement is required; it names a data set for print output. The logical record length (LRECL) is 125. The block size (BLKSIZE) must be 629.

SYSIN
This statement is required; it names the input data set for statements that specify what the utility is to do. The logical record length (LRECL) is 80.

You can use more than one statement of each type. In each statement, separate the operation name (NEWLOG, DELETE, ARCHIVE, CRESTART) from the first parameter by one or more blanks. You can use parameters in any order; separate them by commas with no blanks. Do not split a parameter description across two SYSIN records.

A statement containing an asterisk (*) in column 1 is considered to be a comment, and is ignored. However, it appears in the output listing. To include a comment or sequence number in a SYSIN record, separate it from the last comma by a blank. When a blank follows a comma, the rest of the record is ignored.

Multiple statement operation

When running CSQJU003, a significant error in any statement causes the control statements for the statement in error and all following statements to be skipped. Therefore, BSDS updates cannot occur for any operation specified in the statement in error, or any following statements. However, all the remaining statements are checked for syntax errors.

Adding information about a data set to the BSDS (NEWLOG) on z/OS: z/OS

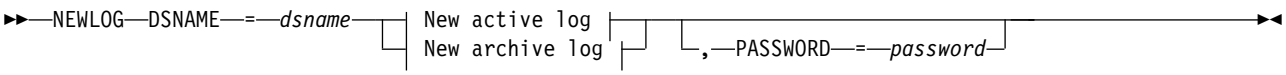
You can use the NEWLOG function of CSQJU003 to add information about a data set to BSDS.

The NEWLOG function declares one of the following data sets:

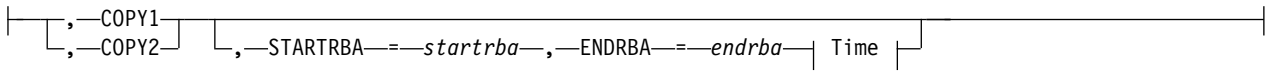
- A VSAM data set that is available for use as an active log data set.
  Use the keywords DSNAME, COPY1, COPY2, and PASSWORD.
- An active log data set that is replacing one that encountered an I/O error.
  Use the keywords DSNAME, COPY1, COPY2, STARTRBA, ENDRBA, and PASSWORD.
- An archive log data set volume.
  Use the keywords DSNAME, COPY1VOL, COPY2VOL, STARTRBA, ENDRBA, STRTLRSN, ENDLRSN, UNIT, CATALOG, and PASSWORD.
  In a queue-sharing group environment, you should always supply LRSN information. Run the print log map utility (CSQJU004) to find RBAs and LRSNs to use for archive log data sets.

A maximum of 310 data sets can be defined for each log copy, either by this NEWLOG function or the MQSC DEFINE LOG command.

NEWLOG

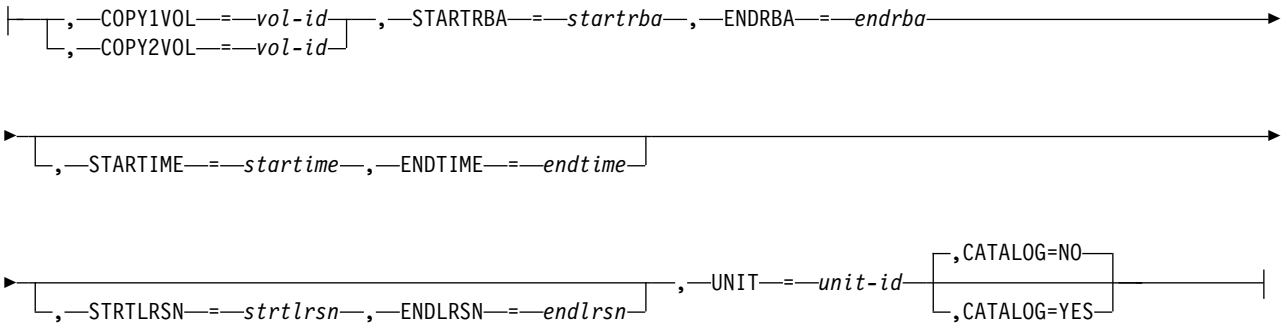


New active log:



Time:

New archive log:



Keywords and parameters

DSNAME= *dsname*
Names a log data set.

dsname can be up to 44 characters long.

PASSWORD= *password*
Assigns a password to the data set. It is stored in the BSDS and later used in any access to the active or archive log data sets.

The password is a data set password, and should follow standard VSAM convention: 1 through 8 alphanumeric characters (A through Z, 0 through 9) or special characters (& * + - . ; ' /).

We recommend that you use an ESM such as RACF to provide your data set security requirements.

COPY1
Makes the data set an active log copy-1 data set.

COPY2
Makes the data set an active log copy-2 data set.

STARTRBA= *startrba*
Gives the log RBA (relative byte address within the log) of the beginning of the replacement active log data set or the archive log data set volume specified by DSNAME.

startrba is a hexadecimal number of up to 16 characters. The value must end with 000. If you use fewer than 16 characters, leading zeros are added. The RBA can be obtained from messages or by printing the log map.

The value of STARTRBA must be a multiple of 4096. (The hexadecimal value must end in 000.)

A value higher than FFFFFFFFF000 cannot be specified for a version 1 format BSDS.

ENDRBA= *endrba*
Gives the log RBA (relative byte address within the log) of the end of the replacement active log data set or the archive log data set volume specified by DSNAME.

endrba is a hexadecimal number of up to 16 characters. The value must end with FFF. If you use fewer than 16 characters, leading zeros are added.

A value higher than FFFFFFFFFFFF cannot be specified for a version 1 format BSDS.

STARTIME= *startime*
Start time of the RBA in the BSDS. This is an optional field. The time stamp format (with valid values in parentheses) is yyyydddhhmmsst, where:

yyyy Indicates the year (1993 through 2099)

ddd Indicates the day of the year (1 through 365; 366 in leap years)

hh Indicates the hour (zero through 23)

mm Indicates the minutes (zero through 59)

ss Indicates the seconds (zero through 59)

t Indicates tenths of a second

If fewer than 14 digits are specified for the STARTIME and ENDTIME parameter, trailing zeros are added.

STARTRBA is required when STARTIME is specified.

ENDTIME= *endtime*
End time of the RBA in the BSDS. This is an optional field. For time stamp format, see the STARTIME option. The ENDTIME value must be greater than or equal to the value of STARTIME.

STRTLRSN= *strtlrsn*
Gives the LRSN (logical record sequence number) of the first complete log record on the new archive data set.

strtlrsn is a hexadecimal number of up to 12 characters. If you use fewer than 12 characters, leading zeros are added.

ENDLRSN= *endlrsn*
Gives the LRSN (logical record sequence number) of the last log record on the new archive data set.

endlrsn is a hexadecimal number of up to 12 characters. If you use fewer than 12 characters, leading zeros are added.

COPY1VOL= *vol-id*
The volume serial of the copy-1 archive log data set named after DSNAME.

COPY2VOL= *vol-id*
The volume serial of the copy-2 archive log data set named after DSNAME.

UNIT= *unit-id*
The device type of the archive log data set named after DSNAME.

CATALOG
Specifies whether the archive log data set is cataloged:

NO The archive log data set is not cataloged. All subsequent allocations of the data set are made using the unit and volume information specified on the function. This is the default.

YES The archive log data set is cataloged. A flag is set in the BSDS indicating this, and all subsequent allocations of the data set are made using the catalog.

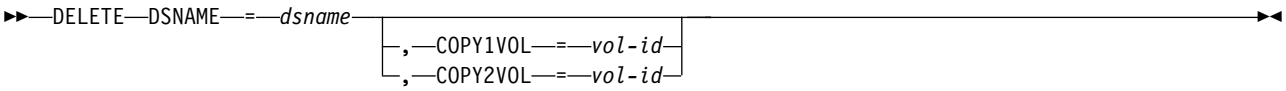
IBM MQ requires that all archive log data sets on DASD be cataloged. Select CATALOG=YES if the archive log data set is on DASD.

Deleting information about a data set from the BSDS (DELETE) on z/OS: z/OS

You can use the DELETE function of CSQJU003 to delete all information about a specified log data set or data set volume from the bootstrap data sets.

For example, you can use this function to delete outdated archive log data sets.

DELETE



Keywords and parameters

DSNAME= *dsname*
Specifies the name of the log data set.

dsname can be up to 44 characters long.

COPY1VOL= *vol-id*
The volume serial number of the copy-1 archive log data set named after DSNAME.

COPY2VOL= *vol-id*
The volume serial number of the copy-2 archive log data set named after DSNAME.

Supplying a password for archive log data sets (ARCHIVE) on z/OS: z/OS

You can use the ARCHIVE function of CSQJU003 to assign a password to all archive data sets created after this operation.

This password is added to the z/OS password data set each time a new archive log data set is created.

Use the NOPASSWD keyword to remove the password protection for all archives created after the archive operation.

Note: Typically, use an external security manager (ESM), such as RACF, if you want to implement security on any IBM MQ data sets.

ARCHIVE

Keywords and parameters

PASSWORD= *password*
Specifies that a password is to be assigned to the archive log data sets.

password specifies the password, which is a data set password and it must follow the standard VSAM convention; that is, 1 through 8 alphanumeric characters (A through Z, 0 through 9) or special characters (& * + - . ; ' /).

NOPASSWD
Specifies that archive password protection is not to be active for all archives created after this operation. No other keyword can be used with NOPASSWD.

Controlling the next restart (CRESTART) on z/OS: z/OS

You can use the CRESTART function of CSQJU003 to control the next restart of the queue manager, either by creating a new conditional restart control record or by canceling the one currently active.

These records limit the scope of the log data used during restart (truncating the log, in effect) . Any existing conditional restart control record governs every restart until one of these events occurs:

- A restart operation completes
- A CRESTART CANCEL is issued
- A new conditional restart control record is created

Attention: This can override IBM MQ efforts to maintain data in a consistent state. Only use this function when implementing the disaster recovery process described in Recovering a single queue manager at an alternative site and Recovering a queue-sharing group at the alternative site, or under the guidance of IBM service.

CRESTART

Keywords and parameters

CREATE
Creates a new conditional restart control record. When the new record is created, the previous control record becomes inactive.

CANCEL
Makes the currently active conditional restart control record inactive. The record remains in the BSDS as historical information.

No other keyword can be used with CANCEL.

ENDRBA= *endrba*
Gives the last RBA of the log to be used during restart (the point at which the log is to be truncated), and the starting RBA of the next active log to be written after restart. Any log information in the bootstrap data set and the active logs, with an RBA greater than *endrba*, is discarded.

endrba is a hexadecimal number of up to 16 digits. If you use fewer than 16 digits, leading zeros are added.

The value of ENDRBA must be a multiple of 4096. (The hexadecimal value must end in 000.)

A value higher than FFFFFFFFF000 cannot be specified for a version 1 format BSDS.

ENDLRSN= *endlrsn*
Gives the LRSN of the last log record to be used during restart (the point at which the log is to be truncated). Any log information in the bootstrap data set and the active logs with an LRSN greater than endlrsn is discarded.

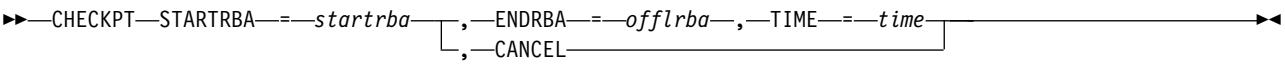
Setting checkpoint records (CHECKPT) on z/OS: z/OS

You can use the CHECKPT function of CSQJU003 to add or delete a record in the BSDS checkpoint queue.

Use the STARTRBA and ENDRBA keywords to add a record, or the STARTRBA and CANCEL keywords to delete a record.

Attention: This can override IBM MQ efforts to maintain data in a consistent state. Only use this function when implementing the disaster recovery process described in Recovering a single queue manager at an alternative site and Recovering a queue-sharing group at the alternative site, or under the guidance of IBM service.

CHECKPT



Keywords and parameters

STARTRBA= *startrba*
Indicates the start checkpoint log record.

startrba is a hexadecimal number of up to 16 digits. If you use fewer than 16 digits, leading zeros are added. The RBA can be obtained from messages or by printing the log map.

A value higher than FFFFFFFFFFFF cannot be specified for a version 1 format BSDS.

ENDRBA= *endrba*
Indicates the end checkpoint log record corresponding to the start checkpoint record.

endrba is a hexadecimal number of up to 16 digits. If you use fewer than 16 digits, leading zeros are added. The RBA can be obtained from messages or by printing the log map.

A value higher than FFFFFFFFFFFF cannot be specified for a version 1 format BSDS.

TIME= *time*
Gives the time the start checkpoint record was written. The time stamp format (with valid values in parentheses) is yyyydddhhmmsst, where:

yyyy Indicates the year (1993 through 2099)

ddd Indicates the day of the year (1 through 365; 366 in leap years)

hh Indicates the hour (zero through 23)

mm Indicates the minutes (zero through 59)

ss Indicates the seconds (zero through 59)

t Indicates tenths of a second

If fewer than 14 digits are specified for the TIME parameter, trailing zeros are added.

CANCEL
Deletes the checkpoint queue record containing a starting RBA that matches the RBA specified by STARTRBA.

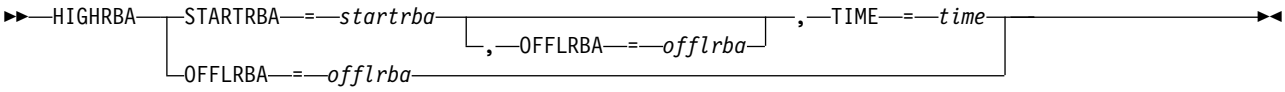
Updating the highest written log RBA (HIGHRBA) on z/OS: z/OS

You can use the HIGHRBA function of CSQJU003 to update the highest written log RBA recorded in the BSDS for either the active or archive log data sets.

Use the STARTRBA keyword to update the active log, and the OFFLRBA keyword to update the archive log.

Attention: This can override IBM MQ efforts to maintain data in a consistent state. Only use this function when implementing the disaster recovery process described in Recovering a single queue manager at an alternative site, or under the guidance of IBM service personnel.

HIGHRBA



Keywords and parameters

STARTRBA= *startrba*
Indicates the log RBA of the highest written log record in the active log data set.

startrba is a hexadecimal number of up to 16 digits. If you use fewer than 16 digits, leading zeros are added. The RBA can be obtained from messages or by printing the log map.

A value higher than FFFFFFFFFFFF cannot be specified for a version 1 format BSDS.

TIME= *time*
Specifies when the log record with the highest RBA was written to the log. The time stamp format (with valid values in parentheses) is yyyydddhhmmsst, where:

yyyy Indicates the year (1993 through 2099)

ddd Indicates the day of the year (1 through 365; 366 in leap years)

hh Indicates the hour (zero through 23)

mm Indicates the minutes (zero through 59)

ss Indicates the seconds (zero through 59)

t Indicates tenths of a second

If fewer than 14 digits are specified for the TIME parameter, trailing zeros are added.

OFFLRBA= *offlrba*
Specifies the highest offloaded RBA in the archive log.

offlrba is a hexadecimal number of up to 16 digits. If you use fewer than 16 digits, leading zeros are added. The value must end with hexadecimal 'FFF'.

A value higher than FFFFFFFFFFFF cannot be specified for a version 1 format BSDS.

The print log map utility (CSQJU004) on z/OS

z/OS

CSQJU004 is the batch utility program used to print log data information from the BSDS.

The IBM MQ print log map utility runs as a z/OS batch program to list the following information:

- The BSDS version
- Log data set name and log RBA association for both copies of all active and archive log data sets
- Active log data sets available for new log data
- Contents of the queue of checkpoint records in the bootstrap data set (BSDS)
- Contents of the quiesce history record
- System and utility time stamps
- Passwords for the active and archive log data sets, if provided

You can run the CSQJU004 program regardless of whether the queue manager is running. However, if the queue manager is running, consistent results from the utility can be ensured only if both the utility and the queue manager are running under control of the same z/OS system.

For further information, see

- Invoking the CSQJU004 utility
- Data definition statements required for the CSQJU004 utility

To use this utility, the user ID of the job must have the requisite security authorization, or, if the BSDS is password protected, the appropriate VSAM password for the data set.

Invoking the CSQJU004 utility

The following example shows the JCL used to invoke the CSQJU004 utility:

```
//JU004   EXEC  PGM=CSQJU004
//STEPLIB  DD   DISP=SHR,DSN=thlqual.SCSQANLE
//         DD   DISP=SHR,DSN=thlqual.SCSQAUTH
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  DISP=SHR,DSN=bsds.dsname
```

Figure 37. Sample JCL to invoke the CSQJU004 utility

The EXEC statement can use an optional parameter TIME(RAW) which changes the way timestamps are formatted.

```
//JU004 EXEC PGM=CSQJU004,PARM='TIME(RAW)'
```

This parameter causes timestamps to be formatted without applying timezone or leap second offsets for the formatting system. You can use this mode of operation when formatting a BSDS created at a remote site, or before a daylight saving time change, for example. The default, no parameter specified, is to format timestamps using the current formatting system's timezone and leap second corrections.

Formatted times affected by this parameter are:

- highest RBA written
- archive log command times
- checkpoint times
- conditional restart record times

Data definition statements

The CSQJU004 utility requires DD statements with the following DDnames:

SYSUT1
This statement is required to specify and allocate the bootstrap data set. If the BSDS must be shared with a concurrently running queue manager subsystem, use DISP=SHR on the DD statement.

SYSPRINT
This statement is required to specify a data set or print spool class for print output. The logical record length (LRECL) is 125 and the record format (RECFM) is VBA.

Finding out what the BSDS contains describes the output.

The log print utility (CSQ1LOGP) on z/OS

z/OS

Use this utility to print information contained in the IBM MQ log data sets or the BSDS.

- Invoking the CSQ1LOGP utility
- Input control parameters
- Usage notes
- The EXTRACT function
  - Example of processing EXTRACT data
- CSQ1LOGP output
  - Detail report
  - Record layouts for the output data sets

Invoking the CSQ1LOGP utility

You run the IBM MQ log print utility as a z/OS batch program. You can specify:

- A bootstrap data set (BSDS)
- Active log data sets (with no BSDS)
- Archive log data sets (with no BSDS)

Sample JCL to invoke the CSQ1LOGP utility is shown in Figure 38 on page 1946, Figure 39 on page 1947, Figure 40 on page 1947 and Figure 41 on page 1947.

These data definition statements must be provided:

SYSPRINT
All error messages, exception conditions, and the detail report are written to this data set. The logical record length (LRECL) is 131.

SYSIN
Input selection criteria can be specified in this data set. See “Input control parameters” on page 1948 for more information.

The logical record length (LRECL) must be 80, but only columns 1 through 72 are significant; columns 73 through 80 are ignored. At most 50 records can be used. Records with an asterisk (*) in column 1 are interpreted as comments and are ignored.

SYSSUMRY
If a summary report is requested, by specifying the parameter **SUMMARY** (YES) or **SUMMARY** (ONLY), the output is written to this data set. The logical record length (LRECL) is 131.

BSDS Name of the bootstrap data set (BSDS).

ACTIVEn
Name of an active log data set you want to print (n=number).

ARCHIVE
Name of an archive log data set you want to print.

If you specify the keyword **EXTRACT** (YES), provide one or more of the following DD statements, depending on what types of data you want to extract. Do not specify an LRECL, as it is set internally by the utility. These DDs are the required DCB parameters for the output data set.

CSQBACK
This data set contains persistent messages written to the log by units of work that were rolled back during the log range specified.

CSQCMT
This data set contains persistent messages written to the log by units of work that were committed during the log range specified.

CSQBOTH
This data set contains persistent messages written to the log by units of work that were either committed or rolled back during the log range specified.

CSQINFLT
This data set contains persistent messages written to the log by units of work that remained in flight during the log range specified.

CSQOBJS
This data set contains information about object alterations that occurred during the log range specified.

For each DD statement, the record format (RECFM) is VB, the logical record length (LRECL) is 32756, and the block size (BLKSIZE) must be 32760.

If you are processing active log data sets, the utility runs even if IBM MQ is running, if the BSDS and active log data sets are defined by using at least SHAREOPTIONS(2 3).

```
//PRTLOG EXEC PGM=CSQ1LOGP
//STEPLIB  DD DISP=SHR,DSN=thlqual.SCSQANLE
//         DD DISP=SHR,DSN=thlqual.SCSQLOAD
//BSDS     DD DSN=qmgr.bsds.dsname,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSSUMRY DD SYSOUT=*
//SYSIN    DD *
* extract records for pageset 3. Produce both summary and detail reports
PAGESET(3)
SUMMARY(YES)
/*
```

Figure 38. Sample JCL to invoke the CSQ1LOGP utility using a BSDS

```
//PRTLOG  EXEC PGM=CSQ1LOGP
//STEPLIB  DD DISP=SHR,DSN=thlqual.SCSQANLE
//         DD DISP=SHR,DSN=thlqual.SCSQLOAD
//ACTIVE1  DD DSN=qmgr.logcopy1.ds01,DISP=SHR
//ACTIVE2  DD DSN=qmgr.logcopy1.ds02,DISP=SHR
//ACTIVE3  DD DSN=qmgr.logcopy1.ds03,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSSUMRY DD SYSOUT=*
//SYSIN DD *
insert your input control statements here, for example:
Urid(urid1)
Urid(urid2)
/*
```

Figure 39. Sample JCL to invoke the CSQ1LOGP utility using active log data sets

```
//PRTLOG  EXEC PGM=CSQ1LOGP
//STEPLIB  DD DISP=SHR,DSN=thlqual.SCSQANLE
//         DD DISP=SHR,DSN=thlqual.SCSQLOAD
//ARCHIVE  DD DSN=qmgr.archive1.ds01,DISP=SHR
//         DD DSN=qmgr.archive1.ds02,DISP=SHR
//         DD DSN=qmgr.archive1.ds03,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSSUMRY DD SYSOUT=*
//SYSIN    DD *
  insert your input control statements here
/*
```

Figure 40. Sample JCL to invoke the CSQ1LOGP utility using archive log data sets

```
//PRTLOG  EXEC PGM=CSQ1LOGP
...
//CSQBACK   DD DSN=backout.dataset,DISP=(NEW,CATLG)
//CSQCMT    DD DSN=commit.dataset,DISP=(NEW,CATLG)
//CSQBOTH   DD DSN=both.dataset,DISP=(NEW,CATLG)
//CSQINFLT  DD DSN=inflight.dataset,DISP=(NEW,CATLG)
//CSQOBJS   DD DSN=objects.dataset,DISP=(NEW,CATLG)
```

Figure 41. Sample JCL showing additional statements for the EXTRACT keyword

The EXEC statement can use an optional parameter TIME(RAW) which changes the way timestamps are formatted.

```
//PRTLOG EXEC PGM=CSQ1LOGP,PARM='TIME(RAW)'
```

This causes timestamps to be formatted without applying timezone or leap second offsets for the formatting system. You can use this mode of operation when formatting log data created at a remote site, or before a daylight saving time change, for example.

If no parameter is specified, the default behavior is to format timestamps using the timezone and leap second corrections of the system doing the formatting.

Formatted times affected by this parameter are those associated with:

- checkpoint time
- restart time
- UR start time

Input control parameters

The keywords that you can use in the SYSIN data set are described in the following list.

You can specify various selection criteria to limit the log records that are processed. These are:

- log range, using RBASTART-RBAEND or LRSNSTART-LRSNEND
- page sets, using PAGESET
- units of recovery, using URID
- record contents, using DATA
- resource manager, using RM

Different types of selection criteria can be combined; only records meeting all the criteria are processed.

LRSNSTART ***(hexadecimal-constant)***
Specifies the logical record sequence number (LRSN) from which to begin processing. You cannot use this keyword together with RBASTART. Use this keyword only if your queue manager is in a queue-sharing group.

LRSN values are always greater than A00000000000; this value is used as the start value if a lower value is specified.

You can also use the forms STARTLRSN or STRTLRSN or LRSNSTRT. Specify this keyword only once.

LRSNEND ***(hexadecimal-constant)***
Specifies the logical record sequence number (LRSN) of the last record to be scanned. The default is FFFFFFFFFFFF (the end of the data sets). You can use this keyword only with LRSNSTART.

You can also use the form ENDLRSN.

Specify this keyword only once.

RBASTART ***(hexadecimal-constant)***
Specifies the log RBA from which to begin processing. You cannot use this keyword together with LRSNSTART.

You can also use the forms STARTRBA or ST. Specify this keyword only once.

RBAEND ***(hexadecimal-constant)***
Specifies the last valid log RBA that is to be processed. If this keyword is omitted, processing continues to the end of the log (FFFFFFFFFFFF if 6 byte RBAs are in use, or FFFFFFFFFFFFFFFF if 8 byte RBAs are in use). You can use this keyword only with RBASTART.

You can also use the forms ENDRBA or EN. Specify this keyword only once.

PAGESET ***(decimal-integer)***
Specifies a page set identifier. The number must be in the range 00 through 99. You can specify a maximum of 10 PAGESET keywords. If PAGESET keywords are specified, only log records associated with the page sets you specify are processed.

URID ***(hexadecimal-constant)***
Specifies a hexadecimal unit of recovery identifier. Changes to data occur in the context of an IBM MQ unit of recovery. A unit of recovery is identified on the log by a BEGIN UR record. The log RBA of that BEGIN UR record is the URID value you must use. If you know the URID for a particular UR that you are interested in, you can limit the extraction of information from the log to that URID.

The hexadecimal constant can consist of 1 through 16 characters (8 bytes), and leading zeros are not required.

You can specify a maximum of 10 URID keywords.

DATA ***(hexadecimal-string)***
Specifies a data string in hexadecimal.

The string can consist of 2 through 48 characters (24 bytes), and must have an even number of characters.

You can specify a maximum of 10 DATA keywords.

If DATA keywords are specified, only log records that contain at least one of the strings are processed.

Note: Though you can use the DATA and EXTRACT parameters together, it is difficult to reliably derive meaning from the output, unless you have a good understanding of the internal implementation of IBM MQ. This is because only the low level individual log records that contain the requested DATA are processed so you do not extract the full output that is logically associated with the data, only the records where that DATA sequence actually appears. For example you might get only records associated with putting messages and not with getting messages, or you might get only the first part of the data for long messages because the rest of the data is in other log records that do not contain the requested DATA string.

RM *(resource_manager)*
Specifies a particular resource manager. Only records associated with this resource manager are processed. Valid values for this keyword are:

RECOVERY
Recovery log manager

DATA Data manager

BUFFER
Buffer manager

IMSBRIDGE
IMS bridge

SUMMARY (YES|NO|ONLY)
Specifies whether a summary report is to be produced or not:

YES Produce a summary report in addition to the detail report.

NO Do not produce a summary report.

ONLY Produce only a summary report (no detail report).

The default is NO.

EXTRACT (YES|NO)
Specifying EXTRACT(YES) causes each log record that meets the input selection criteria to be written to the appropriate output file, as explained on page "The EXTRACT function" on page 1950. The default is NO.

Note: Though you can use the DATA and EXTRACT parameters together, it is difficult to reliably derive meaning from the output, unless you have a good understanding of the internal implementation of IBM MQ. This is because only the low level individual log records that contain the requested DATA are processed so you do not extract the full output that is logically associated with the data, only the records where that DATA sequence actually appears. For example you might get only records associated with putting messages and not with getting messages, or you might get only the first part of the data for long messages because the rest of the data is in other log records that do not contain the requested DATA string.

DECOMPRESS (YES|NO)
Specifies whether any compressed log records will be expanded:

YES Any compressed log records will be expanded before a Search, Print or Extract function is performed

NO Any compressed log records will not be expanded before a Search or Print function is performed. Do not use DECOMPRESS(NO) with the Extract function

The default is YES.

Usage notes

1. If your queue manager is in a queue-sharing group, you can specify the log range required by either LRSNSTART (optionally with LRSNEND) or RBASTART (optionally with RBAEND). You cannot mix LRSN and RBA specifications.

   If you need to coordinate the log information from the different queue managers in the queue-sharing group, use LRSN specifications.
2. If your queue manager is not in a queue-sharing group, you cannot use LRSN specifications; you must use RBA specifications.
3. If you are using a BSDS, RBASTART or LRSNSTART must be specified.
4. CSQ1LOGP starts its processing on the first record containing an LRSN or RBA value greater than or equal to the value specified on LRSNSTART or RBASTART.
5. Normally you are only interested in the most recent additions to the log. Take care to choose a suitable value for the start of the log range, and do not use the defaults. Otherwise, you create an enormous amount of data, most of which is of no interest to you.

The EXTRACT function

Typical uses of the `EXTRACT` parameter are to:

- Review which persistent messages were put to or got from a queue and whether the request was committed. This allows messages to be replayed.
- Review persistent messages that were put or got, but the request was backed out.
- Display which applications backed out rather than committed.
- Discover the volume of persistent data processed by queues, to identify the high use queues.
- Identify which applications set object attributes.
- Re-create object definitions for recovery purposes after a major failure.

When CSQ1LOGP with the `EXTRACT` parameter set is run against a log data set it processes all records in the data set, or all those within a specified range. Processing is as follows:

1. When a commit request is found, if the CSQCMT ddname is present then the data is written to this data set. If the CSQBOTH ddname is present the data is also written to this data set.
2. When a backout request is found, if the CSQBACK ddname is present then the data is written to this data set. If the CSQBOTH ddname is present the data is also written to this data set.
3. When changes to objects are detected, the information is written to the data set identified by the CSQOBJS ddname.
4. When the last record has been processed, information about remaining units of work is written to the data set identified by the CSQINFLT ddname.

If you do not want to collect one or more of these classes of information, then omit the appropriate DD statements.

Example of processing EXTRACT data

The following job uses DFSORT facilities to process the file of committed records to add up the number of bytes put to each queue.

```
//TOOLRUN EXEC PGM=ICETOOL,REGION=1024K
//TOOLMSG  DD SYSOUT=*
//DFSMSG   DD SYSOUT=*
//TOOLIN   DD *
SORT FROM(IN) TO(TEMP1) USING(CTL1)
DISPLAY FROM(TEMP1) LIST(OUT1) ON(5,48,CH) ON(53,4,BI)
/*
//CTL1     DD *
* SELECT THE RECORDS WHICH WERE PUT
  INCLUDE COND=(180,5,CH,EQ,C'MQPUT')
* SORT BY QUEUE NAME
  SORT FIELDS=(112,48,CH,A)
* ONLY COPY THE QUEUE NAME AND SIZE OF USER DATA TO OUTPUT REC
  OUTREC FIELDS=(1,4,112,48,104,4)
* ADD UP THE NUMBER OF BYTES PROCESSED
* SUM FIELDS=(104,4,FI)
/*
//IN       DD DISP=SHR,DSN=commit.dataset
//TEMP1    DD DISP=(NEW,DELETE),DSN=&TEMP1,SPACE=(CYL,(10,10))
//OUT1     DD SYSOUT=*
```

Figure 42. Accumulating bytes put to each queue

This produces output in the following format:

BA1	3605616
BA10	3572328
BA2	3612624
BA3	3579336
BA4	3572328
BA5	3491736
BA6	3574080
BA7	3532032
BA8	3577584
BA9	3539040
SYSTEM.ADMIN.CHANNEL.EVENT	186120
SYSTEM.ADMIN.QMGR.EVENT	384
SYSTEM.CHANNEL.SYNCQ	46488312

The following table lists the samples that are provided to allow you to print and interpret the data generated when EXTRACT(YES) is used

Sample	Description
thlqual.SCSQLOAD(CSQ4LOGS)	Sample C program to report on the UOW activity and object manipulation
thlqual.SCSQC37S(CSQ4LOGS)	Source for sample C program
thlqual.SCSQC370(CSQ4LOGD)	C header file to map records generated when using the EXTRACT(YES) function of CSQ1LOGP
thlqual.SCSQPROC(CSQ4LOGJ)	Sample JCL to run program CSQ4LOGS

CSQ1LOGP output

Detail report

The detail report begins by echoing the input selection criteria specified by SYSIN, and then prints each valid log record encountered. Definitions of keywords in the detail report are as follows:

RM Resource manager that wrote the log record.

TYPE Type of log record.

URID BEGIN UR for this unit of recovery, see the previous description.

LRID Logical record identifier in the form: `AAAAAAAA.BBBBBBCC` where:

AAAAAAAA
Is the page set number.

BBBBBB
Is the relative page number in the page set.

CC Is the relative record number on the page.

LRSN Logical record sequence number (LRSN) of the log record scanned.

SUBTYPE
Subtype of the log record type.

CHANGE LENGTH
Length of the logged change.

CHANGE OFFSET
Start position of the change.

BACKWARD CHAIN
Pointer to the previous page.

FORWARD CHAIN
Pointer to the next page.

RECORD LENGTH
Length of the inserted record.

Record layouts for the output data sets

The data sets produced when the `EXTRACT` keyword is specified contains information about persistent messages. Messages are identified by their queue name and an eight character key. Once a message has been got, the key can be reused by another message, so it is important to ensure that time sequence is maintained. In the records are times. A time stamp can be extracted only from a Begin-UR record or from an MQPUT request. Thus if there is only a long running transaction which is getting messages, the times when the gets occurred are the time the transaction started (the Begin-UR record). If there are many short units of work, or many messages being put, the time is reasonably accurate (within milliseconds). Otherwise the times become less and less accurate.

Note: There is a 4 byte Record Descriptor Word at the front of each record because the files are Variable Blocked format. The first data byte of a variable-length record has relative position 5 and the first 4 bytes contain the record descriptor word. The field names correspond to those in the C header file CSQ4LOGD in thlqual.SCSQC370.

The information in the data sets has the following layout:

Offset		Type	Length	Name	Description
Dec	Hex				
0	0	Character	21	csrecorddate	The approximate time the log was written, in the format yyyy.ddd hh:mm:ss.thm
21	15	Character	7	cstimedelta	Approximate time difference in milliseconds from the start of the unit of work. Right-aligned and padded with blanks.
28	1C	64-bit integer	8	dtodout	Estimated time that the log record was created, in STCK format.
36	24	Character	8	csurid	Queue manager-specific unique identifier of the unit of work that created the log record.
44	2C	Character	12	cscorrelator	Thread correlation identifier
56	38	Character	8	csauth	Authorization identifier (Userid associated with unit of work)
64	40	64-bit integer	8	dtime	Time that the unit of work was started, in STCK format
72	48	Character	8	csresource	Resource name
80	50	Character	8	cscnty	Connection type: one of BATCH, RRSBATCH, IMS, CICS, CHIN, or nulls for an internal task
88	58	Character	8	cscnid	Connection ID of thread that created this unit of work
96	60	Character	3	csstatus	Unit of work type: BUR for begin or CP for checkpoint information
99	63	Integer	4	ldatalen	Length of the message data (if any)
103	67	Character	4	csqmgrname	Name of queue manager
107	6B	Character	48	csqueuename	Name of queue, for get, put, or expired messages. This field can be question marks. Question marks appear when it is not possible to determine the user ID associated with the entry. This typically happens when the begin_ur record or the checkpoint record from which you might get the URID is not in the log range specified in the job, nor on the log data sets used.
155	9B	Character	12	cssqdmcp	Shared queue message key. Blank if not a shared queue
167	A7	Character	8	csdmcp	Non-shared queue message key. Blank if a shared queue.

Offset		Type	Length	Name	Description
175	AF	Character	8	csverb	Activity: **ALTER** the object was changed **DEFINE** the object was created **MQGET** the message was got **MQPUT** the message was put **EXPIRE** the message expired **ABORT2** the message was backed out **PHASE1** the first phase of two-phase commit **PHASE2** the second phase of two-phase commit, or the only phase of one phase commit
183	B7	Character	1	cscmitstatus	Status of unit of work: **B** backed out **C** committed **I** inflight
184	B8	Character	1	csshunt	Shunted indicator: **S** shunted record **N** not shunted
185	B9	Character	8	cslogrba	RBA of log record
193	C1	Character	8	csshuntrba	RBA of shunted log record
201	C9	Character	1	csuowscope	UOW scope in hexadecimal: **01** local **02** shared
202	CA	Integer	4	lsegment	The segment number of the data, starting from 1.
206	CE		Variable		Data part
206	CE	Character	1	csbora	If csverb is ALTER, indicates whether the data is the 'before' or 'after' copy of the object. **B** before **A** after
207	CF	Character	Variable	csvardata	Message or object data. Length as given in ldatalen.

The queue-sharing group utility (CSQ5PQSG) on z/OS

z/OS

You can use the CSQ5PQSG utility program to add queue-sharing group and queue manager definitions to the IBM MQ Db2 tables, and to remove them.

The CSQ5PQSG utility can also be used to verify the consistency of Db2 object definitions for queue manager, CF structure, and shared queue objects, within a queue-sharing group.

- Invoking the queue-sharing group utility
- Syntax, keywords, and parameters
- Example

Invoking the queue-sharing group utility

Figure 43 shows an example of the JCL used to invoke the CSQ5PQSG utility.

```
//S001   EXEC PGM=CSQ5PQSG,REGION=4M,
//     PARM='function,function parameters'
//STEPLIB  DD DSN=thlqual.SCSQANLE,DISP=SHR
//         DD DSN=thlqual.SCSQAUTH,DISP=SHR
//         DD DSN=db2qual.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
```

Figure 43. Sample JCL to invoke the CSQ5PQSG utility

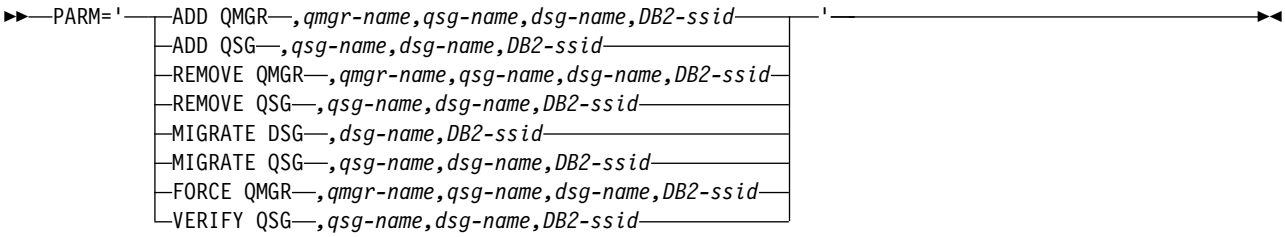
Data definition statements

The CSQ5PQSG utility requires data definition statements with the following DDname:

SYSPRINT
This statement is required; it names the data set for print output. The logical record length (LRECL) is 125.

Syntax, keywords, and parameters

Queue-sharing group utility



A queue-sharing group name (*qsg-name*) can have up to 4 characters, comprising uppercase A-Z, 0-9, $, #, @. It must not start with a numeric. For implementation reasons, names of less than 4 characters are padded internally with @ symbols, so do not use names ending in @.

The queue-sharing group name must be different from any of the queue manager names within the queue-sharing group.

PARM

This field contains the function request followed by the function-specific parameters. These are described in the following text:

ADD QMGR

Add a queue manager record into the CSQ.ADMIN_B_QMGR table. This operation completes successfully only if all the following conditions are met:

- A corresponding queue-sharing group record exists in the CSQ.ADMIN_B_QSG table.
- The queue manager entry does not exist in the CSQ.ADMIN_B_QMGR table as the member of a different queue-sharing group.
- There is no member entry in the XCF group with a different QMGR number value than the one created by the utility when you add a record to the CSQ.ADMIN_B_QMGR table.

If there are members in the XCF group without the corresponding entries in the Db2 table, you can use the utility to add them. Add queue managers in the order that is indicated by the CSQU524I messages that are issued by the queue-sharing group utility (CSQ5PQSG) when it is run with the **VERIFY QSG** parameter.

If a queue manager exists in Db2 table CSQ.ADMIN_B_QMGR, but is missing from MVS XCF group, you can run this utility to restore the appropriate XCF group entry, as indicated by CSQ5010E message.

qmgr-name
The queue manager name

qsg-name
The queue-sharing group name

dsg-name
The Db2 data-sharing group name

DB2-ssid
The Db2 subsystem ID

ADD QSG

Add a queue-sharing group record into the CSQ.ADMIN_B_QSG table.

qsg-name
The queue-sharing group name

dsg-name
The Db2 data-sharing group name

DB2-ssid
The Db2 subsystem ID

REMOVE QMGR

Remove a queue manager record from the CSQ.ADMIN_B_QMGR table. This only completes successfully if the queue manager has either never been started, or terminated normally from its last execution.

qmgr-name
The queue manager name

qsg-name
The queue-sharing group name

dsg-name
The Db2 data-sharing group name

DB2-ssid
The Db2 subsystem ID

REMOVE QSG

Remove a queue-sharing group record from the CSQ.ADMIN_B_QSG table. This only completes successfully if no queue managers are defined to the queue-sharing group.

qsg-name
The queue-sharing group name

dsg-name
The Db2 data-sharing group name

DB2-ssid
The Db2 subsystem ID

MIGRATE DSG

Check that the installation is ready to migrate its Db2 data-sharing group table definitions from an earlier version of IBM WebSphere MQ to IBM MQ Version 9.0.

It verifies that all the queue managers in the data-sharing group have applied the migration and coexistence PTF, and have since been started.

dsg-name
The Db2 data-sharing group name

DB2-ssid
The Db2 subsystem ID

This function does not actually do the migration, which involves several steps. It is included as part of the sample migration job CSQ4570T and CSQ4571T in thlqual SCSQPROC.

Migration requires that a migration PTF is installed on **all** queue managers in the data-sharing group.

MIGRATE QSG

Check that the installation is ready to migrate its Db2 data-sharing group table definitions from an earlier version of IBM WebSphere MQ to IBM MQ Version 9.0.

The MIGRATE QSG and MIGRATE DSG functions perform the same function. The only difference is in the scope of the processing. MIGRATE QSG works on a single queue-sharing group only, MIGRATE DSG works on all queue-sharing groups that are defined within the data-sharing group.

It verifies that:

- All the queue managers in the queue-sharing group have applied the migration & coexistence PTF and have since been started.

qsg-name
The queue-sharing group name

dsg-name
The Db2 data-sharing group name

DB2-ssid
The Db2 subsystem ID

This function does not actually do the migration, which involves several steps. It is included as part of the sample migration job CSQ4570T and CSQ4571T in thlqual SCSQPROC.

Migration requires that a migration PTF is installed on **all** queue managers in the queue-sharing group.

FORCE QMGR

Remove a queue manager record from the CSQ.ADMIN_B_QMGR table, even if the queue manager has terminated abnormally.

Use the **FORCE** option, rather than **REMOVE**, to remove the last queue manager in a queue-sharing group.

Attention: This can override IBM MQ efforts to maintain data in a consistent state. Only use this function when you cannot carry out the procedure for removing a queue manager from a queue-sharing group on page Removing a queue manager from a queue-sharing group.

qmgr-name
The queue manager name

qsg-name
The queue-sharing group name

dsg-name
The Db2 data-sharing group name

DB2-ssid
The Db2 subsystem ID

VERIFY QSG
Validate the consistency of the Db2 object definitions for queue manager, CF structure, and shared queue objects, within the queue-sharing group.

qsg-name
The queue-sharing group name

dsg-name
The Db2 data-sharing group name

DB2-ssid
The Db2 subsystem ID

Example

The following sample JCL adds an entry for queue manager QM01 into queue-sharing group QSG1. It specifies a connection to Db2 subsystem DB2A, which is a member of Db2 data-sharing group DSN510PG.

```
//S001   EXEC PGM=CSQ5PQSG,REGION=4M,
//     PARM='ADD QMGR,QM01,QSG1,DSN510PG,DB2A'
//STEPLIB  DD DSN=thlqual.SCSQANLE,DISP=SHR
//         DD DSN=thlqual.SCSQAUTH,DISP=SHR
//         DD DSN=db2qual.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
```

Figure 44. Using the queue-sharing group utility to add a queue manager into a queue-sharing group

The active log preformat utility (CSQJUFMT) on z/OS

z/OS

You can use the CSQJUFMT utility to format active log data sets before they are used by a queue manager.

If the active log data sets are preformatted by the utility, log write performance is improved on the queue manager's first pass through the active logs. If the utility is not used, the queue manager must format each log control interval at log write time before it is used. On the second and subsequent passes through the active log data sets, the log control intervals already contain data, so need no further formatting, and no performance benefit accrues.

Invoking the CSQJUFMT utility

You can only run the CSQJUFMT program before starting the queue manager that use the logs.

Note: Do not use this utility to format a log data set after the queue manager has started, or data will be lost.

```
EXEC PGM=CSQJUFMT
```

Each step running the CQJUFMT utility formats a single active log data set. Add additional CSQJUFMT steps for each active log being created.

Attention: JCL limits the number of steps in a single job to 255. If you are formatting more than 255 active log data sets, you will need to run multiple jobs.

These DD statements should be provided:

SYSPRINT
This statement is required to specify a data set or print spool class for print output.

SYSUT1
This statement identifies the log data set to be preformatted.

```
//JOBLIB    DD   DISP=SHR,DSN=thlqual.SCSQANLE
//          DD   DISP=SHR,DSN=thlqual.SCSQAUTH
//*
//JUFMT11   EXEC PGM=CSQJUFMT
//SYSPRINT  DD   SYSOUT=*
//SYSUT1    DD   DISP=OLD,DSN=hlq.LOGCOPY1.DS01
//*
//JUFMT21   EXEC PGM=CSQJUFMT
//SYSPRINT  DD   SYSOUT=*
//SYSUT1    DD   DISP=OLD,DSN=hlq.LOGCOPY2.DS01
```

Figure 45. Example of the JCL used to invoke the CSQJUFMT utility

Sample JCL is supplied in thlqual.SCSQPROC (CSQ4LFMT) for preformatting a newly defined dual log data set. It contains two steps, one step to format each of the copies of the log data set.

The dead-letter queue handler utility (CSQUDLQH) on z/OS

z/OS

You can use the default dead-letter utility (CSQUDLQH) to handle message written to the dead-letter queue.

A *dead-letter queue* (DLQ) is a holding queue for messages that cannot be delivered to their destination queues. Every queue manager in a network can have an associated DLQ.

Queue managers, message channel agents, and applications can put messages on the DLQ. All messages on the DLQ can be prefixed with a *dead-letter header* structure, MQDLH. Messages put on the DLQ by a queue manager or by a message channel agent always have a dead-letter header; ensure that applications putting messages on the DLQ also supply a dead-letter header structure. The *Reason* field of the MQDLH structure contains a reason code that identifies why the message is on the DLQ.

Implement a routine that runs regularly to process messages on the DLQ. Such a routine is called a *dead-letter queue handler*. IBM MQ supplies a default *dead-letter queue handler* (DLQ handler) called CSQUDLQH. A user-written *rules table* supplies instructions to the DLQ handler, for processing messages on the DLQ. That is, the DLQ handler matches messages on the DLQ against entries in the rules table. When a DLQ message matches an entry in the rules table, the DLQ handler performs the action associated with that entry.

Invoking the DLQ handler on z/OS: z/OS

Use this topic to understand how to invoke the CSQUDLQH utility program, and its data definition statements.

The CSQUDLQH utility program runs as a z/OS batch program. Specify the name of the dead-letter queue that you want to process and the queue manager on which it resides. You can do this in one of the following two ways (in these examples, the dead-letter queue is called CSQ1.DEAD.QUEUE and the queue manager is called CSQ1):

1. The names can be specified as positional parameters in the PARM parameter of the EXEC statement within the submitted JCL, for example:

```
//READQ EXEC PGM=CSQUDLQH,
//  PARM='CSQ1.DEAD.QUEUE CSQ1'
```

Figure 46. Specifying the queue manager and dead-letter queue names for the dead-letter queue handler in the JCL

2. The names can be specified in the rules table, for example:

```
INPUTQ(CSQ1.DEAD.QUEUE) INPUTQM(CSQ1)
```

Figure 47. Specifying the queue manager and dead-letter queue names for the dead-letter queue handler in the rules table

Any parameters that you specify in the PARM parameter override those in the rules table. If you specify only one parameter in the PARM statement, it is used as the name of the dead-letter queue. The rules table is taken from the SYSIN data set.

For further information on the keywords you can specify, to match and process pattern and action keywords, see “Rules (patterns and actions) on z/OS” on page 1963.

Stopping the DLQ handler

The CSQUDLQH utility is stopped when any of the following conditions is true:

- The dead letter queue is empty for a specified amount of time as configured by the WAIT control data keyword.
- The dead letter queue is set to GET(DISABLED).
- The queue manager is quiesced.
- The CSQUDLQH job is cancelled.

Messages generated during the handling of the queue are written to the standard output when the CSQUDLQH utility ends in a controlled manner. If the handler is cancelled, it does not generate these messages.

Data definition statements

CSQUDLQH requires DD statements with these DDnames:

SYSOUT
: This statement is required; it names the data set for print output. You can specify the logical record length (LRECL) and block size (BLKSIZE) for this output data set.

SYSIN
: This statement is required; it names the input data set containing the rules table that specifies what the utility is to do. The logical record length (LRECL) is 80.

Sample JCL

```
//READQ EXEC PGM=CSQUDLQH,
//      PARM='CSQ1.DEAD.QUEUE CSQ1'
//STEPLIB  DD DSN=thlqual.SCSQAUTH,DISP=SHR
//         DD DSN=thlqual.SCSQLOAD,DISP=SHR
//         DD DSN=thlqual.SCSQANLE,DISP=SHR
//SYSOUT   DD SYSOUT=*
//SYSIN    DD *
INPUTQM(CSQ2) INPUTQ('CSQ2.DEAD.QUEUE')
ACTION(RETRY)
/*
```

Figure 48. Sample JCL to invoke the CSQUDLQH utility. In this example, queue manager CSQ1 and dead-letter queue CSQ1.DEAD.QUEUE are used because the values specified in the PARM statement override the values specified in the SYSIN data set.

The DLQ handler rules table on z/OS: z/OS

The DLQ handler rules table defines how the DLQ handler is to process messages that arrive on the DLQ.

There are two types of entry in a rules table:

- The first entry in the table, which is optional, contains "Control data."
- All other entries in the table are *rules* for the DLQ handler to follow. Each rule consists of a *pattern* (a set of message characteristics) that a message is matched against, and an *action* to be taken when a message on the DLQ matches the specified pattern. There must be at least one rule in a rules table.

Each entry in the rules table comprises one or more keywords.

See "Rules table conventions on z/OS" on page 1966 for information about the syntax of the rules table.

See Rules (patterns and actions) for information about how the pattern-matching, and action keywords control the CSQUDLQH utility

Control data

This section describes the keywords that you can include in a control-data entry in a DLQ handler rules table.

- All keywords are optional.
- If a control-data entry is included in the rules table, it must be the first entry in the table.
- The default value for a keyword, if any, is underlined.
- The vertical line (|) separates alternatives. You can specify only one of these.

INPUTQ (*QueueName***|' ')**
Specifies the name of the DLQ that you want to process:

1. If you specify a queue name in the PARM parameter of the EXEC statement, this overrides any INPUTQ value in the rules table.
2. If you do not specify a queue name in the PARM parameter of the EXEC statement, the INPUTQ value in the rules table is used.
3. If you do not specify a queue name in the PARM parameter of the EXEC statement or the rules table, the dead-letter queue named *qmgr-name*.DEAD.QUEUE is used if it has been defined. If this queue does not exist, the program fails and returns error message CSQU224E, giving the reason code for the error.

INPUTQM (*QueueManagerName***|' ')**
Specifies the name of the queue manager that owns the DLQ named on the INPUTQ keyword.

1. If you specify a queue manager name in the PARM parameter of the EXEC statement, this overrides any INPUTQM value in the rules table.
2. If you do not specify a queue manager name in the PARM parameter of the EXEC statement, the INPUTQM value in the rules table is used.
3. If you do not specify a queue manager name in the PARM parameter of the EXEC statement or the rules table, the default queue manager is used (if one has been defined using CSQBDEFV). If not, the program fails and returns error message CSQU220E, giving the reason code for the error.

RETRYINT (*Interval***|60)**
Specifies the interval, in seconds, at which the DLQ handler should attempt to reprocess messages on the DLQ that could not be processed at the first attempt, and for which repeated attempts have been requested. The DLQ handler reprocesses messages after it has first browsed to the end of the queue.

The default is 60 seconds.

WAIT (YES|NO| *nnn*)
Specifies whether the DLQ handler should wait for further messages to arrive on the DLQ when it detects that there are no further messages that it can process.

YES The DLQ handler waits indefinitely.

NO The DLQ handler terminates when it detects that the DLQ is either empty or contains no messages that it can process.

nnn The DLQ handler waits for *nnn* seconds for new work to arrive after it detects that the queue is either empty or contains no messages that it can process, before terminating.

Specify a value in the range 1 through 999 999.

Specify WAIT (YES) for busy DLQs, and WAIT (NO) or WAIT (*nnn*) for DLQs that have a low level of activity. If the DLQ handler is allowed to terminate, you can use triggering to invoke it when needed.

Rules (patterns and actions) on z/OS: z/OS

The DLQ handler is controlled with a series of pattern-matching and action keywords described here.

Figure 49 shows an example rule from a DLQ handler rules table.

```
PERSIST(MQPER_PERSISTENT) REASON (MQRC_PUT_INHIBITED) +
 ACTION (RETRY) RETRY (3)
```

Figure 49. An example rule from a DLQ handler rules table. This rule instructs the DLQ handler to make three attempts to deliver to its destination queue any persistent message that was put on the DLQ because MQPUT and MQPUT1 were inhibited.

This section describes the keywords that you can include in a rules table. It begins with a description of the pattern-matching keywords (those keywords against which messages on the DLQ are matched). It then describes the action keywords (those keywords that determine how the DLQ handler is to process a matching message).

- All keywords except ACTION are optional.
- The default value for a keyword, if any, is underlined. For most keywords, the default value is asterisk (*), which matches any value.
- The vertical line (|) separates alternatives. You can specify only one of these keywords.

The keywords can be grouped as follows:

- The pattern-matching keywords
- The action keywords

The pattern-matching keywords

The pattern-matching keywords, are described in the following table. You use these keywords to specify values against which messages on the DLQ are matched. All pattern-matching keywords are optional.

APPLIDAT (*ApplIdentityData*|*)
The *ApplIdentityData* value of the message on the DLQ, specified in the message descriptor, MQMD.

APPLNAME (*PutApplName*|*)
The name of the application that issued the MQPUT or MQPUT1 call, as specified in the *PutApplName* field of the message descriptor, MQMD, of the message on the DLQ.

APPLTYPE (*PutApplType*|*)
The *PutApplType* value specified in the message descriptor, MQMD, of the message on the DLQ.

DESTQ (*QueueName***|*)**
The name of the message queue for which the message is destined.

DESTQM (*QueueManagerName***|*)**
The queue manager name for the message queue for which the message is destined.

FEEDBACK (*Feedback***|*)**
Describes the nature of the report when the *MsgType* value is MQMT_REPORT.

You can use symbolic names. For example, you can use the symbolic name MQFB_COA to identify those messages on the DLQ that require confirmation of their arrival on their destination queues. A few symbolic names are not accepted by the utility and lead to a syntax error. In these cases, you can use the corresponding numeric value.

FORMAT (*Format***|*)**
The name that the sender of the message uses to describe the format of the message data.

MSGTYPE (*MsgType***|*)**
The message type of the message on the DLQ.

You can use symbolic names. For example, you can use the symbolic name MQMT_REQUEST to identify those messages on the DLQ that require replies.

PERSIST (*Persistence***|*)**
The persistence value of the message. (The persistence of a message determines whether it survives restarts of the queue manager.)

You can use symbolic names. For example, you can use the symbolic name MQPER_PERSISTENT to identify those messages on the DLQ that are persistent.

REASON (*ReasonCode***|*)**
The reason code that describes why the message was put to the DLQ.

You can use symbolic names. For example, you can use the symbolic name MQRC_Q_FULL to identify those messages placed on the DLQ because their destination queues were full. A few symbolic names are not accepted by the utility and lead to a syntax error. In these cases, you can use the corresponding numeric value.

REPLYQ (*QueueName***|*)**
The reply-to queue name specified in the message descriptor, MQMD, of the message on the DLQ.

REPLYQM (*QueueManagerName***|*)**
The queue manager name of the reply-to queue specified in the REPLYQ keyword.

USERID (*UserIdentifier***|*)**
The user ID of the user who originated the message on the DLQ, as specified in the message descriptor, MQMD.

The action keywords

The action keywords are described in the following table. You use these keywords to describe how a matching message is processed.

ACTION (DISCARD|IGNORE|RETRY|FWD)
The action taken for any message on the DLQ that matches the pattern defined in this rule.

DISCARD
Causes the message to be deleted from the DLQ.

IGNORE
Causes the message to be left on the DLQ.

RETRY
Causes the DLQ handler to try again to put the message on its destination queue.

FWD Causes the DLQ handler to forward the message to the queue named on the FWDQ keyword.

You must specify the ACTION keyword. The number of attempts made to implement an action is governed by the RETRY keyword. The RETRYINT keyword of the control data controls the interval between attempts.

CONVERT (YES|NO)
By default, this keyword is set to CONVERT(YES). When forwarding or retrying a message, the DLQ handler performs an MQGET with MQGMO_CONVERT; that is, it converts the message data to the CCSID and encoding of the queue manager.

However, setting CONVERT(NO) forwards or retries the message without converting the message contents.

FWDQ (*QueueName*|&DESTQ|&REPLYQ)
The name of the message queue to which the message is forwarded when you select the ACTION keyword.

QueueName
This parameter is the name of a message queue. FWDQ(' ') is not valid.

&DESTQ
Takes the queue name from the *DestQName* field in the MQDLH structure.

&REPLYQ
Takes the name from the *ReplyToQ* field in the message descriptor, MQMD. You can specify REPLYQ (?*) in the message pattern to avoid error messages, when a rule specifying FWDQ (&REPLYQ), matches a message with a blank *ReplyToQ* field.

FWDQM (*QueueManagerName*|&DESTQM|&REPLYQM|' ')
The queue manager of the queue to which a message is forwarded.

QueueManagerName
This parameter defines the queue manager name for the queue to which the message is forwarded when you select the ACTION (FWD) keyword.

&DESTQM
Takes the queue manager name from the *DestQMgrName* field in the MQDLH structure.

&REPLYQM
Takes the name from the *ReplyToQMgr* field in the message descriptor, MQMD.

'' The local queue manager.

HEADER (YES|NO)
Whether the MQDLH should remain on a message for which ACTION (FWD) is requested. By default, the MQDLH remains on the message. The HEADER keyword is not valid for actions other than FWD.

PUTAUT (DEF|CTX)
The authority with which messages should be put by the DLQ handler:

DEF Puts messages with the authority of the DLQ handler itself.

CTX Causes the messages to be put with the authority of the user ID in the message context. You must be authorized to assume the identity of other users, if you specify PUTAUT (CTX).

RETRY (*RetryCount*|1)
The number of times that an action should be attempted (at the interval specified on the RETRYINT keyword of the control data). Specify a value in the range 1 through 999 999 999.

Note: The count of attempts made by the DLQ handler to implement any particular rule is specific to the current instance of the DLQ handler; the count does not persist across restarts. If you restart the DLQ handler, the count of attempts made to apply a rule is reset to zero.

Rules table conventions on z/OS: z/OS

Use this topic to understand the conventions used in the CSQUDLQH rule table.

The rules table must adhere to the following conventions regarding its syntax, structure, and contents:

- A rules table must contain at least one rule.
- Keywords can occur in any order.
- A keyword can be included once only in any rule.
- Keywords are not case-sensitive.
- A keyword and its parameter value can be separated from other keywords by at least one blank or comma.
- Any number of blanks can occur at the beginning or end of a rule, and between keywords, punctuation, and values.
- Each rule must begin on a new line.
- For reasons of portability, the significant length of a line should not be greater than 72 characters.
- Use the plus sign (+) as the last nonblank character on a line to indicate that the rule continues from the first nonblank character in the next line. Use the minus sign (-) as the last nonblank character on a line to indicate that the rule continues from the start of the next line. Continuation characters can occur within keywords and parameters.

  For example:

  ```
  APPLNAME('ABC+
    D')
  ```

  results in 'ABCD'.

  ```
  APPLNAME('ABC-
    D')
  ```

  results in 'ABC D'.
- Comment lines, which begin with an asterisk (*), can occur anywhere in the rules table.
- Blank lines are ignored.

Each entry in the DLQ handler rules table comprises one or more keywords and their associated parameters. The parameters must follow these syntax rules:

- Each parameter value must include at least one significant character. The delimiting quotation marks in following examples are not considered significant. For example, these parameters are valid:

  FORMAT('ABC')
  3 significant characters

  FORMAT(ABC)
  3 significant characters

  FORMAT('A')
  1 significant character

  FORMAT(A)
  1 significant character

  FORMAT(' ')
  1 significant character

  These parameters are not valid because they contain no significant characters:

  – `FORMAT('')`

  - FORMAT( )
  - FORMAT()
  - FORMAT
- Wildcard characters are supported. You can use the question mark (?) instead of any single character, except a trailing blank. You can use the asterisk (*) instead of zero or more adjacent characters. The asterisk (*) and the question mark (?) are ***always*** interpreted as wildcard characters in parameter values.
- You cannot include wildcard characters in the parameters of these keywords: ACTION, HEADER, RETRY, FWDQ, FWDQM, and PUTAUT.
- Trailing blanks in parameter values, and in the corresponding fields in the message on the DLQ, are not significant when performing wildcard matches. However, leading and embedded blanks within strings in quotation marks are significant to wildcard matches.
- Numeric parameters cannot include the question mark (?) wildcard character. You can include the asterisk (*) instead of an entire numeric parameter, but the asterisk cannot be included as part of a numeric parameter. For example, these are valid numeric parameters:

  MSGTYPE(2)
  Only reply messages are eligible

  MSGTYPE(*)
  Any message type is eligible

  MSGTYPE('*')
  Any message type is eligible

  However, MSGTYPE('2*') is not valid, because it includes an asterisk (*) as part of a numeric parameter.
- Numeric parameters must be in the range zero through 999 999 999 unless otherwise stated. If the parameter value is in this range, it is accepted, even if it is not currently valid in the field to which the keyword relates. You can use symbolic names for numeric parameters.
- If a string value is shorter than the field in the MQDLH or MQMD to which the keyword relates, the value is padded with blanks to the length of the field. If the value, excluding asterisks, is longer than the field, an error is diagnosed. For example, these are all valid string values for an eight character field:

  'ABCDEFGH'
  8 characters

  'A*C*E*G*I'
  5 characters excluding asterisks

  '*A*C*E*G*I*K*M*O*'
  8 characters excluding asterisks
- Strings that contain blanks, lowercase characters, or special characters other than period (.), forward slash (/), underscore (_), and percent sign (%) must be enclosed in single quotation marks. Lowercase characters not enclosed in quotation marks are folded to uppercase. If the string includes a quotation, two single quotation marks must be used to denote both the beginning and the end of the quotation. When the length of the string is calculated, each occurrence of double quotation marks is counted as a single character.

Processing the rules table on z/OS: z/OS

Use this topic to understand how the CSQUDLQH utility processes the rules table.

The DLQ handler searches the rules table for a rule with a pattern that matches a message on the DLQ. The search begins with the first rule in the table, and continues sequentially through the table. When a rule with a matching pattern is found, the rules table attempts the action from that rule. The DLQ handler increments the retry count for a rule by 1 whenever it attempts to apply that rule. If the first attempt fails, the attempt is repeated until the count of attempts made matches the number specified on the RETRY keyword. If all attempts fail, the DLQ handler searches for the next matching rule in the table.

This process is repeated for subsequent matching rules until an action is successful. When each matching rule has been attempted the number of times specified on its RETRY keyword, and all attempts have failed, ACTION (IGNORE) is assumed. ACTION (IGNORE) is also assumed if no matching rule is found.

For further information, see Ensuring that all DLQ messages are processed.

Note:

1. Matching rule patterns are sought only for messages on the DLQ that begin with an MQDLH. If the dead-letter queue handler encounters one or more messages that are not prefixed by an MQDLH, it issues an information message to report this. Messages that do not contain an MQDLH are not processed by the DLQ handler and remain on the dead-letter queue until dealt with by another method.
2. All pattern keywords can default, so that a rule can consist of an action only. Note, however, that action-only rules are applied to all messages on the queue that have MQDLHs and that have not already been processed in accordance with other rules in the table.
3. The rules table is validated when the DLQ handler starts, and errors flagged at that time. You can change the rules table at any time, but those changes do not come into effect until the DLQ handler is restarted.
4. The DLQ handler does not alter the content of messages, of the MQDLH, or of the message descriptor. The DLQ handler always puts messages to other queues with the message option MQPMO_PASS_ALL_CONTEXT.
5. Consecutive syntax errors in the rules table might not be recognized because the validation of the rules table is designed to eliminate the generation of repetitive errors.
6. The DLQ handler opens the DLQ with the MQOO_INPUT_AS_Q_DEF option.
7. Do not run applications that perform `MQGET` calls against the queue at the same time as the DLQ handler. This includes multiple instances of the DLQ handler. There is typically a one-to-one relationship between the dead-letter queue and the DLQ handler.

Ensuring that all DLQ messages are processed

The DLQ handler keeps a record of all messages on the DLQ that have been seen but not removed. If you use the DLQ handler as a filter to extract a small subset of the messages from the DLQ, the DLQ handler still keeps a record of those messages on the DLQ that it did not process. Also, the DLQ handler cannot guarantee that new messages arriving on the DLQ will be seen, even if the DLQ is defined as first-in first-out (FIFO). Therefore, if the queue is not empty, the DLQ is periodically rescanned to check all messages. For these reasons, ensure that the DLQ contains as few messages as possible. If messages that cannot be discarded or forwarded to other queues (for whatever reason) are allowed to accumulate on the queue, the workload of the DLQ handler increases and the DLQ itself is in danger of filling up.

You can take specific measures to enable the DLQ handler to empty the DLQ. For example, do not use ACTION (IGNORE), which leaves messages on the DLQ. (Remember that ACTION (IGNORE) is assumed for messages that are not explicitly addressed by other rules in the table.) Instead, for those messages that you would otherwise ignore, use an action that moves the messages to another queue. For

example:

```
ACTION (FWD) FWDQ (IGNORED.DEAD.QUEUE) HEADER (YES)
```

Similarly, the final rule in the table should be a catchall to process messages that have not been addressed by earlier rules in the table. For example, the final rule in the table could be something like this:

```
ACTION (FWD) FWDQ (REALLY.DEAD.QUEUE) HEADER (YES)
```

This forwards messages that fall through to the final rule in the table to the queue REALLY.DEAD.QUEUE, where they can be processed manually. If you do not have such a rule, messages are likely to remain on the DLQ indefinitely.

An example DLQ handler rules table on z/OS: z/OS

Use this topic as an example of the DLQ handler rules table.

Here is an example rules table that contains a single control-data entry and several rules:

```
*****************************************************************************
*            An example rules table for the CSQUDLQH utility                *
*****************************************************************************
* Control data entry
* ------------------
* If no queue manager name is supplied as an explicit parameter to CSQUDLQH,
* use the default queue manager.
* If no queue name is supplied as an explicit parameter to CSQUDLQH, use the
* DLQ defined for the queue manager.
*
inputqm(' ')  inputq(' ')

* Rules
* -----

* The first check deals with attempted security violations.
* If a message was placed on the DLQ because the putter did not have the
* appropriate authority for the target queue, forward the message to a queue
* for manual inspection.

REASON(MQRC_NOT_AUTHORIZED) ACTION(FWD) +
FWDQ(DEADQ.MANUAL.SECURITY)

* The next set of rules with ACTION (RETRY) try to deliver the message to the
* intended destination.

* If a message is placed on the DLQ because its destination queue is full,
* attempt to forward the message to its destination queue.  Make 5 attempts at
* approximately 60-second intervals (the default value for RETRYINT).

REASON(MQRC_Q_FULL) ACTION(RETRY) RETRY(5)

* If a message is placed on the DLQ because there has been a problem starting the
* application by triggering, forward the message to another queue for manual
* inspection.

REASON(MQFB_APPL_CANNOT_BE_STARTED) ACTION(FWD) +
FWDQ(DEADQ.MANUAL.TRIGGER)

* If a message is placed on the DLQ because of a put inhibited condition, attempt
* to forward the message to its destination queue.  Make 5 attempts at
* approximately 60-second intervals (the default value for RETRYINT).

REASON(MQRC_PUT_INHIBITED) ACTION(RETRY) RETRY(5)
```

```
* The AAAA corporation often send messages with incorrect addresses. When we find
* a request from the AAAA corporation, we return it to the DLQ (DEADQ) of the
* reply-to queue manager (&REPLYQM). The AAAA DLQ handler attempts to
* redirect the message.

MSGTYPE(MQMT_REQUEST) REPLYQM(AAAA.*) +
  ACTION(FWD) FWDQ(DEADQ) FWDQM(&REPLYQM)

* The BBBB corporation requests that we try sending messages to queue manager
* BBB2 if queue manager BBB1 is unavailable.

DESTQM(BBB1) +
  ACTION(FWD) FWDQ(&DESTQ) FWDQM(BBB2) HEADER(NO)

* The CCCC corporation is very security conscious, and believes that none of its
* messages will ever end up on one of our DLQs.  If we do see a message from a
* CCCC queue manager on our DLQ, we send it to a special destination in the CCCC
* organization where the problem is investigated.

REPLYQM(CCCC.*) +
  ACTION(FWD) FWDQ(ALARM) FWDQM(CCCC.SYSTEM)

* Messages that are not persistent risk being lost when a queue manager terminates.
* If an application is sending nonpersistent messages, it will be able to cope with
* the message being lost, so we can afford to discard the message.

PERSIST(MQPER_NOT_PERSISTENT) ACTION(DISCARD)

* For performance and efficiency reasons, we like to keep the number of messages on
* the DLQ small. If we receive a message that has not been processed by an earlier
* rule in the table, we assume that it requires manual intervention to resolve the
* problem.

* Some problems are best solved at the node where the problem was detected, and
* others are best solved where the message originated. We do not have the message
* origin, but we can use the REPLYQM to identify a node that has some interest
* in this message. Attempt to put the message onto a manual intervention queue
* at the appropriate node. If this fails, put the message on the manual
* intervention queue at this node.

REPLYQM('?*') +
  ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION) FWDQM(&REPLYQM)

ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION)
```

The BSDS conversion utility (CSQJUCNV) on z/OS

z/OS

You can use the CSQJUCNV BSDS conversion utility to convert a version 1 bootstrap data set (BSDS) to version 2. CSQJUCNV runs as a batch job.

A version 1 BSDS supports 6 byte log RBA (Relative Byte Address) values. A version 2 BSDS can be used by queue managers running IBM MQ Version 8.0, and supports 8 byte log RBA values. For more information about the change from 6 byte to 8 byte log RBA, see Larger log Relative Byte Address.

CD A version 2 BSDS can be used only by queue managers that have Version 8.0 new functions enabled with OPMODE. If the queue manager is in a queue-sharing group, all queue managers in the queue-sharing group must either have been started with OPMODE=(NEWFUNC,800), or OPMODE=(NEWFUNC,900), or added into the queue-sharing group at Version 8.0 or Version 9.0, before the BSDS can be converted to version 2.

If the parameters provided specify that the queue manager is in a queue-sharing group, the utility checks that the queue managers are at the correct level, before allowing the conversion of the BSDS to proceed.

For queue managers that are not in a queue-sharing group,the utility does not check that the queue manager has been started in Version 8.0 new function mode.

The converted BSDSs are written to new data sets. These new data sets must be allocated with similar attributes to the current BSDS before the utility is run, and must be empty. A version 2 BSDS contains more data than a version 1 BSDS, therefore, you must ensure that the new data sets are allocated with sufficient available space. See Planning your logging environment, and the associated topics, for the recommended values when defining a new BSDS.

The current BSDSs are not modified and can be used to start the queue manager, should the attempt to convert the BSDSs and restart the queue manager with the new BSDS be unsuccessful.

Important:

1. Only run this utility when the queue manager that owns the BSDS is stopped.
2. Do not attempt to start the queue manager with the new BSDS until the utility has completed successfully. If a queue manager is started with a BSDS that is the output of an unsuccessful or incomplete conversion, it terminates with reason code 00D10121.
3. To use this utility, your user ID of the job must have read and write access to both the old and new BSDSs.

- "Invoking the CSQJUCNV utility"
- "Syntax, keywords, and parameters"
- "Data definition (DD) statements" on page 1972

Invoking the CSQJUCNV utility

The utility runs as a z/OS batch program. Figure 1 shows an example of the JCL used to invoke the CSQJUCNV utility for a queue manager that is a member of a queue-sharing group.

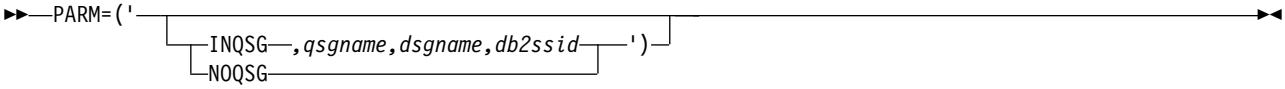
Sample JCL to run the utility is also provided in `thlqual.SCSQPROC(CSQ4BCNV)`.

```
//CONVERT EXEC PGM=CSQJUCNV,REGION=32M,
//            PARM=('INQSG,qsgname,dsgname,db2ssid')
//STEPLIB  DD DSN=thlqual.SCSQAUTH,DISP=SHR
//         DD DSN=thlqual.SCSQANLE,DISP=SHR
//         DD DSN=db2qual.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=hlq.BSDS01,DISP=SHR
//SYSUT2   DD DSN=hlq.BSDS02,DISP=SHR
//SYSUT3   DD DSN=newhlq.BSDS01,DISP=OLD
//SYSUT4   DD DSN=newhlq.BSDS02,DISP=OLD
```

Figure 50. Sample JCL to invoke the CSQJUCNV utility

Syntax, keywords, and parameters

BSDS conversion utility



PARM

This field must contain one of the following parameters to indicate whether the queue manager is a member of a queue-sharing group or not, followed by any function-specific parameters described in the following text:

INQSG

CD The queue manager that owns the BSDS is a member of a queue-sharing group. Specifying this parameter causes the utility to verify that all members of the queue-sharing group have either been started with OPMODE=(NEWFUNC,800),or OPMODE=(NEWFUNC,900), or been added into the queue-sharing group at Version 8.0 or Version 9.0.

See Implementing the larger log Relative Byte Address for details on how you perform this task.

The utility terminates with a nonzero reason code, without writing anything to the output BSDS, if this condition is not met.

qsgname
The queue-sharing group name

dsgname
The Db2 data-sharing group name

db2ssid
The Db2 subsystem ID

NOQSG

CD The queue manager that owns the BSDS is not a member of a queue-sharing group. The BSDS is converted, regardless of whether the queue manager has been started with Version 8.0 new functions enabled.

Attention: Do not specify this parameter for a queue manager that is a member of a queue-sharing group.

Data definition (DD) statements

CSQJUCNV recognizes DD statements with the following DD names:

SYSUT1
Specifies the old BSDS that is to be converted. This statement is required.

SYSUT2
Specifies the second copy of the old BSDS that is to be converted. If you are using dual BSDS, you should specify this.

SYSUT3
Specifies the new, converted BSDS. This statement is required.

SYSUT4
Specifies the second copy of the converted BSDS. This statement is required if the installation uses dual BSDSs; otherwise, it is optional.

SYSPRINT
Contains the output messages from the conversion utility. This statement is required.

The message security policy utility (CSQ0UTIL)

z/OS

The Advanced Message Security policy utility is provided to manage security policies that specify the cryptographic encryption and signature algorithms for encrypting and authenticating messages that flow through queues.

Using this utility program, you can display, define, alter, delete and export security policies.

The CSQ0UTIL utility program runs as a z/OS batch utility that accepts **SYSIN** command input. Sample JCL to run the utility is provided in member CSQ40CFG of thlqual.SCSQPROC.

```
------------------------------------------------------------------------------
//CSQ40CFG JOB 1,CSQ0,CLASS=A,MSGCLASS=X
//CSQ40CFG EXEC PGM=CSQ0UTIL,
//              PARM='ENVAR("_CEE_ENVFILE_S=DD:ENVARS") /'
//STEPLIB  DD DSN=thlqual.SCSQANLE,DISP=SHR
//         DD DSN=thlqual.SCSQAUTH,DISP=SHR
//ENVARS   DD DSN=thlqual.SCSQPROC(CSQ40ENV),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
dspmqspl -m qmgr
/*
------------------------------------------------------------------------------
```

The utility accepts the following commands:

dspmqspl
: Display or export information about one or more security policies.

setmqspl
: Define, alter or remove a security policy

For information on how to use these commands to manage security policies see Managing security policies.

General usage notes

When specifying distinguished names (DNs) that have embedded blanks, you must enclose the entire DN in double quotes ("). For example:

```
-a "CN=John Smith,O=IBM,C=US"
-r "CN=JSmith,O=IBM Australia,C=AU"
```

Arguments that would exceed column 80 of a SYSIN input record can be continued on subsequent SYSIN records provided those arguments are enclosed in double quotes ("), and relevant continuations resume in column 1 of subsequent SYSIN records.

When exporting policy information using **dspmqspl** with the -export parameter the output is written to an additional DD named EXPORT. The EXPORT DD can be SYSOUT=*, a sequential data set, or the member of a partitioned data set. The record format is fixed block and the logical record length is 80. The output is in the form of one or more **setmqspl** commands that can subsequently be used as input to CSQ0UTIL.

To use this utility you need connect authority to the queue manager and access to the queue SYSTEM.PROTECTION.POLICY.QUEUE. If command events have been enabled for the queue manager you need put authority to the queue SYSTEM.ADMIN.COMMAND.EVENT. If configuration events have been enabled for the queue manager you need put authority to the queue SYSTEM.ADMIN.CONFIG.EVENT.

Related information:
Security policies

dspmqspl (display security policy):

Use the **dspmqspl** command to display a list of all policies and details of a named policy.

Syntax

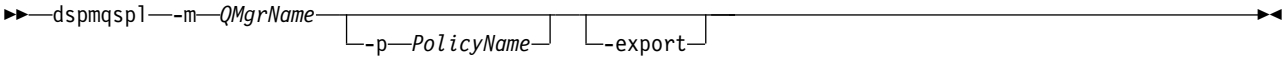


Table 157. dspmqspl command flags

Command flag	Explanation
-m	Queue manager name (mandatory).
-p	Policy name.
-export The output is written to a DD named EXPORT	Adding this flag generates output which can easily be applied to a different queue manager.

V 9.0.0

Examples

The **dspmqspl** command shows the key reuse count for all policies. The following example is the output you receive on Multiplatforms:

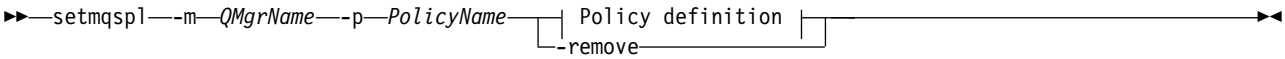
```
Policy Details:
Policy name: PROT
Quality of protection: PRIVACY
Signature algorithm: SHA256
Encryption algorithm: AES256
Signer DNs: -
Recipient DNs:
  CN=Name, O=Organization, C=Country
Toleration: 0
Key Reuse Count: 0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Policy Details:
Policy name: PROT2
Quality of protection: CONFIDENTIALITY
Signature algorithm: NONE
Encryption algorithm: AES256
Signer DNs: -
Recipient DNs:
  CN=Name, O=Organization, C=Country
Toleration: 0
Key Reuse Count: 100
```

z/OS On z/OS, you can use the **dspmqspl** command with the CSQ0UTIL utility. For more information, see The message security policy utility (CSQ0UTIL).

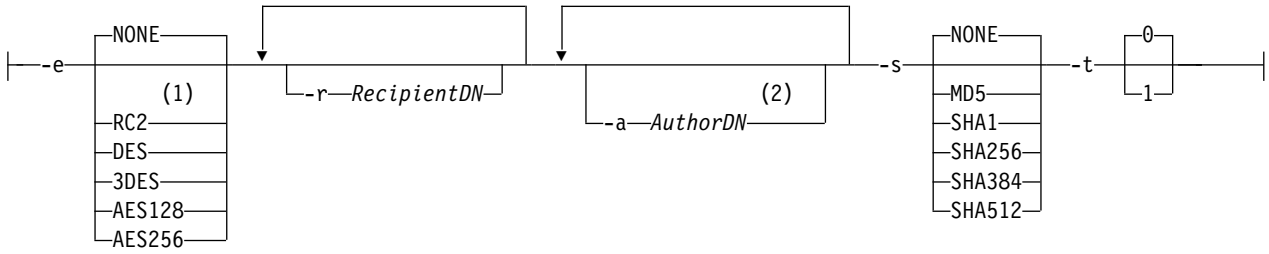
setmqspl (set security policy):

Use the **setmqspl** command to define a new security policy, alter an already existing one, or remove an existing policy.

Syntax



Policy definition:



Notes:

1 If an encryption algorithm is selected, a recipient DN must also be provided.

2 If an author DN is provided, a signing algorithm must also be selected.

Table 158. `setmqspl` command flags

Command flag	Explanation
-m	Queue manager name. This flag is mandatory for all actions on security policies.
-p	Policy name. Set the policy name to the name of the queue you want the policy to apply to.
-s	Digital signature algorithm. Advanced Message Security supports the following values: `MD5`, `SHA1`, `SHA256`, `SHA384`, and `SHA512`. All must be in uppercase. The default value is `NONE`. **Important:** • For the SHA384 and SHA512 cryptographic hash functions, keys used for signing must be longer than 768 bits. • The name of the signature algorithm must be provided in uppercase. • V 9.0.0 From Version 9.0, with the `Confidentiality` policy, the signature algorithm must be `NONE`. For more information about the `Confidentiality` policy, see Qualities of protection available with AMS.

Table 158. `setmqspl` command flags (continued)

Command flag	Explanation
-e	Digital encryption algorithm. Advanced Message Security supports the following encryption algorithms: `RC2`, `DES`, `3DES`, `AES128`, `AES256`. The default value is `NONE`. **Important:** The name of the encryption algorithm must be provided in uppercase
-r	The distinguished name (DN) of the message recipient (the certificate of a provided DN is used to encrypt a given message). Recipients can be specified only if the encryption algorithm is different from `NONE`. Multiple recipients can be included for a message. Each DN must be provided with a separate **-r** flag. **Important:** • DN attribute names must be in uppercase. • Commas must be used as a name separators. • To avoid command interpreter errors, place quotation marks around the DNs. For example: `-r "CN=alice, O=ibm, C=US"`
-a	Signature DN that is validated during message retrieval. Only messages signed by a user with a provided DN are accepted during the retrieval. Signature DNs can be specified only if the signature algorithm is different from `NONE`. Multiple authorized signers can be specified, each authorized signer needs to have a separate **-a** flag. **Important:** The attribute in the DN name must be in upper case. Specify `CN=` rather than `cn=`. The attribute values in the DN are case sensitive so, for example, `CN=USERID1` is different from `CN=userid1`.
-t	The toleration flag indicates whether messages that do not meet the requirements of the policy can still be successfully browsed or retrieved by an application. Toleration may be useful for example when introducing a policy to a queue which already contains unprotected messages. Valid values include: • **0 (default)** Toleration flag off. • **1** Toleration flag on. Toleration is optional and facilitates staged implementation, where policies were applied to queues but those queues may already contain messages that have no policy, or still receive messages from remote systems that do not have the security policy set.

Table 158. `setmqspl` command flags (continued)

Command flag	Explanation
V 9.0.0 **-c**	The key reuse count can be provided as an integer from 1 through 9,999,999. Special values are: • **0** Keys are not reused. • ***** Allows applications to reuse an encryption key an unlimited number of times. If you omit the **-c** parameter when defining a policy, a key reuse count of 0 is assumed for backwards compatibility with previous versions of Advanced Message Security and IBM WebSphere MQ Extended Security Edition. Note that a non-zero key reuse count is only valid for a confidentiality policy. If you attempt to create or modify an integrity or privacy policy, with a non-zero key reuse count, you receive error message `AMQ9091: Key reuse is not valid for policy` and the policy operation fails.
-remove	Delete policy. Only the policy name flag, **-p** is valid for use in combination with this flag.

V 9.0.0

Examples

The following list shows examples of some valid **setmqspl** commands on Multiplatforms:

```
setmqspl -m QMGR -p PROT -s SHA256
setmqspl -m QMGR -p PROT -s SHA256 -a "CN=Alice, O=IBM, C=US"
setmqspl -m QMGR -p PROT -s SHA256 -e AES128 -a "CN=Alice, O=IBM, C=US" -r "CN=Bob, O=IBM, C=GB"
setmqspl -m QMGR -p PROT -e AES128 -r "CN=Bob, O=IBM, C=GB" -c 50
```

The following list shows examples of **setmqspl** commands that are not valid:

- No recipients specified:

```
setmqspl -m QMGR -p PROT -e AES128
```

- Key reuse not valid for an `Integrity` policy:

```
setmqspl -m QMGR -p PROT -s SHA256 -c 1
```

- Key reuse is not valid for a `Privacy` policy:

```
setmqspl -m QMGR -p PROT -s SHA256 -e AES128 -r "CN=Bob, O=IBM, C=GB" -c 1
```

z/OS On z/OS, you can use the **setmqspl** command with the CSQ0UTIL utility. For more information, see The message security policy utility (CSQ0UTIL).

Display queue manager information utility (CSQUDSPM)

z/OS V 9.0.1

CSQUDSPM displays information about queue managers and provides the equivalent function to **dspmq** on Multiplatforms.

Purpose

Use the CSQUDSPM utility to list all IBM MQ subsystems on the LPAR regardless of what version of IBM MQ they are associated with. This is done by walking through the SSCT (Sub System Communications Table) looking for IBM MQ sub-systems.

Sample JCL, CSQ4DSPM, is provided for this purpose; the JCL resides in the SCSQPROC data set.

Packaging

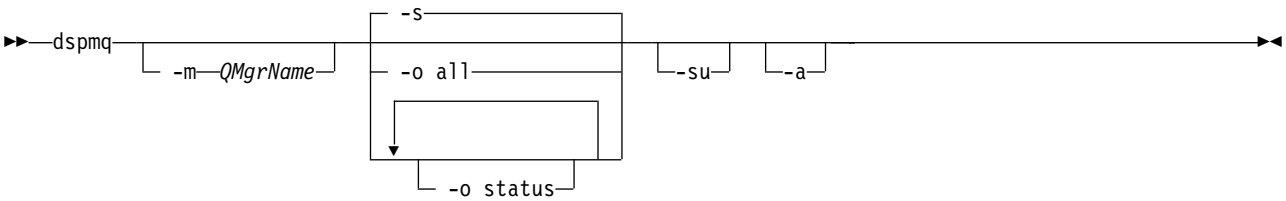
The CSQUDSPM load module is provided in the SCSQAUTH data set with an alias called DSPMQ.

If you need to run CSQUDSPM from USS, you can achieve this by creating an empty file with the same name, and enabling the sticky bit. As long as SCSQAUTH is in the STEPLIB concatenation of the caller, CSQUDSPM will be invoked.

To enable the sticky bit, you can use the following USS commands. In this case the DSPMQ alias is being used.

```
touch dspmq
chmod 755 dspmq
chmod +t dspmq
```

Syntax



Required parameters

None

Optional parameters

-a Displays information about running queue managers only.

-m *QMgrName*
The queue manager for which to display details. If you do not specify a name, all queue managers on the LPAR are displayed.

-s The operational status of the queue managers is displayed. This parameter is the default status setting.

The parameter **-o status** is equivalent to **-s**.

-o all
All details about the queue manager, or queue managers, are displayed.

-o status
The operational status of the queue managers is displayed.

-su
Suppress information about queue managers whose version is unknown.

An unknown version displays an INSTVER `V.R.M` of `0.0.0`.

Command output

Output name	Details
QMNAME	The name of the queue manager consisting of up to four characters. If the queue manager name is less than four characters the string is not padded. This parameter is always output. Examples: QMNAME(MQ21), QMNAME(MQ1)
STATUS	The status of the queue manager. Either `Running` or `Stopped`. This parameter is always output. Examples: STATUS(Running), STATUS(Stopped)
INSTVER	The version that the queue manager was last started up with, in the format `V.R.M`. **Note:** In the case of a queue manager that has not been started since the last IPL of the LPAR, the version of that queue manager cannot be obtained. In that situation, the INSTVER attribute displays a `V.R.M` of `0.0.0`. Examples: INSTVER(8.0.0), INSTVER(9.0.1)
ERLYVER	The version of early code associated with the queue manager. This should be the same for all queue managers in the LPAR in the format `V.R.M`. Examples: ERLYVER(9.0.1)
CMDPFX	The command prefix for the queue manager subsystem. This can be from one to eight characters long, and is not padded. Examples: CMDPFX(!MQ21), CMDPFX(MQ90ATST)
V 9.0.2 QSGNAME	The name of the queue-sharing group, that the queue manager is a member of, consisting of up to four characters. If the queue manager name is less than four characters the string is not padded. This parameter is always output. If the queue manager is not a member of a queue-sharing group then QSGNAME() is displayed. QSGNAME information can only be obtained when the queue manager is running, that is, STATUS(Running). If the queue manager is stopped QSGNAME(Unknown) is displayed. Example: QSGNAME(QSG1)

Examples

1. **Input:**

```
dspmq
```

Output:

```
QMNAME(QM01) STATUS(Stopped)
QMNAME(QM02) STATUS(Running)
QMNAME(QM03) STATUS(Stopped)
QMNAME(QM04) STATUS(Running)
```

2. V 9.0.2 **Input:**

```
dspmq -o all
```

Output:

```
QMNAME(QM01) STATUS(Stopped) INSTVER(0.0.0) ERLYVER(9.0.1) CMDPFX(!QM01) QSGNAME(Unknown)
QMNAME(QM02) STATUS(Running) INSTVER(9.0.1) ERLYVER(9.0.1) CMDPFX(!QM02) QSGNAME(QSG1)
QMNAME(QM03) STATUS(Stopped) INSTVER(9.0.1) ERLYVER(9.0.1) CMDPFX(!QM03) QSGNAME(Unknown)
QMNAME(QM04) STATUS(Running) INSTVER(9.0.1) ERLYVER(9.0.1) CMDPFX(!QM04) QSGNAME()
```

3. V 9.0.2 **Input:**

```
dspmq -o all -su
```

Output:

```
QMNAME(QM02) STATUS(Running) INSTVER(9.0.1) ERLYVER(9.0.1) CMDPFX(!QM02) QSGNAME(QSG1)
QMNAME(QM03) STATUS(Stopped) INSTVER(9.0.1) ERLYVER(9.0.1) CMDPFX(!QM03) QSGNAME(Unknown)
QMNAME(QM04) STATUS(Running) INSTVER(9.0.1) ERLYVER(9.0.1) CMDPFX(!QM04) QSGNAME()
```

Developing applications reference

Use the links provided in this section to help you develop your IBM MQ applications.

- "MQI applications reference" on page 1981
- "User exits, API exits, and installable services reference" on page 3535
- IBM i "IBM i Application Programming Reference (ILE/RPG)" on page 2999
- "SOAP reference" on page 3485
- "Reference material for IBM MQ bridge for HTTP" on page 3823
- "The IBM MQ .NET classes and interfaces" on page 3859
- "IBM MQ C++ classes" on page 3922
- The IBM MQ classes for Java libraries
- IBM MQ classes for JMS

Related information:

Developing applications

MQI applications reference

Use the links provided in this section to help you develop your Message Queue Interface (MQI) applications.

- "Code examples"
- "Constants" on page 2048
- "Data types used in the MQI" on page 2196
- "Function calls" on page 2620
- "Attributes of objects" on page 2792
- "Return codes" on page 2876
- "Rules for validating MQI options" on page 2877
- "Machine encodings" on page 2902
- "Report options and message flags" on page 2905
- "Data conversion" on page 2909
- "Properties specified as MQRFH2 elements" on page 2932
- "Code page conversion" on page 2941

Related concepts:

"User exits, API exits, and installable services reference" on page 3535
Use the links provided in this section to help you develop your User exits, API exits, and installable services applications:

Related reference:

"SOAP reference" on page 3485
IBM MQ transport for SOAP reference information arranged alphabetically.

"Reference material for IBM MQ bridge for HTTP" on page 3823
Reference topics for IBM MQ bridge for HTTP, arranged alphabetically

"The IBM MQ .NET classes and interfaces" on page 3859
IBM MQ .NET classes and interfaces are listed alphabetically. The properties, methods and constructors are described.

"IBM MQ C++ classes" on page 3922
The IBM MQ C++ classes encapsulate the IBM MQ Message Queue Interface (MQI). There is a single C++ header file, **imqi.hpp**, which covers all of these classes.

Related information:

Developing applications

The IBM MQ Classes for Java libraries

../com.ibm.mq.javadoc.doc/WMQJMSClasses/index.html

Code examples

Use the reference information in this section to accomplish the tasks that address your business needs.

C language examples:

This collection of topics is mostly taken from the IBM MQ for z/OS sample applications. They are applicable to all platforms, except where noted.

Connecting to a queue manager:

This example demonstrates how to use the MQCONN call to connect a program to a queue manager in z/OS batch.

This extract is taken from the Browse sample application (program CSQ4BCA1) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
#include <cmqc.h>
.
.
.
static char Parm1[MQ_Q_MGR_NAME_LENGTH] ;

int  main(int argc, char *argv[] )
   {
   /*                                                  */
   /*     Variables for MQ calls                       */
   /*                                                  */
   MQHCONN Hconn;      /* Connection handle            */
   MQLONG  CompCode;   /* Completion code              */
   MQLONG  Reason;     /* Qualifying reason            */

   /* Copy the queue manager name, passed in the       */
   /* parm field, to Parm1                             */
   strncpy(Parm1,argv[1],MQ_Q_MGR_NAME_LENGTH);

   /*                                                  */
   /* Connect to the specified queue manager.          */
   /*   Test the output of the connect call.  If the   */
   /*   call fails, print an error message showing the */
   /*   completion code and reason code, then leave the*/
   /*   program.                                       */
   /*                                                  */
   MQCONN(Parm1,
          &Hconn,
          &CompCode,
          &Reason);
   if ((CompCode != MQCC_OK) | (Reason != MQRC_NONE))
      {
      sprintf(pBuff, MESSAGE_4_E,
              ERROR_IN_MQCONN, CompCode, Reason);
      PrintLine(pBuff);
      RetCode = CSQ4_ERROR;
      goto AbnormalExit2;
      }

.
.
.
}
```

Disconnecting from a queue manager:

This example demonstrates how to use the MQDISC call to disconnect a program from a queue manager in z/OS batch.

The variables used in this code extract are those that were set in "Connecting to a queue manager" on page 1982. This extract is taken from the Browse sample application (program CSQ4BCA1) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
⋮
   /*                                                  */
   /*   Disconnect from the queue manager.  Test the   */
   /*   output of the disconnect call.  If the call    */
   /*   fails, print an error message showing the      */
   /*   completion code and reason code.               */
   /*                                                  */
   MQDISC(&Hconn,
          &CompCode,
          &Reason);
   if ((CompCode != MQCC_OK) || (Reason != MQRC_NONE))
      {
      sprintf(pBuff, MESSAGE_4_E,
              ERROR_IN_MQDISC, CompCode, Reason);
      PrintLine(pBuff);
      RetCode = CSQ4_ERROR;
      }
⋮
```

Creating a dynamic queue:

This example demonstrates how to use the MQOPEN call to create a dynamic queue.

This extract is taken from the Mail Manager sample application (program CSQ4TCD1) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
⋮
MQLONG  HCONN = 0;   /* Connection handle      */
MQHOBJ  HOBJ;        /* MailQ Object handle    */
MQHOBJ  HobjTempQ;   /* TempQ Object Handle    */
MQLONG  CompCode;    /* Completion code        */
MQLONG  Reason;      /* Qualifying reason      */
MQOD    ObjDesc = {MQOD_DEFAULT};
                     /* Object descriptor      */
MQLONG  OpenOptions; /* Options control MQOPEN */

   /*---------------------------------------- */
   /* Initialize the Object Descriptor (MQOD)  */
   /* control block.  (The remaining fields    */
   /* are already initialized.)                */
   /*----------------------------------------*/
   strncpy( ObjDesc.ObjectName,
            SYSTEM_REPLY_MODEL,
            MQ_Q_NAME_LENGTH );
   strncpy( ObjDesc.DynamicQName,
            SYSTEM_REPLY_INITIAL,
            MQ_Q_NAME_LENGTH );
   OpenOptions = MQOO_INPUT_AS_Q_DEF;
   /*----------------------------------------*/
   /* Open the model queue and, therefore,     */
   /* create and open a temporary dynamic      */
   /* queue                                    */
   /*----------------------------------------*/
   MQOPEN( HCONN,
```

```
            &ObjDesc,
            OpenOptions,
            &HobjTempQ,
            &CompCode,
            &Reason );
   if ( CompCode == MQCC_OK ) {

   }
   else {
      /*---------------------------------------*/
      /* Build an error message to report the  */
      /* failure of the opening of the model   */
      /* queue                                 */
      /*---------------------------------------*/
      MQMErrorHandling( "OPEN TEMPQ", CompCode,
                        Reason );
      ErrorFound = TRUE;
   }
   return ErrorFound;
}
.
.
.
```

Opening an existing queue:

This example demonstrates how to use the MQOPEN call to open a queue that has already been defined.

This extract is taken from the Browse sample application (program CSQ4BCA1) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
#include <cmqc.h>
.
.
.
static char Parm1[MQ_Q_MGR_NAME_LENGTH];
.
.
.
int  main(int argc, char *argv[] )
   {
   /*
   /*     Variables for MQ calls                          */
   /*
   MQHCONN Hconn ;            /* Connection handle       */
   MQLONG  CompCode;          /* Completion code         */
   MQLONG  Reason;            /* Qualifying reason       */
   MQOD    ObjDesc = { MQOD_DEFAULT };
                              /* Object descriptor       */
   MQLONG  OpenOptions;       /* Options that control    */
                              /*  the MQOPEN call        */
   MQHOBJ  Hobj;              /* Object handle           */

.
.
.
   /*  Copy the queue name, passed in the parm field,    */
   /*  to Parm2 strncpy(Parm2,argv[2],         */
   /*  MQ_Q_NAME_LENGTH);                                */

.
.
.
   /*                                                    */
   /*  Initialize the object descriptor (MQOD) control   */
   /*  block.  (The initialization default sets StrucId, */
   /*  Version, ObjectType, ObjectQMgrName,              */
   /*  DynamicQName, and AlternateUserid fields)         */
   /*                                                    */
   strncpy(ObjDesc.ObjectName,Parm2,MQ_Q_NAME_LENGTH);

.
.
.
   /*  Initialize the other fields required for the open */
   /*  call (Hobj is set by the MQCONN call).            */
```

```
   /*                                                          */
   OpenOptions  = MQOO_BROWSE;

.
.
.
   /*                                                          */
   /* Open the queue.                                          */
   /*   Test the output of the open call.  If the call         */
   /*   fails, print an error message showing the              */
   /*   completion code and reason code, then bypass           */
   /*   processing, disconnect and leave the program.          */
   /*                                                          */
   MQOPEN(Hconn,
          &ObjDesc,
          OpenOptions,
          &Hobj,
          &CompCode,
          &Reason);
   if ((CompCode != MQCC_OK) || (Reason != MQRC_NONE))
      {
      sprintf(pBuff, MESSAGE_4_E,
              ERROR_IN_MQOPEN, CompCode, Reason);
      PrintLine(pBuff);
      RetCode = CSQ4_ERROR;
      goto AbnormalExit1;       /* disconnect processing */
      }

.
.
.
   } /* end of main */
```

Closing a queue:

This example demonstrates how to use the MQCLOSE call to close a queue.

This extract is taken from the Browse sample application (program CSQ4BCA1) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
.
.
.
   /*                                                          */
   /*   Close the queue.                                       */
   /*   Test the output of the close call.  If the call        */
   /*   fails, print an error message showing the              */
   /*   completion code and reason code.                       */
   /*                                                          */
   MQCLOSE(Hconn,
           &Hobj,
           MQCO_NONE,
           &CompCode,
           &Reason);
   if ((CompCode != MQCC_OK) || (Reason != MQRC_NONE))
      {
      sprintf(pBuff, MESSAGE_4_E,
              ERROR_IN_MQCLOSE, CompCode, Reason);
      PrintLine(pBuff);
      RetCode = CSQ4_ERROR;
      }
.
.
.
```

Putting a message using MQPUT:

This example demonstrates how to use the MQPUT call to put a message on a queue.

This extract is not taken from the sample applications supplied with IBM MQ. For the names and locations of the sample applications, see Sample procedural programs (platforms except z/OS) z/OS and Sample programs for IBM MQ for z/OS .

```
.
.
.
qput()
{
     MQMD    MsgDesc;
     MQPMO   PutMsgOpts;
     MQLONG  CompCode;
     MQLONG  Reason;
     MQHCONN Hconn;
     MQHOBJ  Hobj;
     char message_buffer[] = "MY MESSAGE";
    /*-------------------------------*/
    /*  Set up PMO structure.        */
    /*-------------------------------*/
     memset(&PutMsgOpts, '\0', sizeof(PutMsgOpts));
     memcpy(PutMsgOpts.StrucId, MQPMO_STRUC_ID,
            sizeof(PutMsgOpts.StrucId));
     PutMsgOpts.Version = MQPMO_VERSION_1;
     PutMsgOpts.Options = MQPMO_SYNCPOINT;

    /*-------------------------------*/
    /*  Set up MD structure.         */
    /*-------------------------------*/
    memset(&MsgDesc, '\0', sizeof(MsgDesc));
    memcpy(MsgDesc.StrucId, MQMD_STRUC_ID,
           sizeof(MsgDesc.StrucId));
    MsgDesc.Version     =  MQMD_VERSION_1;
    MsgDesc.Expiry      =  MQEI_UNLIMITED;
    MsgDesc.Report      =  MQRO_NONE;
    MsgDesc.MsgType     =  MQMT_DATAGRAM;
    MsgDesc.Priority    = 1;
    MsgDesc.Persistence = MQPER_PERSISTENT;
    memset(MsgDesc.ReplyToQ,
           '\0',
           sizeof(MsgDesc.ReplyToQ));
    /*-------------------------------------------------*/
    /*  Put the message.                               */
    /*-------------------------------------------------*/
    MQPUT(Hconn, Hobj, &MsgDesc, &PutMsgOpts,
          sizeof(message_buffer), message_buffer,
          &CompCode, &Reason);
    /*-----------------------------------*/
    /*  Check completion and reason codes. */
    /*-----------------------------------*/
    switch (CompCode)
        {
         case MQCC_OK:
             break;
         case MQCC_FAILED:
             switch (Reason)
                 {
                  case MQRC_Q_FULL:
                  case MQRC_MSG_TOO_BIG_FOR_Q:
                      break;
                  default:
                      break; /* Perform error processing */
                  }
             break;
```

```
         default:
             break;          /* Perform error processing */
        }
}
```

Putting a message using MQPUT1:

This example demonstrates how to use the MQPUT1 call to open a queue, put a single message on the queue, then close the queue.

This extract is taken from the Credit Check sample application (program CSQ4CCB5) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
.
.
.
MQLONG   Hconn;              /* Connection handle        */
MQHOBJ   Hobj_CheckQ;        /* Object handle            */
MQLONG   CompCode;           /* Completion code          */
MQLONG   Reason;             /* Qualifying reason        */
MQOD     ObjDesc    = {MQOD_DEFAULT};
                             /* Object descriptor        */
MQMD     MsgDesc    = {MQMD_DEFAULT};
                             /* Message descriptor       */
MQLONG   OpenOptions;        /* Control the MQOPEN call  */

MQGMO    GetMsgOpts = {MQGMO_DEFAULT};
                             /* Get Message Options      */
MQLONG   MsgBuffLen;         /* Length of message buffer */
CSQ4BCAQ MsgBuffer;          /* Message structure        */
MQLONG   DataLen;            /* Length of message        */

MQPMO    PutMsgOpts = {MQPMO_DEFAULT};
                             /* Put Message Options      */
CSQ4BQRM PutBuffer;          /* Message structure        */
MQLONG   PutBuffLen = sizeof(PutBuffer);
                             /* Length of message buffer */
.
.
.
void Process_Query(void)
   {
   /*                                                     */
   /* Build the reply message                             */
   /*                                                     */

.
.
.
   /*                                                     */
   /* Set the object descriptor, message descriptor and   */
   /* put message options to the values required to       */
   /* create the reply message.                           */
   /*                                                     */
   strncpy(ObjDesc.ObjectName, MsgDesc.ReplyToQ,
           MQ_Q_NAME_LENGTH);
   strncpy(ObjDesc.ObjectQMgrName, MsgDesc.ReplyToQMgr,
           MQ_Q_MGR_NAME_LENGTH);
   MsgDesc.MsgType = MQMT_REPLY;
   MsgDesc.Report  = MQRO_NONE;
   memset(MsgDesc.ReplyToQ, ' ', MQ_Q_NAME_LENGTH);
   memset(MsgDesc.ReplyToQMgr, ' ', MQ_Q_MGR_NAME_LENGTH);
   memcpy(MsgDesc.MsgId, MQMI_NONE, sizeof(MsgDesc.MsgId));
   PutMsgOpts.Options = MQPMO_SYNCPOINT +
                        MQPMO_PASS_IDENTITY_CONTEXT;
   PutMsgOpts.Context = Hobj_CheckQ;
   PutBuffLen = sizeof(PutBuffer);
   MQPUT1(Hconn,
          &ObjDesc,
          &MsgDesc,
```

```
        &PutMsgOpts,
        PutBuffLen,
        &PutBuffer,
        &CompCode,
        &Reason);

   if (CompCode != MQCC_OK)
      {
      strncpy(TS_Operation, "MQPUT1",
              sizeof(TS_Operation));
      strncpy(TS_ObjName, ObjDesc.ObjectName,
              MQ_Q_NAME_LENGTH);
      Record_Call_Error();
      Forward_Msg_To_DLQ();
      }
   return;
   }

.
.
.
```

Getting a message:

This example demonstrates how to use the MQGET call to remove a message from a queue.

This extract is taken from the Browse sample application (program CSQ4BCA1) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
#include "cmqc.h"
.
.
.
#define BUFFERLENGTH 80
.
.
.
int main(int argc, char *argv[] )
   {
   /*                                                  */
   /*     Variables for MQ calls                       */
   /*                                                  */
   MQHCONN Hconn ;            /* Connection handle      */
   MQLONG  CompCode;          /* Completion code        */
   MQLONG  Reason;            /* Qualifying reason      */
   MQHOBJ  Hobj;              /* Object handle          */
   MQMD    MsgDesc  = { MQMD_DEFAULT };
                              /* Message descriptor     */
   MQLONG  DataLength ;       /* Length of the message  */
   MQCHAR  Buffer[BUFFERLENGTH+1];
                              /* Area for message data  */
   MQGMO   GetMsgOpts = { MQGMO_DEFAULT };
                              /* Options which control  */
                              /*  the MQGET call        */
   MQLONG  BufferLength = BUFFERLENGTH ;
                              /* Length of buffer       */

.
.
.
   /*     No need to change the message descriptor     */
   /*     (MQMD) control block because initialization  */
   /*     default sets all the fields.                 */
   /*                                                  */
   /*     Initialize the get message options (MQGMO)   */
   /*     control block (the copy file initializes all */
   /*     the other fields).                           */
   /*                                                  */
   GetMsgOpts.Options = MQGMO_NO_WAIT      +
                        MQGMO_BROWSE_FIRST +
                        MQGMO_ACCEPT_TRUNCATED_MSG;
   /*                                                  */
   /* Get the first message.                           */
```

```
   /*   Test for the output of the call is carried out   */
   /*   in the 'for' loop.                               */
   /*                                                    */
   MQGET(Hconn,
         Hobj,
         &MsgDesc,
         &GetMsgOpts,
         BufferLength,
         Buffer,
         &DataLength,
         &CompCode,
         &Reason);
   /*                                                    */
   /* Process the message and get the next message,      */
   /* until no messages remaining.                       */

.
.
.
   /*     If the call fails for any other reason,        */
   /*     print an error message showing the completion  */
   /*     code and reason code.                          */
   /*                                                    */
   if ( (CompCode == MQCC_FAILED) &&
        (Reason   == MQRC_NO_MSG_AVAILABLE) )
      {

.
.
.
      }
   else
      {
      sprintf(pBuff, MESSAGE_4_E,
              ERROR_IN_MQGET, CompCode, Reason);
      PrintLine(pBuff);
      RetCode = CSQ4_ERROR;
      }

.
.
.
   } /* end of main */
```

Getting a message using the wait option:

This example demonstrates how to use the wait option of the MQGET call.

This code accepts truncated messages. This extract is taken from the Credit Check sample application (program CSQ4CCB5) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
.
.
.
MQLONG   Hconn;              /* Connection handle        */
MQHOBJ   Hobj_CheckQ;        /* Object handle            */
MQLONG   CompCode;           /* Completion code          */
MQLONG   Reason;             /* Qualifying reason        */
MQOD     ObjDesc    = {MQOD_DEFAULT};
                             /* Object descriptor        */
MQMD     MsgDesc    = {MQMD_DEFAULT};
                             /* Message descriptor       */
MQLONG   OpenOptions;
                             /* Control the MQOPEN call  */
MQGMO    GetMsgOpts = {MQGMO_DEFAULT};
                             /* Get Message Options      */
MQLONG   MsgBuffLen;         /* Length of message buffer */
CSQ4BCAQ MsgBuffer;          /* Message structure        */
MQLONG   DataLen;            /* Length of message        */
.
.
.
void main(void)
```

```
   {
.
.
.
   /*                                                          */
   /* Initialize options and open the queue for input          */
   /*                                                          */
.
.
.
      /*                                                       */
      /* Get and process messages                              */
      /*                                                       */
      GetMsgOpts.Options = MQGMO_WAIT +
                           MQGMO_ACCEPT_TRUNCATED_MSG +
                           MQGMO_SYNCPOINT;
      GetMsgOpts.WaitInterval = WAIT_INTERVAL;
      MsgBuffLen = sizeof(MsgBuffer);
      memcpy(MsgDesc.MsgId, MQMI_NONE,
             sizeof(MsgDesc.MsgId));
      memcpy(MsgDesc.CorrelId, MQCI_NONE,
             sizeof(MsgDesc.CorrelId));
      /*                                                       */
      /* Make the first MQGET call outside the loop            */
      /*                                                       */
      MQGET(Hconn,
            Hobj_CheckQ,
            &MsgDesc,
            &GetMsgOpts,
            MsgBuffLen,
            &MsgBuffer,
            &DataLen,
            &CompCode,
            &Reason);
.
.
.
      /*                                                       */
      /* Test the output of the MQGET call.  If the call       */
      /* failed, send an error message showing the             */
      /* completion code and reason code, unless the           */
      /* reason code is NO_MSG AVAILABLE.                      */
      /*                                                       */
      if (Reason != MQRC_NO_MSG_AVAILABLE)
         {
         strncpy(TS_Operation, "MQGET", sizeof(TS_Operation));
         strncpy(TS_ObjName, ObjDesc.ObjectName,
                 MQ_Q_NAME_LENGTH);
         Record_Call_Error();
         }
.
.
.
```

Getting a message using signaling:

Signaling is available only with IBM MQ for z/OS .

This example demonstrates how to use the MQGET call to set a signal so that you are notified when a suitable message arrives on a queue. This extract is not taken from the sample applications supplied with IBM MQ.

```
⋮
get_set_signal()
{
     MQMD    MsgDesc;
     MQGMO   GetMsgOpts;
     MQLONG  CompCode;
     MQLONG  Reason;
     MQHCONN Hconn;
     MQHOBJ  Hobj;
     MQLONG  BufferLength;
     MQLONG  DataLength;
     char message_buffer[100];
     long  int q_ecb, work_ecb;
     short int signal_sw, endloop;
     long  int mask = 255;

     /*---------------------------*/
     /*  Set up GMO structure.    */
     /*---------------------------*/
     memset(&GetMsgOpts,'\0',sizeof(GetMsgOpts));
     memcpy(GetMsgOpts.StrucId, MQGMO_STRUC_ID,
            sizeof(GetMsgOpts.StrucId);
     GetMsgOpts.Version      = MQGMO_VERSION_1;
     GetMsgOpts.WaitInterval = 1000;
     GetMsgOpts.Options      = MQGMO_SET_SIGNAL +
                               MQGMO_BROWSE_FIRST;
     q_ecb              = 0;
     GetMsgOpts.Signal1      = &q_ecb;
     /*---------------------------*/
     /*  Set up MD structure.     */
     /*---------------------------*/
     memset(&MsgDesc,'\0',sizeof(MsgDesc));
     memcpy(MsgDesc.StrucId, MQMD_STRUC_ID,
            sizeof(MsgDesc.StrucId);
     MsgDesc.Version  = MQMD_VERSION_1;
     MsgDesc.Report   = MQRO_NONE;
     memcpy(MsgDesc.MsgId,MQMI_NONE,
            sizeof(MsgDesc.MsgId));
     memcpy(MsgDesc.CorrelId,MQCI_NONE,
            sizeof(MsgDesc.CorrelId));
    /*-------------------------------------------------*/
    /*  Issue the MQGET call.                          */
    /*-------------------------------------------------*/
     BufferLength = sizeof(message_buffer);
     signal_sw   = 0;

     MQGET(Hconn, Hobj, &MsgDesc, &GetMsgOpts,
           BufferLength, message_buffer, &DataLength,
           &CompCode, &Reason);
    /*-------------------------------------*/
    /*  Check completion and reason codes. */
    /*-------------------------------------*/
     switch (CompCode)
         {
          case (MQCC_OK):          /* Message retrieved  */
              break;
          case (MQCC_WARNING):
              switch (Reason)
```

```
          {
           case (MQRC_SIGNAL_REQUEST_ACCEPTED):
               signal_sw = 1;
               break;
           default:
               break;   /* Perform error processing  */
          }
          break;
      case (MQCC_FAILED):
          switch (Reason)
          {
           case (MQRC_Q_MGR_NOT_AVAILABLE):
           case (MQRC_CONNECTION_BROKEN):
           case (MQRC_Q_MGR_STOPPING):
               break;
           default:
               break;  /* Perform error processing.  */
          }
          break;
      default:
          break;       /* Perform error processing.  */
     }
/*-----------------------------------------------------*/
/* If the SET_SIGNAL was accepted, set up a loop to  */
/* check whether a message has arrived at one second */
/* intervals. The loop ends if a message arrives or  */
/* the wait interval specified in the MQGMO          */
/* structure has expired.                            */
/*                                                   */
/* If a message arrives on the queue, another MQGET  */
/* must be issued to retrieve the message. If other  */
/* MQM calls have been made in the intervening       */
/* period, this may necessitate reinitializing the   */
/* MQMD and MQGMO structures.                        */
/* In this code, no intervening calls                */
/* have been made, so the only change required to    */
/* the structures is to specify MQGMO_NO_WAIT,       */
/* since we now know the message is there.           */
/*                                                   */
/* This code uses the EXEC CICS DELAY command to     */
/* suspend the program for a second. A batch program */
/* may achieve the same effect by calling an         */
/* assembler language subroutine which issues a      */
/* z/OS STIMER macro.                                */
/*-----------------------------------------------------*/
if (signal_sw == 1)
   {
    endloop = 0;
    do
      {
       EXEC CICS DELAY FOR HOURS(0) MINUTES(0) SECONDS(1);
       work_ecb = q_ecb & mask;
       switch (work_ecb)
           {
            case (MQEC_MSG_ARRIVED):
                endloop = 1;
                mqgmo_options = MQGMO_NO_WAIT;
                MQGET(Hconn, Hobj, &MsgDesc, &GetMsgOpts,
                      BufferLength, message_buffer,
                      &DataLength, &CompCode, &Reason);
                if (CompCode != MQCC_OK)
                    ;          /* Perform error processing. */
                break;
            case (MQEC_WAIT_INTERVAL_EXPIRED):
            case (MQEC_WAIT_CANCELED):
                endloop = 1;
                break;
```

```
                default:
                    break;
                }
            } while (endloop == 0);
         }
      return;
}
```

Inquiring about the attributes of an object:

This example demonstrates how to use the MQINQ call to inquire about the attributes of a queue.

This extract is taken from the Queue Attributes sample application (program CSQ4CCC1) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
#include <cmqc.h>      /* MQ API header file       */
:
:
#define NUMBEROFSELECTORS  2

const MQHCONN Hconn = MQHC_DEF_HCONN;
:
:
static void InquireGetAndPut(char   *Message,
                             PMQHOBJ pHobj,
                             char   *Object)

   {
   /*      Declare local variables                        */
   /*                                                     */
   MQLONG  SelectorCount = NUMBEROFSELECTORS;
                                /* Number of selectors  */
   MQLONG  IntAttrCount  = NUMBEROFSELECTORS;
                                /* Number of int attrs  */
   MQLONG  CharAttrLength = 0;
                     /* Length of char attribute buffer  */
   MQCHAR *CharAttrs ;
                     /* Character attribute buffer       */
   MQLONG  SelectorsTable[NUMBEROFSELECTORS];
                                /* attribute selectors  */
   MQLONG  IntAttrsTable[NUMBEROFSELECTORS];
                                /* integer attributes   */
   MQLONG  CompCode;            /* Completion code      */
   MQLONG  Reason;              /* Qualifying reason    */
   /*                                                     */
   /*     Open the queue.  If successful, do the inquire */
   /*     call.                                          */
   /*                                                     */
      /*                                                  */
      /*   Initialize the variables for the inquire      */
      /*   call:                                         */
      /*    - Set SelectorsTable to the attributes whose */
      /*      status is                                   */
      /*       required                                   */
      /*    - All other variables are already set        */
      /*                                                  */
      SelectorsTable[0] = MQIA_INHIBIT_GET;
      SelectorsTable[1] = MQIA_INHIBIT_PUT;
      /*                                                  */
      /*   Issue the inquire call                         */
      /*     Test the output of the inquire call. If the */
      /*     call failed, display an error message       */
      /*     showing the completion code and reason code,*/
      /*     otherwise display the status of the         */
      /*     INHIBIT-GET and INHIBIT-PUT attributes      */
      /*                                                  */
      MQINQ(Hconn,
```

```
            *pHobj,
            SelectorCount,
            SelectorsTable,
            IntAttrCount,
            IntAttrsTable,
            CharAttrLength,
            CharAttrs,
            &CompCode,
            &Reason);
      if (CompCode != MQCC_OK)
         {
         sprintf(Message, MESSAGE_4_E,
                 ERROR_IN_MQINQ, CompCode, Reason);
         SetMsg(Message);
         }
      else
         {
           /* Process the changes */
         } /* end if CompCode */
```

Setting the attributes of a queue:

This example demonstrates how to use the MQSET call to change the attributes of a queue.

This extract is taken from the Queue Attributes sample application (program CSQ4CCC1) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
#include <cmqc.h>      /* MQ API header file      */
:
#define NUMBEROFSELECTORS  2

const MQHCONN Hconn = MQHC_DEF_HCONN;

static void InhibitGetAndPut(char   *Message,
                             PMQHOBJ pHobj,
                             char   *Object)
   {
   /*                                                  */
   /*      Declare local variables                     */
   /*                                                  */
   MQLONG  SelectorCount = NUMBEROFSELECTORS;
                               /* Number of selectors  */
   MQLONG  IntAttrCount  = NUMBEROFSELECTORS;
                               /* Number of int attrs  */
   MQLONG  CharAttrLength = 0;
                     /* Length of char attribute buffer  */
   MQCHAR *CharAttrs ;
                     /* Character attribute buffer       */
   MQLONG  SelectorsTable[NUMBEROFSELECTORS];
                               /* attribute selectors  */
   MQLONG  IntAttrsTable[NUMBEROFSELECTORS];
                               /* integer attributes   */
   MQLONG  CompCode;           /* Completion code      */
   MQLONG  Reason;             /* Qualifying reason    */

:
   /*                                                  */
   /*     Open the queue.  If successful, do the       */
   /*     inquire call.                                */
   /*                                                  */

:
      /*                                               */
      /*   Initialize the variables for the set call:  */
      /*    - Set SelectorsTable to the attributes to be */
```

```
      /*      set                                          */
      /*    - Set IntAttrsTable to the required status     */
      /*    - All other variables are already set          */
      /*                                                   */
      SelectorsTable[0] = MQIA_INHIBIT_GET;
      SelectorsTable[1] = MQIA_INHIBIT_PUT;
      IntAttrsTable[0]  = MQQA_GET_INHIBITED;
      IntAttrsTable[1]  = MQQA_PUT_INHIBITED;

⋮
      /*                                                   */
      /*   Issue the set call.                             */
      /*     Test the output of the set call.  If the      */
      /*     call fails, display an error message          */
      /*     showing the completion code and reason        */
      /*     code; otherwise move INHIBITED to the         */
      /*     relevant screen map fields                    */
      /*                                                   */
      MQSET(Hconn,
            *pHobj,
            SelectorCount,
            SelectorsTable,
            IntAttrCount,
            IntAttrsTable,
            CharAttrLength,
            CharAttrs,
            &CompCode,
            &Reason);
      if (CompCode != MQCC_OK)
         {
         sprintf(Message, MESSAGE_4_E,
                 ERROR_IN_MQSET, CompCode, Reason);
         SetMsg(Message);
         }
      else
         {
           /* Process the changes */
         } /* end if CompCode */
```

Retrieving status information with MQSTAT:

This example demonstrates how to issue an asynchronous MQPUT and retrieve the status information with MQSTAT.

This extract is taken from the Calling MQSTAT sample application (program amqsapt0) supplied with IBM MQ for Windows systems. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
/*********************************************************************/
/*                                                                   */
/* Program name: AMQSAPT0                                            */
/*                                                                   */
/* Description: Sample C program that asynchronously puts messages  */
/*              to a message queue (example using MQPUT & MQSTAT).   */
/*                                                                   */
/* Licensed Materials - Property of IBM                              */
/*                                                                   */
/* 63H9336                                                           */
/* (c) Copyright IBM Corp. 2006 All Rights Reserved.                 */
/*                                                                   */
/* US Government Users Restricted Rights - Use, duplication or       */
/* disclosure restricted by GSA ADP Schedule Contract with           */
/* IBM Corp.                                                         */
/*                                                                   */
/*********************************************************************/
/*                                                                   */
```

```
/* Function:                                                          */
/*                                                                    */
/*   AMQSAPT0 is a sample C program to put messages on a message      */
/*   queue with asynchronous response option, querying the success    */
/*   of the put operations with MQSTAT.                               */
/*                                                                    */
/*      -- messages are sent to the queue named by the parameter      */
/*                                                                    */
/*      -- gets lines from StdIn, and adds each to target             */
/*         queue, taking each line of text as the content             */
/*         of a datagram message; the sample stops when a null        */
/*         line (or EOF) is read.                                     */
/*         New-line characters are removed.                           */
/*         If a line is longer than 99 characters it is broken up     */
/*         into 99-character pieces. Each piece becomes the           */
/*         content of a datagram message.                             */
/*         If the length of a line is a multiple of 99 plus 1, for    */
/*         example, 199, the last piece will only contain a           */
/*         new-line character so will terminate the input.            */
/*                                                                    */
/*      -- writes a message for each MQI reason other than            */
/*         MQRC_NONE; stops if there is a MQI completion code         */
/*         of MQCC_FAILED                                             */
/*                                                                    */
/*      -- summarizes the overall success of the put operations       */
/*         through a call to MQSTAT to query MQSTAT_TYPE_ASYNC_ERROR*/
/*                                                                    */
/*    Program logic:                                                  */
/*         MQOPEN target queue for OUTPUT                             */
/*         while end of input file not reached,                       */
/*         .  read next line of text                                  */
/*         .  MQPUT datagram message with text line as data           */
/*         MQCLOSE target queue                                       */
/*         MQSTAT connection                                          */
/*                                                                    */
/*                                                                    */
/**********************************************************************/
/*                                                                    */
/*   AMQSAPT0 has the following parameters                            */
/*       required:                                                    */
/*                 (1) The name of the target queue                   */
/*       optional:                                                    */
/*                 (2) Queue manager name                             */
/*                 (3) The open options                               */
/*                 (4) The close options                              */
/*                 (5) The name of the target queue manager           */
/*                 (6) The name of the dynamic queue                  */
/*                                                                    */
/**********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
   /* includes for MQI */
#include <cmqc.h>

int main(int argc, char **argv)
{
  /*  Declare file and character for sample input                   */
  FILE *fp;

  /*   Declare MQI structures needed                                */
  MQOD     od = {MQOD_DEFAULT};    /* Object Descriptor             */
  MQMD     md = {MQMD_DEFAULT};    /* Message Descriptor            */
  MQPMO   pmo = {MQPMO_DEFAULT};   /* put message options           */
  MQSTS   sts = {MQSTS_DEFAULT};   /* status information            */
     /** note, sample uses defaults where it can **/
  MQHCONN  Hcon;                   /* connection handle             */
```

```
MQHOBJ   Hobj;                  /* object handle                */
MQLONG   O_options;             /* MQOPEN options               */
MQLONG   C_options;             /* MQCLOSE options              */
MQLONG   CompCode;              /* completion code              */
MQLONG   OpenCode;              /* MQOPEN completion code       */
MQLONG   Reason;                /* reason code                  */
MQLONG   CReason;               /* reason code for MQCONN       */
MQLONG   messlen;               /* message length               */
char     buffer[100];           /* message buffer               */
char     QMName[50];            /* queue manager name           */

printf("Sample AMQSAPT0 start\n");
if (argc < 2)
{
  printf("Required parameter missing - queue name\n");
  exit(99);
}

/******************************************************************/
/*                                                                */
/*   Connect to queue manager                                     */
/*                                                                */
/******************************************************************/
QMName[0] = 0;    /* default */
if (argc > 2)
  strcpy(QMName, argv[2]);
MQCONN(QMName,                  /* queue manager                */
       &Hcon,                   /* connection handle            */
       &Compcode,               /* completion code              */
       &Reason);                /* reason code                  */
/* report reason and stop if it failed     */
if (CompCode == MQCC_FAILED)
{
  printf("MQCONN ended with reason code %d\n", CReason);
  exit( (int)CReason );
}

/******************************************************************/
/*                                                                */
/*   Use parameter as the name of the target queue                */
/*                                                                */
/******************************************************************/
strncpy(od.ObjectName, argv[1], (size_t)MQ_Q_NAME_LENGTH);
printf("target queue is %s\n", od.ObjectName);

if (argc > 5)
{
  strncpy(od.ObjectQMgrName, argv[5], (size_t) MQ_Q_MGR_NAME_LENGTH);
  printf("target queue manager is %s\n", od.ObjectQMgrName);
}

if (argc > 6)
{
  strncpy(od.DynamicQName, argv[6], (size_t) MQ_Q_NAME_LENGTH);
  printf("dynamic queue name is %s\n", od.DynamicQName);
}

/******************************************************************/
/*                                                                */
/*   Open the target message queue for output                     */
/*                                                                */
/******************************************************************/
if (argc > 3)
{
  O_options = atoi( argv[3] );
  printf("open  options are %d\n", O_options);
}
```

```
else
{
  O_options = MQOO_OUTPUT            /* open queue for output     */
            | MQOO_FAIL_IF_QUIESCING /* but not if MQM stopping   */
            ;                        /* = 0x2010 = 8208 decimal   */
}

MQOPEN(Hcon,                      /* connection handle            */
       &od,                       /* object descriptor for queue  */
       O_options,                 /* open options                 */
       &Hobj,                     /* object handle                */
       &OpenCode,                 /* MQOPEN completion code       */
       &Reason);                  /* reason code                  */

/* report reason, if any; stop if failed      */
if (Reason != MQRC_NONE)
{
  printf("MQOPEN ended with reason code %d\n", Reason);
}

if (OpenCode == MQCC_FAILED)
{
  printf("unable to open queue for output\n");
}

/*********************************************************************/
/*                                                                   */
/*   Read lines from the file and put them to the message queue      */
/*   Loop until null line or end of file, or there is a failure      */
/*                                                                   */
/*********************************************************************/
CompCode = OpenCode;        /* use MQOPEN result for initial test */
fp = stdin;

memcpy(md.Format,           /* character string format            */
       MQFMT_STRING, (size_t)MQ_FORMAT_LENGTH);

/*********************************************************************/
/* These options specify that put operation should occur             */
/* asynchronously and the application will check the success         */
/* using MQSTAT at a later time.                                     */
/*********************************************************************/
md.Persistence = MQPER_NOT_PERSISTENT;
pmo.Options |= MQPMO_ASYNC_RESPONSE;

/*********************************************************************/
/* These options cause the MsgId and CorrelId to be replaced, so  */
/* that there is no need to reset them before each MQPUT             */
/*********************************************************************/
pmo.Options |= MQPMO_NEW_MSG_ID;
pmo.Options |= MQPMO_NEW_CORREL_ID;

while (CompCode != MQCC_FAILED)
{
  if (fgets(buffer, sizeof(buffer), fp) != NULL)
  {
    messlen = (MQLONG)strlen(buffer); /* length without null      */
    if (buffer[messlen-1] == '\n')  /* last char is a new-line    */
    {
      buffer[messlen-1]  = '\0';    /* replace new-line with null */
      --messlen;                    /* reduce buffer length       */
    }
  }
  else messlen = 0;        /* treat EOF same as null line         */

  /*******************************************************************/
  /*                                                                 */
```

```
  /*   Put each buffer to the message queue                       */
  /*                                                              */
  /****************************************************************/
  if (messlen > 0)
  {
    MQPUT(Hcon,                /* connection handle               */
          Hobj,                /* object handle                   */
          &md,                 /* message descriptor              */
          &pmo,                /* default options (datagram)      */
          messlen,             /* message length                  */
          buffer,              /* message buffer                  */
          &CompCode,           /* completion code                 */
          &Reason);            /* reason code                     */

    /* report reason, if any */
    if (Reason != MQRC_NONE)
    {
      printf("MQPUT ended with reason code %d\n", Reason);
    }
  }
  else   /* satisfy end condition when empty line is read */
    CompCode = MQCC_FAILED;
}

/****************************************************************/
/*                                                              */
/*   Close the target queue (if it was opened)                  */
/*                                                              */
/****************************************************************/
if (OpenCode != MQCC_FAILED)
{
  if (argc > 4)
  {
    C_options = atoi( argv[4] );
    printf("close options are %d\n", C_options);
  }
  else
  {
    C_options = MQCO_NONE;        /* no close options             */
  }

  MQCLOSE(Hcon,                   /* connection handle            */
          &Hobj,                  /* object handle                */
          C_options,
          &CompCode,              /* completion code              */
          &Reason);               /* reason code                  */

  /* report reason, if any     */
  if (Reason != MQRC_NONE)
  {
    printf("MQCLOSE ended with reason code %d\n", Reason);
  }
}

/****************************************************************/
/*                                                              */
/*   Query how many asynchronous puts succeeded                 */
/*                                                              */
/****************************************************************/
MQSTAT(&Hcon,                   /* connection handle            */
       MQSTAT_TYPE_ASYNC_ERROR, /* status type                  */
       &Sts,                    /* MQSTS structure              */
       &CompCode,               /* completion code              */
       &Reason);                /* reason code                  */

/* report reason, if any     */
if (Reason != MQRC_NONE)
```

```
  {
    printf("MQSTAT ended with reason code %d\n", Reason);
  }
  else
  {
    /* Display results */
    printf("Succeeded putting %d messages\n",
           sts.PutSuccessCount);
    printf("%d messages were put with a warning\n",
           sts.PutWarningCount);
    printf("Failed to put %d messages\n",
           sts.PutFailureCount);

    if(sts.CompCode == MQCC_WARNING)
    {
      printf("The first warning that occurred had reason code %d\n",
             sts.Reason);
    }
    else if(sts.CompCode == MQCC_FAILED)
    {
      printf("The first error that occurred had reason code %d\n",
             sts.Reason);
    }
  }

  /******************************************************************/
  /*                                                                */
  /*   Disconnect from MQM if not already connected                 */
  /*                                                                */
  /******************************************************************/
  if (CReason != MQRC_ALREADY_CONNECTED)
  {
    MQDISC(&Hcon,                   /* connection handle            */
           &CompCode,               /* completion code              */
           &Reason);                /* reason code                  */

    /* report reason, if any     */
    if (Reason != MQRC_NONE)
    {
      printf("MQDISC ended with reason code %d\n", Reason);
    }
  }

  /******************************************************************/
  /*                                                                */
  /* END OF AMQSAPT0                                                */
  /*                                                                */
  /******************************************************************/
  printf("Sample AMQSAPT0 end\n");
  return(0);
}
```

COBOL examples:

This collection of topics is taken from the IBM MQ for z/OS sample applications. They are applicable to all platforms, except where noted.

Connecting to a queue manager:

This example demonstrates how to use the MQCONN call to connect a program to a queue manager in z/OS batch.

This extract is taken from the Browse sample application (program CSQ4BVA1) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
* -------------------------------------------------------*
WORKING-STORAGE SECTION.
* -------------------------------------------------------*
*    W02 - Data fields derived from the PARM field
 01  W02-MQM                    PIC X(48) VALUE SPACES.
*    W03 - MQM API fields
 01  W03-HCONN                  PIC S9(9) BINARY.
 01  W03-COMPCODE               PIC S9(9) BINARY.
 01  W03-REASON                 PIC S9(9) BINARY.
*
*    MQV contains constants (for filling in the control
*    blocks)
*    and return codes (for testing the result of a call)
*
 01  W05-MQM-CONSTANTS.
 COPY CMQV SUPPRESS.

:
*    Separate into the relevant fields any data passed
*    in the PARM statement
*
     UNSTRING PARM-STRING DELIMITED BY ALL ','
                              INTO W02-MQM
                                   W02-OBJECT.

:
*    Connect to the specified queue manager.
*
     CALL 'MQCONN' USING W02-MQM
                         W03-HCONN
                         W03-COMPCODE
                         W03-REASON.
*
*    Test the output of the connect call.  If the call
*    fails, print an error message showing the
*    completion code and reason code.
*
     IF (W03-COMPCODE NOT = MQCC-OK) THEN

:
     END-IF.

:
```

Disconnecting from a queue manager:

This example demonstrates how to use the MQDISC call to disconnect a program from a queue manager in z/OS batch.

The variables used in this code extract are those that were set in "Connecting to a queue manager" on page 2001. This extract is taken from the Browse sample application (program CSQ4BVA1) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
⋮
*
* Disconnect from the queue manager
*
     CALL 'MQDISC' USING W03-HCONN
                         W03-COMPCODE
                         W03-REASON.
*
*    Test the output of the disconnect call.  If the
*    call fails, print an error message showing the
*    completion code and reason code.
*
     IF (W03-COMPCODE NOT = MQCC-OK) THEN
⋮
           END-IF.
⋮
```

Creating a dynamic queue:

This example demonstrates how to use the MQOPEN call to create a dynamic queue.

This extract is taken from the Credit Check sample application (program CSQ4CVB1) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
⋮
* -------------------------------------------------------*
WORKING-STORAGE SECTION.
* -------------------------------------------------------*
*
*    W02 - Queues processed in this program
*
 01  W02-MODEL-QNAME        PIC X(48) VALUE
     'CSQ4SAMP.B1.MODEL                          '.
 01  W02-NAME-PREFIX        PIC X(48) VALUE
     'CSQ4SAMP.B1.*                              '.
 01  W02-TEMPORARY-Q         PIC X(48).
*
*    W03 - MQM API fields
*
 01  W03-HCONN       PIC S9(9) BINARY VALUE ZERO.
 01  W03-OPTIONS     PIC S9(9) BINARY.
 01  W03-HOBJ        PIC S9(9) BINARY.
 01  W03-COMPCODE    PIC S9(9) BINARY.
 01  W03-REASON      PIC S9(9) BINARY.
*
*    API control blocks
*
 01  MQM-OBJECT-DESCRIPTOR.
     COPY CMQODV.
*
*    CMQV contains constants (for setting or testing
*    field values) and return codes (for testing the
*    result of a call)
```

```
*
 01  MQM-CONSTANTS.
 COPY CMQV SUPPRESS.
* ----------------------------------------------------------*
 PROCEDURE DIVISION.
* ----------------------------------------------------------*
:
* ----------------------------------------------------------*
 OPEN-TEMP-RESPONSE-QUEUE SECTION.
* ----------------------------------------------------------*
*
*  This section creates a temporary dynamic queue
*  using a model queue
*
* ----------------------------------------------------------*
*
* Change three fields in the Object Descriptor (MQOD)
* control block.  (MQODV initializes the other fields)
*
     MOVE MQOT-Q           TO MQOD-OBJECTTYPE.
     MOVE W02-MODEL-QNAME  TO MQOD-OBJECTNAME.
     MOVE W02-NAME-PREFIX  TO MQOD-DYNAMICQNAME.
*
     COMPUTE W03-OPTIONS = MQOO-INPUT-EXCLUSIVE.
*
     CALL 'MQOPEN' USING W03-HCONN
                         MQOD
                         W03-OPTIONS
                         W03-HOBJ-MODEL
                         W03-COMPCODE
                         W03-REASON.
*
     IF W03-COMPCODE NOT = MQCC-OK
          MOVE 'MQOPEN'      TO M01-MSG4-OPERATION
          MOVE W03-COMPCODE  TO M01-MSG4-COMPCODE
          MOVE W03-REASON    TO M01-MSG4-REASON
          MOVE M01-MESSAGE-4 TO M00-MESSAGE
     ELSE
          MOVE MQOD-OBJECTNAME TO W02-TEMPORARY-Q
     END-IF.
*
 OPEN-TEMP-RESPONSE-QUEUE-EXIT.
*
*    Return to performing section.
*
     EXIT.
     EJECT
*
```

Opening an existing queue:

This example demonstrates how to use the MQOPEN call to open an existing queue.

This extract is taken from the Browse sample application (program CSQ4BVA1) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
⋮
* -------------------------------------------------------*
 WORKING-STORAGE SECTION.
* -------------------------------------------------------*
*
*    W01 - Fields derived from the command area input
*
 01  W01-OBJECT                  PIC X(48).
*
*    W02 - MQM API fields
*
 01  W02-HCONN        PIC S9(9) BINARY VALUE ZERO.
 01  W02-OPTIONS      PIC S9(9) BINARY.
 01  W02-HOBJ         PIC S9(9) BINARY.
 01  W02-COMPCODE     PIC S9(9) BINARY.
 01  W02-REASON       PIC S9(9) BINARY.
*
*    CMQODV defines the object descriptor (MQOD)
*
 01  MQM-OBJECT-DESCRIPTOR.
     COPY CMQODV.
*
* CMQV contains constants (for setting or testing
* field values) and return codes (for testing the
* result of a call)
*
 01  MQM-CONSTANTS.
 COPY CMQV SUPPRESS.
* -------------------------------------------------------*
 E-OPEN-QUEUE SECTION.
* -------------------------------------------------------*
*                                                        *
* This section opens the queue                           *
*
*    Initialize the Object Descriptor (MQOD) control
*    block
*    (The copy file initializes the remaining fields.)
*
     MOVE MQOT-Q         TO MQOD-OBJECTTYPE.
     MOVE W01-OBJECT     TO MQOD-OBJECTNAME.
*
*    Initialize W02-OPTIONS to open the queue for both
*    inquiring about and setting attributes
*
     COMPUTE W02-OPTIONS = MQOO-INQUIRE + MQOO-SET.

*
*    Open the queue
*
     CALL 'MQOPEN' USING W02-HCONN
                         MQOD
                         W02-OPTIONS
                         W02-HOBJ
                         W02-COMPCODE
                         W02-REASON.
*
*    Test the output from the open
*
*    If the completion code is not OK, display a
```

```
*    separate error message for each of the following
*    errors:
*
*  Q-MGR-NOT-AVAILABLE - MQM is not available
*  CONNECTION-BROKEN   - MQM is no longer connected to CICS
*  UNKNOWN-OBJECT-NAME - The queue does not exist
*  NOT-AUTHORIZED      - The user is not authorized to open
*                        the queue
*
* For any other error, display an error message
* showing the completion and reason codes
*
  IF W02-COMPCODE NOT = MQCC-OK
     EVALUATE TRUE
*
       WHEN W02-REASON = MQRC-Q-MGR-NOT-AVAILABLE
            MOVE M01-MESSAGE-6 TO M00-MESSAGE
*
       WHEN W02-REASON = MQRC-CONNECTION-BROKEN
            MOVE M01-MESSAGE-6 TO M00-MESSAGE
*
       WHEN W02-REASON = MQRC-UNKNOWN-OBJECT-NAME
            MOVE M01-MESSAGE-2 TO M00-MESSAGE
*
       WHEN W02-REASON = MQRC-NOT-AUTHORIZED
            MOVE M01-MESSAGE-3 TO M00-MESSAGE
*
       WHEN OTHER
            MOVE 'MQOPEN'      TO M01-MSG4-OPERATION
            MOVE W02-COMPCODE  TO M01-MSG4-COMPCODE
            MOVE W02-REASON    TO M01-MSG4-REASON
            MOVE M01-MESSAGE-4 TO M00-MESSAGE
        END-EVALUATE
     END-IF.
 E-EXIT.
*
*    Return to performing section
*
     EXIT.
     EJECT
```

Closing a queue:

This example demonstrates how to use the MQCLOSE call.

The variables used in this code extract are those that were set in “Connecting to a queue manager” on page 2001. This extract is taken from the Browse sample application (program CSQ4BVA1) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
⋮
*
*    Close the queue
*
     MOVE MQCO-NONE TO W03-OPTIONS.
*
     CALL 'MQCLOSE' USING W03-HCONN
                          W03-HOBJ
                          W03-OPTIONS
                          W03-COMPCODE
                          W03-REASON.
*
*    Test the output of the MQCLOSE call.  If the call
*    fails, print an error message showing the
*    completion code and reason code.
*
```

```
     IF (W03-COMPCODE NOT = MQCC-OK) THEN
        MOVE 'CLOSE'       TO W04-MSG4-TYPE
        MOVE W03-COMPCODE  TO W04-MSG4-COMPCODE
        MOVE W03-REASON    TO W04-MSG4-REASON
        MOVE W04-MESSAGE-4 TO W00-PRINT-DATA
        PERFORM PRINT-LINE
        MOVE W06-CSQ4-ERROR TO W00-RETURN-CODE
     END-IF.
*
```

Putting a message using MQPUT:

This example demonstrates how to use the MQPUT call using context.

This extract is taken from the Credit Check sample application (program CSQ4CVB1) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
⋮
* -------------------------------------------------------*
 WORKING-STORAGE SECTION.
* -------------------------------------------------------*
*
*    W02 - Queues processed in this program
*
 01  W02-TEMPORARY-Q             PIC X(48).
*
*    W03 - MQM API fields
*
 01  W03-HCONN           PIC S9(9) BINARY VALUE ZERO.
 01  W03-HOBJ-INQUIRY    PIC S9(9) BINARY.
 01  W03-OPTIONS         PIC S9(9) BINARY.
 01  W03-BUFFLEN         PIC S9(9) BINARY.
 01  W03-COMPCODE        PIC S9(9) BINARY.
 01  W03-REASON          PIC S9(9) BINARY.
*
 01  W03-PUT-BUFFER.
*
     05 W03-CSQ4BIIM.
     COPY CSQ4VB1.
*
*    API control blocks
*
 01  MQM-MESSAGE-DESCRIPTOR.
     COPY CMQMDV.
 01  MQM-PUT-MESSAGE-OPTIONS.
     COPY CMQPMOV.
*
*    MQV contains constants (for filling in the
*    control blocks) and return codes (for testing
*    the result of a call).
*
 01  MQM-CONSTANTS.
     COPY CMQV SUPPRESS.
* -------------------------------------------------------*
 PROCEDURE DIVISION.
* -------------------------------------------------------*
⋮
*    Open queue and build message.
⋮
*
* Set the message descriptor and put-message options to
* the values required to create the message.
* Set the length of the message.
*
```

```
  MOVE MQMT-REQUEST         TO MQMD-MSGTYPE.
  MOVE MQCI-NONE            TO MQMD-CORRELID.
  MOVE MQMI-NONE            TO MQMD-MSGID.
  MOVE W02-TEMPORARY-Q      TO MQMD-REPLYTOQ.
  MOVE SPACES               TO MQMD-REPLYTOQMGR.
  MOVE 5                    TO MQMD-PRIORITY.
  MOVE MQPER-NOT-PERSISTENT TO MQMD-PERSISTENCE.
  COMPUTE MQPMO-OPTIONS     =  MQPMO-NO-SYNCPOINT +
                               MQPMO-DEFAULT-CONTEXT.
  MOVE LENGTH OF CSQ4BIIM-MSG TO W03-BUFFLEN.
*
     CALL 'MQPUT' USING W03-HCONN
                        W03-HOBJ-INQUIRY
                        MQMD
                        MQPMO
                        W03-BUFFLEN
                        W03-PUT-BUFFER
                        W03-COMPCODE
                        W03-REASON.
     IF W03-COMPCODE NOT = MQCC-OK

⋮
     END-IF.
```

Putting a message using MQPUT1:

This example demonstrates how to use the MQPUT1 call.

This extract is taken from the Credit Check sample application (program CSQ4CVB5) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
⋮
* -------------------------------------------------------*
 WORKING-STORAGE SECTION.
* -------------------------------------------------------*
*
*    W03 - MQM API fields
*
 01  W03-HCONN          PIC S9(9) BINARY VALUE ZERO.
 01  W03-OPTIONS        PIC S9(9) BINARY.
 01  W03-COMPCODE       PIC S9(9) BINARY.
 01  W03-REASON         PIC S9(9) BINARY.
 01  W03-BUFFLEN        PIC S9(9) BINARY.
*
 01  W03-PUT-BUFFER.
     05 W03-CSQ4BQRM.
     COPY CSQ4VB4.
*
*    API control blocks
*
 01  MQM-OBJECT-DESCRIPTOR.
     COPY CMQODV.
 01  MQM-MESSAGE-DESCRIPTOR.
     COPY CMQMDV.
 01  MQM-PUT-MESSAGE-OPTIONS.
     COPY CMQPMOV.
*
* CMQV contains constants (for filling in the
* control blocks) and return codes (for testing
* the result of a call).
*
 01  MQM-MQV.
 COPY CMQV SUPPRESS.
* -------------------------------------------------------*
```

```
 PROCEDURE DIVISION.
* ---------------------------------------------------------*
:
:
*    Get the request message.
:
:
* ---------------------------------------------------------*
 PROCESS-QUERY SECTION.
* ---------------------------------------------------------*
:
:
*    Build the reply message.
:
:
*
* Set the object descriptor, message descriptor and
* put-message options to the values required to create
* the message.
* Set the length of the message.
*
  MOVE MQMD-REPLYTOQ    TO MQOD-OBJECTNAME.
  MOVE MQMD-REPLYTOQMGR TO MQOD-OBJECTQMGRNAME.
  MOVE MQMT-REPLY       TO MQMD-MSGTYPE.
  MOVE SPACES           TO MQMD-REPLYTOQ.
  MOVE SPACES           TO MQMD-REPLYTOQMGR.
  MOVE LOW-VALUES       TO MQMD-MSGID.
  COMPUTE MQPMO-OPTIONS = MQPMO-SYNCPOINT +
                          MQPMO-PASS-IDENTITY-CONTEXT.
  MOVE W03-HOBJ-CHECKQ  TO MQPMO-CONTEXT.
  MOVE LENGTH OF CSQ4BQRM-MSG TO W03-BUFFLEN.
*
     CALL 'MQPUT1' USING W03-HCONN
                         MQOD
                         MQMD
                         MQPMO
                         W03-BUFFLEN
                         W03-PUT-BUFFER
                         W03-COMPCODE
                         W03-REASON.
     IF W03-COMPCODE NOT = MQCC-OK
         MOVE 'MQPUT1'          TO M02-OPERATION
         MOVE MQOD-OBJECTNAME   TO M02-OBJECTNAME
         PERFORM RECORD-CALL-ERROR
         PERFORM FORWARD-MSG-TO-DLQ
     END-IF.
*
```

Getting a message:

This example demonstrates how to use the MQGET call to remove a message from a queue.

This extract is taken from the Credit Check sample application (program CSQ4CVB1) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
:
:
* ---------------------------------------------------------*
 WORKING-STORAGE SECTION.
* ---------------------------------------------------------*
*
*    W03 - MQM API fields
*
 01  W03-HCONN            PIC S9(9) BINARY VALUE ZERO.
 01  W03-HOBJ-RESPONSE    PIC S9(9) BINARY.
 01  W03-OPTIONS          PIC S9(9) BINARY.
 01  W03-BUFFLEN          PIC S9(9) BINARY.
 01  W03-DATALEN          PIC S9(9) BINARY.
 01  W03-COMPCODE         PIC S9(9) BINARY.
```

```
 01  W03-REASON           PIC S9(9) BINARY.
*
 01  W03-GET-BUFFER.
     05 W03-CSQ4BAM.
     COPY CSQ4VB2.
*
*    API control blocks
*
 01  MQM-MESSAGE-DESCRIPTOR.
     COPY CMQMDV.
 01  MQM-GET-MESSAGE-OPTIONS.
     COPY CMQGMOV.
*
*    MQV contains constants (for filling in the
*    control blocks) and return codes (for testing
*    the result of a call).
*
 01  MQM-CONSTANTS.
     COPY CMQV SUPPRESS.
* ---------------------------------------------------------*
 A-MAIN SECTION.
* ---------------------------------------------------------*
:
:
*    Open response queue.
:
:
* ---------------------------------------------------------*
 PROCESS-RESPONSE-SCREEN SECTION.
* ---------------------------------------------------------*
*                                                          *
*  This section gets a message from the response queue.    *
*                                                          *
*  When a correct response is received, it is              *
*  transferred to the map for display; otherwise           *
*  an error message is built.                              *
*                                                          *
* ---------------------------------------------------------*

*
*    Set get-message options
*
  COMPUTE MQGMO-OPTIONS = MQGMO-SYNCPOINT +
                          MQGMO-ACCEPT-TRUNCATED-MSG +
                          MQGMO-NO-WAIT.
*
* Set msgid and correlid in MQMD to nulls so that any
* message will qualify.
* Set length to available buffer length.
*
     MOVE MQMI-NONE TO MQMD-MSGID.
     MOVE MQCI-NONE TO MQMD-CORRELID.
     MOVE LENGTH OF W03-GET-BUFFER TO W03-BUFFLEN.
*
     CALL 'MQGET' USING W03-HCONN
                        W03-HOBJ-RESPONSE
                        MQMD
                        MQGMO
                        W03-BUFFLEN
                        W03-GET-BUFFER
                        W03-DATALEN
                        W03-COMPCODE
                        W03-REASON.
     EVALUATE TRUE
         WHEN W03-COMPCODE NOT = MQCC-FAILED

:
:
*            Process the message
```

```
⋮
        WHEN (W03-COMPCODE = MQCC-FAILED AND
              W03-REASON = MQRC-NO-MSG-AVAILABLE)
                MOVE M01-MESSAGE-9 TO M00-MESSAGE
                PERFORM CLEAR-RESPONSE-SCREEN
*
        WHEN OTHER
            MOVE 'MQGET  '     TO M01-MSG4-OPERATION
            MOVE W03-COMPCODE  TO M01-MSG4-COMPCODE
            MOVE W03-REASON    TO M01-MSG4-REASON
            MOVE M01-MESSAGE-4 TO M00-MESSAGE
            PERFORM CLEAR-RESPONSE-SCREEN
    END-EVALUATE.
```

Getting a message using the wait option:

This example demonstrates how to use the MQGET call with the wait option and accepting truncated messages.

This extract is taken from the Credit Check sample application (program CSQ4CVB5) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
⋮
* -------------------------------------------------------*
 WORKING-STORAGE SECTION.
* -------------------------------------------------------*
*
*    W00 - General work fields
*
 01  W00-WAIT-INTERVAL   PIC S9(09) BINARY VALUE 30000.
*
*    W03 - MQM API fields
*
 01  W03-HCONN           PIC S9(9) BINARY VALUE ZERO.
 01  W03-OPTIONS         PIC S9(9) BINARY.
 01  W03-HOBJ-CHECKQ     PIC S9(9) BINARY.
 01  W03-COMPCODE        PIC S9(9) BINARY.
 01  W03-REASON          PIC S9(9) BINARY.
 01  W03-DATALEN         PIC S9(9) BINARY.
 01  W03-BUFFLEN         PIC S9(9) BINARY.
*
 01  W03-MSG-BUFFER.
     05 W03-CSQ4BCAQ.
     COPY CSQ4VB3.
*
*    API control blocks
*
 01  MQM-MESSAGE-DESCRIPTOR.
     COPY CMQMDV.
 01  MQM-GET-MESSAGE-OPTIONS.
     COPY CMQGMOV.
*
*    CMQV contains constants (for filling in the
*    control blocks) and return codes (for testing
*    the result of a call).
*
 01  MQM-MQV.
 COPY CMQV SUPPRESS.
* -------------------------------------------------------*
 PROCEDURE DIVISION.
* -------------------------------------------------------*
```

```
:
*    Open input queue.
:
*
*    Get and process messages.
*
  COMPUTE MQGMO-OPTIONS = MQGMO-WAIT +
                          MQGMO-ACCEPT-TRUNCATED-MSG +
                          MQGMO-SYNCPOINT.
  MOVE LENGTH OF W03-MSG-BUFFER TO W03-BUFFLEN.
  MOVE W00-WAIT-INTERVAL TO MQGMO-WAITINTERVAL.
  MOVE MQMI-NONE TO MQMD-MSGID.
  MOVE MQCI-NONE TO MQMD-CORRELID.
*
*    Make the first MQGET call outside the loop.
*
     CALL 'MQGET' USING W03-HCONN
                        W03-HOBJ-CHECKQ
                        MQMD
                        MQGMO
                        W03-BUFFLEN
                        W03-MSG-BUFFER
                        W03-DATALEN
                        W03-COMPCODE
                        W03-REASON.
*
*    Test the output of the MQGET call using the
*    PERFORM loop that follows.
*
*    Perform whilst no failure occurs
*      - process this message
*      - reset the call parameters
*      - get another message
*    End-perform
*

*
*    Test the output of the MQGET call.  If the call
*    fails, send an error message showing the
*    completion code and reason code, unless the
*    completion code is NO-MSG-AVAILABLE.
*
     IF (W03-COMPCODE NOT = MQCC-FAILED) OR
        (W03-REASON NOT = MQRC-NO-MSG-AVAILABLE)
         MOVE 'MQGET '          TO M02-OPERATION
         MOVE MQOD-OBJECTNAME   TO M02-OBJECTNAME
               PERFORM RECORD-CALL-ERROR
     END-IF.
:
```

Getting a message using signaling:

This example demonstrates how to use the MQGET call with signaling. This extract is taken from the Credit Check sample application (program CSQ4CVB2) supplied with IBM MQ for z/OS.

Signaling is available only with IBM MQ for z/OS .

```
⋮
* -------------------------------------------------------*
 WORKING-STORAGE SECTION.
* -------------------------------------------------------*
*
*    W00 - General work fields

⋮
 01  W00-WAIT-INTERVAL    PIC S9(09) BINARY VALUE 30000.
*
*    W03 - MQM API fields
*
 01  W03-HCONN            PIC S9(9) BINARY VALUE ZERO.
 01  W03-HOBJ-REPLYQ      PIC S9(9) BINARY.
 01  W03-COMPCODE         PIC S9(9) BINARY.
 01  W03-REASON           PIC S9(9) BINARY.
 01  W03-DATALEN          PIC S9(9) BINARY.
 01  W03-BUFFLEN          PIC S9(9) BINARY.

⋮
 01  W03-GET-BUFFER.
     05 W03-CSQ4BQRM.
     COPY CSQ4VB4.
*
     05 W03-CSQ4BIIM REDEFINES W03-CSQ4BQRM.
     COPY CSQ4VB1.
*
     05 W03-CSQ4BPGM REDEFINES W03-CSQ4BIIM.
     COPY CSQ4VB5.

⋮
*    API control blocks
*
 01  MQM-MESSAGE-DESCRIPTOR.
     COPY CMQMDV.
 01  MQM-GET-MESSAGE-OPTIONS.
     COPY CMQGMOV.

⋮
*    MQV contains constants (for filling in the
*    control blocks) and return codes (for testing
*    the result of a call).
*
 01  MQM-MQV.
 COPY CMQV SUPPRESS.
* -------------------------------------------------------*
 LINKAGE SECTION.
* -------------------------------------------------------*
 01  L01-ECB-ADDR-LIST.
     05  L01-ECB-ADDR1        POINTER.
     05  L01-ECB-ADDR2        POINTER.
*
 01  L02-ECBS.
     05  L02-INQUIRY-ECB1     PIC S9(09) BINARY.
     05  L02-REPLY-ECB2       PIC S9(09) BINARY.
 01  REDEFINES L02-ECBS.
     05                       PIC  X(02).
     05  L02-INQUIRY-ECB1-CC  PIC S9(04) BINARY.
```

```
     05                      PIC  X(02).
     05  L02-REPLY-ECB2-CC    PIC S9(04) BINARY.
*
* ------------------------------------------------------*
 PROCEDURE DIVISION.
* ------------------------------------------------------*
:
* Initialize variables, open queues, set signal on
* inquiry queue.
:
* ------------------------------------------------------*
 PROCESS-SIGNAL-ACCEPTED SECTION.
* ------------------------------------------------------*
*  This section gets a message with signal.  If a       *
*  message is received, process it.  If the signal      *
*  is set or is already set, the program goes into      *
*  an operating system wait.                            *
*  Otherwise an error is reported and call error set.   *
* ------------------------------------------------------*
*
  PERFORM REPLYQ-GETSIGNAL.
*
  EVALUATE TRUE
      WHEN (W03-COMPCODE = MQCC-OK AND
              W03-REASON = MQRC-NONE)
          PERFORM PROCESS-REPLYQ-MESSAGE
*
      WHEN (W03-COMPCODE = MQCC-WARNING AND
              W03-REASON = MQRC-SIGNAL-REQUEST-ACCEPTED)
           OR
           (W03-COMPCODE = MQCC-FAILED AND
              W03-REASON = MQRC-SIGNAL-OUTSTANDING)
          PERFORM EXTERNAL-WAIT
*
      WHEN OTHER
          MOVE 'MQGET SIGNAL'  TO M02-OPERATION
          MOVE MQOD-OBJECTNAME TO M02-OBJECTNAME
          PERFORM RECORD-CALL-ERROR
          MOVE W06-CALL-ERROR  TO W06-CALL-STATUS
  END-EVALUATE.
*
 PROCESS-SIGNAL-ACCEPTED-EXIT.
*    Return to performing section
     EXIT.
     EJECT
*
* ------------------------------------------------------*
 EXTERNAL-WAIT SECTION.
* ------------------------------------------------------*
*  This section performs an external CICS wait on two   *
*  ECBs until at least one is posted.  It then calls    *
*  the sections to handle the posted ECB.               *
* ------------------------------------------------------*
     EXEC CICS WAIT EXTERNAL
         ECBLIST(W04-ECB-ADDR-LIST-PTR)
         NUMEVENTS(2)
     END-EXEC.
*
* At least one ECB must have been posted to get to this
* point. Test which ECB has been posted and perform
* the appropriate section.
*
     IF L02-INQUIRY-ECB1 NOT = 0
         PERFORM TEST-INQUIRYQ-ECB
     ELSE
         PERFORM TEST-REPLYQ-ECB
```

```
     END-IF.
*
 EXTERNAL-WAIT-EXIT.
*
*    Return to performing section.
*
     EXIT.
     EJECT

:
* ------------------------------------------------------*
 REPLYQ-GETSIGNAL SECTION.
* ------------------------------------------------------*
*                                                       *
* This section performs an MQGET call (in syncpoint with *
* signal) on the reply queue.  The signal field in the   *
* MQGMO is set to the address of the ECB.                *
* Response handling is done by the performing section.   *
*                                                       *
* ------------------------------------------------------*
*
     COMPUTE MQGMO-OPTIONS         =  MQGMO-SYNCPOINT +
                                      MQGMO-SET-SIGNAL.
     MOVE W00-WAIT-INTERVAL        TO MQGMO-WAITINTERVAL.
     MOVE LENGTH OF W03-GET-BUFFER TO W03-BUFFLEN.
*
     MOVE ZEROS                    TO L02-REPLY-ECB2.
     SET MQGMO-SIGNAL1 TO ADDRESS OF L02-REPLY-ECB2.

*
*    Set msgid and correlid to nulls so that any message
*    will qualify.
*
     MOVE MQMI-NONE TO MQMD-MSGID.
     MOVE MQCI-NONE TO MQMD-CORRELID.
*
     CALL 'MQGET' USING W03-HCONN
                        W03-HOBJ-REPLYQ
                        MQMD
                        MQGMO
                        W03-BUFFLEN
                        W03-GET-BUFFER
                        W03-DATALEN
                        W03-COMPCODE
                        W03-REASON.
*
 REPLYQ-GETSIGNAL-EXIT.
*
*    Return to performing section.
*
     EXIT.
     EJECT
*

:
```

Inquiring about the attributes of an object:

This example demonstrates how to use the MQINQ call to inquire about the attributes of a queue.

This extract is taken from the Queue Attributes sample application (program CSQ4CVC1) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS).

```
⋮
* -------------------------------------------------------*
 WORKING-STORAGE SECTION.
* -------------------------------------------------------*
*
*    W02 - MQM API fields
*
 01  W02-SELECTORCOUNT    PIC S9(9) BINARY VALUE 2.
 01  W02-INTATTRCOUNT     PIC S9(9) BINARY VALUE 2.
 01  W02-CHARATTRLENGTH   PIC S9(9) BINARY VALUE ZERO.
 01  W02-CHARATTRS        PIC X     VALUE LOW-VALUES.
 01  W02-HCONN            PIC S9(9) BINARY VALUE ZERO.
 01  W02-HOBJ             PIC S9(9) BINARY.
 01  W02-COMPCODE         PIC S9(9) BINARY.
 01  W02-REASON           PIC S9(9) BINARY.
 01  W02-SELECTORS-TABLE.
     05  W02-SELECTORS    PIC S9(9) BINARY OCCURS 2 TIMES
 01  W02-INTATTRS-TABLE.
     05  W02-INTATTRS     PIC S9(9) BINARY OCCURS 2 TIMES
*
*    CMQODV defines the object descriptor (MQOD).
*
 01  MQM-OBJECT-DESCRIPTOR.
     COPY CMQODV.
*
* CMQV contains constants (for setting or testing field
* values) and return codes (for testing the result of a
* call).
*
 01  MQM-CONSTANTS.
 COPY CMQV SUPPRESS.
* -------------------------------------------------------*
 PROCEDURE DIVISION.
* -------------------------------------------------------*
*
*    Get the queue name and open the queue.
*

⋮
*
*    Initialize the variables for the inquiry call:
*     - Set W02-SELECTORS-TABLE to the attributes whose
*    status is required
*     - All other variables are already set
*
     MOVE MQIA-INHIBIT-GET TO W02-SELECTORS(1).
     MOVE MQIA-INHIBIT-PUT TO W02-SELECTORS(2).
*
*    Inquire about the attributes.
*
     CALL 'MQINQ' USING W02-HCONN,
                        W02-HOBJ,
                        W02-SELECTORCOUNT,
                        W02-SELECTORS-TABLE,
                        W02-INTATTRCOUNT,
                        W02-INTATTRS-TABLE,
                        W02-CHARATTRLENGTH,
                        W02-CHARATTRS,
```

```
                        W02-COMPCODE,
                        W02-REASON.
*
* Test the output from the inquiry:
*
* - If the completion code is not OK, display an error
*   message showing the completion and reason codes
*
* - Otherwise, move the correct attribute status into
*   the relevant screen map fields
*
     IF W02-COMPCODE NOT = MQCC-OK
        MOVE 'MQINQ'       TO M01-MSG4-OPERATION
        MOVE W02-COMPCODE  TO M01-MSG4-COMPCODE
        MOVE W02-REASON    TO M01-MSG4-REASON
        MOVE M01-MESSAGE-4 TO M00-MESSAGE
*
     ELSE
*       Process the changes.

⋮
           END-IF.

⋮
```

Setting the attributes of a queue:

This example demonstrates how to use the MQSET call to change the attributes of a queue.

This extract is taken from the Queue Attributes sample application (program CSQ4CVC1) supplied with IBM MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample procedural programs (platforms except z/OS)

```
⋮
* ------------------------------------------------------------*
 WORKING-STORAGE SECTION.
* ------------------------------------------------------------*
*
*    W02 - MQM API fields
*
 01  W02-SELECTORCOUNT    PIC S9(9) BINARY VALUE 2.
 01  W02-INTATTRCOUNT     PIC S9(9) BINARY VALUE 2.
 01  W02-CHARATTRLENGTH   PIC S9(9) BINARY VALUE ZERO.
 01  W02-CHARATTRS        PIC X     VALUE LOW-VALUES.
 01  W02-HCONN            PIC S9(9) BINARY VALUE ZERO.
 01  W02-HOBJ             PIC S9(9) BINARY.
 01  W02-COMPCODE         PIC S9(9) BINARY.
 01  W02-REASON           PIC S9(9) BINARY.
 01  W02-SELECTORS-TABLE.
     05  W02-SELECTORS    PIC S9(9) BINARY OCCURS 2 TIMES.
 01  W02-INTATTRS-TABLE.
     05  W02-INTATTRS     PIC S9(9) BINARY OCCURS 2 TIMES.
*
*    CMQODV defines the object descriptor (MQOD).
*
 01  MQM-OBJECT-DESCRIPTOR.
     COPY CMQODV.
*
* CMQV contains constants (for setting or testing
* field values) and return codes (for testing the
* result of a call).
*
 01  MQM-CONSTANTS.
 COPY CMQV SUPPRESS.
```

```
* -------------------------------------------------------*
 PROCEDURE DIVISION.
* -------------------------------------------------------*
*
*    Get the queue name and open the queue.
*
:
*
*
* Initialize the variables required for the set call:
* - Set W02-SELECTORS-TABLE to the attributes to be set
* - Set W02-INTATTRS-TABLE to the required status
* - All other variables are already set
*
     MOVE MQIA-INHIBIT-GET   TO W02-SELECTORS(1).
     MOVE MQIA-INHIBIT-PUT   TO W02-SELECTORS(2).
     MOVE MQQA-GET-INHIBITED TO W02-INTATTRS(1).
     MOVE MQQA-PUT-INHIBITED TO W02-INTATTRS(2).
*
*    Set the attributes.
*
     CALL 'MQSET' USING W02-HCONN,
                        W02-HOBJ,
                        W02-SELECTORCOUNT,
                        W02-SELECTORS-TABLE,
                        W02-INTATTRCOUNT,
                        W02-INTATTRS-TABLE,
                        W02-CHARATTRLENGTH,
                        W02-CHARATTRS,
                        W02-COMPCODE,
                        W02-REASON.
*
* Test the output from the call:
*
*  - If the completion code is not OK, display an error
*    message showing the completion and reason codes
*
*  - Otherwise, move 'INHIBITED' into the relevant
*    screen map fields
*
     IF W02-COMPCODE NOT = MQCC-OK
        MOVE 'MQSET'       TO M01-MSG4-OPERATION
        MOVE W02-COMPCODE  TO M01-MSG4-COMPCODE
        MOVE W02-REASON    TO M01-MSG4-REASON
        MOVE M01-MESSAGE-4 TO M00-MESSAGE
     ELSE
*
*       Process the changes.
:
     END-IF.
```

System/390 assembler-language examples:

This collection of topics is mostly taken from the IBM MQ for z/OS sample applications.

Connecting to a queue manager:

This example demonstrates how to use the MQCONN call to connect a program to a queue manager in z/OS batch.

This extract is taken from the Browse sample program (CSQ4BAA1) supplied with IBM MQ for z/OS.

```
⋮
WORKAREA DSECT
*
PARMLIST CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
COMPCODE DS    F              Completion code
REASON   DS    F              Reason code
HCONN    DS    F              Connection handle
         ORG
PARMADDR DS    F              Address of parm field
PARMLEN  DS    H              Length of parm field
*
MQMNAME  DS    CL48           Queue manager name
*
*
*********************************************************
*  SECTION NAME : MAINPARM                              *
*********************************************************
MAINPARM DS    0H
         MVI   MQMNAME,X'40'
         MVC   MQMNAME+1(L'MQMNAME-1),MQMNAME
*
* Space out first byte and initialize
*
*
* Code to address and verify parameters passed omitted
*
*
PARM1MVE DS    0H
         SR    R1,R3           Length of data
         LA    R4,MQMNAME      Address for target
         BCTR  R1,R0           Reduce for execute
         EX    R1,MOVEPARM     Move the data
*
*********************************************************
* EXECUTES                                              *
*********************************************************
MOVEPARM MVC   0(*-*,R4),0(R3)
*
         EJECT
*********************************************************
*  SECTION NAME : MAINCONN                              *
*********************************************************
*
*
MAINCONN DS    0H
         XC    HCONN,HCONN       Null connection handle
*
         CALL  MQCONN,                          X
               (MQMNAME,                        X
               HCONN,                           X
               COMPCODE,                        X
               REASON),                         X
               MF=(E,PARMLIST),VL
```

```
*
         LA    R0,MQCC_OK       Expected compcode
         C     R0,COMPCODE      As expected?
         BER   R6               Yes .. return to caller
*
         MVC   INF4_TYP,=CL10'CONNECT   '
         BAL   R7,ERRCODE             Translate error
         LA    R0,8                   Set exit code
         ST    R0,EXITCODE            to 8
         B     ENDPROG                End the program
*
```

Disconnecting from a queue manager:

This example demonstrates how to use the MQDISC call to disconnect a program from a queue manager in z/OS batch.

This extract is not taken from the sample applications supplied with IBM MQ.

```
⋮
*
*        ISSUE MQI DISC REQUEST USING REENTRANT FORM
*        OF CALL MACRO
*
*        HCONN WAS SET BY A PREVIOUS MQCONN REQUEST
*        R5 = WORK REGISTER
*
DISC     DS    0H
         CALL  MQDISC,                      X
               (HCONN,                      X
               COMPCODE,                    X
               REASON),                     X
               VL,MF=(E,CALLLST)
*
         LA    R5,MQCC_OK
         C     R5,COMPCODE
         BNE   BADCALL

⋮
BADCALL  DS    0H
⋮
*                   CONSTANTS
*
         CMQA
*
*        WORKING STORAGE (RE-ENTRANT)
*
WEG3     DSECT
*
CALLLST  CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
HCONN    DS   F
COMPCODE DS   F
REASON   DS   F
*
*
LEG3     EQU  *-WKEG3
         END
```

Creating a dynamic queue:

This example demonstrates how to use the MQOPEN call to create a dynamic queue.

This extract is not taken from the sample applications supplied with IBM MQ.

```
:
*
*    R5 = WORK REGISTER.
*
OPEN     DS   0H
*
         MVC  WOD_AREA,MQOD_AREA INITIALIZE WORKING VERSION OF
*                                  MQOD WITH DEFAULTS
         MVC  WOD_OBJECTNAME,MOD_Q    COPY IN THE MODEL Q NAME
         MVC  WOD_DYNAMICQNAME,DYN_Q  COPY IN THE DYNAMIC Q NAME
         L    R5,=AL4(MQOO_OUTPUT)    OPEN FOR OUTPUT AND
         A    R5,=AL4(MQOO_INQUIRE)   INQUIRE
         ST   R5,OPTIONS
*
* ISSUE MQI OPEN REQUEST USING REENTRANT
* FORM OF CALL MACRO
*
         CALL MQOPEN,                           X
               (HCONN,                          X
               WOD,                             X
               OPTIONS,                         X
               HOBJ,                            X
               COMPCODE,                        X
               REASON),VL,MF=(E,CALLLST)
*
         LA  R5,MQCC_OK                CHECK THE COMPLETION CODE
         C   R5,COMPCODE               FROM THE REQUEST AND BRANCH
         BNE BADCALL                   TO ERROR ROUTINE IF NOT MQCC_OK
*
         MVC  TEMP_Q,WOD_OBJECTNAME  SAVE NAME OF TEMPORARY Q
*                             CREATED BY OPEN OF MODEL Q
*
:
BADCALL  DS   0H
:
*
*
*   CONSTANTS:
*
MOD_Q  DC   CL48'QUERY.REPLY.MODEL'  MODEL QUEUE NAME
DYN_Q  DC   CL48'QUERY.TEMPQ.*'      DYNAMIC QUEUE NAME
*
       CMQODA DSECT=NO,LIST=YES  CONSTANT VERSION OF MQOD
       CMQA                      MQI VALUE EQUATES
*
*   WORKING STORAGE
*
         DFHEISTG
HCONN    DS F                      CONNECTION HANDLE
OPTIONS  DS F                      OPEN OPTIONS
HOBJ     DS F                      OBJECT HANDLE
COMPCODE DS F                      MQI COMPLETION CODE
REASON   DS F                      MQI REASON CODE
TEMP_Q   DS CL(MQ_Q_NAME_LENGTH)   SAVED QNAME AFTER OPEN
*
WOD      CMQODA DSECT=NO,LIST=YES  WORKING VERSION OF MQOD
*
CALLLST  CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L  LIST FORM
                                                 OF CALL
*                                                 MACRO
```

```
⋮
        END
```

Opening an existing queue:

This example demonstrates how to use the MQOPEN call to open a queue that has already been defined.

It shows how to specify two options. This extract is not taken from the sample applications supplied with IBM MQ.

```
⋮
*
*    R5 = WORK REGISTER.
*
OPEN     DS   0H
*
         MVC  WOD_AREA,MQOD_AREA  INITIALIZE WORKING VERSION OF
*                                 MQOD WITH DEFAULTS
         MVC  WOD_OBJECTNAME,Q_NAME    SPECIFY Q NAME TO OPEN
         LA   R5,MQOO_INPUT_EXCLUSIVE  OPEN FOR MQGET CALLS
*
         ST   R5,OPTIONS
*
* ISSUE MQI OPEN REQUEST USING REENTRANT FORM
* OF CALL MACRO
*
         CALL MQOPEN,                          X
               (HCONN,                         X
               WOD,                            X
               OPTIONS,                        X
               HOBJ,                           X
               COMPCODE,                       X
               REASON),VL,MF=(E,CALLLST)
*
         LA  R5,MQCC_OK         CHECK THE COMPLETION CODE
         C   R5,COMPCODE        FROM THE REQUEST AND BRANCH
         BNE BADCALL            TO ERROR ROUTINE IF NOT MQCC_OK
*
⋮
BADCALL  DS   0H
⋮
*
*
*   CONSTANTS:
*
Q_NAME   DC   CL48'REQUEST.QUEUE'  NAME OF QUEUE TO OPEN
*
         CMQODA DSECT=NO,LIST=YES  CONSTANT VERSION OF MQOD
         CMQA                      MQI VALUE EQUATES
*
*   WORKING STORAGE
*
         DFHEISTG
HCONN    DS F           CONNECTION HANDLE
OPTIONS  DS F           OPEN OPTIONS
HOBJ     DS F           OBJECT HANDLE
COMPCODE DS F           MQI COMPLETION CODE
REASON   DS F           MQI REASON CODE
*
WOD  CMQODA DSECT=NO,LIST=YES  WORKING VERSION OF MQOD
*
CALLLST  CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L  LIST FORM
                                                 OF CALL
*                                                 MACRO
```

```
⋮
         END
```

Closing a queue:

This example demonstrates how to use the MQCLOSE call to close a queue.

This extract is not taken from the sample applications supplied with IBM MQ.

```
⋮
*
* ISSUE MQI CLOSE REQUEST USING REENTRANT FROM OF
* CALL MACRO
*
*        HCONN WAS SET BY A PREVIOUS MQCONN REQUEST
*        HOBJ  WAS SET BY A PREVIOUS MQOPEN REQUEST
*        R5 = WORK REGISTER
*
CLOSE    DS    0H
         LA    R5,MQCO_NONE        NO SPECIAL CLOSE OPTIONS
         ST    R5,OPTIONS              ARE REQUIRED.
*
         CALL  MQCLOSE,                       X
               (HCONN,                        X
               HOBJ,                          X
               OPTIONS,                       X
               COMPCODE,                      X
               REASON),                       X
               VL,MF=(E,CALLLST)
*
         LA    R5,MQCC_OK
         C     R5,COMPCODE
         BNE   BADCALL
*

⋮
BADCALL  DS    0H

⋮
*                   CONSTANTS
*
         CMQA
*
*        WORKING STORAGE (REENTRANT)
*
WEG4     DSECT
*
CALLLST  CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
HCONN    DS   F
HOBJ     DS   F
OPTIONS  DS   F
COMPCODE DS   F
REASON   DS   F
*
*
LEG4     EQU  *-WKEG4
         END
```

Putting a message using MQPUT:

This example demonstrates how to use the MQPUT call to put a message on a queue.

This extract is not taken from the sample applications supplied with IBM MQ.

```
:
*     CONNECT TO QUEUE MANAGER
*
CONN    DS  0H
:
*
*     OPEN A QUEUE
*
OPEN    DS  0H
:
*
*     R4,R5,R6,R7 = WORK REGISTER.
*
PUT  DS  0H
     LA   R4,MQMD           SET UP ADDRESSES AND
     LA   R5,MQMD_LENGTH    LENGTH FOR USE BY MVCL
     LA   R6,WMD            INSTRUCTION, AS MQMD IS
     LA   R7,WMD_LENGTH     OVER 256 BYES LONG.
     MVCL R6,R4             INITIALIZE WORKING VERSION
*                           OF MESSAGE DESCRIPTOR
*
     MVC  WPMO_AREA,MQPMO_AREA  INITIALIZE WORKING MQPMO
*

     LA   R5,BUFFER_LEN   RETRIEVE THE BUFFER LENGTH
     ST   R5,BUFFLEN      AND SAVE IT FOR MQM USE
*
     MVC  BUFFER,TEST_MSG    SET THE MESSAGE TO BE PUT
*
*    ISSUE MQI PUT REQUEST USING REENTRANT FORM
*    OF CALL MACRO
*
*        HCONN WAS SET BY PREVIOUS MQCONN REQUEST
*        HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST
*
         CALL MQPUT,                          X
               (HCONN,                        X
               HOBJ,                          X
               WMD,                           X
               WPMO,                          X
               BUFFLEN,                       X
               BUFFER,                        X
               COMPCODE,                      X
               REASON),VL,MF=(E,CALLLST)
*
         LA  R5,MQCC_OK
         C   R5,COMPCODE
         BNE BADCALL
*

:
BADCALL DS  0H
:
*
*   CONSTANTS
*
CMQMDA DSECT=NO,LIST=YES,PERSISTENCE=MQPER_PERSISTENT
CMQPMOA DSECT=NO,LIST=YES
CMQA
TEST_MSG DC CL80'THIS IS A TEST MESSAGE'
```

```
*
*   WORKING STORAGE DSECT
*
WORKSTG DSECT
*
COMPCODE DS F
REASON  DS F
BUFFLEN DS F
OPTIONS DS F
HCONN  DS F
HOBJ   DS F
*
BUFFER  DS CL80
BUFFER_LEN EQU *-BUFFER
*
WMD   CMQMDA DSECT=NO,LIST=NO
WPMO   CMQPMOA DSECT=NO,LIST=NO
*
CALLLST CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
.
.
.
END
```

Putting a message using MQPUT1:

This example demonstrates how to use the MQPUT1 call to open a queue, put a single message on the queue, then close the queue.

This extract is not taken from the sample applications supplied with IBM MQ.

```
.
.
.
*
*     CONNECT TO QUEUE MANAGER
*
CONN    DS  0H
.
.
.
*
*     R4,R5,R6,R7 = WORK REGISTER.
*
PUT      DS  0H
*
         MVC  WOD_AREA,MQOD_AREA     INITIALIZE WORKING VERSION OF
*                                    MQOD WITH DEFAULTS
         MVC  WOD_OBJECTNAME,Q_NAME  SPECIFY Q NAME FOR PUT1
*
         LA   R4,MQMD           SET UP ADDRESSES AND
         LA   R5,MQMD_LENGTH    LENGTH FOR USE BY MVCL
         LA   R6,WMD            INSTRUCTION, AS MQMD IS
         LA   R7,WMD_LENGTH     OVER 256 BYES LONG.
         MVCL R6,R4             INITIALIZE WORKING VERSION
*                               OF MESSAGE DESCRIPTOR
*
         MVC  WPMO_AREA,MQPMO_AREA       INITIALIZE WORKING MQPMO
*

         LA   R5,BUFFER_LEN           RETRIEVE THE BUFFER LENGTH
         ST   R5,BUFFLEN              AND SAVE IT FOR MQM USE
*
         MVC  BUFFER,TEST_MSG         SET THE MESSAGE TO BE PUT
*
* ISSUE MQI PUT REQUEST USING REENTRANT FORM OF CALL MACRO
*
*        HCONN WAS SET BY PREVIOUS MQCONN REQUEST
*        HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST
*
```

```
         CALL MQPUT1,                                  X
               (HCONN,                                 X
                LMQOD,                                 X
                LMQMD,                                 X
                LMQPMO,                                X
                BUFFERLENGTH,                          X
                BUFFER,                                X
                COMPCODE,                              X
                REASON),VL,MF=(E,CALLLST)
*
         LA  R5,MQCC_OK
         C   R5,COMPCODE
         BNE BADCALL
*
:
:
BADCALL DS  0H
:
:
*
*      CONSTANTS
*
  CMQMDA DSECT=NO,LIST=YES,PERSISTENCE=MQPER_PERSISTENT
  CMQPMOA DSECT=NO,LIST=YES
  CMQODA DSECT=NO,LIST=YES
  CMQA
*
TEST_MSG DC CL80'THIS IS ANOTHER TEST MESSAGE'
Q_NAME   DC CL48'TEST.QUEUE.NAME'
*
*      WORKING STORAGE DSECT
*
WORKSTG  DSECT
*
COMPCODE DS F
REASON   DS F
BUFFLEN  DS F
OPTIONS  DS F
HCONN    DS F
HOBJ     DS F
*
BUFFER   DS CL80
BUFFER_LEN EQU *-BUFFER
*
WOD      CMQODA DSECT=NO,LIST=YES    WORKING VERSION OF MQOD
WMD      CMQMDA DSECT=NO,LIST=NO
WPMO     CMQPMOA DSECT=NO,LIST=NO
*
CALLLST  CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
:
:
         END
```

Getting a message:

This example demonstrates how to use the MQGET call to remove a message from a queue.

This extract is not taken from the sample applications supplied with IBM MQ.

```
:
*
*   CONNECT TO QUEUE MANAGER
*
CONN    DS  0H
:
*
*     OPEN A QUEUE FOR GET
*
OPEN    DS  0H
:
*
*     R4,R5,R6,R7 = WORK REGISTER.
*
GET  DS  0H
     LA   R4,MQMD               SET UP ADDRESSES AND
     LA   R5,MQMD_LENGTH        LENGTH FOR USE BY MVCL
     LA   R6,WMD                INSTRUCTION, AS MQMD IS
     LA   R7,WMD_LENGTH         OVER 256 BYES LONG.
     MVCL R6,R4                 INITIALIZE WORKING VERSION
*                                  OF MESSAGE DESCRIPTOR
*
     MVC  WGMO_AREA,MQGMO_AREA  INITIALIZE WORKING MQGMO
*

     LA   R5,BUFFER_LEN         RETRIEVE THE BUFFER LENGTH
     ST   R5,BUFFLEN            AND SAVE IT FOR MQM USE
*
*
* ISSUE MQI GET REQUEST USING REENTRANT FORM OF CALL MACRO
*
*        HCONN WAS SET BY PREVIOUS MQCONN REQUEST
*        HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST
*
         CALL  MQGET,                           X
               (HCONN,                          X
               HOBJ,                            X
               WMD,                             X
               WGMO,                            X
               BUFFLEN,                         X
               BUFFER,                          X
               DATALEN,                         X
               COMPCODE,                        X
               REASON),                         X
               VL,MF=(E,CALLLST)
*
         LA  R5,MQCC_OK
         C   R5,COMPCODE
         BNE BADCALL
*

:
BADCALL DS  0H
:
*
*      CONSTANTS
*
         CMQMDA DSECT=NO,LIST=YES
         CMQGMOA DSECT=NO,LIST=YES
         CMQA
```

```
*
*      WORKING STORAGE DSECT
*
WORKSTG  DSECT
*
COMPCODE DS F
REASON   DS F
BUFFLEN  DS F
DATALEN  DS F
OPTIONS  DS F
HCONN    DS F
HOBJ     DS F
*
BUFFER   DS CL80
BUFFER_LEN EQU *-BUFFER
*
WMD      CMQMDA DSECT=NO,LIST=NO
WGMO     CMQGMOA DSECT=NO,LIST=NO
*
CALLLST  CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
:
:
END
```

Getting a message using the wait option:

This example demonstrates how to use the wait option of the MQGET call.

This code accepts truncated messages. This extract is not taken from the sample applications supplied with IBM MQ.

```
:
:
*     CONNECT TO QUEUE MANAGER
CONN    DS  0H
:
:
*     OPEN A QUEUE FOR GET
OPEN    DS  0H
:
:
*     R4,R5,R6,R7 = WORK REGISTER.
GET  DS  0H
     LA   R4,MQMD              SET UP ADDRESSES AND
     LA   R5,MQMD_LENGTH       LENGTH FOR USE BY MVCL
     LA   R6,WMD               INSTRUCTION, AS MQMD IS
     LA   R7,WMD_LENGTH        OVER 256 BYES LONG.
     MVCL R6,R4                INITIALIZE WORKING VERSION
*                                  OF MESSAGE DESCRIPTOR
*
     MVC  WGMO_AREA,MQGMO_AREA   INITIALIZE WORKING MQGMO
     L    R5,=AL4(MQGMO_WAIT)
     A    R5,=AL4(MQGMO_ACCEPT_TRUNCATED_MSG)
     ST   R5,WGMO_OPTIONS
     MVC  WGMO_WAITINTERVAL,TWO_MINUTES   WAIT UP TO TWO
                                          MINUTES BEFORE
                                          FAILING THE
                                          CALL
*
     LA   R5,BUFFER_LEN     RETRIEVE THE BUFFER LENGTH
     ST   R5,BUFFLEN        AND SAVE IT FOR MQM USE
*
*  ISSUE MQI GET REQUEST USING REENTRANT FORM OF CALL MACRO
*
*        HCONN WAS SET BY PREVIOUS MQCONN REQUEST
*        HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST
*
         CALL  MQGET,                          X
```

```
              (HCONN,                           X
              HOBJ,                             X
              WMD,                              X
              WGMO,                             X
              BUFFLEN,                          X
              BUFFER,                           X
              DATALEN,                          X
              COMPCODE,                         X
              REASON),                          X
              VL,MF=(E,CALLLST)
*
  LA  R5,MQCC_OK                  DID THE MQGET REQUEST
  C   R5,COMPCODE                   WORK OK?
  BE  GETOK                       YES, SO GO AND PROCESS.
  LA  R5,MQCC_WARNING             NO, SO CHECK FOR A WARNING.
  C   R5,COMPCODE                 IS THIS A WARNING?
  BE  CHECK_W                     YES, SO CHECK THE REASON.
*
  LA  R5,MQRC_NO_MSG_AVAILABLE   IT MUST BE AN ERROR.
                                 IS IT DUE TO AN EMPTY
  C   R5,REASON                  QUEUE?
  BE  NOMSG                      YES, SO HANDLE THE ERROR
  B   BADCALL                    NO, SO GO TO ERROR ROUTINE
*
CHECK_W  DS  0H
         LA  R5,MQRC_TRUNCATED_MSG_ACCEPTED   IS THIS A
                                              TRUNCATED
         C   R5,REASON                        MESSAGE?
         BE  GETOK                   YES, SO GO AND PROCESS.
         B   BADCALL                 NO, SOME OTHER WARNING
*
NOMSG    DS  0H

.
.
.
GETOK    DS  0H

.
.
.
BADCALL DS  0H
.
.
.
*
*      CONSTANTS
*
         CMQMDA DSECT=NO,LIST=YES
         CMQGMOA DSECT=NO,LIST=YES
         CMQA
*
TWO_MINUTES DC F'120000'       GET WAIT INTERVAL
*
*      WORKING STORAGE DSECT
*
WORKSTG  DSECT
*
COMPCODE DS F
REASON   DS F
BUFFLEN  DS F
DATALEN  DS F
OPTIONS  DS F
HCONN    DS F
HOBJ     DS F
*
BUFFER   DS CL80
BUFFER_LEN EQU *-BUFFER
*
WMD      CMQMDA DSECT=NO,LIST=NO
WGMO     CMQGMOA DSECT=NO,LIST=NO
*
```

```
CALLLST  CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
⋮
         END
```

Getting a message using signaling:

This example demonstrates how to use the MQGET call to set a signal so that you are notified when a suitable message arrives on a queue.

This extract is not taken from the sample applications supplied with IBM MQ.

```
⋮
*
*     CONNECT TO QUEUE MANAGER
*
CONN    DS  0H
⋮
*
*     OPEN A QUEUE FOR GET
*
OPEN    DS  0H
⋮
*
*     R4,R5,R6,R7 = WORK REGISTER.
*
GET  DS  0H
     LA   R4,MQMD                SET UP ADDRESSES AND
     LA   R5,MQMD_LENGTH         LENGTH FOR USE BY MVCL
     LA   R6,WMD                 INSTRUCTION, AS MQMD IS
     LA   R7,WMD_LENGTH          OVER 256 BYES LONG.
     MVCL R6,R4                  INITIALIZE WORKING VERSION
*                                   OF MESSAGE DESCRIPTOR
*
     MVC  WGMO_AREA,MQGMO_AREA   INITIALIZE WORKING MQGMO
     LA   R5,MQGMO_SET_SIGNAL
     ST   R5,WGMO_OPTIONS
     MVC  WGMO_WAITINTERVAL,FIVE_MINUTES   WAIT UP TO FIVE
                                           MINUTES BEFORE
*                                          FAILING THE CALL
*
     XC   SIG_ECB,SIG_ECB   CLEAR THE ECB
     LA   R5,SIG_ECB        GET THE ADDRESS OF THE ECB
     ST   R5,WGMO_SIGNAL1   AND PUT IT IN THE WORKING
*                                 MQGMO
*

     LA   R5,BUFFER_LEN     RETRIEVE THE BUFFER LENGTH
     ST   R5,BUFFLEN        AND SAVE IT FOR MQM USE
*
*
*    ISSUE MQI GET REQUEST USING REENTRANT FORM OF CALL MACRO
*
*        HCONN WAS SET BY PREVIOUS MQCONN REQUEST
*        HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST
*
         CALL  MQGET,                          X
               (HCONN,                         X
               HOBJ,                           X
               WMD,                            X
               WGMO,                           X
               BUFFLEN,                        X
```

```
              BUFFER,                          X
              DATALEN,                         X
              COMPCODE,                        X
              REASON),                         X
              VL,MF=(E,CALLLST)
*
         LA  R5,MQCC_OK        DID THE MQGET REQUEST
         C   R5,COMPCODE       WORK OK?
         BE  GETOK             YES, SO GO AND PROCESS.
         LA  R5,MQCC_WARNING   NO, SO CHECK FOR A WARNING.
         C   R5,COMPCODE       IS THIS A WARNING?
         BE  CHECK_W           YES, SO CHECK THE REASON.
         B   BADCALL           NO, SO GO TO ERROR ROUTINE
*
CHECK_W  DS  0H
         LA  R5,MQRC_SIGNAL_REQUEST_ACCEPTED
         C   R5,REASON   SIGNAL REQUEST SIGNAL SET?
         BNE BADCALL     NO, SOME ERROR OCCURRED
         B   DOWORK      YES, SO DO SOMETHING
*                          ELSE
*
CHECKSIG DS  0H
         CLC SIG_ECB+1(3),=AL3(MQEC_MSG_ARRIVED)
                               IS A MESSAGE AVAILABLE?
         BE  GET               YES, SO GO AND GET IT
*
         CLC SIG_ECB+1(3),=AL3(MQEC_WAIT_INTERVAL_EXPIRED)
                           HAVE WE WAITED LONG ENOUGH?
         BE  NOMSG         YES, SO SAY NO MSG AVAILABLE
         B   BADCALL       IF IT'S ANYTHING ELSE
*                          GO TO ERROR ROUTINE.
*
DOWORK   DS  0H

.
.
.
         TM  SIG_ECB,X'40'   HAS THE SIGNAL ECB BEEN POSTED?
         BO  CHECKSIG        YES, SO GO AND CHECK WHY
         B   DOWORK          NO, SO GO AND DO MORE WORK
*
NOMSG    DS  0H

.
.
.
GETOK    DS  0H

.
.
.
BADCALL DS  0H

.
.
.
*
*      CONSTANTS
*
         CMQMDA DSECT=NO,LIST=YES
         CMQGMOA DSECT=NO,LIST=YES
         CMQA
*
FIVE_MINUTES DC F'300000'        GET SIGNAL INTERVAL
*
*      WORKING STORAGE DSECT
*
WORKSTG  DSECT
*
COMPCODE DS F
REASON   DS F
BUFFLEN  DS F
DATALEN  DS F
```

```
OPTIONS  DS F
HCONN    DS F
HOBJ     DS F
SIG_ECB  DS F
*
BUFFER   DS CL80
BUFFER_LEN EQU *-BUFFER
*
WMD      CMQMDA DSECT=NO,LIST=NO
WGMO     CMQGMOA DSECT=NO,LIST=NO
*
CALLLST  CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
⋮
END
```

Inquiring about and setting the attributes of a queue:

This example demonstrates how to use the MQINQ call to inquire about the attributes of a queue and to use the MQSET call to change the attributes of a queue.

This extract is taken from the Queue Attributes sample application (program CSQ4CAC1) supplied with IBM MQ for z/OS.

```
⋮
DFHEISTG DSECT

⋮
OBJDESC  CMQODA LIST=YES   Working object descriptor
*
SELECTORCOUNT   DS F       Number of selectors
INTATTRCOUNT    DS F       Number of integer attributes
CHARATTRLENGTH  DS F       char attributes length
CHARATTRS       DS C       Area for char attributes
*
OPTIONS  DS   F            Command options
HCONN    DS   F            Handle of connection
HOBJ     DS   F            Handle of object
COMPCODE DS   F            Completion code
REASON   DS   F            Reason code
SELECTOR DS   2F           Array of selectors
INTATTRS DS   2F           Array of integer attributes

⋮
OBJECT   DS   CL(MQ_Q_NAME_LENGTH)   Name of queue

⋮
CALLLIST CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*********************************************************
*                  PROGRAM EXECUTION STARTS HERE        *
⋮
CSQ4CAC1 DFHEIENT CODEREG=(R3),DATAREG=(R13)

⋮
*        Initialize the variables for the set call
*
         SR   R0,R0               Clear register zero
         ST   R0,CHARATTRLENGTH   Set char length to zero
         LA   R0,2                Load to set
         ST   R0,SELECTORCOUNT    selectors add
         ST   R0,INTATTRCOUNT     integer attributes
*
         LA   R0,MQIA_INHIBIT_GET Load q attribute selector
         ST   R0,SELECTOR+0       Place in field
```

```
         LA   R0,MQIA_INHIBIT_PUT Load q attribute selector
         ST   R0,SELECTOR+4       Place in field
*
UPDTEST  DS   0H
         CLC  ACTION,CINHIB       Are we inhibiting?
         BE   UPDINHBT            Yes branch to section
*
         CLC  ACTION,CALLOW       Are we allowing?
         BE   UPDALLOW            Yes branch to section
*
         MVC  M00_MSG,M01_MSG1    Invalid request
         BR   R6                  Return to caller
*
UPDINHBT DS   0H
         MVC  UPDTYPE,CINHIBIT      Indicate action type
         LA   R0,MQQA_GET_INHIBITED Load attribute value
         ST   R0,INTATTRS+0         Place in field
         LA   R0,MQQA_PUT_INHIBITED Load attribute value
         ST   R0,INTATTRS+4         Place in field
         B    UPDCALL               Go and do call
*
UPDALLOW DS   0H
         MVC  UPDTYPE,CALLOWED      Indicate action type
         LA   R0,MQQA_GET_ALLOWED   Load attribute value
         ST   R0,INTATTRS+0         Place in field
         LA   R0,MQQA_PUT_ALLOWED   Load attribute value
         ST   R0,INTATTRS+4         Place in field
         B    UPDCALL               Go and do call
*
UPDCALL  DS   0H
         CALL MQSET,                          C
               (HCONN,                        C
               HOBJ,                          C
               SELECTORCOUNT,                 C
               SELECTOR,                      C
               INTATTRCOUNT,                  C
               INTATTRS,                      C
               CHARATTRLENGTH,                C
               CHARATTRS,                     C
               COMPCODE,                      C
               REASON),                       C
               VL,MF=(E,CALLLIST)
*
         LA   R0,MQCC_OK    Load expected compcode
         C    R0,COMPCODE   Was set successful?

.
.
.
*  SECTION NAME : INQUIRE                                *
*  FUNCTION     : Inquires on the objects attributes     *
*  CALLED BY    : PROCESS                                *
*  CALLS        : OPEN, CLOSE, CODES                     *
*  RETURN       : To Register 6                          *
INQUIRE  DS   0H

.
.
.
*        Initialize the variables for the inquire call
*
         SR   R0,R0               Clear register zero
         ST   R0,CHARATTRLENGTH   Set char length to zero
         LA   R0,2                Load to set
         ST   R0,SELECTORCOUNT    selectors add
         ST   R0,INTATTRCOUNT     integer attributes
*
         LA   R0,MQIA_INHIBIT_GET Load attribute value
         ST   R0,SELECTOR+0       Place in field
         LA   R0,MQIA_INHIBIT_PUT Load attribute value
```

```
         ST   R0,SELECTOR+4       Place in field
         CALL MQINQ,                          C
               (HCONN,                        C
               HOBJ,                          C
               SELECTORCOUNT,                 C
               SELECTOR,                      C
               INTATTRCOUNT,                  C
               INTATTRS,                      C
               CHARATTRLENGTH,                C
               CHARATTRS,                     C
               COMPCODE,                      C
               REASON),                       C
               VL,MF=(E,CALLLIST)
         LA   R0,MQCC_OK        Load expected compcode
         C    R0,COMPCODE       Was inquire successful?
```

⋮

PL/I examples:

The use of PL/I is supported by z/OS only. This collection of topics demonstrates techniques using PL/I examples.

Connecting to a queue manager:

This example demonstrates how to use the MQCONN call to connect a program to a queue manager in z/OS batch.

This extract is not taken from the sample applications supplied with IBM MQ.

```
%INCLUDE SYSLIB(CMQP);
%INCLUDE SYSLIB(CMQEPP);
:
/*****************************************************/
/* STRUCTURE BASED ON PARAMETER INPUT AREA (PARAM)  */
/*****************************************************/
DCL 1 INPUT_PARAM      BASED(ADDR(PARAM)),
      2 PARAM_LENGTH   FIXED BIN(15),
      2 PARAM_MQMNAME  CHAR(48);
```

⋮

```
/*****************************************************/
/* WORKING STORAGE DECLARATIONS                      */
/*****************************************************/
DCL MQMNAME                  CHAR(48);
DCL COMPCODE                 BINARY FIXED (31);
DCL REASON                   BINARY FIXED (31);
DCL HCONN                    BINARY FIXED (31);
```

⋮

```
/*****************************************************/
/* COPY QUEUE MANAGER NAME PARAMETER                 */
/* TO LOCAL STORAGE                                  */
/*****************************************************/
MQMNAME = ' ';
MQMNAME = SUBSTR(PARAM_MQMNAME,1,PARAM_LENGTH);
```

⋮

```
/*****************************************************/
/* CONNECT FROM THE QUEUE MANAGER                    */
/*****************************************************/
CALL MQCONN (MQMNAME,     /* MQM SYSTEM NAME         */
             HCONN,       /* CONNECTION HANDLE       */
             COMPCODE,    /* COMPLETION CODE         */
             REASON);     /* REASON CODE             */
```

```
     /*****************************************************/
     /* TEST THE COMPLETION CODE OF THE CONNECT CALL.     */
     /* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE     */
     /* SHOWING THE COMPLETION CODE AND THE REASON CODE. */
     /*****************************************************/
     IF COMPCODE ¬= MQCC_OK
        THEN DO;

⋮
          CALL ERROR_ROUTINE;
        END;
```

Disconnecting from a queue manager:

This example demonstrates how to use the MQDISC call to disconnect a program from a queue manager in z/OS batch.

This extract is not taken from the sample applications supplied with IBM MQ.

```
     %INCLUDE SYSLIB(CMQP);
     %INCLUDE SYSLIB(CMQEPP);
     :
     /*****************************************************/
     /* WORKING STORAGE DECLARATIONS                      */
     /*****************************************************/
     DCL COMPCODE                  BINARY FIXED (31);
     DCL REASON                    BINARY FIXED (31);
     DCL HCONN                     BINARY FIXED (31);

⋮
     /*****************************************************/
     /* DISCONNECT FROM THE QUEUE MANAGER                 */
     /*****************************************************/
     CALL MQDISC (HCONN,       /* CONNECTION HANDLE      */
                  COMPCODE,    /* COMPLETION CODE        */
                  REASON);     /* REASON CODE            */

/************************************************************/
/* TEST THE COMPLETION CODE OF THE DISCONNECT CALL.         */
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE            */
/* SHOWING THE COMPLETION CODE AND THE REASON CODE.         */
/************************************************************/
     IF COMPCODE ¬= MQCC_OK
        THEN DO;

⋮
          CALL ERROR_ROUTINE;
        END;
```

Creating a dynamic queue:

This example demonstrates how to use the MQOPEN call to create a dynamic queue.

This extract is not taken from the sample applications supplied with IBM MQ.

```
     %INCLUDE SYSLIB(CMQP);
     %INCLUDE SYSLIB(CMQEPP);
     :
/*******************************************************/
/* WORKING STORAGE DECLARATIONS                        */
/*******************************************************/
DCL COMPCODE                 BINARY FIXED (31);
DCL REASON                   BINARY FIXED (31);
DCL HCONN                    BINARY FIXED (31);
DCL HOBJ                     BINARY FIXED (31);
DCL OPTIONS                  BINARY FIXED (31);
:
DCL MODEL_QUEUE_NAME     CHAR(48) INIT('PL1.REPLY.MODEL');
DCL DYNAMIC_NAME_PREFIX  CHAR(48) INIT('PL1.TEMPQ.*');
DCL DYNAMIC_QUEUE_NAME   CHAR(48) INIT(' ');
:
/*******************************************************/
/* LOCAL COPY OF OBJECT DESCRIPTOR                     */
/*******************************************************/
DCL 1 LMQOD  LIKE MQOD;
:
/*******************************************************/
/* SET UP OBJECT DESCRIPTOR FOR OPEN OF REPLY QUEUE    */
/*******************************************************/
LMQOD.OBJECTTYPE =MQOT_Q;
LMQOD.OBJECTNAME = MODEL_QUEUE_NAME;
LMQOD.DYNAMICQNAME = DYNAMIC_NAME_PREFIX;
OPTIONS = MQOO_INPUT_EXCLUSIVE;

     CALL MQOPEN (HCONN,
                  LMQOD,
                  OPTIONS,
                  HOBJ,
                  COMPCODE,
                  REASON);

/*******************************************************/
/* TEST THE COMPLETION CODE OF THE OPEN CALL.          */
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE       */
/* SHOWING THE COMPLETION CODE AND THE REASON CODE.    */
/* IF THE CALL HAS SUCCEEDED THEN EXTRACT THE NAME OF  */
/* THE NEWLY CREATED DYNAMIC QUEUE FROM THE OBJECT     */
/* DESCRIPTOR.                                         */
/*******************************************************/
     IF COMPCODE ¬= MQCC_OK
        THEN DO;

:
          CALL ERROR_ROUTINE;
        END;
        ELSE
          DYNAMIC_QUEUE_NAME = LMQOD_OBJECTNAME;
```

Opening an existing queue:

This example demonstrates how to use the MQOPEN call to open an existing queue.

This extract is not taken from the sample applications supplied with IBM MQ.

```
%INCLUDE SYSLIB(CMQP);
%INCLUDE SYSLIB(CMQEPP);
:
/*******************************************************/
/* WORKING STORAGE DECLARATIONS                        */
/*******************************************************/
DCL COMPCODE           BINARY FIXED (31);
DCL REASON             BINARY FIXED (31);
DCL HCONN              BINARY FIXED (31);
DCL HOBJ               BINARY FIXED (31);
DCL OPTIONS            BINARY FIXED (31);
:
DCL QUEUE_NAME         CHAR(48) INIT('PL1.LOCAL.QUEUE');
:
/*******************************************************/
/* LOCAL COPY OF OBJECT DESCRIPTOR                     */
/*******************************************************/
DCL 1 LMQOD  LIKE MQOD;
:
/*******************************************************/
/* SET UP OBJECT DESCRIPTOR FOR OPEN OF REPLY QUEUE    */
/*******************************************************/
LMQOD.OBJECTTYPE = MQOT_Q;
LMQOD.OBJECTNAME = QUEUE_NAME;
OPTIONS = MQOO_INPUT_EXCLUSIVE;

CALL MQOPEN (HCONN,
             LMQOD,
             OPTIONS,
             HOBJ,
             COMPCODE,
             REASON);

/*******************************************************/
/* TEST THE COMPLETION CODE OF THE OPEN CALL.          */
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE       */
/* SHOWING THE COMPLETION CODE AND THE REASON CODE.    */
/*******************************************************/
     IF COMPCODE ¬= MQCC_OK
        THEN DO;

:
          CALL ERROR_ROUTINE;
        END;
```

Closing a queue:

This example demonstrates how to use the MQCLOSE call.

This extract is not taken from the sample applications supplied with IBM MQ.

```
%INCLUDE SYSLIB(CMQP);
%INCLUDE SYSLIB(CMQEPP);
:
/*******************************************************/
/* WORKING STORAGE DECLARATIONS                        */
/*******************************************************/
DCL COMPCODE                 BINARY FIXED (31);
DCL REASON                   BINARY FIXED (31);
DCL HCONN                    BINARY FIXED (31);
DCL HOBJ                     BINARY FIXED (31);
DCL OPTIONS                  BINARY FIXED (31);
:
/*******************************************************/
/* SET CLOSE OPTIONS                                   */
/*******************************************************/
OPTIONS=MQCO_NONE;

/*******************************************************/
/* CLOSE QUEUE                                         */
/*******************************************************/
   CALL MQCLOSE (HCONN,      /* CONNECTION HANDLE       */
                 HOBJ,       /* OBJECT HANDLE           */
                 OPTIONS,    /* CLOSE OPTIONS           */
                 COMPCODE,   /* COMPLETION CODE         */
                 REASON);    /* REASON CODE             */

/*******************************************************/
/* TEST THE COMPLETION CODE OF THE CLOSE CALL.         */
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE       */
/* SHOWING THE COMPLETION CODE AND THE REASON CODE.    */
/*******************************************************/
     IF COMPCODE ¬= MQCC_OK
        THEN DO;

:
          CALL ERROR_ROUTINE;
        END;
```

Putting a message using MQPUT:

This example demonstrates how to use the MQPUT call using context.

This extract is not taken from the sample applications supplied with IBM MQ.

```
%INCLUDE SYSLIB(CMQP);
%INCLUDE SYSLIB(CMQEPP);
:
/*******************************************************/
/* WORKING STORAGE DECLARATIONS                        */
/*******************************************************/
DCL COMPCODE                 BINARY FIXED (31);
DCL REASON                   BINARY FIXED (31);
DCL HCONN                    BINARY FIXED (31);
DCL HOBJ                     BINARY FIXED (31);
DCL OPTIONS                  BINARY FIXED (31);
DCL BUFFLEN                  BINARY FIXED (31);
DCL BUFFER                   CHAR(80);
:
DCL PL1_TEST_MESSAGE         CHAR(80)
INIT('*****     THIS IS A TEST MESSAGE     *****');
```

```
⋮
*******************************************************/
/* LOCAL COPY OF MESSAGE DESCRIPTOR                    */
/* AND PUT MESSAGE OPTIONS                             */
/*******************************************************/
DCL 1 LMQMD  LIKE MQMD;
DCL 1 LMQPMO LIKE MQPMO;
⋮
/*******************************************************/
/* SET UP MESSAGE DESCRIPTOR                           */
/*******************************************************/
LMQMD.MSGTYPE = MQMT_DATAGRAM;
LMQMD.PRIORITY = 1;
LMQMD.PERSISTENCE = MQPER_PERSISTENT;
LMQMD.REPLYTOQ = ' ';
LMQMD.REPLYTOQMGR = ' ';
LMQMD.MSGID = MQMI_NONE;
LMQMD.CORRELID = MQCI_NONE;

/*******************************************************/
/* SET UP PUT MESSAGE OPTIONS                          */
/*******************************************************/
LMQPMO.OPTIONS = MQPMO_NO_SYNCPOINT;

/*******************************************************/
/* SET UP LENGTH OF MESSAGE BUFFER AND THE MESSAGE     */
/*******************************************************/
BUFFLEN = LENGTH(BUFFER);
BUFFER = PL1_TEST_MESSAGE;
/*******************************************************/
/*                                                     */
/* HCONN WAS SET BY PREVIOUS MQCONN REQUEST.           */
/* HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST.            */
/*                                                     */
/*******************************************************/
CALL MQPUT (HCONN,
            HOBJ,
            LMQMD,
            LMQPMO,
            BUFFLEN,
            BUFFER,
            COMPCODE,
            REASON);
/*******************************************************/
/* TEST THE COMPLETION CODE OF THE PUT CALL.           */
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE       */
/* SHOWING THE COMPLETION CODE AND THE REASON CODE.    */
/*******************************************************/
    IF COMPCODE ¬= MQCC_OK
       THEN DO;

⋮
         CALL ERROR_ROUTINE;
        END;
```

Putting a message using MQPUT1:

This example demonstrates how to use the MQPUT1 call.

This extract is not taken from the sample applications supplied with IBM MQ.

```
%INCLUDE SYSLIB(CMQEPP);
%INCLUDE SYSLIB(CMQP);
:
:
/*********************************************************/
/* WORKING STORAGE DECLARATIONS             */
/*********************************************************/
DCL COMPCODE      BINARY FIXED (31);
DCL REASON       BINARY FIXED (31);
DCL HCONN        BINARY FIXED (31);
DCL OPTIONS       BINARY FIXED (31);
DCL BUFFLEN       BINARY FIXED (31);
DCL BUFFER       CHAR(80);
:
DCL REPLY_TO_QUEUE   CHAR(48) INIT('PL1.REPLY.QUEUE');
DCL QUEUE_NAME     CHAR(48) INIT('PL1.LOCAL.QUEUE');
DCL PL1_TEST_MESSAGE  CHAR(80)
INIT('***** THIS IS ANOTHER TEST MESSAGE *****');
:
:
/*********************************************************/
/* LOCAL COPY OF OBJECT DESCRIPTOR, MESSAGE DESCRIPTOR  */
/* AND PUT MESSAGE OPTIONS                */
/*********************************************************/
DCL 1 LMQOD LIKE MQOD;
DCL 1 LMQMD LIKE MQMD;
DCL 1 LMQPMO LIKE MQPMO;
:
:
/*********************************************************/
/* SET UP OBJECT DESCRIPTOR AS REQUIRED.         */
/*********************************************************/
LMQOD.OBJECTTYPE = MQOT_Q;
LMQOD.OBJECTNAME = QUEUE_NAME;

/*********************************************************/
/* SET UP MESSAGE DESCRIPTOR AS REQUIRED.        */
/*********************************************************/
LMQMD.MSGTYPE = MQMT_REQUEST;
LMQMD.PRIORITY = 5;
LMQMD.PERSISTENCE = MQPER_PERSISTENT;
LMQMD.REPLYTOQ = REPLY_TO_QUEUE;
LMQMD.REPLYTOQMGR = ' ';
LMQMD.MSGID = MQMI_NONE;
LMQMD.CORRELID = MQCI_NONE;
/*********************************************************/
/* SET UP PUT MESSAGE OPTIONS AS REQUIRED        */
/*********************************************************/
LMQPMO.OPTIONS = MQPMO_NO_SYNCPOINT;

/*********************************************************/
/* SET UP LENGTH OF MESSAGE BUFFER AND THE MESSAGE    */
/*********************************************************/
BUFFLEN = LENGTH(BUFFER);
BUFFER = PL1_TEST_MESSAGE;

CALL MQPUT1 (HCONN,
LMQOD,
LMQMD,
LMQPMO,
BUFFLEN,
BUFFER,
COMPCODE,
```

```
REASON);

/*******************************************************/
/* TEST THE COMPLETION CODE OF THE PUT1 CALL.      */
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE SHOWING */
/* THE COMPLETION CODE AND THE REASON CODE.       */
/*******************************************************/
IF COMPCODE ¬= MQCC_OK
THEN DO;
⋮
CALL ERROR_ROUTINE;
END;
```

Getting a message:

This example demonstrates how to use the MQGET call to remove a message from a queue.

This extract is not taken from the sample applications supplied with IBM MQ.

```
     %INCLUDE SYSLIB(CMQP);
     %INCLUDE SYSLIB(CMQEPP);

⋮
/*******************************************************/
/* WORKING STORAGE DECLARATIONS                        */
/*******************************************************/
     DCL COMPCODE                 BINARY FIXED (31);
     DCL REASON                   BINARY FIXED (31);
     DCL HCONN                    BINARY FIXED (31);
     DCL HOBJ                     BINARY FIXED (31);
     DCL BUFFLEN                  BINARY FIXED (31);
     DCL DATALEN                  BINARY FIXED (31);
     DCL BUFFER                   CHAR(80);
⋮
/*******************************************************/
/* LOCAL COPY OF MESSAGE DESCRIPTOR AND                */
/* GET MESSAGE OPTIONS                                 */
/*******************************************************/
     DCL 1 LMQMD  LIKE MQMD;
     DCL 1 LMQGMO LIKE MQGMO;

⋮
/*******************************************************/
/* SET UP MESSAGE DESCRIPTOR AS REQUIRED.              */
/* MSGID AND CORRELID IN MQMD SET TO NULLS SO FIRST    */
/* AVAILABLE MESSAGE WILL BE RETRIEVED.                */
/*******************************************************/
     LMQMD.MSGID = MQMI_NONE;
     LMQMD.CORRELID = MQCI_NONE;

/*******************************************************/
/* SET UP GET MESSAGE OPTIONS AS REQUIRED.             */
/*******************************************************/
     LMQGMO.OPTIONS = MQGMO_NO_SYNCPOINT;

/*******************************************************/
/* SET UP LENGTH OF MESSAGE BUFFER.                    */
/*******************************************************/
     BUFFLEN = LENGTH(BUFFER);
/*******************************************************/
/*                                                     */
/* HCONN WAS SET BY PREVIOUS MQCONN REQUEST.           */
/* HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST.            */
/*                                                     */
/*******************************************************/
```

```
     CALL MQGET (HCONN,
                 HOBJ,
                 LMQMD,
                 LMQGMO,
                 BUFFERLEN,
                 BUFFER,
                 DATALEN,
                 COMPCODE,
                 REASON);

/*******************************************************/
/* TEST THE COMPLETION CODE OF THE GET CALL.           */
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE       */
/* SHOWING THE COMPLETION CODE AND THE REASON CODE.    */
/*******************************************************/
     IF COMPCODE ¬= MQCC_OK
        THEN DO;

⋮
          CALL ERROR_ROUTINE;
        END;
```

Getting a message using the wait option:

This example demonstrates how to use the MQGET call with the wait option and accepting truncated messages.

This extract is not taken from the sample applications supplied with IBM MQ.

```
     %INCLUDE SYSLIB(CMQP);
     %INCLUDE SYSLIB(CMQEPP);

⋮
/*******************************************************/
/* WORKING STORAGE DECLARATIONS                        */
/*******************************************************/
     DCL COMPCODE                 BINARY FIXED (31);
     DCL REASON                   BINARY FIXED (31);
     DCL HCONN                    BINARY FIXED (31);
     DCL HOBJ                     BINARY FIXED (31);
     DCL BUFFLEN                  BINARY FIXED (31);
     DCL DATALEN                  BINARY FIXED (31);
     DCL BUFFER                   CHAR(80);

⋮
/*******************************************************/
/* LOCAL COPY OF MESSAGE DESCRIPTOR AND GET MESSAGE    */
/* OPTIONS                                             */
/*******************************************************/
     DCL 1 LMQMD  LIKE MQMD;
     DCL 1 LMQGMO LIKE MQGMO;

⋮
/*******************************************************/
/* SET UP MESSAGE DESCRIPTOR AS REQUIRED.              */
/* MSGID AND CORRELID IN MQMD SET TO NULLS SO FIRST    */
/* AVAILABLE MESSAGE WILL BE RETRIEVED.                */
/*******************************************************/
     LMQMD.MSGID = MQMI_NONE;
     LMQMD.CORRELID = MQCI_NONE;

/*******************************************************/
/* SET UP GET MESSAGE OPTIONS AS REQUIRED.             */
/* WAIT INTERVAL SET TO ONE MINUTE.                    */
/*******************************************************/
     LMQGMO.OPTIONS = MQGMO_WAIT +
```

```
                    MQGMO_ACCEPT_TRUNCATED_MSG +
                    MQGMO_NO_SYNCPOINT;
     LMQGMO.WAITINTERVAL=60000;

/*******************************************************/
/* SET UP LENGTH OF MESSAGE BUFFER.                    */
/*******************************************************/
     BUFFLEN = LENGTH(BUFFER);
/*******************************************************/
/*                                                     */
/* HCONN WAS SET BY PREVIOUS MQCONN REQUEST.           */
/* HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST.            */
/*                                                     */
/*******************************************************/

     CALL MQGET (HCONN,
                 HOBJ,
                 LMQMD,
                 LMQGMO,
                 BUFFERLEN,
                 BUFFER,
                 DATALEN,
                 COMPCODE,
                 REASON);

/*******************************************************/
/* TEST THE COMPLETION CODE OF THE GET CALL.           */
/* TAKE APPROPRIATE ACTION BASED ON COMPLETION CODE AND  */
/* REASON CODE.                                        */
/*******************************************************/

     SELECT(COMPCODE);
       WHEN (MQCC_OK) DO;   /* GET WAS SUCCESSFUL */

⋮
       END;
       WHEN (MQCC_WARNING) DO;
         IF REASON = MQRC_TRUNCATED_MSG_ACCEPTED
           THEN DO;         /* GET WAS SUCCESSFUL */

⋮
           END;
           ELSE DO;

⋮
             CALL ERROR_ROUTINE;
           END;
       END;
       WHEN (MQCC_FAILED) DO;

⋮
           CALL ERROR_ROUTINE;
         END;
       END;
       OTHERWISE;
     END;
```

Getting a message using signaling:

A code extract that demonstrates how to use the MQGET call with signaling.

Signaling is available only with IBM MQ for z/OS .

This extract is not taken from the sample applications supplied with IBM MQ.

```
     %INCLUDE SYSLIB(CMQP);
     %INCLUDE SYSLIB(CMQEPP);
     :
/*******************************************************/
/* WORKING STORAGE DECLARATIONS                        */
/*******************************************************/
     DCL COMPCODE                 BINARY FIXED (31);
     DCL REASON                   BINARY FIXED (31);
     DCL HCONN                    BINARY FIXED (31);
     DCL HOBJ                     BINARY FIXED (31);
     DCL DATALEN                  BINARY FIXED (31);
     DCL BUFFLEN                  BINARY FIXED (31);
     DCL BUFFER                   CHAR(80);

.
.
.
     DCL ECB_FIXED           FIXED BIN(31);
     DCL 1 ECB_OVERLAY BASED(ADDR(ECB_FIXED)),
           3 ECB_WAIT    BIT,
           3 ECB_POSTED  BIT,
           3 ECB_FLAG3_8 BIT(6),
           3 ECB_CODE  PIC'999';

.
.
.
/*******************************************************/
/* LOCAL COPY OF MESSAGE DESCRIPTOR AND GET MESSAGE    */
/* OPTIONS                                             */
/*******************************************************/
     DCL 1 LMQMD  LIKE MQMD;
     DCL 1 LMQGMO LIKE MQGMO;

.
.
.
/*******************************************************/
/* CLEAR ECB FIELD.                                    */
/*******************************************************/
     ECB_FIXED = 0;

.
.
.
/*******************************************************/
/* SET UP MESSAGE DESCRIPTOR AS REQUIRED.              */
/* MSGID AND CORRELID IN MQMD SET TO NULLS SO FIRST    */
/* AVAILABLE MESSAGE WILL BE RETRIEVED.                */
/*******************************************************/
     LMQMD.MSGID = MQMI_NONE;
     LMQMD.CORRELID = MQCI_NONE;
/*******************************************************/
/* SET UP GET MESSAGE OPTIONS AS REQUIRED.             */
/* WAIT INTERVAL SET TO ONE MINUTE.                    */
/*******************************************************/
     LMQGMO.OPTIONS = MQGMO_SET_SIGNAL +
                      MQGMO_NO_SYNCPOINT;
     LMQGMO.WAITINTERVAL=60000;
     LMQGMO.SIGNAL1 = ADDR(ECB_FIXED);
/*******************************************************/
/* SET UP LENGTH OF MESSAGE BUFFER.                    */
/* CALL MESSAGE RETRIEVAL ROUTINE.                     */
/*******************************************************/
     BUFFLEN = LENGTH(BUFFER);
```

```
     CALL GET_MSG;

/*******************************************************/
/* TEST THE COMPLETION CODE OF THE GET CALL.           */
/* TAKE APPROPRIATE ACTION BASED ON COMPLETION CODE AND  */
/* REASON CODE.                                        */
/*******************************************************/

     SELECT;
       WHEN ((COMPCODE = MQCC_OK) &
             (REASON = MQCC_NONE)) DO

.
.
.
         CALL MSG_ROUTINE;

.
.
.
       END;
       WHEN ((COMPCODE = MQCC_WARNING) &
             (REASON = MQRC_SIGNAL_REQUEST_ACCEPTED)) DO;

.
.
.
         CALL DO_WORK;

.
.
.
       END;
       WHEN  ((COMPCODE = MQCC_FAILED) &
             (REASON = MQRC_SIGNAL_OUTSTANDING)) DO;

.
.
.
         CALL DO_WORK;

.
.
.
       END;
       OTHERWISE DO;        /* FAILURE CASE */
/*******************************************************/
/* ISSUE AN ERROR MESSAGE SHOWING THE COMPLETION CODE    */
/* AND THE REASON CODE.                                */
/*******************************************************/

.
.
.
          CALL ERROR_ROUTINE;

.
.
.
       END;
     END;

.
.
.
 DO_WORK: PROC;

.
.
.
     IF ECB_POSTED
        THEN DO;
          SELECT(ECB_CODE);
            WHEN(MQEC_MSG_ARRIVED) DO;

.
.
.
              CALL GET_MSG;

.
.
.
            END;
            WHEN(MQEC_WAIT_INTERVAL_EXPIRED) DO;

.
.
.
              CALL NO_MSG;
```

```
:
           END;
           OTHERWISE DO;        /* FAILURE CASE */
/*******************************************************/
/* ISSUE AN ERROR MESSAGE SHOWING THE COMPLETION CODE  */
/* AND THE REASON CODE.                                */
/*******************************************************/

:
             CALL ERROR_ROUTINE;

:
          END;

         END;

       END;

:
 END DO_WORK;

 GET_MSG: PROC;
/*******************************************************/
/*                                                     */
/* HCONN WAS SET BY PREVIOUS MQCONN REQUEST.           */
/* HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST.            */
/* MD AND GMO SET UP AS REQUIRED.                      */
/*                                                     */
/*******************************************************/

     CALL MQGET (HCONN,
                 HOBJ,
                 LMQMD,
                 LMQGMO,
                 BUFFLEN,
                 BUFFER,
                 DATALEN,
                 COMPCODE,
                 REASON);

 END GET_MSG;

 NO_MSG: PROC;

:
 END NO_MSG;
```

Inquiring about the attributes of an object:

This example demonstrates how to use the MQINQ call to inquire about the attributes of a queue.

This extract is not taken from the sample applications supplied with IBM MQ.

```
     %INCLUDE SYSLIB(CMQP);
     %INCLUDE SYSLIB(CMQEPP);
     :
/*******************************************************/
/* WORKING STORAGE DECLARATIONS                        */
/*******************************************************/
     DCL COMPCODE                 BINARY FIXED (31);
     DCL REASON                   BINARY FIXED (31);
     DCL HCONN                    BINARY FIXED (31);
     DCL HOBJ                     BINARY FIXED (31);
     DCL OPTIONS                  BINARY FIXED (31);
     DCL SELECTORCOUNT            BINARY FIXED (31);
     DCL INTATTRCOUNT             BINARY FIXED (31);
     DCL 1 SELECTOR_TABLE,
         3 SELECTORS(5)           BINARY FIXED (31);
     DCL 1 INTATTR_TABLE,
         3 INTATTRS(5)            BINARY FIXED (31);
     DCL CHARATTRLENGTH           BINARY FIXED (31);
     DCL CHARATTRS                CHAR(100);

⋮

/*******************************************************/
/* SET VARIABLES FOR INQUIRE CALL                      */
/* INQUIRE ON THE CURRENT QUEUE DEPTH                  */
/*******************************************************/

     SELECTORS(01) = MQIA_CURRENT_Q_DEPTH;

     SELECTORCOUNT  = 1;
     INTATTRCOUNT   = 1;

     CHARATTRLENGTH = 0;
/*******************************************************/
/*                                                     */
/* HCONN WAS SET BY PREVIOUS MQCONN REQUEST.           */
/* HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST.            */
/*                                                     */
/*******************************************************/
     CALL MQINQ (HCONN,
                 HOBJ,
                 SELECTORCOUNT,
                 SELECTORS,
                 INTATTRCOUNT,
                 INTATTRS,
                 CHARATTRLENGTH,
                 CHARATTRS,
                 COMPCODE,
                 REASON);
/*******************************************************/
/* TEST THE COMPLETION CODE OF THE INQUIRE CALL.       */
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE SHOWING */
/* THE COMPLETION CODE AND THE REASON CODE.            */
/*******************************************************/
     IF COMPCODE ¬= MQCC_OK
        THEN DO;

⋮

          CALL ERROR_ROUTINE;
        END;
```

Setting the attributes of a queue:

This example demonstrates how to use the MQSET call to change the attributes of a queue.

This extract is not taken from the sample applications supplied with IBM MQ.

```
     %INCLUDE SYSLIB(CMQP);
     %INCLUDE SYSLIB(CMQEPP);
     :
/*******************************************************/
/* WORKING STORAGE DECLARATIONS                        */
/*******************************************************/
     DCL COMPCODE                 BINARY FIXED (31);
     DCL REASON                   BINARY FIXED (31);
     DCL HCONN                    BINARY FIXED (31);
     DCL HOBJ                     BINARY FIXED (31);
     DCL OPTIONS                  BINARY FIXED (31);
     DCL SELECTORCOUNT            BINARY FIXED (31);
     DCL INTATTRCOUNT             BINARY FIXED (31);
     DCL 1 SELECTOR_TABLE,
         3 SELECTORS(5)           BINARY FIXED (31);
     DCL 1 INTATTR_TABLE,
         3 INTATTRS(5)            BINARY FIXED (31);
     DCL CHARATTRLENGTH           BINARY FIXED (31);
     DCL CHARATTRS                CHAR(100);

⋮

/*******************************************************/
/* SET VARIABLES FOR SET CALL                          */
/* SET GET AND PUT INHIBITED                           */
/*******************************************************/

     SELECTORS(01) = MQIA_INHIBIT_GET;
     SELECTORS(02) = MQIA_INHIBIT_PUT;

     INTATTRS(01) = MQQA_GET_INHIBITED;
     INTATTRS(02) = MQQA_PUT_INHIBITED;

     SELECTORCOUNT  = 2;
     INTATTRCOUNT   = 2;

     CHARATTRLENGTH = 0;
/*******************************************************/
/*                                                     */
/* HCONN WAS SET BY PREVIOUS MQCONN REQUEST.           */
/* HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST.            */
/*                                                     */
/*******************************************************/
     CALL MQSET (HCONN,
                 HOBJ,
                 SELECTORCOUNT,
                 SELECTORS,
                 INTATTRCOUNT,
                 INTATTRS,
                 CHARATTRLENGTH,
                 CHARATTRS,
                 COMPCODE,
                 REASON);

/*******************************************************/
/* TEST THE COMPLETION CODE OF THE SET CALL.           */
/* IF THE CALL HAS FAILED ISSUE AN ERROR MESSAGE SHOWING */
/* THE COMPLETION CODE AND THE REASON CODE.            */
/*******************************************************/
     IF COMPCODE ¬= MQCC_OK
```

```
        THEN DO;
⋮
          CALL ERROR_ROUTINE;
        END;
```

Constants

Use the reference information in this section to accomplish the tasks that address your business needs.

IBM MQ COPY, header, include, and module files:

This information is general-use programming interface information.

This section contains information to help you use the MQI for various programming languages, as follows.

C header files:

Header files are provided to help you write C application programs that use the MQI. These header files are summarized in the table:

Table 159. C header files - call prototypes, data types, return codes, constants, and structures

File name	Description	IBM i	UNIX and Linux systems	Windows	z/OS
Call prototypes, data types, return codes, constants, and structures					
CMQC	MQI definitions	C	C	C	C
CMQBC	MQAI definitions	C	C	C	
CMQEC	Interface Entry Points definition (includes CMQC, CMQXC and CMQZC)		C	C	
CMQCFC	PCF definitions	C	C	C	C
CMQPSC	Publish/Subscribe definitions	C	C	C	C
CMQXC	Channel and exit definitions	C	C	C	C
CMQZC	Installable services definitions	C	C	C	
Key: C= Files provided					

COBOL COPY files:

Various COPY files are provided to help you write COBOL application programs that use the MQI. These files are summarized in the table:

Table 160. COBOL copy files - return codes, constants, and structures

File name	Description	IBM i	UNIX	Windows	z/OS
Return codes and constants					
CMQx	MQI definitions	V	V	V	V
CMQCFx	PCF definitions	V	V	V	V
CMQPSx	Publish/Subscribe definitions	V	V	V	V
CMQXx	Channel and exit definitions	V	V	V	V
Structures					

Table 160. COBOL copy files - return codes, constants, and structures (continued)

File name	Description	IBM i	UNIX	Windows	z/OS
CMQAIRx	MQAIR - Authentication information record		V L	V L	
CMQBOx	MQBO - Begin options	V L	V L	V L	
CMQCDx	MQCD - Channel definition	V L	V L	V L	V L
CMQCFBFx	MQCFBF - PCF byte string filter parameter	V L	V L	V L	V L
CMQCFBSx	MQCFBS - PCF byte string parameter	V L	V L	V L	V L
CMQCFGRx	MQCFGR - PCF group parameter	V L	V L	V L	V L
CMQCFHx	MQCFH - PCF header	V L	V L	V L	V L
CMQCFIFx	MQCFIF - PCF integer filter parameter	V L	V L	V L	V L
CMQCFILx	MQCFIL - PCF integer list parameter	V L	V L	V L	V L
CMQCFINx	MQCFIN - PCF integer parameter	V L	V L	V L	V L
CMQCFSFx	MQCFSF - PCF string filter parameter	V L	V L	V L	V L
CMQCFSLx	MQCFSL - PCF string list parameter	V L	V L	V L	V L
CMQCFSTx	MQCFST - PCF string parameter	V L	V L	V L	V L
CMQCFXLx	MQCFIL64 - PCF 64-bit integer list parameter	V L	V L	V L	V L
CMQCFXNx	MQCFIN64 - PCF 64-bit integer parameter	V L	V L	V L	V L
CMQCHRVx	MQCHARV - Variable length string	V L	V L	V L	V L
CMQCIHx	MQCIH - CICS bridge header	V L	V L	V L	V L
CMQCNOx	MQCNO - Connect options	V L	V L	V L	V L
CMQCSPx	MQCSP - Security parameters	V L	V L	V L	V L
CMQCXPx	MQCXP - Channel exit parameters	V L			V L
CMQDHx	MQDH - Distribution header	V L	V L	V L	V L
CMQDLHx	MQDLH - Dead-letter header	V L	V L	V L	V L
CMQDXPx	MQDXP - Data conversion exit parameters	V L		V L	
CMQEPHx	MQEPH - Embedded PCF header	V L	V L	V L	V L
CMQGMOx	MQGMO - Get message options	V L	V L	V L	V L
CMQIIHx	MQIIH - IMS information header	V L	V L	V L	V L
CMQMDx	MQMD - Message descriptor	V L	V L	V L	V L
CMQMD1x	MQMD1 - Message descriptor version 1	V L	V L	V L	V L
CMQMD2x	MQMD2 - Message descriptor version 2	V L	V L	V L	V L
CMQMDEx	MQMDE - Message descriptor extended	V L	V L	V L	V L
CMQODx	MQOD - Object descriptor	V L	V L	V L	V L

Table 160. COBOL copy files - return codes, constants, and structures (continued)

File name	Description	IBM i	UNIX	Windows	z/OS
CMQORx	MQOR - Object record	V L	V L	V L	V L
CMQPMOx	MQPMO - Put message options	V L	V L	V L	V L
CMQRFHx	MQRFH - Rules and formatting header	V L	V L	V L	V L
CMQRFH2x	MQRFH2 - Rules and formatting header 2	V L	V L	V L	V L
CMQRMHx	MQRMH - Reference message header	V L	V L	V L	V L
CMQRRx	MQRR - Response record	V L	V L	V L	
CMQSCOx	MQSCO - TLS configuraton options		V L	V L	
CMQTMx	MQTM - Trigger message	V L		V L	V L
CMQTMCx	MQTMC - Trigger message character	V L	V L		
CMQTMC2x	MQTMC2 - Trigger message 2 character	V L	V L	V L	V L
CMQWIHx	MQWIH - Work information header	V L	V L	V L	V L
CMQXQHx	MQXQH - Transmission queue header	V L	V L	V L	V L
Key: • Files with initial values provided, x=V • Files without initial values provided, x=L					

PL/I include files:

The following INCLUDE files are provided for the PL/I programming language. These files are available on z/OS only.

Table 161. PL/I include files - data types, return codes, constants, and structures

File name	Description	IBM i	UNIX	Windows	z/OS
Data types, return codes, constants, and structures					
CMQP	MQI definitions				P
CMQCFP	PCF definitions				P
CMQEPP	Entry point definitions				P
CMQPSP	Publish/Subscribe definitions				P
CMQXP	Channel and exit definitions				P
Key: P= File provided					

RPG copy files:

The following COPY files are provided for the RPG programming language. These files are available only on IBM i.

Table 162. RPG copy files - return codes, constants, and structures

File name	Description	IBM i	UNIX	Windows	z/OS
Return codes and constants					
CMQx	MQI definitions	G R			
CMQCFx	PCF definitions	G			
CMQPSx	Publish/Subscribe definitions	G			
CMQXx	Channel and exit definitions	G R			
Structures					
CMQBOx	MQBO - Begin options	G H			
CMQCDx	MQCD - Channel definition	G H R			
CMQCFBFx	MQCFBF - PCF byte string filter parameter	G H			
CMQCFBSx	MQCFBS - PCF byte string parameter	G H			
CMQCFGRx	MQCFGR - PCF group parameter	G H			
CMQCFHx	MQCFH - PCF header	G H			
CMQCFIFx	MQCFIF - PCF integer filter parameter	G H			
CMQCFILx	MQCFIL - PCF integer list parameter	G H			
CMQCFINx	MQCFIN - PCF integer parameter	G H			
CMQCFSFx	MQCFSF - PCF string filter parameter	G H			
CMQCFSLx	MQCFSL - PCF string list parameter	G H			
CMQCFSTx	MQCFST - PCF string parameter	G H			
CMQCFXLx	MQCFIL64 - PCF 64-bit integer list parameter	G H			
CMQCFXNx	MQCFIN64 - PCF 64-bit integer parameter	G H			
CMQCHARVx	MQCHARV - Variable length string	G H			
CMQCIHx	MQCIH - CICS bridge header	G H			
CMQCNOx	MQCNO - Connect options	G H			
CMQCSPx	MQCSP - Security parameters	G H			
CMQCXPx	MQCXP - Channel exit parameters	G H R			
CMQDHx	MQDH - Distribution header	G H R			
CMQDLHx	MQDLH - Dead-letter header	G H R			
CMQDXPx	MQDXP - Data conversion exit parameters	G H R			
CMQEPHx	MQEPH - Embedded PCF header	G H			
CMQGMOx	MQGMO - Get message options	G H R			
CMQIIHx	MQIIH - IMS information header	G H R			
CMQMDx	MQMD - Message descriptor	G H R			
CMQMD1x	MQMD1 - Message descriptor version 1	G H R			

Table 162. RPG copy files - return codes, constants, and structures (continued)

File name	Description	IBM i	UNIX	Windows	z/OS
CMQMD2x	MQMD2 - Message descriptor version 2	G H			
CMQMDEx	MQMDE - Message descriptor extended	G H R			
CMQODx	MQOD - Object descriptor	G H R			
CMQORx	MQOR - Object record	G H R			
CMQPMOx	MQPMO - Put message options	G H R			
CMQPXPx	MQPXP - Publish/Subscribe routing exit parameters	G H			
CMQRFHx	MQRFH - Rules and formatting header	G H			
CMQRFH2x	MQRFH2 - Rules and formatting header 2	G H			
CMQRMHx	MQRMH - Reference message header	G H R			
CMQRRx	MQRR - Response record	G H R			
CMQTMx	MQTM - Trigger message	G H R			
CMQTMCx	MQTMC - Trigger message character	G H R			
CMQTMC2x	MQTMC2 - Trigger message 2 character	G H R			
CMQWIHx	MQWIH - Work information header	G H			
CMQXQHx	MQXQH - Transmission queue header	G H R			

Key:
- File for static linkage, initialized, provided x=G
- File for static linkage, not initialized, provided x=H
- File for dynamic linkage, initialized, provided, x=R

Visual Basic module files:

Header (or form) files are provided to help you write Visual Basic application programs that use the MQI. These header files are supplied in 32-bit versions only and are summarized in the table:

Table 163. Visual Basic module files - call declarations, data types, return codes, constants, and structures

File name	Description	IBM i	UNIX and Linux systems	Windows	z/OS
Call declarations, data types, return codes, constants, and structures					
CMQB	MQI definitions			B	
CMQBB	MQAI definitions			B	
CMQCFB	PCF definitions			B	
CMQXB	Channel and exit definitions			B	
Key: B= File provided					

z/OS Assembler COPY files:

Various COPY files are provided to help you write z/OS Assembler application programs that use the MQI. These files are summarized in the table:

Table 164. z/OS Assembler copy files - data types, return codes, constants, and structures

File name	Description	IBM i	UNIX	Windows	z/OS
Data types, return codes, and constants					
CMQA	MQI definitions				A
CMQCFA	PCF definitions				A
CMQPSA	Publish/Subscribe definitions				A
CMQVERA	Structure version control				A
CMQXA	Channel and exit definitions				A
Structures					
CMQCDA	MQCD - Channel definition				
CMQCFBFA	MQCFBF - PCF byte string filter parameter				
CMQCFBSA	MQCFBS - PCF byte string parameter				A
CMQCFGRA	MQCFGR - PCF group parameter				A
CMQCFHA	MQCFH - PCF header				A
CMQCFIFA	MQCFIF - PCF integer filter parameter				A
CMQCFILA	MQCFIL - PCF integer list parameter				A
CMQCFINA	MQCFIN - PCF integer parameter				A
CMQCFSFA	MQCFSF - PCF string filter parameter				A
CMQCFSLA	MQCFSL - PCF string list parameter				A
CMQCFSTA	MQCFST - PCF string parameter				A
CMQCFXLA	MQCFIL64 - PCF 64-bit integer list parameter				A
CMQCFXNA	MQCFIN64 - PCF 64-bit integer parameter				A
CMQCHARVA	MQCHARV - Variable length string				A
CMQCIHA	MQCIH - CICS bridge header				A
CMQCNOA	MQCNO - Connect options				A
CMQCSPA	MQCSP - Security parameters				A
CMQCXPA	MQCXP - Channel exit parameters				A
CMQDHA	MQDH - Distribution header				A
CMQDLHA	MQDLH - Dead-letter header				A
CMQDXPA	MQDXP - Data conversion exit parameters				A
CMQEPHA	MQEPH - Embedded PCF header				A
CMQGMOA	MQGMO - Get message options				A
CMQIIHA	MQIIH - IMS information header				A

Table 164. z/OS Assembler copy files - data types, return codes, constants, and structures (continued)

File name	Description	IBM i	UNIX	Windows	z/OS
CMQMDA	MQMD - Message descriptor				A
CMQMD1A	MQMD1 - Message descriptor version 1				A
CMQMD2A	MQMD2 - Message descriptor version 2				A
CMQMDEA	MQMDE - Message descriptor extended				A
CMQODA	MQOD - Object descriptor				A
CMQORA	MQOR - Object record				A
CMQPMOA	MQPMO - Put message options				A
CMQRFHA	MQRFH - Rules and formatting header				A
CMQRFH2A	MQRFH2 - Rules and formatting header 2				A
CMQRMHA	MQRMH - Reference message header				A
CMQTMA	MQTM - Trigger message				A
CMQTMC2A	MQTMC2 - Trigger message 2 character				A
CMQWCRA	MQWCR - Cluster workload cluster record				A
CMQWDRA	MQWDR - Cluster workload destination record				A
CMQWDR1A	MQWDR1 - Cluster workload destination record version 1				A
CMQWDR2A	MQWDR2 - Cluster workload destination record version 2				A
CMQWIHA	MQWIH - Work information header				A
CMQWQRA	MQWQR - Cluster workload queue record				A
CMQWQR1A	MQWQR1 - Cluster workload queue record version 1				A
CMQWQR2A	MQWQR2 - Cluster workload queue record version 2				A
CMQWXPA	MQWXP - Cluster workload exit parameters				A
CMQWXP1A	MQWXP1 - Cluster workload exit parameters version 1				A
CMQWXP2A	MQWXP2 - Cluster workload exit parameters version 2				A
CMQWXP3A	MQWXP3 - Cluster workload exit parameters version 3				A
CMQXPA	MQXP - CICS API-crossing exit parameters				A
CMQXQHA	MQXQH - Transmission queue header				A

Table 164. z/OS Assembler copy files - data types, return codes, constants, and structures (continued)

File name	Description	IBM i	UNIX	Windows	z/OS
CMQXWDA	MQXWD - Exit wait descriptor				A
Key: A= File provided					

Constants:

List of all the constants

MQ_* (String Lengths):

MQ_ABEND_CODE_LENGTH	4	X'00000004'
MQ_ACCOUNTING_TOKEN_LENGTH	32	X'00000020'
MQ_APPL_IDENTITY_DATA_LENGTH	32	X'00000020'
MQ_APPL_NAME_LENGTH	28	X'0000001C'
MQ_APPL_ORIGIN_DATA_LENGTH	4	X'00000004'
MQ_APPL_TAG_LENGTH	28	X'0000001C'
MQ_ARM_SUFFIX_LENGTH	2	X'00000002'
MQ_ATTENTION_ID_LENGTH	4	X'00000004'
MQ_AUTH_INFO_CONN_NAME_LENGTH	264	X'00000108'
MQ_AUTH_INFO_DESC_LENGTH	64	X'00000040'
MQ_AUTH_INFO_NAME_LENGTH	48	X'00000030'
MQ_AUTH_INFO_OCSP_URL_LENGTH	256	X'00000100'
MQ_AUTHENTICATOR_LENGTH	8	X'00000008'
MQ_AUTO_REORG_CATALOG_LENGTH	44	X'0000002C'
MQ_AUTO_REORG_TIME_LENGTH	4	X'00000004'
MQ_BATCH_INTERFACE_ID_LENGTH	8	X'00000008'
MQ_BRIDGE_NAME_LENGTH	24	X'00000018'
MQ_CANCEL_CODE_LENGTH	4	X'00000004'
MQ_CF_STRUC_DESC_LENGTH	64	X'00000040'
MQ_CF_STRUC_NAME_LENGTH	12	X'0000000C'
MQ_CHANNEL_DATE_LENGTH	12	X'0000000C'
MQ_CHANNEL_DESC_LENGTH	64	X'00000040'
MQ_CHANNEL_NAME_LENGTH	20	X'00000014'
MQ_CHANNEL_TIME_LENGTH	8	X'00000008'
MQ_CHINIT_SERVICE_PARM_LENGTH	32	X'00000020'
MQ_CICS_FILE_NAME_LENGTH	8	X'00000008'
MQ_CLIENT_ID_LENGTH	23	X'00000017'
MQ_CLUSTER_NAME_LENGTH	48	X'00000030'
MQ_CONN_NAME_LENGTH	264	X'00000108'
MQ_CONN_TAG_LENGTH	128	X'00000080'
MQ_CONNECTION_ID_LENGTH	24	X'00000018'
MQ_CORREL_ID_LENGTH	24	X'00000018'

MQ_CREATION_DATE_LENGTH	12	X'0000000C'
MQ_CREATION_TIME_LENGTH	8	X'00000008'
MQ_DATE_LENGTH	12	X'0000000C'
MQ_DISTINGUISHED_NAME_LENGTH	1024	X'00000400'
MQ_DNS_GROUP_NAME_LENGTH	18	X'00000012'
MQ_EXIT_DATA_LENGTH	32	X'00000020'
MQ_EXIT_INFO_NAME_LENGTH	48	X'00000030'
MQ_EXIT_NAME_LENGTH	(value differs by platform or version)	
MQ_EXIT_PD_AREA_LENGTH	48	X'00000030'
MQ_EXIT_USER_AREA_LENGTH	16	X'00000010'
MQ_FACILITY_LENGTH	8	X'00000008'
MQ_FACILITY_LIKE_LENGTH	4	X'00000004'
MQ_FORMAT_LENGTH	8	X'00000008'
MQ_FUNCTION_LENGTH	4	X'00000004'
MQ_GROUP_ID_LENGTH	24	X'00000018'
MQ_LDAP_PASSWORD_LENGTH	32	X'00000020'
MQ_LISTENER_NAME_LENGTH	48	X'00000030'
MQ_LISTENER_DESC_LENGTH	64	X'00000040'
MQ_LOCAL_ADDRESS_LENGTH	48	X'00000030'
MQ_LTERM_OVERRIDE_LENGTH	8	X'00000008'
MQ_LU_NAME_LENGTH	8	X'00000008'
MQ_LUWID_LENGTH	16	X'00000010'
MQ_MAX_EXIT_NAME_LENGTH	128	X'00000080'
MQ_MAX_MCA_USER_ID_LENGTH	64	X'00000040'
MQ_MAX_PROPERTY_NAME_LENGTH	4095	X'00000FFF'
MQ_MAX_USER_ID_LENGTH	64	X'00000040'
MQ_MCA_JOB_NAME_LENGTH	28	X'0000001C'
MQ_MCA_NAME_LENGTH	20	X'00000014'
MQ_MCA_USER_DATA_LENGTH	32	X'00000020'
MQ_MCA_USER_ID_LENGTH	(value differs by platform or version)	
MQ_MFS_MAP_NAME_LENGTH	8	X'00000008'
MQ_MODE_NAME_LENGTH	8	X'00000008'
MQ_MSG_HEADER_LENGTH	4000	X'00000FA0'
MQ_MSG_ID_LENGTH	24	X'00000018'
MQ_MSG_TOKEN_LENGTH	16	X'00000010'
MQ_NAMELIST_DESC_LENGTH	64	X'00000040'
MQ_NAMELIST_NAME_LENGTH	48	X'00000030'
MQ_OBJECT_INSTANCE_ID_LENGTH	24	X'00000018'
MQ_OBJECT_NAME_LENGTH	48	X'00000030'
MQ_PASS_TICKET_APPL_LENGTH	8	X'00000008'

MQ_PASSWORD_LENGTH	12	X'0000000C'
MQ_PROCESS_APPL_ID_LENGTH	256	X'00000100'
MQ_PROCESS_DESC_LENGTH	64	X'00000040'
MQ_PROCESS_ENV_DATA_LENGTH	128	X'00000080'
MQ_PROCESS_NAME_LENGTH	48	X'00000030'
MQ_PROCESS_USER_DATA_LENGTH	128	X'00000080'
MQ_PROGRAM_NAME_LENGTH	20	X'00000014'
MQ_PUT_APPL_NAME_LENGTH	28	X'0000001C'
MQ_PUT_DATE_LENGTH	8	X'00000008'
MQ_PUT_TIME_LENGTH	8	X'00000008'
MQ_Q_DESC_LENGTH	64	X'00000040'
MQ_Q_MGR_DESC_LENGTH	64	X'00000040'
MQ_Q_MGR_IDENTIFIER_LENGTH	48	X'00000030'
MQ_Q_MGR_NAME_LENGTH	48	X'00000030'
MQ_Q_NAME_LENGTH	48	X'00000030'
MQ_QSG_NAME_LENGTH	4	X'00000004'
MQ_REMOTE_SYS_ID_LENGTH	4	X'00000004'
MQ_SECURITY_ID_LENGTH	40	X'00000028'
MQ_SELECTOR_LENGTH	10240	X'00002800'
MQ_SERVICE_ARGS_LENGTH	255	X'000000FF'
MQ_SERVICE_COMMAND_LENGTH	255	X'000000FF'
MQ_SERVICE_DESC_LENGTH	64	X'00000040'
MQ_SERVICE_NAME_LENGTH	32	X'00000020'
MQ_SERVICE_PATH_LENGTH	255	X'000000FF'
MQ_SERVICE_STEP_LENGTH	8	X'00000008'
MQ_SHORT_CONN_NAME_LENGTH	20	X'00000014'
MQ_SHORT_DNAME_LENGTH	256	X'00000100'
MQ_SSL_CIPHER_SPEC_LENGTH	32	X'00000020'
MQ_SSL_CRYPTO_HARDWARE_LENGTH	256	X'00000100'
MQ_SSL_HANDSHAKE_STAGE_LENGTH	32	X'00000020'
MQ_SSL_KEY_LIBRARY_LENGTH	44	X'0000002C'
MQ_SSL_KEY_MEMBER_LENGTH	8	X'00000008'
MQ_SSL_KEY_REPOSITORY_LENGTH	256	X'00000100'
MQ_SSL_PEER_NAME_LENGTH	1024	X'00000400'
MQ_SSL_SHORT_PEER_NAME_LENGTH	256	X'00000100'
MQ_START_CODE_LENGTH	4	X'00000004'
MQ_STORAGE_CLASS_DESC_LENGTH	64	X'00000040'
MQ_STORAGE_CLASS_LENGTH	8	X'00000008'
MQ_SUB_IDENTITY_LENGTH	128	X'00000080'
MQ_SUB_POINT_LENGTH	128	X'00000080'
MQ_SUITE_B_128_BIT	2	X'00000002'
MQ_SUITE_B_192_BIT	4	X'00000004'

MQ_SUITE_B_NONE	1	X'00000001'
MQ_SUITE_B_NOT_AVAILABLE	0	X'00000000'
MQ_TCP_NAME_LENGTH	8	X'00000008'
MQ_TIME_LENGTH	8	X'00000008'
MQ_TOPIC_DESC_LENGTH	64	X'00000040'
MQ_TOPIC_NAME_LENGTH	48	X'00000030'
MQ_TOPIC_STR_LENGTH	10240	X'00002800'
MQ_TOTAL_EXIT_DATA_LENGTH	999	X'000003E7'
MQ_TOTAL_EXIT_NAME_LENGTH	999	X'000003E7'
MQ_TP_NAME_LENGTH	64	X'00000040'
MQ_TPIPE_NAME_LENGTH	8	X'00000008'
MQ_TRAN_INSTANCE_ID_LENGTH	16	X'00000010'
MQ_TRANSACTION_ID_LENGTH	4	X'00000004'
MQ_TRIGGER_DATA_LENGTH	64	X'00000040'
MQ_TRIGGER_PROGRAM_NAME_LENGTH	8	X'00000008'
MQ_TRIGGER_TERM_ID_LENGTH	4	X'00000004'
MQ_TRIGGER_TRANS_ID_LENGTH	4	X'00000004'
MQ_USER_ID_LENGTH	12	X'0000000C'
MQ_VERSION_LENGTH	8	X'00000008'
MQ_XCF_GROUP_NAME_LENGTH	8	X'00000008'
MQ_XCF_MEMBER_NAME_LENGTH	16	X'00000010'

MQ_* (Command format String Lengths):

MQ_ARCHIVE_PFX_LENGTH	36	X'00000024'
MQ_ARCHIVE_UNIT_LENGTH	8	X'00000008'
MQ_ASID_LENGTH	4	X'00000004'
MQ_AUTH_PROFILE_NAME_LENGTH	48	X'00000030'
MQ_CF_LEID_LENGTH	12	X'0000000C'
MQ_COMMAND_MQSC_LENGTH	32768	X'00008000'
MQ_DATA_SET_NAME_LENGTH	44	X'0000002C'
MQ_DB2_NAME_LENGTH	4	X'00000004'
MQ_DSG_NAME_LENGTH	8	X'00000008'
MQ_ENTITY_NAME_LENGTH	1024	X'00000400'
MQ_ENV_INFO_LENGTH	96	X'00000060'
MQ_IP_ADDRESS_LENGTH	48	X'00000030'
MQ_LOG_CORREL_ID_LENGTH	8	X'00000008'
MQ_LOG_EXTENT_NAME_LENGTH	24	X'00000018'
MQ_LOG_PATH_LENGTH	1024	X'00000400'
MQ_LRSN_LENGTH	12	X'0000000C'
MQ_ORIGIN_NAME_LENGTH	8	X'00000008'
MQ_PSB_NAME_LENGTH	8	X'00000008'

MQ_PST_ID_LENGTH	8	X'00000008'
MQ_Q_MGR_CPF_LENGTH	4	X'00000004'
MQ_RESPONSE_ID_LENGTH	24	X'00000018'
MQ_RBA_LENGTH	16	X'00000010'
MQ_SECURITY_PROFILE_LENGTH	40	X'00000028'
MQ_SERVICE_COMPONENT_LENGTH	48	X'00000030'
MQ_SUB_NAME_LENGTH	10240	X'00002800'
MQ_SYSP_SERVICE_LENGTH	32	X'00000020'
MQ_SYSTEM_NAME_LENGTH	8	X'00000008'
MQ_TASK_NUMBER_LENGTH	8	X'00000008'
MQ_TPIPE_PFX_LENGTH	4	X'00000004'
MQ_UOW_ID_LENGTH	256	X'00000100'
MQ_USER_DATA_LENGTH	10240	X'00002800'
MQ_VOLSER_LENGTH	6	X'00000006'

MQACH_* (API exit chain area header structure):

MQACH_STRUC_ID	"ACHb"	
MQACH_STRUC_ID_ARRAY	'A','C','H','b'	
MQACH_VERSION_1	1	X'00000001'
MQACH_CURRENT_VERSION	1	X'00000001'
MQACH_LENGTH_1	(value differs by platform or version)	
MQACH_CURRENT_LENGTH	(value differs by platform or version)	

MQACT_* (Accounting Token):

MQACT_NONE	X'00...00'	(32 nulls)
MQACT_NONE_ARRAY	'\0','\0',...	(32 nulls)

MQACT_* (Command format Action Options)

MQACT_FORCE_REMOVE	1	X'00000001'
MQACT_ADVANCE_LOG	2	X'00000002'
MQACT_COLLECT_STATISTICS	3	X'00000003'
MQACT_PUBSUB	4	X'00000004'

MQACTP_* (Action):

MQACTP_NEW	0	X'00000000'
MQACTP_FORWARD	1	X'00000001'
MQACTP_REPLY	2	X'00000002'
MQACTP_REPORT	3	X'00000003'

MQACTT_* (Accounting Token Types):

MQACTT_UNKNOWN	X'00'	
MQACTT_CICS_LUOW_ID	X'01'	
MQACTT_OS2_DEFAULT	X'04'	
MQACTT_DOS_DEFAULT	X'05'	
MQACTT_UNIX_NUMERIC_ID	X'06'	
MQACTT_OS400_ACCOUNT_TOKEN	X'08'	
MQACTT_WINDOWS_DEFAULT	X'09'	
MQACTT_NT_SECURITY_ID	X'0B'	
MQACTT_USER	X'19'	

MQADOPT_* (Adopt New MCA Checks and Adopt New MCA Types):
Adopt New MCA Checks

MQADOPT_CHECK_NONE	0	X'00000000'
MQADOPT_CHECK_ALL	1	X'00000001'
MQADOPT_CHECK_Q_MGR_NAME	2	X'00000002'
MQADOPT_CHECK_NET_ADDR	4	X'00000004'

Adopt New MCA Types

MQADOPT_TYPE_NO	0	X'00000000'
MQADOPT_TYPE_ALL	1	X'00000001'
MQADOPT_TYPE_SVR	2	X'00000002'
MQADOPT_TYPE_SDR	4	X'00000004'
MQADOPT_TYPE_RCVR	8	X'00000008'
MQADOPT_TYPE_CLUSRCVR	16	X'00000010'

MQAIR_* (Authentication information record structure):

MQAIR_STRUC_ID	"AIRb"	
MQAIR_STRUC_ID_ARRAY	'A','I','R','b'	
MQAIR_VERSION_1	1	X'00000001'
MQAIR_VERSION_2	2	X'00000002'
MQAIR_CURRENT_VERSION	2	X'00000002'

MQAIT_* (Authentication Information Type):

MQAIT_ALL	0	X'00000000'
MQAIT_CRL_LDAP	1	X'00000001'
MQAIT_OCSP	2	X'00000002'
MQAIT_IDPW_OS	3	X'00000003'
MQAIT_IDPW_LDAP	4	X'00000004'

MQAS_* (Command format Asynchronous State Values):

MQAS_NONE	0	X'00000000'
MQAS_STARTED	1	X'00000001'
MQAS_START_WAIT	2	X'00000002'
MQAS_STOPPED	3	X'00000003'
MQAS_SUSPENDED	4	X'00000004'
MQAS_SUSPENDED_TEMPORARY	5	X'00000005'
MQAS_ACTIVE	6	X'00000006'
MQAS_INACTIVE	7	X'00000007'

MQAT_* (Put Application Types):

MQAT_UNKNOWN	-1	X'FFFFFFFF'
MQAT_NO_CONTEXT	0	X'00000000'
MQAT_CICS	1	X'00000001'
MQAT_MVS	2	X'00000002'
MQAT_OS390	2	X'00000002'
MQAT_ZOS	2	X'00000002'
MQAT_IMS	3	X'00000003'
MQAT_OS2	4	X'00000004'
MQAT_DOS	5	X'00000005'
MQAT_AIX	6	X'00000006'
MQAT_UNIX	6	X'00000006'
MQAT_QMGR	7	X'00000007'
MQAT_OS400	8	X'00000008'
MQAT_WINDOWS	9	X'00000009'
MQAT_CICS_VSE	10	X'0000000A'
MQAT_WINDOWS_NT	11	X'0000000B'
MQAT_VMS	12	X'0000000C'
MQAT_GUARDIAN	13	X'0000000D'
MQAT_NSK	13	X'0000000D'
MQAT_VOS	14	X'0000000E'
MQAT_OPEN_TP1	15	X'0000000F'
MQAT_VM	18	X'00000012'
MQAT_IMS_BRIDGE	19	X'00000013'
MQAT_XCF	20	X'00000014'

MQAT_CICS_BRIDGE	21	X'00000015'
MQAT_NOTES_AGENT	22	X'00000016'
MQAT_TPF	23	X'00000017'
MQAT_USER	25	X'00000019'
MQAT_BROKER	26	X'0000001A'
MQAT_QMGR_PUBLISH	26	X'0000001A'
MQAT_JAVA	28	X'0000001C'
MQAT_DQM	29	X'0000001D'
MQAT_CHANNEL_INITIATOR	30	X'0000001E'
MQAT_WLM	31	X'0000001F'
MQAT_BATCH	32	X'00000020'
MQAT_RRS_BATCH	33	X'00000021'
MQAT_SIB	34	X'00000022'
MQAT_DEFAULT	(value differs by platform or version)	
MQAT_USER_FIRST	65536	X'00010000'
MQAT_USER_LAST	999999999	X'3B9AC9FF'

MQAUTH_* (Command format Authority Values):

MQAUTH_NONE	0	X'00000000'
MQAUTH_ALT_USER_AUTHORITY	1	X'00000001'
MQAUTH_BROWSE	2	X'00000002'
MQAUTH_CHANGE	3	X'00000003'
MQAUTH_CLEAR	4	X'00000004'
MQAUTH_CONNECT	5	X'00000005'
MQAUTH_CREATE	6	X'00000006'
MQAUTH_DELETE	7	X'00000007'
MQAUTH_DISPLAY	8	X'00000008'
MQAUTH_INPUT	9	X'00000009'
MQAUTH_INQUIRE	10	X'0000000A'
MQAUTH_OUTPUT	11	X'0000000B'
MQAUTH_PASS_ALL_CONTEXT	12	X'0000000C'
MQAUTH_PASS_IDENTITY_CONTEXT	13	X'0000000D'
MQAUTH_SET	14	X'0000000E'
MQAUTH_SET_ALL_CONTEXT	15	X'0000000F'
MQAUTH_SET_IDENTITY_CONTEXT	16	X'00000010'
MQAUTH_CONTROL	17	X'00000011'
MQAUTH_CONTROL_EXTENDED	18	X'00000012'
MQAUTH_PUBLISH	19	X'00000013'
MQAUTH_SUBSCRIBE	20	X'00000014'
MQAUTH_RESUME	21	X'00000015'

MQAUTH_SYSTEM	22	X'00000016'

MQAUTHOPT_* (Command format Authority Options):

MQAUTHOPT_CUMULATIVE	256	X'00000100'
MQAUTHOPT_ENTITY_EXPLICIT	1	X'00000001'
MQAUTHOPT_ENTITY_SET	2	X'00000002'
MQAUTHOPT_NAME_ALL_MATCHING	32	X'00000020'
MQAUTHOPT_NAME_AS_WILDCARD	64	X'00000040'
MQAUTHOPT_NAME_EXPLICIT	16	X'00000010'

MQAXC_* (API exit context structure):

MQAXC_STRUC_ID	"AXCb"	
MQAXC_STRUC_ID_ARRAY	'A','X','C','b'	
MQAXC_VERSION_1	1	X'00000001'
MQAXC_CURRENT_VERSION	1	X'00000001'

MQAXP_* (API exit parameter structure):

MQAXP_STRUC_ID	"AXPb"	
MQAXP_STRUC_ID_ARRAY	'A','X','P','b'	
MQAXP_VERSION_1	1	X'00000001'
MQAXP_VERSION_2	2	X'00000002'
MQAXP_CURRENT_VERSION	2	X'00000002'

MQBA_* (Byte Attribute Selectors):

MQBA_FIRST	6001	X'00001771'
MQBA_LAST	8000	X'00001F40'

MQBACF_* (Command format Byte Parameter Types):

MQBACF_FIRST	7001	X'00001B59'
MQBACF_EVENT_ACCOUNTING_TOKEN	7001	X'00001B59'
MQBACF_EVENT_SECURITY_ID	7002	X'00001B5A'
MQBACF_RESPONSE_SET	7003	X'00001B5B'
MQBACF_RESPONSE_ID	7004	X'00001B5C'
MQBACF_EXTERNAL_UOW_ID	7005	X'00001B5D'
MQBACF_CONNECTION_ID	7006	X'00001B5E'
MQBACF_GENERIC_CONNECTION_ID	7007	X'00001B5F'
MQBACF_ORIGIN_UOW_ID	7008	X'00001B60'
MQBACF_Q_MGR_UOW_ID	7009	X'00001B61'
MQBACF_ACCOUNTING_TOKEN	7010	X'00001B62'

MQBACF_CORREL_ID	7011	X'00001B63'
MQBACF_GROUP_ID	7012	X'00001B64'
MQBACF_MSG_ID	7013	X'00001B65'
MQBACF_CF_LEID	7014	X'00001B66'
MQBACF_DESTINATION_CORREL_ID	7015	X'00001B67'
MQBACF_SUB_ID	7016	X'00001B68'
MQBACF_LAST_USED	7016	X'00001B68'

MQBL_* (Buffer Length for mqAddString and mqSetString):

MQBL_NULL_TERMINATED	-1	X'FFFFFFFF'

MQBMHO_* (Buffer to message handle options and structure):
Buffer to message handle options structure

MQBMHO_STRUC_ID	"BMHO"	
MQBMHO_STRUC_ID_ARRAY	'B','M','H','O'	
MQBMHO_VERSION_1	1	X'00000001'
MQBMHO_CURRENT_VERSION	1	X'00000001'

Buffer To Message Handle Options

MQBMHO_NONE	0	X'00000000'
MQBMHO_DELETE_PROPERTIES	1	X'00000001'

MQBND_* (Default Bindings):

MQBND_BIND_ON_OPEN	0	X'00000000'
MQBND_BIND_NOT_FIXED	1	X'00000001'
MQBND_BIND_ON_GROUP	2	X'00000002'

MQBO_* (Begin options and structure):
Begin options structure

MQBO_STRUC_ID	"BObb"	
MQBO_STRUC_ID_ARRAY	'B','O','b','b'	
MQBO_VERSION_1	1	X'00000001'
MQBO_CURRENT_VERSION	1	X'00000001'

Begin Options

MQBO_NONE	0	X'00000000'

MQBT_* (Command format Bridge Types):

MQBT_OTMA	1	X'00000001'

MQCA_* (Character Attribute Selectors):

MQCA_ADMIN_TOPIC_NAME	2105	X'00000839'
MQCA_ALTERATION_DATE	2027	X'000007EB'
MQCA_ALTERATION_TIME	2028	X'000007EC'
MQCA_APPL_ID	2001	X'000007D1'
MQCA_AUTH_INFO_CONN_NAME	2053	X'00000805'
MQCA_AUTH_INFO_DESC	2046	X'000007FE'
MQCA_AUTH_INFO_NAME	2045	X'000007FD'
MQCA_AUTH_INFO_OCSP_URL	2109	X'0000083D'
MQCA_AUTO_REORG_CATALOG	2091	X'0000082B'
MQCA_AUTO_REORG_START_TIME	2090	X'0000082A'
MQCA_BACKOUT_REQ_Q_NAME	2019	X'000007E3'
MQCA_BASE_OBJECT_NAME	2002	X'000007D2'
MQCA_BASE_Q_NAME	2002	X'000007D2'
MQCA_BATCH_INTERFACE_ID	2068	X'00000814'
MQCA_CF_STRUC_DESC	2052	X'00000804'
MQCA_CF_STRUC_NAME	2039	X'000007F7'
MQCA_CHANNEL_AUTO_DEF_EXIT	2026	X'000007EA'
MQCA_CHILD	2101	X'00000835'
MQCA_CHINIT_SERVICE_PARM	2076	X'0000081C'
MQCA_CICS_FILE_NAME	2060	X'0000080C'
MQCA_CLUS_CHL_NAME	2124	X'0000084C'
MQCA_CLUSTER_DATE	2037	X'000007F5'
MQCA_CLUSTER_NAME	2029	X'000007ED'
MQCA_CLUSTER_NAMELIST	2030	X'000007EE'
MQCA_CLUSTER_Q_MGR_NAME	2031	X'000007EF'
MQCA_CLUSTER_TIME	2038	X'000007F6'
MQCA_CLUSTER_WORKLOAD_DATA	2034	X'000007F2'
MQCA_CLUSTER_WORKLOAD_EXIT	2033	X'000007F1'
MQCA_COMMAND_INPUT_Q_NAME	2003	X'000007D3'
MQCA_COMMAND_REPLY_Q_NAME	2067	X'00000813'
MQCA_CREATION_DATE	2004	X'000007D4'
MQCA_CREATION_TIME	2005	X'000007D5'
MQCA_DEAD_LETTER_Q_NAME	2006	X'000007D6'
MQCA_DEF_XMIT_Q_NAME	2025	X'000007E9'
MQCA_DNS_GROUP	2071	X'00000817'

MQCA_ENV_DATA	2007	X'000007D7'
MQCA_FIRST	2001	X'000007D1'
MQCA_IGQ_USER_ID	2041	X'000007F9'
MQCA_INITIATION_Q_NAME	2008	X'000007D8'
MQCA_LAST	4000	X'00000FA0'
MQCA_LAST_USED	2109	X'0000083D'
MQCA_LDAP_PASSWORD	2048	X'00000800'
MQCA_LDAP_USER_NAME	2047	X'000007FF'
MQCA_LU_GROUP_NAME	2072	X'00000818'
MQCA_LU_NAME	2073	X'00000819'
MQCA_LU62_ARM_SUFFIX	2074	X'0000081A'
MQCA_MODEL_DURABLE_Q	2096	X'00000830'
MQCA_MODEL_NON_DURABLE_Q	2097	X'00000831'
MQCA_MONITOR_Q_NAME	2066	X'00000812'
MQCA_NAMELIST_DESC	2009	X'000007D9'
MQCA_NAMELIST_NAME	2010	X'000007DA'
MQCA_NAMES	2020	X'000007E4'
MQCA_PARENT	2102	X'00000836'
MQCA_PASS_TICKET_APPL	2086	X'00000826'
MQCA_PROCESS_DESC	2011	X'000007DB'
MQCA_PROCESS_NAME	2012	X'000007DC'
MQCA_Q_DESC	2013	X'000007DD'
MQCA_Q_MGR_DESC	2014	X'000007DE'
MQCA_Q_MGR_IDENTIFIER	2032	X'000007F0'
MQCA_Q_MGR_NAME	2015	X'000007DF'
MQCA_Q_NAME	2016	X'000007E0'
MQCA_QSG_NAME	2040	X'000007F8'
MQCA_REMOTE_Q_MGR_NAME	2017	X'000007E1'
MQCA_REMOTE_Q_NAME	2018	X'000007E2'
MQCA_REPOSITORY_NAME	2035	X'000007F3'
MQCA_REPOSITORY_NAMELIST	2036	X'000007F4'
MQCA_RESUME_DATE	2098	X'00000832'
MQCA_RESUME_TIME	2099	X'00000833'
MQCA_SERVICE_DESC	2078	X'0000081E'
MQCA_SERVICE_NAME	2077	X'0000081D'
MQCA_SERVICE_START_ARGS	2080	X'00000820'
MQCA_SERVICE_START_COMMAND	2079	X'0000081F'
MQCA_SERVICE_STOP_ARGS	2082	X'00000822'
MQCA_SERVICE_STOP_COMMAND	2081	X'00000821'
MQCA_STDERR_DESTINATION	2084	X'00000824'
MQCA_STDOUT_DESTINATION	2083	X'00000823'
MQCA_SSL_CRL_NAMELIST	2050	X'00000802'

MQCA_SSL_CRYPTO_HARDWARE	2051	X'00000803'
MQCA_SSL_KEY_LIBRARY	2069	X'00000815'
MQCA_SSL_KEY_MEMBER	2070	X'00000816'
MQCA_SSL_KEY_REPOSITORY	2049	X'00000801'
MQCA_STORAGE_CLASS	2022	X'000007E6'
MQCA_STORAGE_CLASS_DESC	2042	X'000007FA'
MQCA_SYSTEM_LOG_Q_NAME	2065	X'00000811'
MQCA_TCP_NAME	2075	X'0000081B'
MQCA_TOPIC_DESC	2093	X'0000082D'
MQCA_TOPIC_NAME	2092	X'0000082C'
MQCA_TOPIC_STRING_FILTER	2108	X'0000083C'
MQCA_TOPIC_STRING	2094	X'0000082E'
MQCA_TPIPE_NAME	2085	X'00000825'
MQCA_TRIGGER_CHANNEL_NAME	2064	X'00000810'
MQCA_TRIGGER_DATA	2023	X'000007E7'
MQCA_TRIGGER_PROGRAM_NAME	2062	X'0000080E'
MQCA_TRIGGER_TERM_ID	2063	X'0000080F'
MQCA_TRIGGER_TRANS_ID	2061	X'0000080D'
MQCA_USER_DATA	2021	X'000007E5'
MQCA_USER_LIST	4000	X'00000FA0'
MQCA_VERSION	2120	X'00000848'
MQCA_XCF_GROUP_NAME	2043	X'000007FB'
MQCA_XCF_MEMBER_NAME	2044	X'000007FC'
MQCA_XMIT_Q_NAME	2024	X'000007E8'

MQCACF_* (Command format Character Parameter Types):

MQCACF_FIRST	3001	X'00000BB9'
MQCACF_FROM_Q_NAME	3001	X'00000BB9'
MQCACF_TO_Q_NAME	3002	X'00000BBA'
MQCACF_FROM_PROCESS_NAME	3003	X'00000BBB'
MQCACF_TO_PROCESS_NAME	3004	X'00000BBC'
MQCACF_FROM_NAMELIST_NAME	3005	X'00000BBD'
MQCACF_TO_NAMELIST_NAME	3006	X'00000BBE'
MQCACF_FROM_CHANNEL_NAME	3007	X'00000BBF'
MQCACF_TO_CHANNEL_NAME	3008	X'00000BC0'
MQCACF_FROM_AUTH_INFO_NAME	3009	X'00000BC1'
MQCACF_TO_AUTH_INFO_NAME	3010	X'00000BC2'
MQCACF_Q_NAMES	3011	X'00000BC3'
MQCACF_PROCESS_NAMES	3012	X'00000BC4'
MQCACF_NAMELIST_NAMES	3013	X'00000BC5'
MQCACF_ESCAPE_TEXT	3014	X'00000BC6'

MQCACF_LOCAL_Q_NAMES	3015	X'00000BC7'
MQCACF_MODEL_Q_NAMES	3016	X'00000BC8'
MQCACF_ALIAS_Q_NAMES	3017	X'00000BC9'
MQCACF_REMOTE_Q_NAMES	3018	X'00000BCA'
MQCACF_SENDER_CHANNEL_NAMES	3019	X'00000BCB'
MQCACF_SERVER_CHANNEL_NAMES	3020	X'00000BCC'
MQCACF_REQUESTER_CHANNEL_NAMES	3021	X'00000BCD'
MQCACF_RECEIVER_CHANNEL_NAMES	3022	X'00000BCE'
MQCACF_OBJECT_Q_MGR_NAME	3023	X'00000BCF'
MQCACF_APPL_NAME	3024	X'00000BD0'
MQCACF_USER_IDENTIFIER	3025	X'00000BD1'
MQCACF_AUX_ERROR_DATA_STR_1	3026	X'00000BD2'
MQCACF_AUX_ERROR_DATA_STR_2	3027	X'00000BD3'
MQCACF_AUX_ERROR_DATA_STR_3	3028	X'00000BD4'
MQCACF_BRIDGE_NAME	3029	X'00000BD5'
MQCACF_STREAM_NAME	3030	X'00000BD6'
MQCACF_TOPIC	3031	X'00000BD7'
MQCACF_PARENT_Q_MGR_NAME	3032	X'00000BD8'
MQCACF_CORREL_ID	3033	X'00000BD9'
MQCACF_PUBLISH_TIMESTAMP	3034	X'00000BDA'
MQCACF_STRING_DATA	3035	X'00000BDB'
MQCACF_SUPPORTED_STREAM_NAME	3036	X'00000BDC'
MQCACF_REG_TOPIC	3037	X'00000BDD'
MQCACF_REG_TIME	3038	X'00000BDE'
MQCACF_REG_USER_ID	3039	X'00000BDF'
MQCACF_CHILD_Q_MGR_NAME	3040	X'00000BE0'
MQCACF_REG_STREAM_NAME	3041	X'00000BE1'
MQCACF_REG_Q_MGR_NAME	3042	X'00000BE2'
MQCACF_REG_Q_NAME	3043	X'00000BE3'
MQCACF_REG_CORREL_ID	3044	X'00000BE4'
MQCACF_EVENT_USER_ID	3045	X'00000BE5'
MQCACF_OBJECT_NAME	3046	X'00000BE6'
MQCACF_EVENT_Q_MGR	3047	X'00000BE7'
MQCACF_AUTH_INFO_NAMES	3048	X'00000BE8'
MQCACF_EVENT_APPL_IDENTITY	3049	X'00000BE9'
MQCACF_EVENT_APPL_NAME	3050	X'00000BEA'
MQCACF_EVENT_APPL_ORIGIN	3051	X'00000BEB'
MQCACF_SUBSCRIPTION_NAME	3052	X'00000BEC'
MQCACF_REG_SUB_NAME	3053	X'00000BED'
MQCACF_SUBSCRIPTION_IDENTITY	3054	X'00000BEE'
MQCACF_REG_SUB_IDENTITY	3055	X'00000BEF'
MQCACF_SUBSCRIPTION_USER_DATA	3056	X'00000BF0'

MQCACF_REG_SUB_USER_DATA	3057	X'00000BF1'
MQCACF_APPL_TAG	3058	X'00000BF2'
MQCACF_DATA_SET_NAME	3059	X'00000BF3'
MQCACF_UOW_START_DATE	3060	X'00000BF4'
MQCACF_UOW_START_TIME	3061	X'00000BF5'
MQCACF_UOW_LOG_START_DATE	3062	X'00000BF6'
MQCACF_UOW_LOG_START_TIME	3063	X'00000BF7'
MQCACF_UOW_LOG_EXTENT_NAME	3064	X'00000BF8'
MQCACF_PRINCIPAL_ENTITY_NAMES	3065	X'00000BF9'
MQCACF_GROUP_ENTITY_NAMES	3066	X'00000BFA'
MQCACF_AUTH_PROFILE_NAME	3067	X'00000BFB'
MQCACF_ENTITY_NAME	3068	X'00000BFC'
MQCACF_SERVICE_COMPONENT	3069	X'00000BFD'
MQCACF_RESPONSE_Q_MGR_NAME	3070	X'00000BFE'
MQCACF_CURRENT_LOG_EXTENT_NAME	3071	X'00000BFF'
MQCACF_RESTART_LOG_EXTENT_NAME	3072	X'00000C00'
MQCACF_MEDIA_LOG_EXTENT_NAME	3073	X'00000C01'
MQCACF_LOG_PATH	3074	X'00000C02'
MQCACF_COMMAND_MQSC	3075	X'00000C03'
MQCACF_Q_MGR_CPF	3076	X'00000C04'
MQCACF_USAGE_LOG_RBA	3078	X'00000C06'
MQCACF_USAGE_LOG_LRSN	3079	X'00000C07'
MQCACF_COMMAND_SCOPE	3080	X'00000C08'
MQCACF_ASID	3081	X'00000C09'
MQCACF_PSB_NAME	3082	X'00000C0A'
MQCACF_PST_ID	3083	X'00000C0B'
MQCACF_TASK_NUMBER	3084	X'00000C0C'
MQCACF_TRANSACTION_ID	3085	X'00000C0D'
MQCACF_Q_MGR_UOW_ID	3086	X'00000C0E'
MQCACF_ORIGIN_NAME	3088	X'00000C10'
MQCACF_ENV_INFO	3089	X'00000C11'
MQCACF_SECURITY_PROFILE	3090	X'00000C12'
MQCACF_CONFIGURATION_DATE	3091	X'00000C13'
MQCACF_CONFIGURATION_TIME	3092	X'00000C14'
MQCACF_FROM_CF_STRUC_NAME	3093	X'00000C15'
MQCACF_TO_CF_STRUC_NAME	3094	X'00000C16'
MQCACF_CF_STRUC_NAMES	3095	X'00000C17'
MQCACF_FAIL_DATE	3096	X'00000C18'
MQCACF_FAIL_TIME	3097	X'00000C19'
MQCACF_BACKUP_DATE	3098	X'00000C1A'
MQCACF_BACKUP_TIME	3099	X'00000C1B'
MQCACF_SYSTEM_NAME	3100	X'00000C1C'

MQCACF_CF_STRUC_BACKUP_START	3101	X'00000C1D'
MQCACF_CF_STRUC_BACKUP_END	3102	X'00000C1E'
MQCACF_CF_STRUC_LOG_Q_MGRS	3103	X'00000C1F'
MQCACF_FROM_STORAGE_CLASS	3104	X'00000C20'
MQCACF_TO_STORAGE_CLASS	3105	X'00000C21'
MQCACF_STORAGE_CLASS_NAMES	3106	X'00000C22'
MQCACF_DSG_NAME	3108	X'00000C24'
MQCACF_DB2_NAME	3109	X'00000C25'
MQCACF_SYSP_CMD_USER_ID	3110	X'00000C26'
MQCACF_SYSP_OTMA_GROUP	3111	X'00000C27'
MQCACF_SYSP_OTMA_MEMBER	3112	X'00000C28'
MQCACF_SYSP_OTMA_DRU_EXIT	3113	X'00000C29'
MQCACF_SYSP_OTMA_TPIPE_PFX	3114	X'00000C2A'
MQCACF_SYSP_ARCHIVE_PFX1	3115	X'00000C2B'
MQCACF_SYSP_ARCHIVE_UNIT1	3116	X'00000C2C'
MQCACF_SYSP_LOG_CORREL_ID	3117	X'00000C2D'
MQCACF_SYSP_UNIT_VOLSER	3118	X'00000C2E'
MQCACF_SYSP_Q_MGR_TIME	3119	X'00000C2F'
MQCACF_SYSP_Q_MGR_DATE	3120	X'00000C30'
MQCACF_SYSP_Q_MGR_RBA	3121	X'00000C31'
MQCACF_SYSP_LOG_RBA	3122	X'00000C32'
MQCACF_SYSP_SERVICE	3123	X'00000C33'
MQCACF_FROM_LISTENER_NAME	3124	X'00000C34'
MQCACF_TO_LISTENER_NAME	3125	X'00000C35'
MQCACF_FROM_SERVICE_NAME	3126	X'00000C36'
MQCACF_TO_SERVICE_NAME	3127	X'00000C37'
MQCACF_LAST_PUT_DATE	3128	X'00000C38'
MQCACF_LAST_PUT_TIME	3129	X'00000C39'
MQCACF_LAST_GET_DATE	3130	X'00000C3A'
MQCACF_LAST_GET_TIME	3131	X'00000C3B'
MQCACF_OPERATION_DATE	3132	X'00000C3C'
MQCACF_OPERATION_TIME	3133	X'00000C3D'
MQCACF_ACTIVITY_DESC	3134	X'00000C3E'
MQCACF_APPL_IDENTITY_DATA	3135	X'00000C3F'
MQCACF_APPL_ORIGIN_DATA	3136	X'00000C40'
MQCACF_PUT_DATE	3137	X'00000C41'
MQCACF_PUT_TIME	3138	X'00000C42'
MQCACF_REPLY_TO_Q	3139	X'00000C43'
MQCACF_REPLY_TO_Q_MGR	3140	X'00000C44'
MQCACF_RESOLVED_Q_NAME	3141	X'00000C45'
MQCACF_STRUC_ID	3142	X'00000C46'
MQCACF_VALUE_NAME	3143	X'00000C47'

MQCACF_SERVICE_START_DATE	3144	X'00000C48'
MQCACF_SERVICE_START_TIME	3145	X'00000C49'
MQCACF_SYSP_OFFLINE_RBA	3146	X'00000C4A'
MQCACF_SYSP_ARCHIVE_PFX2	3147	X'00000C4B'
MQCACF_SYSP_ARCHIVE_UNIT2	3148	X'00000C4C'
MQCACF_TO_TOPIC_NAME	3149	X'00000C4D'
MQCACF_FROM_TOPIC_NAME	3150	X'00000C4E'
MQCACF_TOPIC_NAMES	3151	X'00000C4F'
MQCACF_SUB_NAME	3152	X'00000C50'
MQCACF_DESTINATION_Q_MGR	3153	X'00000C51'
MQCACF_DESTINATION	3154	X'00000C52'
MQCACF_SUB_USER_ID	3156	X'00000C54'
MQCACF_SUB_USER_DATA	3159	X'00000C57'
MQCACF_SUB_SELECTOR	3160	X'00000C58'
MQCACF_LAST_PUB_DATE	3161	X'00000C59'
MQCACF_LAST_PUB_TIME	3162	X'00000C5A'
MQCACF_FROM_SUB_NAME	3163	X'00000C5B'
MQCACF_TO_SUB_NAME	3164	X'00000C5C'
MQCACF_LAST_MSG_TIME	3167	X'00000C5F'
MQCACF_LAST_MSG_DATE	3168	X'00000C60'
MQCACF_SUBSCRIPTION_POINT	3169	X'00000C61'
MQCACF_FILTER	3170	X'00000C62'
MQCACF_NONE	3171	X'00000C63'
MQCACF_ADMIN_TOPIC_NAMES	3172	X'00000C64'
MQCACF_LAST_USED	3172	X'00000C64'

MQCACH_* (Command format Character Channel Parameter Types):

MQCACH_FIRST	3501	X'00000DAD'
MQCACH_CHANNEL_NAME	3501	X'00000DAD'
MQCACH_DESC	3502	X'00000DAE'
MQCACH_MODE_NAME	3503	X'00000DAF'
MQCACH_TP_NAME	3504	X'00000DB0'
MQCACH_XMIT_Q_NAME	3505	X'00000DB1'
MQCACH_CONNECTION_NAME	3506	X'00000DB2'
MQCACH_MCA_NAME	3507	X'00000DB3'
MQCACH_SEC_EXIT_NAME	3508	X'00000DB4'
MQCACH_MSG_EXIT_NAME	3509	X'00000DB5'
MQCACH_SEND_EXIT_NAME	3510	X'00000DB6'
MQCACH_RCV_EXIT_NAME	3511	X'00000DB7'
MQCACH_CHANNEL_NAMES	3512	X'00000DB8'
MQCACH_SEC_EXIT_USER_DATA	3513	X'00000DB9'

MQCACH_MSG_EXIT_USER_DATA	3514	X'00000DBA'
MQCACH_SEND_EXIT_USER_DATA	3515	X'00000DBB'
MQCACH_RCV_EXIT_USER_DATA	3516	X'00000DBC'
MQCACH_USER_ID	3517	X'00000DBD'
MQCACH_PASSWORD	3518	X'00000DBE'
MQCACH_LOCAL_ADDRESS	3520	X'00000DC0'
MQCACH_LOCAL_NAME	3521	X'00000DC1'
MQCACH_LAST_MSG_TIME	3524	X'00000DC4'
MQCACH_LAST_MSG_DATE	3525	X'00000DC5'
MQCACH_MCA_USER_ID	3527	X'00000DC7'
MQCACH_CHANNEL_START_TIME	3528	X'00000DC8'
MQCACH_CHANNEL_START_DATE	3529	X'00000DC9'
MQCACH_MCA_JOB_NAME	3530	X'00000DCA'
MQCACH_LAST_LUWID	3531	X'00000DCB'
MQCACH_CURRENT_LUWID	3532	X'00000DCC'
MQCACH_FORMAT_NAME	3533	X'00000DCD'
MQCACH_MR_EXIT_NAME	3534	X'00000DCE'
MQCACH_MR_EXIT_USER_DATA	3535	X'00000DCF'
MQCACH_SSL_CIPHER_SPEC	3544	X'00000DD8'
MQCACH_SSL_PEER_NAME	3545	X'00000DD9'
MQCACH_SSL_HANDSHAKE_STAGE	3546	X'00000DDA'
MQCACH_SSL_SHORT_PEER_NAME	3547	X'00000DDB'
MQCACH_REMOTE_APPL_TAG	3548	X'00000DDC'
MQCACH_SSL_CERT_USER_ID	3549	X'00000DDD'
MQCACH_SSL_CERT_ISSUER_NAME	3550	X'00000DDE'
MQCACH_LU_NAME	3551	X'00000DDF'
MQCACH_IP_ADDRESS	3552	X'00000DE0'
MQCACH_TCP_NAME	3553	X'00000DE1'
MQCACH_LISTENER_NAME	3554	X'00000DE2'
MQCACH_LISTENER_DESC	3555	X'00000DE3'
MQCACH_LISTENER_START_DATE	3556	X'00000DE4'
MQCACH_LISTENER_START_TIME	3557	X'00000DE5'
MQCACH_SSL_KEY_RESET_DATE	3558	X'00000DE6'
MQCACH_SSL_KEY_RESET_TIME	3559	X'00000DE7'
MQCACH_LAST_USED	3559	X'00000DE7'

MQCADSD_* (CICS information header ADS Descriptors):

MQCADSD_NONE	0	X'00000000'
MQCADSD_SEND	1	X'00000001'
MQCADSD_RECV	16	X'00000010'
MQCADSD_MSGFORMAT	256	X'00000100'

MQCAFTY_* (Connection Affinity Values):

MQCAFTY_NONE	0	X'00000000'
MQCAFTY_PREFERRED	1	X'00000001'

MQCAMO_* (Command format Character Monitoring Parameter Types):

MQCAMO_FIRST	2701	X'00000A8D'
MQCAMO_CLOSE_DATE	2701	X'00000A8D'
MQCAMO_CLOSE_TIME	2702	X'00000A8E'
MQCAMO_CONN_DATE	2703	X'00000A8F'
MQCAMO_CONN_TIME	2704	X'00000A90'
MQCAMO_DISC_DATE	2705	X'00000A91'
MQCAMO_DISC_TIME	2706	X'00000A92'
MQCAMO_END_DATE	2707	X'00000A93'
MQCAMO_END_TIME	2708	X'00000A94'
MQCAMO_OPEN_DATE	2709	X'00000A95'
MQCAMO_OPEN_TIME	2710	X'00000A96'
MQCAMO_START_DATE	2711	X'00000A97'
MQCAMO_START_TIME	2712	X'00000A98'
MQCAMO_LAST_USED	2712	X'00000A98'

MQCBC_* (MQCBC constants structure):

MQCBC_STRUC_ID	"CBCb"	
MQCBC_STRUC_ID_ARRAY	'C','B','C','b'	
MQCBC_VERSION_1	1	X'00000001'
MQCBC_CURRENT_VERSION	1	X'00000001'

MQCBCF_* (MQCBC constants Flags):

MQCBCF_NONE	0	X'00000000'
MQCBCF_READA_BUFFER_EMPTY	1	X'00000001'

MQCBCT_* (MQCBC constants Callback type):

MQCBCT_START_CALL	1	X'00000001'
MQCBCT_STOP_CALL	2	X'00000002'
MQCBCT_REGISTER_CALL	3	X'00000003'
MQCBCT_DEREGISTER_CALL	4	X'00000004'
MQCBCT_EVENT_CALL	5	X'00000005'
MQCBCT_MSG_REMOVED	6	X'00000006'
MQCBCT_MSG_NOT_REMOVED	7	X'00000007'

MQCBD_* (MQCBD constants structure):

MQCBD_STRUC_ID	"CBDb"	
MQCBD_STRUC_ID_ARRAY	'C','B','D','b'	
MQCBD_VERSION_1	1	X'00000001'
MQCBD_CURRENT_VERSION	1	X'00000001'

MQCBDO_* (MQCBD constants Callback Options):

MQCBDO_NONE	0	X'00000000'
MQCBDO_START_CALL	1	X'00000001'
MQCBDO_STOP_CALL	4	X'00000004'
MQCBDO_REGISTER_CALL	256	X'00000100'
MQCBDO_DEREGISTER_CALL	512	X'00000200'
MQCBDO_FAIL_IF_QUIESCING	8192	X'00002000'

MQCBO_* (Create-Bag Options for mqCreateBag):

MQCBO_NONE	0	X'00000000'
MQCBO_USER_BAG	0	X'00000000'
MQCBO_ADMIN_BAG	1	X'00000001'
MQCBO_COMMAND_BAG	16	X'00000010'
MQCBO_SYSTEM_BAG	32	X'00000020'
MQCBO_GROUP_BAG	64	X'00000040'
MQCBO_LIST_FORM_ALLOWED	2	X'00000002'
MQCBO_LIST_FORM_INHIBITED	0	X'00000000'
MQCBO_REORDER_AS_REQUIRED	4	X'00000004'
MQCBO_DO_NOT_REORDER	0	X'00000000'
MQCBO_CHECK_SELECTORS	8	X'00000008'
MQCBO_DO_NOT_CHECK_SELECTORS	0	X'00000000'

MQCBT_* (MQCBD constants This is the type of the Callback Function):

MQCBT_MESSAGE_CONSUMER	1	X'00000001'
MQCBT_EVENT_HANDLER	2	X'00000002'

MQCC_* (completion codes):

MQCC_OK	0	X'00000000'
MQCC_WARNING	1	X'00000001'
MQCC_FAILED	2	X'00000002'
MQCC_UNKNOWN	-1	X'FFFFFFFF'

MQCCSI_* (Coded Character Set Identifiers):

MQCCSI_UNDEFINED	0	X'00000000'
MQCCSI_DEFAULT	0	X'00000000'
MQCCSI_Q_MGR	0	X'00000000'
MQCCSI_INHERIT	-2	X'FFFFFFFE'
MQCCSI_EMBEDDED	-1	X'FFFFFFFF'
MQCCSI_APPL	-3	X'FFFFFFFD'

MQCCT_* (CICS information header Conversational Task Options):

MQCCT_YES	1	X'00000001'
MQCCT_NO	0	X'00000000'

MQCD_* (Channel definition structure):

MQCD_VERSION_1	1	X'00000001'
MQCD_VERSION_2	2	X'00000002'
MQCD_VERSION_3	3	X'00000003'
MQCD_VERSION_4	4	X'00000004'
MQCD_VERSION_5	5	X'00000005'
MQCD_VERSION_6	6	X'00000006'
MQCD_VERSION_7	7	X'00000007'
MQCD_VERSION_8	8	X'00000008'
MQCD_VERSION_9	9	X'00000009'
MQCD_VERSION_10	10	X'0000000A'
MQCD_VERSION_11	11	X'0000000B'
MQCD_CURRENT_VERSION	11	X'0000000B'
MQCD_LENGTH_4	(value differs by platform or version)	
MQCD_LENGTH_5	(value differs by platform or version)	
MQCD_LENGTH_6	(value differs by platform or version)	

MQCD_LENGTH_7	(value differs by platform or version)	
MQCD_LENGTH_8	(value differs by platform or version)	
MQCD_LENGTH_9	(value differs by platform or version)	
MQCD_LENGTH_10	(value differs by platform or version)	
MQCD_LENGTH_11	(value differs by platform or version)	
MQCD_CURRENT_LENGTH	(value differs by platform or version)	

MQCDC_* (Channel Data Conversion):

MQCDC_SENDER_CONVERSION	1	X'00000001'
MQCDC_NO_SENDER_CONVERSION	0	X'00000000'

MQCERT_* (Certificate Validation Policy Type):

MQ_CERT_VAL_POLICY_DEFAULT	0	X'00000000'
MQ_CERT_VAL_POLICY_ANY	0	X'00000000'
MQ_CERT_VAL_POLICY_RFC5280	1	X'00000001'

MQCF_* (Capability Flags):

MQCF_NONE	0	X'00000000'
MQCF_DIST_LISTS	1	X'00000001'

MQCFAC_* (CICS information header Facility):

MQCFAC_NONE	X'00...00'	(8 nulls)
MQCFAC_NONE_ARRAY	'\0','\0',...	(8 nulls)

MQCFBF_* (Command format byte string filter parameter structure):

MQCFBF_STRUC_LENGTH_FIXED	20	X'00000014'

MQCFBS_* (Command format byte string parameter structure):

MQCFBS_STRUC_LENGTH_FIXED	16	X'00000010'

MQCFC_* (Command format header Control Options):

MQCFC_LAST	1	X'00000001'
MQCFC_NOT_LAST	0	X'00000000'

MQCFGR_* (Command format group parameter structure):

MQCFGR_STRUC_LENGTH	16	X'00000010'

MQCFH_* (Command format header structure):

MQCFH_STRUC_LENGTH	36	X'00000024'
MQCFH_VERSION_1	1	X'00000001'
MQCFH_VERSION_2	2	X'00000002'
MQCFH_VERSION_3	3	X'00000003'
MQCFH_CURRENT_VERSION	3	X'00000003'

MQCFIF_* (Command format integer filter parameter structure):

MQCFIF_STRUC_LENGTH	20	X'00000014'

MQCFIL_* (Command format integer list parameter structure):

MQCFIL_STRUC_LENGTH_FIXED	16	X'00000010'

MQCFIL64_* (Command format 64-bit integer list parameter structure):

MQCFIL64_STRUC_LENGTH_FIXED	16	X'00000010'

MQCFIN_* (Command format integer parameter structure):

MQCFIN_STRUC_LENGTH	16	X'00000010'

MQCFIN64_* (Command format 64-bit integer parameter structure):

MQCFIN64_STRUC_LENGTH	24	X'00000018'

MQCFO_* (Command format Refresh Repository Options and Command format Remove Queues Options):
Command format Refresh Repository Options

MQCFO_REFRESH_REPOSITORY_YES	1	X'00000001'
MQCFO_REFRESH_REPOSITORY_NO	0	X'00000000'

Command format Remove Queues Options

MQCFO_REMOVE_QUEUES_YES	1	X'00000001'
MQCFO_REMOVE_QUEUES_NO	0	X'00000000'

MQCFOP_* (Command format Filter Operators):

MQCFOP_LESS	1	X'00000001'
MQCFOP_EQUAL	2	X'00000002'
MQCFOP_GREATER	4	X'00000004'
MQCFOP_NOT_LESS	6	X'00000006'
MQCFOP_NOT_EQUAL	5	X'00000005'
MQCFOP_NOT_GREATER	3	X'00000003'
MQCFOP_LIKE	18	X'00000012'
MQCFOP_NOT_LIKE	21	X'00000015'
MQCFOP_CONTAINS	10	X'0000000A'
MQCFOP_EXCLUDES	13	X'0000000D'
MQCFOP_CONTAINS_GEN	26	X'0000001A'
MQCFOP_EXCLUDES_GEN	29	X'0000001D'

MQCFR_* (CF Recoverability):

MQCFR_YES	1	X'00000001'
MQCFR_NO	0	X'00000000'

MQCFSF_* (Command format string filter parameter structure):

MQCFSF_STRUC_LENGTH_FIXED	24	X'00000018'

MQCFSL_* (Command format string list parameter structure):

MQCFSL_STRUC_LENGTH_FIXED	24	X'00000018'

MQCFST_* (Command format string parameter structure):

MQCFST_STRUC_LENGTH_FIXED	20	X'00000014'

MQCFSTATUS_* (Command format CF Status):

MQCFSTATUS_NOT_FOUND	0	X'00000000'
MQCFSTATUS_ACTIVE	1	X'00000001'
MQCFSTATUS_IN_RECOVER	2	X'00000002'
MQCFSTATUS_IN_BACKUP	3	X'00000003'
MQCFSTATUS_FAILED	4	X'00000004'
MQCFSTATUS_NONE	5	X'00000005'
MQCFSTATUS_UNKNOWN	6	X'00000006'
MQCFSTATUS_ADMIN_INCOMPLETE	20	X'00000014'
MQCFSTATUS_NEVER_USED	21	X'00000015'
MQCFSTATUS_NO_BACKUP	22	X'00000016'
MQCFSTATUS_NOT_FAILED	23	X'00000017'
MQCFSTATUS_NOT_RECOVERABLE	24	X'00000018'
MQCFSTATUS_XES_ERROR	25	X'00000019'

MQCFT_* (Command format Types of Structure):

MQCFT_NONE	0	X'00000000'
MQCFT_COMMAND	1	X'00000001'
MQCFT_RESPONSE	2	X'00000002'
MQCFT_INTEGER	3	X'00000003'
MQCFT_STRING	4	X'00000004'
MQCFT_INTEGER_LIST	5	X'00000005'
MQCFT_STRING_LIST	6	X'00000006'
MQCFT_EVENT	7	X'00000007'
MQCFT_USER	8	X'00000008'
MQCFT_BYTE_STRING	9	X'00000009'
MQCFT_TRACE_ROUTE	10	X'0000000A'
MQCFT_REPORT	12	X'0000000C'
MQCFT_INTEGER_FILTER	13	X'0000000D'
MQCFT_STRING_FILTER	14	X'0000000E'
MQCFT_BYTE_STRING_FILTER	15	X'0000000F'
MQCFT_COMMAND_XR	16	X'00000010'
MQCFT_XR_MSG	17	X'00000011'
MQCFT_XR_ITEM	18	X'00000012'
MQCFT_XR_SUMMARY	19	X'00000013'
MQCFT_GROUP	20	X'00000014'
MQCFT_STATISTICS	21	X'00000015'
MQCFT_ACCOUNTING	22	X'00000016'
MQCFT_INTEGER64	23	X'00000017'
MQCFT_INTEGER64_LIST	25	X'00000019'

MQCFTYPE_* (Command format CF Types):

MQCFTYPE_APPL	0	X'00000000'
MQCFTYPE_ADMIN	1	X'00000001'

MQCFUNC_* (CICS information header Functions):

MQCFUNC_MQCONN	"CONN"	
MQCFUNC_MQGET	"GETb"	
MQCFUNC_MQINQ	"INQb"	
MQCFUNC_MQOPEN	"OPEN"	
MQCFUNC_MQPUT	"PUTb"	
MQCFUNC_MQPUT1	"PUT1"	
MQCFUNC_NONE	"bbbb"	
MQCFUNC_MQCONN_ARRAY	'C','O','N','N'	
MQCFUNC_MQGET_ARRAY	'G','E','T','b'	
MQCFUNC_MQINQ_ARRAY	'I','N','Q','b'	
MQCFUNC_MQOPEN_ARRAY	'O','P','E','N'	
MQCFUNC_MQPUT_ARRAY	'P','U','T','b'	
MQCFUNC_MQPUT1_ARRAY	'P','U','T','1'	
MQCFUNC_NONE_ARRAY	'b','b','b','b'	

MQCGWI_* (CICS information header Get Wait Interval):

MQCGWI_DEFAULT	-2	X'FFFFFFFE'

MQCHAD_* (Channel Auto Definition):

MQCHAD_DISABLED	0	X'00000000'
MQCHAD_ENABLED	1	X'00000001'

MQCHIDS_* (Command format Indoubt Status):

MQCHIDS_NOT_INDOUBT	0	X'00000000'
MQCHIDS_INDOUBT	1	X'00000001'

MQCHLD_* (Command format Channel Dispositions):

MQCHLD_ALL	-1	X'FFFFFFFF'
MQCHLD_DEFAULT	1	X'00000001'
MQCHLD_SHARED	2	X'00000002'
MQCHLD_PRIVATE	4	X'00000004'
MQCHLD_FIXSHARED	5	X'00000005'

MQCHS_* (Command format Channel Status):

MQCHS_INACTIVE	0	X'00000000'
MQCHS_BINDING	1	X'00000001'
MQCHS_STARTING	2	X'00000002'
MQCHS_RUNNING	3	X'00000003'
MQCHS_STOPPING	4	X'00000004'
MQCHS_RETRYING	5	X'00000005'
MQCHS_STOPPED	6	X'00000006'
MQCHS_REQUESTING	7	X'00000007'
MQCHS_PAUSED	8	X'00000008'
MQCHS_INITIALIZING	13	X'0000000D'
MQCHS_SWITCHING	14	X'0000000E'

MQCHSH_* (Command format Channel Shared Restart Options):

MQCHSH_RESTART_NO	0	X'00000000'
MQCHSH_RESTART_YES	1	X'00000001'

MQCHSR_* (Command format Channel Stop Options):

MQCHSR_STOP_NOT_REQUESTED	0	X'00000000'
MQCHSR_STOP_REQUESTED	1	X'00000001'

MQCHSSTATE_* (Command format Channel Substates):

MQCHSSTATE_OTHER	0	X'00000000'
MQCHSSTATE_END_OF_BATCH	100	X'00000064'
MQCHSSTATE_SENDING	200	X'000000C8'
MQCHSSTATE_RECEIVING	300	X'0000012C'
MQCHSSTATE_SERIALIZING	400	X'00000190'
MQCHSSTATE_RESYNCHING	500	X'000001F4'
MQCHSSTATE_HEARTBEATING	600	X'00000258'
MQCHSSTATE_IN_SCYEXIT	700	X'000002BC'
MQCHSSTATE_IN_RCVEXIT	800	X'00000320'
MQCHSSTATE_IN_SENDEXIT	900	X'00000384'
MQCHSSTATE_IN_MSGEXIT	1000	X'000003E8'
MQCHSSTATE_IN_MREXIT	1100	X'0000044C'
MQCHSSTATE_IN_CHADEXIT	1200	X'000004B0'
MQCHSSTATE_NET_CONNECTING	1250	X'000004E2'
MQCHSSTATE_SSL_HANDSHAKING	1300	X'00000514'
MQCHSSTATE_NAME_SERVER	1400	X'00000578'
MQCHSSTATE_IN_MQPUT	1500	X'000005DC'
MQCHSSTATE_IN_MQGET	1600	X'00000640'
MQCHSSTATE_IN_MQI_CALL	1700	X'000006A4'

MQCHSSTATE_COMPRESSING	1800	X'00000708'

MQCHT_* (Channel Types):

MQCHT_SENDER	1	X'00000001'
MQCHT_SERVER	2	X'00000002'
MQCHT_RECEIVER	3	X'00000003'
MQCHT_REQUESTER	4	X'00000004'
MQCHT_ALL	5	X'00000005'
MQCHT_CLNTCONN	6	X'00000006'
MQCHT_SVRCONN	7	X'00000007'
MQCHT_CLUSRCVR	8	X'00000008'
MQCHT_CLUSSDR	9	X'00000009'

MQCHTAB_* (Command format Channel Table Types):

MQCHTAB_Q_MGR	1	X'00000001'
MQCHTAB_CLNTCONN	2	X'00000002'

MQCI_* (Correlation Identifier):

MQCI_NONE	X'00...00'	(24 nulls)
MQCI_NONE_ARRAY	'\0','\0',...	(24 nulls)
MQCI_NEW_SESSION	X'414D5121...'	
MQCI_NEW_SESSION_ARRAY	'\x41','\x4D','\51','\ x21',...	

MQCIH_* (CICS information header structure and Flags):
CICS information header structure

MQCIH_STRUC_ID	"CIHb"	
MQCIH_STRUC_ID_ARRAY	'C','I','H','b'	
MQCIH_VERSION_1	1	X'00000001'
MQCIH_VERSION_2	2	X'00000002'
MQCIH_CURRENT_VERSION	2	X'00000002'
MQCIH_LENGTH_1	164	X'000000A4'
MQCIH_LENGTH_2	180	X'000000B4'
MQCIH_CURRENT_LENGTH	180	X'000000B4'

CICS information header Flags

MQCIH_NONE	0	X'00000000'
MQCIH_PASS_EXPIRATION	1	X'00000001'
MQCIH_UNLIMITED_EXPIRATION	0	X'00000000'
MQCIH_REPLY_WITHOUT_NULLS	2	X'00000002'
MQCIH_REPLY_WITH_NULLS	0	X'00000000'
MQCIH_SYNC_ON_RETURN	4	X'00000004'
MQCIH_NO_SYNC_ON_RETURN	0	X'00000000'

MQCLCT_* (Cluster Cache Types):

MQCLCT_STATIC	0	X'00000000'
MQCLCT_DYNAMIC	1	X'00000001'

MQCLRS_* (Command format Clear Topic String Scope):

MQCLRS_LOCAL	1	X'00000001'
MQCLRS_GLOBAL	2	X'00000002'

MQCLRT_* (Command format Clear Topic String Type):

MQCLRT_RETAINED	1	X'00000001'

MQCLT_* (CICS information header Link Types):

MQCLT_PROGRAM	1	X'00000001'
MQCLT_TRANSACTION	2	X'00000002'

MQCLWL_* (Cluster Workload):

MQCLWL_USEQ_LOCAL	0	X'00000000'
MQCLWL_USEQ_ANY	1	X'00000001'
MQCLWL_USEQ_AS_Q_MGR	-3	X'FFFFFFFD'

MQCLXQ_* (Cluster transmission queue type):

`MQCLXQ_*` are the values you can set in the `DEFCLXQ` queue manager attribute. The **DFTCLXQ** attribute controls which transmission queue is selected by default by cluster-sender channels to get messages from, to send the messages to cluster-receiver channels.

MQCLXQ_SCTQ	0	X'00000000'
MQCLXQ_CHANNEL	1	X'00000001'

Related reference:

"DefClusterXmitQueueType (MQLONG)" on page 2811
The DefClusterXmitQueueType attribute controls which transmission queue is selected by default by cluster-sender channels to get messages from, to send the messages to cluster-receiver channels.

"MQINQ - Inquire object attributes" on page 2700
The MQINQ call returns an array of integers and a set of character strings containing the attributes of an object.

Related information:

Change Queue Manager
The Change Queue Manager (MQCMD_CHANGE_Q_MGR) command changes the specified attributes of the queue manager.

Inquire Queue Manager
The Inquire Queue Manager (**MQCMD_INQUIRE_Q_MGR**) command inquires about the attributes of a queue manager.

Inquire Queue Manager (Response)
The response to the Inquire Queue Manager (**MQCMD_INQUIRE_Q_MGR**) command consists of the response header followed by the *QMgrName* structure and the requested combination of attribute parameter structures.

MQCMD_* (Command Codes):

MQCMD_NONE	0	X'00000000'
MQCMD_CHANGE_Q_MGR	1	X'00000001'
MQCMD_INQUIRE_Q_MGR	2	X'00000002'
MQCMD_CHANGE_PROCESS	3	X'00000003'
MQCMD_COPY_PROCESS	4	X'00000004'
MQCMD_CREATE_PROCESS	5	X'00000005'
MQCMD_DELETE_PROCESS	6	X'00000006'
MQCMD_INQUIRE_PROCESS	7	X'00000007'
MQCMD_CHANGE_Q	8	X'00000008'
MQCMD_CLEAR_Q	9	X'00000009'
MQCMD_COPY_Q	10	X'0000000A'
MQCMD_CREATE_Q	11	X'0000000B'
MQCMD_DELETE_Q	12	X'0000000C'
MQCMD_INQUIRE_Q	13	X'0000000D'
MQCMD_REFRESH_Q_MGR	16	X'00000010'
MQCMD_RESET_Q_STATS	17	X'00000011'
MQCMD_INQUIRE_Q_NAMES	18	X'00000012'
MQCMD_INQUIRE_PROCESS_NAMES	19	X'00000013'
MQCMD_INQUIRE_CHANNEL_NAMES	20	X'00000014'
MQCMD_CHANGE_CHANNEL	21	X'00000015'
MQCMD_COPY_CHANNEL	22	X'00000016'
MQCMD_CREATE_CHANNEL	23	X'00000017'

MQCMD_DELETE_CHANNEL	24	X'00000018'
MQCMD_INQUIRE_CHANNEL	25	X'00000019'
MQCMD_PING_CHANNEL	26	X'0000001A'
MQCMD_RESET_CHANNEL	27	X'0000001B'
MQCMD_START_CHANNEL	28	X'0000001C'
MQCMD_STOP_CHANNEL	29	X'0000001D'
MQCMD_START_CHANNEL_INIT	30	X'0000001E'
MQCMD_START_CHANNEL_LISTENER	31	X'0000001F'
MQCMD_CHANGE_NAMELIST	32	X'00000020'
MQCMD_COPY_NAMELIST	33	X'00000021'
MQCMD_CREATE_NAMELIST	34	X'00000022'
MQCMD_DELETE_NAMELIST	35	X'00000023'
MQCMD_INQUIRE_NAMELIST	36	X'00000024'
MQCMD_INQUIRE_NAMELIST_NAMES	37	X'00000025'
MQCMD_ESCAPE	38	X'00000026'
MQCMD_RESOLVE_CHANNEL	39	X'00000027'
MQCMD_PING_Q_MGR	40	X'00000028'
MQCMD_INQUIRE_Q_STATUS	41	X'00000029'
MQCMD_INQUIRE_CHANNEL_STATUS	42	X'0000002A'
MQCMD_CONFIG_EVENT	43	X'0000002B'
MQCMD_Q_MGR_EVENT	44	X'0000002C'
MQCMD_PERFM_EVENT	45	X'0000002D'
MQCMD_CHANNEL_EVENT	46	X'0000002E'
MQCMD_DELETE_PUBLICATION	60	X'0000003C'
MQCMD_DEREGISTER_PUBLISHER	61	X'0000003D'
MQCMD_DEREGISTER_SUBSCRIBER	62	X'0000003E'
MQCMD_PUBLISH	63	X'0000003F'
MQCMD_REGISTER_PUBLISHER	64	X'00000040'
MQCMD_REGISTER_SUBSCRIBER	65	X'00000041'
MQCMD_REQUEST_UPDATE	66	X'00000042'
MQCMD_BROKER_INTERNAL	67	X'00000043'
MQCMD_ACTIVITY_MSG	69	X'00000045'
MQCMD_INQUIRE_CLUSTER_Q_MGR	70	X'00000046'
MQCMD_RESUME_Q_MGR_CLUSTER	71	X'00000047'
MQCMD_SUSPEND_Q_MGR_CLUSTER	72	X'00000048'
MQCMD_REFRESH_CLUSTER	73	X'00000049'
MQCMD_RESET_CLUSTER	74	X'0000004A'
MQCMD_TRACE_ROUTE	75	X'0000004B'
MQCMD_REFRESH_SECURITY	78	X'0000004E'
MQCMD_CHANGE_AUTH_INFO	79	X'0000004F'
MQCMD_COPY_AUTH_INFO	80	X'00000050'
MQCMD_CREATE_AUTH_INFO	81	X'00000051'

MQCMD_DELETE_AUTH_INFO	82	X'00000052'
MQCMD_INQUIRE_AUTH_INFO	83	X'00000053'
MQCMD_INQUIRE_AUTH_INFO_NAMES	84	X'00000054'
MQCMD_INQUIRE_CONNECTION	85	X'00000055'
MQCMD_STOP_CONNECTION	86	X'00000056'
MQCMD_INQUIRE_AUTH_RECS	87	X'00000057'
MQCMD_INQUIRE_ENTITY_AUTH	88	X'00000058'
MQCMD_DELETE_AUTH_REC	89	X'00000059'
MQCMD_SET_AUTH_REC	90	X'0000005A'
MQCMD_LOGGER_EVENT	91	X'0000005B'
MQCMD_RESET_Q_MGR	92	X'0000005C'
MQCMD_CHANGE_LISTENER	93	X'0000005D'
MQCMD_COPY_LISTENER	94	X'0000005E'
MQCMD_CREATE_LISTENER	95	X'0000005F'
MQCMD_DELETE_LISTENER	96	X'00000060'
MQCMD_INQUIRE_LISTENER	97	X'00000061'
MQCMD_INQUIRE_LISTENER_STATUS	98	X'00000062'
MQCMD_COMMAND_EVENT	99	X'00000063'
MQCMD_CHANGE_SECURITY	100	X'00000064'
MQCMD_CHANGE_CF_STRUC	101	X'00000065'
MQCMD_CHANGE_STG_CLASS	102	X'00000066'
MQCMD_CHANGE_TRACE	103	X'00000067'
MQCMD_ARCHIVE_LOG	104	X'00000068'
MQCMD_BACKUP_CF_STRUC	105	X'00000069'
MQCMD_CREATE_BUFFER_POOL	106	X'0000006A'
MQCMD_CREATE_PAGE_SET	107	X'0000006B'
MQCMD_CREATE_CF_STRUC	108	X'0000006C'
MQCMD_CREATE_STG_CLASS	109	X'0000006D'
MQCMD_COPY_CF_STRUC	110	X'0000006E'
MQCMD_COPY_STG_CLASS	111	X'0000006F'
MQCMD_DELETE_CF_STRUC	112	X'00000070'
MQCMD_DELETE_STG_CLASS	113	X'00000071'
MQCMD_INQUIRE_ARCHIVE	114	X'00000072'
MQCMD_INQUIRE_CF_STRUC	115	X'00000073'
MQCMD_INQUIRE_CF_STRUC_STATUS	116	X'00000074'
MQCMD_INQUIRE_CMD_SERVER	117	X'00000075'
MQCMD_INQUIRE_CHANNEL_INIT	118	X'00000076'
MQCMD_INQUIRE_QSG	119	X'00000077'
MQCMD_INQUIRE_LOG	120	X'00000078'
MQCMD_INQUIRE_SECURITY	121	X'00000079'
MQCMD_INQUIRE_STG_CLASS	122	X'0000007A'
MQCMD_INQUIRE_SYSTEM	123	X'0000007B'

MQCMD_INQUIRE_THREAD	124	X'0000007C'
MQCMD_INQUIRE_TRACE	125	X'0000007D'
MQCMD_INQUIRE_USAGE	126	X'0000007E'
MQCMD_MOVE_Q	127	X'0000007F'
MQCMD_RECOVER_BSDS	128	X'00000080'
MQCMD_RECOVER_CF_STRUC	129	X'00000081'
MQCMD_RESET_TPIPE	130	X'00000082'
MQCMD_RESOLVE_INDOUBT	131	X'00000083'
MQCMD_RESUME_Q_MGR	132	X'00000084'
MQCMD_REVERIFY_SECURITY	133	X'00000085'
MQCMD_SET_ARCHIVE	134	X'00000086'
MQCMD_SET_LOG	136	X'00000088'
MQCMD_SET_SYSTEM	137	X'00000089'
MQCMD_START_CMD_SERVER	138	X'0000008A'
MQCMD_START_Q_MGR	139	X'0000008B'
MQCMD_START_TRACE	140	X'0000008C'
MQCMD_STOP_CHANNEL_INIT	141	X'0000008D'
MQCMD_STOP_CHANNEL_LISTENER	142	X'0000008E'
MQCMD_STOP_CMD_SERVER	143	X'0000008F'
MQCMD_STOP_Q_MGR	144	X'00000090'
MQCMD_STOP_TRACE	145	X'00000091'
MQCMD_SUSPEND_Q_MGR	146	X'00000092'
MQCMD_INQUIRE_CF_STRUC_NAMES	147	X'00000093'
MQCMD_INQUIRE_STG_CLASS_NAMES	148	X'00000094'
MQCMD_CHANGE_SERVICE	149	X'00000095'
MQCMD_COPY_SERVICE	150	X'00000096'
MQCMD_CREATE_SERVICE	151	X'00000097'
MQCMD_DELETE_SERVICE	152	X'00000098'
MQCMD_INQUIRE_SERVICE	153	X'00000099'
MQCMD_INQUIRE_SERVICE_STATUS	154	X'0000009A'
MQCMD_START_SERVICE	155	X'0000009B'
MQCMD_STOP_SERVICE	156	X'0000009C'
MQCMD_DELETE_BUFFER_POOL	157	X'0000009D'
MQCMD_DELETE_PAGE_SET	158	X'0000009E'
MQCMD_CHANGE_BUFFER_POOL	159	X'0000009F'
MQCMD_CHANGE_PAGE_SET	160	X'000000A0'
MQCMD_INQUIRE_Q_MGR_STATUS	161	X'000000A1'
MQCMD_CREATE_LOG	162	X'000000A2'
MQCMD_STATISTICS_MQI	164	X'000000A4'
MQCMD_STATISTICS_Q	165	X'000000A5'
MQCMD_STATISTICS_CHANNEL	166	X'000000A6'
MQCMD_ACCOUNTING_MQI	167	X'000000A7'

MQCMD_ACCOUNTING_Q	168	X'000000A8'
MQCMD_INQUIRE_AUTH_SERVICE	169	X'000000A9'
MQCMD_CHANGE_TOPIC	170	X'000000AA'
MQCMD_COPY_TOPIC	171	X'000000AB'
MQCMD_CREATE_TOPIC	172	X'000000AC'
MQCMD_DELETE_TOPIC	173	X'000000AD'
MQCMD_INQUIRE_TOPIC	174	X'000000AE'
MQCMD_INQUIRE_TOPIC_NAMES	175	X'000000AF'
MQCMD_INQUIRE_SUBSCRIPTION	176	X'000000B0'
MQCMD_CREATE_SUBSCRIPTION	177	X'000000B1'
MQCMD_CHANGE_SUBSCRIPTION	178	X'000000B2'
MQCMD_DELETE_SUBSCRIPTION	179	X'000000B3'
MQCMD_COPY_SUBSCRIPTION	181	X'000000B5'
MQCMD_INQUIRE_SUB_STATUS	182	X'000000B6'
MQCMD_INQUIRE_TOPIC_STATUS	183	X'000000B7'
MQCMD_CLEAR_TOPIC_STRING	184	X'000000B8'
MQCMD_INQUIRE_PUBSUB_STATUS	185	X'000000B9'
MQCMD_PURGE_CHANNEL	195	X'000000C3'

MQCMDI_* (Command format Command Information Values):

MQCMDI_CMDSCOPE_ACCEPTED	1	X'00000001'
MQCMDI_CMDSCOPE_GENERATED	2	X'00000002'
MQCMDI_CMDSCOPE_COMPLETED	3	X'00000003'
MQCMDI_QSG_DISP_COMPLETED	4	X'00000004'
MQCMDI_COMMAND_ACCEPTED	5	X'00000005'
MQCMDI_CLUSTER_REQUEST_QUEUED	6	X'00000006'
MQCMDI_CHANNEL_INIT_STARTED	7	X'00000007'
MQCMDI_RECOVER_STARTED	11	X'0000000B'
MQCMDI_BACKUP_STARTED	12	X'0000000C'
MQCMDI_RECOVER_COMPLETED	13	X'0000000D'
MQCMDI_SEC_TIMER_ZERO	14	X'0000000E'
MQCMDI_REFRESH_CONFIGURATION	16	X'00000010'
MQCMDI_SEC_SIGNOFF_ERROR	17	X'00000011'
MQCMDI_IMS_BRIDGE_SUSPENDED	18	X'00000012'
MQCMDI_DB2_SUSPENDED	19	X'00000013'
MQCMDI_DB2_OBSOLETE_MSGS	20	X'00000014'
MQCMDI_SEC_UPPERCASE	21	X'00000015'
MQCMDI_SEC_MIXEDCASE	22	X'00000016'

MQCMDL_* (Command Levels):

MQCMDL_LEVEL_1	100
MQCMDL_LEVEL_101	101
MQCMDL_LEVEL_110	110
MQCMDL_LEVEL_114	114
MQCMDL_LEVEL_120	120
MQCMDL_LEVEL_200	200
MQCMDL_LEVEL_201	201
MQCMDL_LEVEL_210	210
MQCMDL_LEVEL_211	211
MQCMDL_LEVEL_220	220
MQCMDL_LEVEL_221	221
MQCMDL_LEVEL_230	230
MQCMDL_LEVEL_320	320
MQCMDL_LEVEL_420	420
MQCMDL_LEVEL_500	500
MQCMDL_LEVEL_510	510
MQCMDL_LEVEL_520	520
MQCMDL_LEVEL_530	530
MQCMDL_LEVEL_531	531
MQCMDL_LEVEL_600	600
MQCMDL_LEVEL_700	700
MQCMDL_LEVEL_701	701
MQCMDL_LEVEL_710	710
MQCMDL_LEVEL_711	711
MQCMDL_LEVEL_750	750
MQCMDL_LEVEL_800	800
MQCMDL_LEVEL_801	801
MQCMDL_LEVEL_802	802
 MQCMDL_LEVEL_900	900
MQCMDL_LEVEL_901	901
 MQCMDL_LEVEL_902	902

MQCMHO_* (Create message handle options and structure):
Create message handle options structure

MQCMHO_STRUC_ID	"CMHO"	
MQCMHO_STRUC_ID_ARRAY	'C','M','H','O'	
MQCMHO_VERSION_1	1	X'00000001'
MQCMHO_CURRENT_VERSION	1	X'00000001'

Create Message Handle Options

MQCMHO_DEFAULT_VALIDATION	0	X'00000000'
MQCMHO_NO_VALIDATION	1	X'00000001'
MQCMHO_VALIDATE	2	X'00000002'
MQCMHO_NONE	0	X'00000000'

MQCNO_* (Connect options and structure):
Connect options structure

MQCNO_STRUC_ID	"CNOb"	
MQCNO_STRUC_ID_ARRAY	'C','N','O','b'	
MQCNO_VERSION_1	1	X'00000001'
MQCNO_VERSION_2	2	X'00000002'
MQCNO_VERSION_3	3	X'00000003'
MQCNO_VERSION_4	4	X'00000004'
MQCNO_VERSION_5	5	X'00000005'
MQCNO_CURRENT_VERSION	5	X'00000005'

Connect Options

MQCNO_STANDARD_BINDING	0	X'00000000'
MQCNO_FASTPATH_BINDING	1	X'00000001'
MQCNO_SERIALIZE_CONN_TAG_Q_MGR	2	X'00000002'
MQCNO_SERIALIZE_CONN_TAG_QSG	4	X'00000004'
MQCNO_RESTRICT_CONN_TAG_Q_MGR	8	X'00000008'
MQCNO_RESTRICT_CONN_TAG_QSG	16	X'00000010'
MQCNO_HANDLE_SHARE_NONE	32	X'00000020'
MQCNO_HANDLE_SHARE_BLOCK	64	X'00000040'
MQCNO_HANDLE_SHARE_NO_BLOCK	128	X'00000080'
MQCNO_SHARED_BINDING	256	X'00000100'
MQCNO_ISOLATED_BINDING	512	X'00000200'
MQCNO_LOCAL_BINDING	1024	X'00000400'
MQCNO_CLIENT_BINDING	2048	X'00000800'
MQCNO_ACCOUNTING_MQI_ENABLED	4096	X'00001000'
MQCNO_ACCOUNTING_MQI_DISABLED	8192	X'00002000'
MQCNO_ACCOUNTING_Q_ENABLED	16384	X'00004000'
MQCNO_ACCOUNTING_Q_DISABLED	32768	X'00008000'

MQCNO_NO_CONV_SHARING	65536	X'00010000'
MQCNO_ALL_CONVS_SHARE	262144	X'00040000'
MQCNO_CD_FOR_OUTPUT_ONLY	524288	X'00080000'
MQCNO_USE_CD_SELECTION	1048576	X'00100000'
MQCNO_RECONNECT	16777216	X'01000000'
MQCNO_RECONNECT_AS_DEF	0	X'00000000'
MQCNO_RECONNECT_DISABLED	33554432	X'02000000'
MQCNO_RECONNECT_Q_MGR	67108864	X'04000000'
MQCNO_ACTIVITY_TRACE_ENABLED	134217728	X'08000000'
MQCNO_ACTIVITY_TRACE_DISABLED	268435456	X'10000000'
MQCNO_NONE	0	X'00000000'

MQCO_* (Close Options):

MQCO_IMMEDIATE	0	X'00000000'
MQCO_NONE	0	X'00000000'
MQCO_DELETE	1	X'00000001'
MQCO_DELETE_PURGE	2	X'00000002'
MQCO_KEEP_SUB	4	X'00000004'
MQCO_REMOVE_SUB	8	X'00000008'
MQCO_QUIESCE	32	X'00000020'

MQCODL_* (CICS information header Output Data Length):

MQCODL_AS_INPUT	-1	X'FFFFFFFF'

MQCOMPRESS_* (Channel Compression):

MQCOMPRESS_NOT_AVAILABLE	-1	X'FFFFFFFF'
MQCOMPRESS_NONE	0	X'00000000'
MQCOMPRESS_RLE	1	X'00000001'
MQCOMPRESS_ZLIBFAST	2	X'00000002'
MQCOMPRESS_ZLIBHIGH	4	X'00000004'
MQCOMPRESS_SYSTEM	8	X'00000008'
MQCOMPRESS_ANY	268435455	X'0FFFFFFF'

MQCONNID_* (Connection Identifier):

MQCONNID_NONE	X'00...00'	(24 nulls)
MQCONNID_NONE_ARRAY	'\0','\0',...	(24 nulls)

MQCOPY_* (Property Copy Options):

MQCOPY_NONE	0	X'00000000'
MQCOPY_ALL	1	X'00000001'
MQCOPY_FORWARD	2	X'00000002'
MQCOPY_PUBLISH	4	X'00000004'
MQCOPY_REPLY	8	X'00000008'
MQCOPY_REPORT	16	X'00000010'
MQCOPY_DEFAULT	22	X'00000016'

MQCQT_* (Cluster Queue Types):

MQCQT_LOCAL_Q	1	X'00000001'
MQCQT_ALIAS_Q	2	X'00000002'
MQCQT_REMOTE_Q	3	X'00000003'
MQCQT_Q_MGR_ALIAS	4	X'00000004'

MQCRC_* (CICS information header Return Codes):

MQCRC_OK	0	X'00000000'
MQCRC_CICS_EXEC_ERROR	1	X'00000001'
MQCRC_MQ_API_ERROR	2	X'00000002'
MQCRC_BRIDGE_ERROR	3	X'00000003'
MQCRC_BRIDGE_ABEND	4	X'00000004'
MQCRC_APPLICATION_ABEND	5	X'00000005'
MQCRC_SECURITY_ERROR	6	X'00000006'
MQCRC_PROGRAM_NOT_AVAILABLE	7	X'00000007'
MQCRC_BRIDGE_TIMEOUT	8	X'00000008'
MQCRC_TRANSID_NOT_AVAILABLE	9	X'00000009'

MQCS_* (MQCBC constants Consumer state):

MQCS_NONE	0	X'00000000'
MQCS_SUSPENDED_TEMPORARY	1	X'00000001'
MQCS_SUSPENDED_USER_ACTION	2	X'00000002'
MQCS_SUSPENDED	3	X'00000003'
MQCS_STOPPED	4	X'00000004'

MQCSC_* (CICS information header Start Codes):

MQCSC_START	"Sbbb"	
MQCSC_STARTDATA	"SDbb"	
MQCSC_TERMINPUT	"TDbb"	
MQCSC_NONE	"bbbb"	
MQCSC_START_ARRAY	'S','b','b','b'	
MQCSC_STARTDATA_ARRAY	'S','D','b','b'	
MQCSC_TERMINPUT_ARRAY	'T','D','b','b'	
MQCSC_NONE_ARRAY	'b','b','b','b'	

MQCSP_* (Connection security parameters structure and Authentication Types):
Connection security parameters structure

MQCSP_STRUC_ID	"CSPb"	
MQCSP_STRUC_ID_ARRAY	'C','S','P','b'	
MQCSP_VERSION_1	1	X'00000001'
MQCSP_CURRENT_VERSION	1	X'00000001'

Connection security parameters Authentication Types

MQCSP_AUTH_NONE	0	X'00000000'
MQCSP_AUTH_USER_ID_AND_PWD	1	X'00000001'

MQCSRV_* (Command Server Options):

MQCSRV_CONVERT_NO	0	X'00000000'
MQCSRV_CONVERT_YES	1	X'00000001'
MQCSRV_DLQ_NO	0	X'00000000'
MQCSRV_DLQ_YES	1	X'00000001'

MQCT_* (Queue Manager Connection Tag):

MQCT_NONE	X'00...00'	(128 nulls)
MQCT_NONE_ARRAY	'\0','\0',...	(128 nulls)

MQCTES_* (CICS information header Task End Status):

MQCTES_NOSYNC	0	X'00000000'
MQCTES_COMMIT	256	X'00000100'
MQCTES_BACKOUT	4352	X'00001100'
MQCTES_ENDTASK	65536	X'00010000'

MQCTLO_* (MQCTL options structure and Consumer Control Options):
MQCTL options structure

MQCTLO_STRUC_ID	"CTLO"	
MQCTLO_STRUC_ID_ARRAY	'C','T','L','O'	
MQCTLO_VERSION_1	1	X'00000001'
MQCTLO_CURRENT_VERSION	1	X'00000001'

MQCTL options Consumer Control Options

MQCTLO_NONE	0	X'00000000'
MQCTLO_THREAD_AFFINITY	1	X'00000001'
MQCTLO_FAIL_IF_QUIESCING	8192	X'00002000'

MQCUOWC_* (CICS information header Unit-of-Work Controls):

MQCUOWC_ONLY	273	X'00000111'
MQCUOWC_CONTINUE	65536	X'00010000'
MQCUOWC_FIRST	17	X'00000011'
MQCUOWC_MIDDLE	16	X'00000010'
MQCUOWC_LAST	272	X'00000110'
MQCUOWC_COMMIT	256	X'00000100'
MQCUOWC_BACKOUT	4352	X'00001100'

MQCXP_* (Channel exit parameter structure):

MQCXP_STRUC_ID	"CXPb"	
MQCXP_STRUC_ID_ARRAY	'C','X','P','b'	
MQCXP_VERSION_1	1	X'00000001'
MQCXP_VERSION_2	2	X'00000002'
MQCXP_VERSION_3	3	X'00000003'
MQCXP_VERSION_4	4	X'00000004'
MQCXP_VERSION_5	5	X'00000005'
MQCXP_VERSION_6	6	X'00000006'
MQCXP_VERSION_7	7	X'00000007'
MQCXP_VERSION_8	8	X'00000008'
MQCXP_VERSION_9	9	X'00000009'
MQCXP_CURRENT_VERSION	9	X'00000009'

MQDC_* (Destination Class):

MQDC_MANAGED	1	X'00000001'
MQDC_PROVIDED	2	X'00000002'

MQDCC_* (Conversion Options, and Masks and Factors):
Conversion Options

MQDCC_DEFAULT_CONVERSION	1	X'00000001'
MQDCC_FILL_TARGET_BUFFER	2	X'00000002'
MQDCC_INT_DEFAULT_CONVERSION	4	X'00000004'
MQDCC_SOURCE_ENC_NATIVE	(value differs by platform or version)	
MQDCC_SOURCE_ENC_NORMAL	16	X'00000010'
MQDCC_SOURCE_ENC_REVERSED	32	X'00000020'
MQDCC_SOURCE_ENC_UNDEFINED	0	X'00000000'
MQDCC_TARGET_ENC_NATIVE	(value differs by platform or version)	
MQDCC_TARGET_ENC_NORMAL	256	X'00000100'
MQDCC_TARGET_ENC_REVERSED	512	X'00000200'
MQDCC_TARGET_ENC_UNDEFINED	0	X'00000000'
MQDCC_NONE	0	X'00000000'

Conversion Options Masks and Factors

MQDCC_SOURCE_ENC_MASK	240	X'000000F0'
MQDCC_TARGET_ENC_MASK	3840	X'00000F00'
MQDCC_SOURCE_ENC_FACTOR	16	X'00000010'
MQDCC_TARGET_ENC_FACTOR	256	X'00000100'

MQDELO_* (Publish/Subscribe Delete Options):

MQDELO_NONE	0	X'00000000'
MQDELO_LOCAL	4	X'00000004'

MQDH_* (Distribution header structure):

MQDH_STRUC_ID	"DHbb"	
MQDH_STRUC_ID_ARRAY	'D','H','b','b'	
MQDH_VERSION_1	1	X'00000001'
MQDH_CURRENT_VERSION	1	X'00000001'

MQDHF_* (Distribution header Flags):

MQDHF_NEW_MSG_IDS	1	X'00000001'
MQDHF_NONE	0	X'00000000'

MQDISCONNECT_* (Command format Disconnect Types):

MQDISCONNECT_NORMAL	0	X'00000000'
MQDISCONNECT_IMPLICIT	1	X'00000001'
MQDISCONNECT_Q_MGR	2	X'00000002'

MQDL_* (Distribution Lists):

MQDL_SUPPORTED	1	X'00000001'
MQDL_NOT_SUPPORTED	0	X'00000000'

MQDLH_* (Dead-letter header structure):

MQDLH_STRUC_ID	"DLHb"	
MQDLH_STRUC_ID_ARRAY	'D','L','H','b'	
MQDLH_VERSION_1	1	X'00000001'
MQDLH_CURRENT_VERSION	1	X'00000001'

MQDLV_* (Persistent/Non-persistent Message Delivery):

MQDLV_AS_PARENT	0	X'00000000'
MQDLV_ALL	1	X'00000001'
MQDLV_ALL_DUR	2	X'00000002'
MQDLV_ALL_AVAIL	3	X'00000003'

MQDMHO_* (Delete message handle options and structure):
Delete message handle options structure

MQDMHO_STRUC_ID	"DMHO"	
MQDMHO_STRUC_ID_ARRAY	'D','M','H','O'	
MQDMHO_VERSION_1	1	X'00000001'
MQDMHO_CURRENT_VERSION	1	X'00000001'

Delete Message Handle Options

MQDMHO_NONE	0	X'00000000'

MQDMPO_* (Delete message property options and structure):
Delete message property options structure

MQDMPO_STRUC_ID	"DMPO"	
MQDMPO_STRUC_ID_ARRAY	'D','M','P','O'	
MQDMPO_VERSION_1	1	X'00000001'
MQDMPO_CURRENT_VERSION	1	X'00000001'

Delete Message Property Options

MQDMPO_DEL_FIRST	0	X'00000000'
MQDMPO_DEL_PROP_UNDER_CURSOR	1	X'00000001'
MQDMPO_NONE	0	X'00000000'

MQDNSWLM_* (DNS WLM):

MQDNSWLM_NO	0	X'00000000'
MQDNSWLM_YES	1	X'00000001'

MQDT_* (Destination Types):

MQDT_APPL	1	X'00000001'
MQDT_BROKER	2	X'00000002'

MQDXP_* (Conversion exit parameter structure):

MQDXP_STRUC_ID	"DXPb"	
MQDXP_STRUC_ID_ARRAY	'D','X','P','b'	
MQDXP_VERSION_1	1	X'00000001'
MQDXP_VERSION_2	2	X'00000002'
MQDXP_CURRENT_VERSION	2	X'00000002'

MQEC_* (Signal Values):

MQEC_MSG_ARRIVED	2	X'00000002'
MQEC_WAIT_INTERVAL_EXPIRED	3	X'00000003'
MQEC_WAIT_CANCELED	4	X'00000004'
MQEC_Q_MGR_QUIESCING	5	X'00000005'
MQEC_CONNECTION_QUIESCING	6	X'00000006'

MQEI_* (Expiry):

MQEI_UNLIMITED	-1	X'FFFFFFFF'

MQENC_* (Encoding):
MQENC_* (Encoding)

MQENC_NATIVE	IBM i	273	X'00000111'
	Linux	546	X'00000222'
	Linux on SPARC	273	X'00000111'
	Linux on x86	546	X'00000222'
	Solaris on SPARC	273	X'00000111'
	UNIX	273	X'00000111'
	Windows	546	X'00000222'
	Micro Focus COBOL on Windows	17	X'00000011'
	z/OS	785	X'00000311'

MQENC_* (Encoding Masks)

MQENC_INTEGER_MASK	15	X'0000000F'
MQENC_DECIMAL_MASK	240	X'000000F0'
MQENC_FLOAT_MASK	3840	X'00000F00'
MQENC_RESERVED_MASK	-4096	X'FFFFF000'

MQENC_* (Encodings for Binary Integers)

MQENC_INTEGER_UNDEFINED	0	X'00000000'
MQENC_INTEGER_NORMAL	1	X'00000001'
MQENC_INTEGER_REVERSED	2	X'00000002'

MQENC_* (Encodings for Packed Decimal Integers)

MQENC_DECIMAL_UNDEFINED	0	X'00000000'
MQENC_DECIMAL_NORMAL	16	X'00000010'
MQENC_DECIMAL_REVERSED	32	X'00000020'

MQENC_* (Encodings for Floating Point Numbers)

MQENC_FLOAT_UNDEFINED	0	X'00000000'
MQENC_FLOAT_IEEE_NORMAL	256	X'00000100'
MQENC_FLOAT_IEEE_REVERSED	512	X'00000200'
MQENC_FLOAT_S390	768	X'00000300'
MQENC_FLOAT_TNS	1024	X'00000400'

MQEPH_* (Embedded command format header structure and Flags):
Embedded command format header structure

MQEPH_STRUC_ID	"EPHb"	
MQEPH_STRUC_ID_ARRAY	'E','P','H','b'	
MQEPH_STRUC_LENGTH_FIXED	68	X'00000044'
MQEPH_VERSION_1	1	X'00000001'
MQEPH_CURRENT_VERSION	1	X'00000001'

Embedded command format header Flags

MQEPH_NONE	0	X'00000000'
MQEPH_CCSID_EMBEDDED	1	X'00000001'

MQET_* (Command format Escape Types):

MQET_MQSC	1	X'00000001'

MQEVO_* (Command format Event Origins):

MQEVO_OTHER	0	X'00000000'
MQEVO_CONSOLE	1	X'00000001'
MQEVO_INIT	2	X'00000002'
MQEVO_MSG	3	X'00000003'
MQEVO_MQSET	4	X'00000004'
MQEVO_INTERNAL	5	X'00000005'
MQEVO_MQSUB	6	X'00000006'
MQEVO_CTLMSG	7	X'00000007'
V 9.0.1 MQEVO_REST	8	X'00000008'

MQEVR_* (Command format Event Recording):

MQEVR_DISABLED	0	X'00000000'
MQEVR_ENABLED	1	X'00000001'
MQEVR_EXCEPTION	2	X'00000002'
MQEVR_NO_DISPLAY	3	X'00000003'

MQEXPI_* (Expiration Scan Interval):

MQEXPI_OFF	0	X'00000000'

MQFB_* (Feedback Values):

MQFB_NONE	0	X'00000000'
MQFB_SYSTEM_FIRST	1	X'00000001'
MQFB_QUIT	256	X'00000100'
MQFB_EXPIRATION	258	X'00000102'
MQFB_COA	259	X'00000103'
MQFB_COD	260	X'00000104'
MQFB_CHANNEL_COMPLETED	262	X'00000106'
MQFB_CHANNEL_FAIL_RETRY	263	X'00000107'
MQFB_CHANNEL_FAIL	264	X'00000108'
MQFB_APPL_CANNOT_BE_STARTED	265	X'00000109'
MQFB_TM_ERROR	266	X'0000010A'
MQFB_APPL_TYPE_ERROR	267	X'0000010B'
MQFB_STOPPED_BY_MSG_EXIT	268	X'0000010C'
MQFB_ACTIVITY	269	X'0000010D'
MQFB_XMIT_Q_MSG_ERROR	271	X'0000010F'
MQFB_PAN	275	X'00000113'
MQFB_NAN	276	X'00000114'
MQFB_STOPPED_BY_CHAD_EXIT	277	X'00000115'
MQFB_STOPPED_BY_PUBSUB_EXIT	279	X'00000117'
MQFB_NOT_A_REPOSITORY_MSG	280	X'00000118'
MQFB_BIND_OPEN_CLUSRCVR_DEL	281	X'00000119'
MQFB_MAX_ACTIVITIES	282	X'0000011A'
MQFB_NOT_FORWARDED	283	X'0000011B'
MQFB_NOT_DELIVERED	284	X'0000011C'
MQFB_UNSUPPORTED_FORWARDING	285	X'0000011D'
MQFB_UNSUPPORTED_DELIVERY	286	X'0000011E'
MQFB_DATA_LENGTH_ZERO	291	X'00000123'
MQFB_DATA_LENGTH_NEGATIVE	292	X'00000124'
MQFB_DATA_LENGTH_TOO_BIG	293	X'00000125'
MQFB_BUFFER_OVERFLOW	294	X'00000126'
MQFB_LENGTH_OFF_BY_ONE	295	X'00000127'
MQFB_IIH_ERROR	296	X'00000128'
MQFB_NOT_AUTHORIZED_FOR_IMS	298	X'0000012A'
MQFB_IMS_ERROR	300	X'0000012C'
MQFB_IMS_FIRST	301	X'0000012D'
MQFB_IMS_LAST	399	X'0000018F'
MQFB_CICS_INTERNAL_ERROR	401	X'00000191'
MQFB_CICS_NOT_AUTHORIZED	402	X'00000192'
MQFB_CICS_BRIDGE_FAILURE	403	X'00000193'
MQFB_CICS_CORREL_ID_ERROR	404	X'00000194'
MQFB_CICS_CCSID_ERROR	405	X'00000195'
MQFB_CICS_ENCODING_ERROR	406	X'00000196'

MQFB_CICS_CIH_ERROR	407	X'00000197'
MQFB_CICS_UOW_ERROR	408	X'00000198'
MQFB_CICS_COMMAREA_ERROR	409	X'00000199'
MQFB_CICS_APPL_NOT_STARTED	410	X'0000019A'
MQFB_CICS_APPL_ABENDED	411	X'0000019B'
MQFB_CICS_DLQ_ERROR	412	X'0000019C'
MQFB_CICS_UOW_BACKED_OUT	413	X'0000019D'
MQFB_PUBLICATIONS_ON_REQUEST	501	X'000001F5'
MQFB_SUBSCRIBER_IS_PUBLISHER	502	X'000001F6'
MQFB_MSG_SCOPE_MISMATCH	503	X'000001F7'
MQFB_SELECTOR_MISMATCH	504	X'000001F8'
MQFB_IMS_NACK_1A_REASON_FIRST	600	X'00000258'
MQFB_IMS_NACK_1A_REASON_LAST	855	X'00000357'
MQFB_SYSTEM_LAST	65535	X'0000FFFF'
MQFB_APPL_FIRST	65536	X'00010000'
MQFB_APPL_LAST	999999999	X'3B9AC9FF'

MQFC_* (Command format Force Options):

MQFC_YES	1	X'00000001'
MQFC_NO	0	X'00000000'

MQFMT_* (Formats):

MQFMT_NONE	"bbbbbbbb"
MQFMT_ADMIN	"MQADMINb"
MQFMT_CHANNEL_COMPLETED	"MQCHCOMb"
MQFMT_CICS	"MQCICSbb"
MQFMT_COMMAND_1	"MQCMD1bb"
MQFMT_COMMAND_2	"MQCMD2bb"
MQFMT_DEAD_LETTER_HEADER	"MQDEADbb"
MQFMT_DIST_HEADER	"MQHDISTb"
MQFMT_EMBEDDED_PCF	"MQHEPCFb"
MQFMT_EVENT	"MQEVENTb"
MQFMT_IMS	"MQIMSbbb"
MQFMT_IMS_VAR_STRING	"MQIMSVSb"
MQFMT_MD_EXTENSION	"MQHMDEbb"
MQFMT_PCF	"MQPCFbbb"
MQFMT_REF_MSG_HEADER	"MQHREFbb"
MQFMT_RF_HEADER	"MQHRFbbb"
MQFMT_RF_HEADER_1	"MQHRFbbb"
MQFMT_RF_HEADER_2	"MQHRF2bb"
MQFMT_STRING	"MQSTRbbb"

MQFMT_TRIGGER	"MQTRIGbb"
MQFMT_WORK_INFO_HEADER	"MQHWIHbb"
MQFMT_XMIT_Q_HEADER	"MQXMITbb"
MQFMT_NONE_ARRAY	'b','b','b','b','b','b','b','b'
MQFMT_ADMIN_ARRAY	'M','Q','A','D','M','I','N','b'
MQFMT_CHANNEL_COMPLETED_ARRAY	'M','Q','C','H','C','O','M','b'
MQFMT_CICS_ARRAY	'M','Q','C','I','C','S','b','b'
MQFMT_COMMAND_1_ARRAY	'M','Q','C','M','D','1','b','b'
MQFMT_COMMAND_2_ARRAY	'M','Q','C','M','D','2','b','b'
MQFMT_DEAD_LETTER_HEADER_ARRAY	'M','Q','D','E','A','D','b','b'
MQFMT_DIST_HEADER_ARRAY	'M','Q','H','D','I','S','T','b'
MQFMT_EMBEDDED_PCF_ARRAY	'M','Q','H','E','P','C','F','b'
MQFMT_EVENT_ARRAY	'M','Q','E','V','E','N','T','b'
MQFMT_IMS_ARRAY	'M','Q','I','M','S','b','b','b'
MQFMT_IMS_VAR_STRING_ARRAY	'M','Q','I','M','S','V','S','b'
MQFMT_MD_EXTENSION_ARRAY	'M','Q','H','M','D','E','b','b'
MQFMT_PCF_ARRAY	'M','Q','P','C','F','b','b','b'
MQFMT_REF_MSG_HEADER_ARRAY	'M','Q','H','R','E','F','b','b'
MQFMT_RF_HEADER_ARRAY	'M','Q','H','R','F','b','b','b'
MQFMT_RF_HEADER_1_ARRAY	'M','Q','H','R','F','b','b','b'
MQFMT_RF_HEADER_2_ARRAY	'M','Q','H','R','F','2','b','b'
MQFMT_STRING_ARRAY	'M','Q','S','T','R','b','b','b'
MQFMT_TRIGGER_ARRAY	'M','Q','T','R','I','G','b','b'
MQFMT_WORK_INFO_HEADER_ARRAY	'M','Q','H','W','I','H','b','b'
MQFMT_XMIT_Q_HEADER_ARRAY	'M','Q','X','M','I','T','b','b'

MQGA_* (Group Attribute Selectors):

MQGA_FIRST	8001	X'00001F41'
MQGA_LAST	9000	X'00002328'

MQGACF_* (Command format Group Parameter Types):

MQGACF_FIRST	8001	X'00001F41'
MQGACF_COMMAND_CONTEXT	8001	X'00001F41'
MQGACF_COMMAND_DATA	8002	X'00001F42'
MQGACF_TRACE_ROUTE	8003	X'00001F43'
MQGACF_OPERATION	8004	X'00001F44'
MQGACF_ACTIVITY	8005	X'00001F45'
MQGACF_EMBEDDED_MQMD	8006	X'00001F46'
MQGACF_MESSAGE	8007	X'00001F47'
MQGACF_MQMD	8008	X'00001F48'
MQGACF_VALUE_NAMING	8009	X'00001F49'

MQGACF_Q_ACCOUNTING_DATA	8010	X'00001F4A'
MQGACF_Q_STATISTICS_DATA	8011	X'00001F4B'
MQGACF_CHL_STATISTICS_DATA	8012	X'00001F4C'
MQGACF_LAST_USED	8012	X'00001F4C'

MQGI_* (Group Identifier):

MQGI_NONE	X'00...00'	(24 nulls)
MQGI_NONE_ARRAY	'\0','\0',...	(24 nulls)

MQGMO_* (Get message options and structure):
Get message options structure

MQGMO_STRUC_ID	"GMOb"	
MQGMO_STRUC_ID_ARRAY	'G','M','O','b'	
MQGMO_VERSION_1	1	X'00000001'
MQGMO_VERSION_2	2	X'00000002'
MQGMO_VERSION_3	3	X'00000003'
MQGMO_VERSION_4	4	X'00000004'
MQGMO_CURRENT_VERSION	4	X'00000004'

Get Message Options

MQGMO_WAIT	1	X'00000001'
MQGMO_NO_WAIT	0	X'00000000'
MQGMO_SET_SIGNAL	8	X'00000008'
MQGMO_FAIL_IF_QUIESCING	8192	X'00002000'
MQGMO_SYNCPOINT	2	X'00000002'
MQGMO_SYNCPOINT_IF_PERSISTENT	4096	X'00001000'
MQGMO_NO_SYNCPOINT	4	X'00000004'
MQGMO_MARK_SKIP_BACKOUT	128	X'00000080'
MQGMO_BROWSE_FIRST	16	X'00000010'
MQGMO_BROWSE_NEXT	32	X'00000020'
MQGMO_BROWSE_MSG_UNDER_CURSOR	2048	X'00000800'
MQGMO_BROWSE_HANDLE	17825808	X'01100010'
MQGMO_BROWSE_CO_OP	18874384	X'01200010'
MQGMO_MSG_UNDER_CURSOR	256	X'00000100'
MQGMO_LOCK	512	X'00000200'
MQGMO_UNLOCK	1024	X'00000400'
MQGMO_ACCEPT_TRUNCATED_MSG	64	X'00000040'
MQGMO_CONVERT	16384	X'00004000'
MQGMO_LOGICAL_ORDER	32768	X'00008000'
MQGMO_COMPLETE_MSG	65536	X'00010000'

MQGMO_ALL_MSGS_AVAILABLE	131072	X'00020000'
MQGMO_ALL_SEGMENTS_AVAILABLE	262144	X'00040000'
MQGMO_MARK_BROWSE_HANDLE	1048576	X'00100000'
MQGMO_MARK_BROWSE_CO_OP	2097152	X'00200000'
MQGMO_UNMARK_BROWSE_CO_OP	4194304	X'00400000'
MQGMO_UNMARK_BROWSE_HANDLE	8388608	X'00800000'
MQGMO_UNMARKED_BROWSE_MSG	16777216	X'01000000'
MQGMO_PROPERTIES_FORCE_MQRFH2	33554432	X'02000000'
MQGMO_NO_PROPERTIES	67108864	X'04000000'
MQGMO_PROPERTIES_IN_HANDLE	134217728	X'08000000'
MQGMO_PROPERTIES_COMPATIBILITY	268435456	X'10000000'
MQGMO_PROPERTIES_AS_Q_DEF	0	X'00000000'
MQGMO_NONE	0	X'00000000'

MQGS_* (Group Status):

MQGS_NOT_IN_GROUP	'b' (blank)	
MQGS_MSG_IN_GROUP	'G'	
MQGS_LAST_MSG_IN_GROUP	'L'	

MQHA_* (Handle Selectors):

MQHA_FIRST	4001	X'00000FA1'
MQHA_BAG_HANDLE	4001	X'00000FA1'
MQHA_LAST_USED	4001	X'00000FA1'
MQHA_LAST	6000	X'00001770'

MQHB_* (Bag Handles):

MQHB_UNUSABLE_HBAG	-1	X'FFFFFFFF'
MQHB_NONE	-2	X'FFFFFFFE'

MQHC_* (Connection Handles):

MQHC_DEF_HCONN	0	X'00000000'
MQHC_UNUSABLE_HCONN	-1	X'FFFFFFFF'
MQHC_UNASSOCIATED_HCONN	-3	X'FFFFFFFD'

MQHM_* (Message handle):

MQHM_UNUSABLE_HMSG	-1	X'FFFFFFFF'
MQHM_NONE	0	X'00000000'

MQHO_* (Object Handle):

MQHO_UNUSABLE_HOBJ	-1	X'FFFFFFFF'
MQHO_NONE	0	X'00000000'

MQHSTATE_* (Command format Handle States):

MQHSTATE_INACTIVE	0	X'00000000'
MQHSTATE_ACTIVE	1	X'00000001'

MQIA_* (Integer Attribute Selectors):

MQIA_ACCOUNTING_CONN_OVERRIDE	136	X'00000088'
MQIA_ACCOUNTING_INTERVAL	135	X'00000087'
MQIA_ACCOUNTING_MQI	133	X'00000085'
MQIA_ACCOUNTING_Q	134	X'00000086'
MQIA_ACTIVE_CHANNELS	100	X'00000064'
MQIA_ACTIVITY_CONN_OVERRIDE	239	X'000000EF'
MQIA_ACTIVITY_RECORDING	138	X'0000008A'
MQIA_ACTIVITY_TRACE	240	X'000000F0'
MQIA_ADOPTNEWMCA_CHECK	102	X'00000066'
MQIA_ADOPTNEWMCA_INTERVAL	104	X'00000068'
MQIA_ADOPTNEWMCA_TYPE	103	X'00000067'
MQIA_ADOPT_CONTEXT	260	X'00000104'
	273	X'00000111'
 MQIA_ADVANCED_CAPABILITY MQIA_AMQP_CAPABILITY	265	X'00000109'
MQIA_APPL_TYPE	1	X'00000001'
MQIA_ARCHIVE	60	X'0000003C'
MQIA_AUTHENTICATION_FAIL_DELAY	259	X'00000103'
MQIA_AUTHENTICATION_METHOD	266	X'0000010A'
MQIA_AUTH_INFO_TYPE	66	X'00000042'
MQIA_AUTHORITY_EVENT	47	X'0000002F'
MQIA_AUTO_REORG_INTERVAL	174	X'000000AE'
MQIA_AUTO_REORGANIZATION	173	X'000000AD'
MQIA_BACKOUT_THRESHOLD	22	X'00000016'
MQIA_BASE_TYPE	193	X'000000C1'
MQIA_BATCH_INTERFACE_AUTO	86	X'00000056'
MQIA_BRIDGE_EVENT	74	X'0000004A'

MQIA_CF_LEVEL	70	X'00000046'
MQIA_CF_RECOVER	71	X'00000047'
MQIA_CHANNEL_AUTO_DEF	55	X'00000037'
MQIA_CHANNEL_AUTO_DEF_EVENT	56	X'00000038'
MQIA_CHANNEL_EVENT	73	X'00000049'
MQIA_CHECK_CLIENT_BINDING	258	X'00000102'
MQIA_CHECK_LOCAL_BINDING	257	X'00000101'
MQIA_CHINIT_ADAPTERS	101	X'00000065'
MQIA_CHINIT_CONTROL	119	X'00000077'
MQIA_CHINIT_DISPATCHERS	105	X'00000069'
MQIA_CHINIT_TRACE_AUTO_START	117	X'00000075'
MQIA_CHINIT_TRACE_TABLE_SIZE	118	X'00000076'
MQIA_CLUSTER_OBJECT_STATE	256	X'00000100'
MQIA_CLUSTER_PUB_ROUTE	255	X'000000FF'
MQIA_CLUSTER_Q_TYPE	59	X'0000003B'
MQIA_CLUSTER_WORKLOAD_LENGTH	58	X'0000003A'
MQIA_CLWL_MRU_CHANNELS	97	X'00000061'
MQIA_CLWL_Q_RANK	95	X'0000005F'
MQIA_CLWL_Q_PRIORITY	96	X'00000060'
MQIA_CLWL_USEQ	98	X'00000062'
MQIA_CMD_SERVER_AUTO	87	X'00000057'
MQIA_CMD_SERVER_CONTROL	120	X'00000078'
MQIA_CMD_SERVER_CONVERT_MSG	88	X'00000058'
MQIA_CMD_SERVER_DLQ_MSG	89	X'00000059'
MQIA_CODED_CHAR_SET_ID	2	X'00000002'
MQIA_COMM_EVENT	232	X'000000E8'
MQIA_COMMAND_EVENT	99	X'00000063'
MQIA_COMMAND_LEVEL	31	X'0000001F'
MQIA_CONFIGURATION_EVENT	51	X'00000033'
MQIA_CPI_LEVEL	27	X'0000001B'
MQIA_CURRENT_Q_DEPTH	3	X'00000003'
MQIA_DEF_BIND	61	X'0000003D'
MQIA_DEF_CLUSTER_XMIT_Q_TYPE	250	X'000000FA'
MQIA_DEF_INPUT_OPEN_OPTION	4	X'00000004'
MQIA_DEF_PERSISTENCE	5	X'00000005'
MQIA_DEF_PRIORITY	6	X'00000006'
MQIA_DEF_PUT_RESPONSE_TYPE	184	X'000000B8'
MQIA_DEF_READ_AHEAD	188	X'000000BC'
MQIA_DEFINITION_TYPE	7	X'00000007'
MQIA_DISPLAY_TYPE	262	X'00000106'
MQIA_DIST_LISTS	34	X'00000022'
MQIA_DNS_WLM	106	X'0000006A'

MQIA_DURABLE_SUB	175	X'000000AF'
MQIA_EXPIRY_INTERVAL	39	X'00000027'
MQIA_FIRST	1	X'00000001'
MQIA_GROUP_UR	221	X'000000DD'
MQIA_HARDEN_GET_BACKOUT	8	X'00000008'
MQIA_HIGH_Q_DEPTH	36	X'00000024'
MQIA_IGQ_PUT_AUTHORITY	65	X'00000041'
MQIA_INDEX_TYPE	57	X'00000039'
MQIA_INHIBIT_EVENT	48	X'00000030'
MQIA_INHIBIT_GET	9	X'00000009'
MQIA_INHIBIT_PUB	181	X'000000B5'
MQIA_INHIBIT_PUT	10	X'0000000A'
MQIA_INHIBIT_SUB	182	X'000000B6'
MQIA_INTRA_GROUP_queuing	64	X'00000040'
MQIA_IP_ADDRESS_VERSION	93	X'0000005D'
 MQIA_KEY_REUSE_COUNT	267	X'0000010B'
MQIA_LAST	2000	X'000007D0'
 MQIA_LAST_USED	267	X'0000010B'
MQIA_LDAP_AUTHORMD	263	X'00000107'
MQIA_LDAP_NESTGRP	264	X'00000108'
MQIA_LDAP_SECURE_COMM	261	X'00000105'
MQIA_LISTENER_PORT_NUMBER	85	X'00000055'
MQIA_LISTENER_TIMER	107	X'0000006B'
MQIA_LOGGER_EVENT	94	X'0000005E'
MQIA_LU62_CHANNELS	108	X'0000006C'
MQIA_LOCAL_EVENT	49	X'00000031'
MQIA_MSG_MARK_BROWSE_INTERVAL	68	X'00000044'
MQIA_MAX_CHANNELS	109	X'0000006D'
MQIA_MAX_CLIENTS	172	X'000000AC'
MQIA_MAX_GLOBAL_LOCKS	83	X'00000053'
MQIA_MAX_HANDLES	11	X'0000000B'
MQIA_MAX_LOCAL_LOCKS	84	X'00000054'
MQIA_MAX_MSG_LENGTH	13	X'0000000D'
MQIA_MAX_OPEN_Q	80	X'00000050'
MQIA_MAX_PRIORITY	14	X'0000000E'
MQIA_MAX_PROPERTIES_LENGTH	192	X'000000C0'
MQIA_MAX_Q_DEPTH	15	X'0000000F'
MQIA_MAX_Q_TRIGGERS	90	X'0000005A'
MQIA_MAX_RECOVERY_TASKS	171	X'000000AB'
MQIA_MAX_UNCOMMITTED_MSGS	33	X'00000021'
MQIA_MCAST_BRIDGE	233	X'000000E9'

MQIA_MONITOR_INTERVAL	81	X'00000051'
MQIA_MONITORING_AUTO_CLUSSDR	124	X'0000007C'
MQIA_MONITORING_CHANNEL	122	X'0000007A'
MQIA_MONITORING_Q	123	X'0000007B'
MQIA_MSG_DELIVERY_SEQUENCE	16	X'00000010'
MQIA_MSG_DEQ_COUNT	38	X'00000026'
MQIA_MSG_ENQ_COUNT	37	X'00000025'
MQIA_NAME_COUNT	19	X'00000013'
MQIA_NAMELIST_TYPE	72	X'00000048'
MQIA_NPM_CLASS	78	X'0000004E'
MQIA_NPM_DELIVERY	196	X'000000C4'
MQIA_OPEN_INPUT_COUNT	17	X'00000011'
MQIA_OPEN_OUTPUT_COUNT	18	X'00000012'
MQIA_OUTBOUND_PORT_MAX	140	X'0000008C'
MQIA_OUTBOUND_PORT_MIN	110	X'0000006E'
MQIA_PAGESET_ID	62	X'0000003E'
MQIA_PERFORMANCE_EVENT	53	X'00000035'
MQIA_PLATFORM	32	X'00000020'
MQIA_PM_DELIVERY	195	X'000000C3'
MQIA_PROPERTY_CONTROL	190	X'000000BE'
MQIA_PROT_POLICY_CAPABILITY	251	X'000000FB'
MQIA_PROXY_SUB	199	X'000000C7'
MQIA_PUB_COUNT	215	X'000000D7'
MQIA_PUB_SCOPE	219	X'000000DB'
MQIA_PUBSUB_CLUSTER	249	X'000000F9'
MQIA_PUBSUB_MAXMSG_RETRY_COUNT	206	X'000000CE'
MQIA_PUBSUB_MODE	187	X'000000BB'
MQIA_PUBSUB_NP_MSG	203	X'000000CB'
MQIA_PUBSUB_NP_RESP	205	X'000000CD'
MQIA_PUBSUB_SYNC_PT	207	X'000000CF'
MQIA_Q_DEPTH_HIGH_EVENT	43	X'0000002B'
MQIA_Q_DEPTH_HIGH_LIMIT	40	X'00000028'
MQIA_Q_DEPTH_LOW_EVENT	44	X'0000002C'
MQIA_Q_DEPTH_LOW_LIMIT	41	X'00000029'
MQIA_Q_DEPTH_MAX_EVENT	42	X'0000002A'
MQIA_Q_SERVICE_INTERVAL	54	X'00000036'
MQIA_Q_SERVICE_INTERVAL_EVENT	46	X'0000002E'
MQIA_Q_TYPE	20	X'00000014'
MQIA_Q_USERS	82	X'00000052'
MQIA_QMGR_CFCONLOS	245	X'000000F5'
MQIA_QMOPT_CONS_COMMS_MSGS	155	X'0000009B'
MQIA_QMOPT_CONS_CRITICAL_MSGS	154	X'0000009A'

MQIA_QMOPT_CONS_ERROR_MSGS	153	X'00000099'
MQIA_QMOPT_CONS_INFO_MSGS	151	X'00000097'
MQIA_QMOPT_CONS_REORG_MSGS	156	X'0000009C'
MQIA_QMOPT_CONS_SYSTEM_MSGS	157	X'0000009D'
MQIA_QMOPT_CONS_WARNING_MSGS	152	X'00000098'
MQIA_QMOPT_CSMT_ON_ERROR	150	X'00000096'
MQIA_QMOPT_INTERNAL_DUMP	170	X'000000AA'
MQIA_QMOPT_LOG_COMMS_MSGS	162	X'000000A2'
MQIA_QMOPT_LOG_CRITICAL_MSGS	161	X'000000A1'
MQIA_QMOPT_LOG_ERROR_MSGS	160	X'000000A0'
MQIA_QMOPT_LOG_INFO_MSGS	158	X'0000009E'
MQIA_QMOPT_LOG_REORG_MSGS	163	X'000000A3'
MQIA_QMOPT_LOG_SYSTEM_MSGS	164	X'000000A4'
MQIA_QMOPT_LOG_WARNING_MSGS	159	X'0000009F'
MQIA_QMOPT_TRACE_COMMS	166	X'000000A6'
MQIA_QMOPT_TRACE_CONVERSION	168	X'000000A8'
MQIA_QMOPT_TRACE_REORG	167	X'000000A7'
MQIA_QMOPT_TRACE_MQI_CALLS	165	X'000000A5'
MQIA_QMOPT_TRACE_SYSTEM	169	X'000000A9'
MQIA_QSG_DISP	63	X'0000003F'
MQIA_READ_AHEAD	189	X'000000BD'
MQIA_RECEIVE_TIMEOUT	111	X'0000006F'
MQIA_RECEIVE_TIMEOUT_MIN	113	X'00000071'
MQIA_RECEIVE_TIMEOUT_TYPE	112	X'00000070'
MQIA_REMOTE_EVENT	50	X'00000032'
MQIA_RETENTION_INTERVAL	21	X'00000015'
MQIA_REVERSE_DNS_LOOKUP	254	X'000000FE'
MQIA_SCOPE	45	X'0000002D'
MQIA_SECURITY_CASE	141	X'0000008D'
MQIA_SERVICE_CONTROL	139	X'0000008B'
MQIA_SERVICE_TYPE	121	X'00000079'
MQIA_SHAREABILITY	23	X'00000017'
MQIA_SHARED_Q_Q_MGR_NAME	77	X'0000004D'
MQIA_SSL_EVENT	75	X'0000004B'
MQIA_SSL_FIPS_REQUIRED	92	X'0000005C'
MQIA_SSL_RESET_COUNT	76	X'0000004C'
MQIA_SSL_TASKS	69	X'00000045'
MQIA_START_STOP_EVENT	52	X'00000034'
MQIA_STATISTICS_CHANNEL	129	X'00000081'
MQIA_STATISTICS_AUTO_CLUSSDR	130	X'00000082'
MQIA_STATISTICS_INTERVAL	131	X'00000083'
MQIA_STATISTICS_MQI	127	X'0000007F'

MQIA_STATISTICS_Q	128	X'00000080'
MQIA_SUB_COUNT	204	X'000000CC'
MQIA_SUB_SCOPE	218	X'000000DA'
MQIA_SYNCPOINT	30	X'0000001E'
MQIA_TCP_CHANNELS	114	X'00000072'
MQIA_TCP_KEEP_ALIVE	115	X'00000073'
MQIA_TCP_STACK_TYPE	116	X'00000074'
MQIA_TIME_SINCE_RESET	35	X'00000023'
MQIA_TOPIC_DEF_PERSISTENCE	185	X'000000B9'
MQIA_TOPIC_NODE_COUNT	253	X'000000FD'
MQIA_TOPIC_TYPE	208	X'000000D0'
MQIA_TRACE_ROUTE_RECORDING	137	X'00000089'
MQIA_TREE_LIFE_TIME	183	X'000000B7'
MQIA_TRIGGER_CONTROL	24	X'00000018'
MQIA_TRIGGER_DEPTH	29	X'0000001D'
MQIA_TRIGGER_INTERVAL	25	X'00000019'
MQIA_TRIGGER_MSG_PRIORITY	26	X'0000001A'
MQIA_TRIGGER_TYPE	28	X'0000001C'
MQIA_TRIGGER_RESTART	91	X'0000005B'
MQIA_USAGE	12	X'0000000C'
MQIA_USE_DEAD_LETTER_Q	234	X'000000EA'
MQIA_USER_LIST	2000	X'000007D0'
MQIA_WILDCARD_OPERATION	216	X'000000D8'
MQIA_XR_CAPABILITY	243	X'000000F3'

MQIACF_* (Command format Integer Parameter Types):

MQIACF_FIRST	1001	X'000003E9'
MQIACF_Q_MGR_ATTRS	1001	X'000003E9'
MQIACF_Q_ATTRS	1002	X'000003EA'
MQIACF_PROCESS_ATTRS	1003	X'000003EB'
MQIACF_NAMELIST_ATTRS	1004	X'000003EC'
MQIACF_FORCE	1005	X'000003ED'
MQIACF_REPLACE	1006	X'000003EE'
MQIACF_PURGE	1007	X'000003EF'
MQIACF_QUIESCE	1008	X'000003F0'
MQIACF_MODE	1008	X'000003F0'
MQIACF_ALL	1009	X'000003F1'
MQIACF_EVENT_APPL_TYPE	1010	X'000003F2'
MQIACF_EVENT_ORIGIN	1011	X'000003F3'
MQIACF_PARAMETER_ID	1012	X'000003F4'
MQIACF_ERROR_ID	1013	X'000003F5'

MQIACF_ERROR_IDENTIFIER	1013	X'000003F5'
MQIACF_SELECTOR	1014	X'000003F6'
MQIACF_CHANNEL_ATTRS	1015	X'000003F7'
MQIACF_OBJECT_TYPE	1016	X'000003F8'
MQIACF_ESCAPE_TYPE	1017	X'000003F9'
MQIACF_ERROR_OFFSET	1018	X'000003FA'
MQIACF_AUTH_INFO_ATTRS	1019	X'000003FB'
MQIACF_REASON_QUALIFIER	1020	X'000003FC'
MQIACF_COMMAND	1021	X'000003FD'
MQIACF_OPEN_OPTIONS	1022	X'000003FE'
MQIACF_OPEN_TYPE	1023	X'000003FF'
MQIACF_PROCESS_ID	1024	X'00000400'
MQIACF_THREAD_ID	1025	X'00000401'
MQIACF_Q_STATUS_ATTRS	1026	X'00000402'
MQIACF_UNCOMMITTED_MSGS	1027	X'00000403'
MQIACF_HANDLE_STATE	1028	X'00000404'
MQIACF_AUX_ERROR_DATA_INT_1	1070	X'0000042E'
MQIACF_AUX_ERROR_DATA_INT_2	1071	X'0000042F'
MQIACF_CONV_REASON_CODE	1072	X'00000430'
MQIACF_BRIDGE_TYPE	1073	X'00000431'
MQIACF_INQUIRY	1074	X'00000432'
MQIACF_WAIT_INTERVAL	1075	X'00000433'
MQIACF_OPTIONS	1076	X'00000434'
MQIACF_BROKER_OPTIONS	1077	X'00000435'
MQIACF_REFRESH_TYPE	1078	X'00000436'
MQIACF_SEQUENCE_NUMBER	1079	X'00000437'
MQIACF_INTEGER_DATA	1080	X'00000438'
MQIACF_REGISTRATION_OPTIONS	1081	X'00000439'
MQIACF_PUBLICATION_OPTIONS	1082	X'0000043A'
MQIACF_CLUSTER_INFO	1083	X'0000043B'
MQIACF_Q_MGR_DEFINITION_TYPE	1084	X'0000043C'
MQIACF_Q_MGR_TYPE	1085	X'0000043D'
MQIACF_ACTION	1086	X'0000043E'
MQIACF_SUSPEND	1087	X'0000043F'
MQIACF_BROKER_COUNT	1088	X'00000440'
MQIACF_APPL_COUNT	1089	X'00000441'
MQIACF_ANONYMOUS_COUNT	1090	X'00000442'
MQIACF_REG_REG_OPTIONS	1091	X'00000443'
MQIACF_DELETE_OPTIONS	1092	X'00000444'
MQIACF_CLUSTER_Q_MGR_ATTRS	1093	X'00000445'
MQIACF_REFRESH_INTERVAL	1094	X'00000446'
MQIACF_REFRESH_REPOSITORY	1095	X'00000447'

MQIACF_REMOVE_QUEUES	1096	X'00000448'
MQIACF_OPEN_INPUT_TYPE	1098	X'0000044A'
MQIACF_OPEN_OUTPUT	1099	X'0000044B'
MQIACF_OPEN_SET	1100	X'0000044C'
MQIACF_OPEN_INQUIRE	1101	X'0000044D'
MQIACF_OPEN_BROWSE	1102	X'0000044E'
MQIACF_Q_STATUS_TYPE	1103	X'0000044F'
MQIACF_Q_HANDLE	1104	X'00000450'
MQIACF_Q_STATUS	1105	X'00000451'
MQIACF_SECURITY_TYPE	1106	X'00000452'
MQIACF_CONNECTION_ATTRS	1107	X'00000453'
MQIACF_CONNECT_OPTIONS	1108	X'00000454'
MQIACF_CONN_INFO_TYPE	1110	X'00000456'
MQIACF_CONN_INFO_CONN	1111	X'00000457'
MQIACF_CONN_INFO_HANDLE	1112	X'00000458'
MQIACF_CONN_INFO_ALL	1113	X'00000459'
MQIACF_AUTH_PROFILE_ATTRS	1114	X'0000045A'
MQIACF_AUTHORIZATION_LIST	1115	X'0000045B'
MQIACF_AUTH_ADD_AUTHS	1116	X'0000045C'
MQIACF_AUTH_REMOVE_AUTHS	1117	X'0000045D'
MQIACF_ENTITY_TYPE	1118	X'0000045E'
MQIACF_COMMAND_INFO	1120	X'00000460'
MQIACF_CMDSCOPE_Q_MGR_COUNT	1121	X'00000461'
MQIACF_Q_MGR_SYSTEM	1122	X'00000462'
MQIACF_Q_MGR_EVENT	1123	X'00000463'
MQIACF_Q_MGR_DQM	1124	X'00000464'
MQIACF_Q_MGR_CLUSTER	1125	X'00000465'
MQIACF_QSG_DISPS	1126	X'00000466'
MQIACF_UOW_STATE	1128	X'00000468'
MQIACF_SECURITY_ITEM	1129	X'00000469'
MQIACF_CF_STRUC_STATUS	1130	X'0000046A'
MQIACF_UOW_TYPE	1132	X'0000046C'
MQIACF_CF_STRUC_ATTRS	1133	X'0000046D'
MQIACF_EXCLUDE_INTERVAL	1134	X'0000046E'
MQIACF_CF_STATUS_TYPE	1135	X'0000046F'
MQIACF_CF_STATUS_SUMMARY	1136	X'00000470'
MQIACF_CF_STATUS_CONNECT	1137	X'00000471'
MQIACF_CF_STATUS_BACKUP	1138	X'00000472'
MQIACF_CF_STRUC_TYPE	1139	X'00000473'
MQIACF_CF_STRUC_SIZE_MAX	1140	X'00000474'
MQIACF_CF_STRUC_SIZE_USED	1141	X'00000475'
MQIACF_CF_STRUC_ENTRIES_MAX	1142	X'00000476'

MQIACF_CF_STRUC_ENTRIES_USED	1143	X'00000477'
MQIACF_CF_STRUC_BACKUP_SIZE	1144	X'00000478'
MQIACF_MOVE_TYPE	1145	X'00000479'
MQIACF_MOVE_TYPE_MOVE	1146	X'0000047A'
MQIACF_MOVE_TYPE_ADD	1147	X'0000047B'
MQIACF_Q_MGR_NUMBER	1148	X'0000047C'
MQIACF_Q_MGR_STATUS	1149	X'0000047D'
MQIACF_Db2_CONN_STATUS	1150	X'0000047E'
MQIACF_SECURITY_ATTRS	1151	X'0000047F'
MQIACF_SECURITY_TIMEOUT	1152	X'00000480'
MQIACF_SECURITY_INTERVAL	1153	X'00000481'
MQIACF_SECURITY_SWITCH	1154	X'00000482'
MQIACF_SECURITY_SETTING	1155	X'00000483'
MQIACF_STORAGE_CLASS_ATTRS	1156	X'00000484'
MQIACF_USAGE_TYPE	1157	X'00000485'
MQIACF_BUFFER_POOL_ID	1158	X'00000486'
MQIACF_USAGE_TOTAL_PAGES	1159	X'00000487'
MQIACF_USAGE_UNUSED_PAGES	1160	X'00000488'
MQIACF_USAGE_PERSIST_PAGES	1161	X'00000489'
MQIACF_USAGE_NONPERSIST_PAGES	1162	X'0000048A'
MQIACF_USAGE_RESTART_EXTENTS	1163	X'0000048B'
MQIACF_USAGE_EXPAND_COUNT	1164	X'0000048C'
MQIACF_PAGESET_STATUS	1165	X'0000048D'
MQIACF_USAGE_TOTAL_BUFFERS	1166	X'0000048E'
MQIACF_USAGE_DATA_SET_TYPE	1167	X'0000048F'
MQIACF_USAGE_PAGESET	1168	X'00000490'
MQIACF_USAGE_DATA_SET	1169	X'00000491'
MQIACF_USAGE_BUFFER_POOL	1170	X'00000492'
MQIACF_MOVE_COUNT	1171	X'00000493'
MQIACF_EXPIRY_Q_COUNT	1172	X'00000494'
MQIACF_CONFIGURATION_OBJECTS	1173	X'00000495'
MQIACF_CONFIGURATION_EVENTS	1174	X'00000496'
MQIACF_SYSP_TYPE	1175	X'00000497'
MQIACF_SYSP_DEALLOC_INTERVAL	1176	X'00000498'
MQIACF_SYSP_MAX_ARCHIVE	1177	X'00000499'
MQIACF_SYSP_MAX_READ_TAPES	1178	X'0000049A'
MQIACF_SYSP_IN_BUFFER_SIZE	1179	X'0000049B'
MQIACF_SYSP_OUT_BUFFER_SIZE	1180	X'0000049C'
MQIACF_SYSP_OUT_BUFFER_COUNT	1181	X'0000049D'
MQIACF_SYSP_ARCHIVE	1182	X'0000049E'
MQIACF_SYSP_DUAL_ACTIVE	1183	X'0000049F'
MQIACF_SYSP_DUAL_ARCHIVE	1184	X'000004A0'

MQIACF_SYSP_DUAL_BSDS	1185	X'000004A1'
MQIACF_SYSP_MAX_CONNS	1186	X'000004A2'
MQIACF_SYSP_MAX_CONNS_FORE	1187	X'000004A3'
MQIACF_SYSP_MAX_CONNS_BACK	1188	X'000004A4'
MQIACF_SYSP_EXIT_INTERVAL	1189	X'000004A5'
MQIACF_SYSP_EXIT_TASKS	1190	X'000004A6'
MQIACF_SYSP_CHKPOINT_COUNT	1191	X'000004A7'
MQIACF_SYSP_OTMA_INTERVAL	1192	X'000004A8'
MQIACF_SYSP_Q_INDEX_DEFER	1193	X'000004A9'
MQIACF_SYSP_Db2_TASKS	1194	X'000004AA'
MQIACF_SYSP_RESLEVEL_AUDIT	1195	X'000004AB'
MQIACF_SYSP_ROUTING_CODE	1196	X'000004AC'
MQIACF_SYSP_SMF_ACCOUNTING	1197	X'000004AD'
MQIACF_SYSP_SMF_STATS	1198	X'000004AE'
MQIACF_SYSP_SMF_INTERVAL	1199	X'000004AF'
MQIACF_SYSP_TRACE_CLASS	1200	X'000004B0'
MQIACF_SYSP_TRACE_SIZE	1201	X'000004B1'
MQIACF_SYSP_WLM_INTERVAL	1202	X'000004B2'
MQIACF_SYSP_ALLOC_UNIT	1203	X'000004B3'
MQIACF_SYSP_ARCHIVE_RETAIN	1204	X'000004B4'
MQIACF_SYSP_ARCHIVE_WTOR	1205	X'000004B5'
MQIACF_SYSP_BLOCK_SIZE	1206	X'000004B6'
MQIACF_SYSP_CATALOG	1207	X'000004B7'
MQIACF_SYSP_COMPACT	1208	X'000004B8'
MQIACF_SYSP_ALLOC_PRIMARY	1209	X'000004B9'
MQIACF_SYSP_ALLOC_SECONDARY	1210	X'000004BA'
MQIACF_SYSP_PROTECT	1211	X'000004BB'
MQIACF_SYSP_QUIESCE_INTERVAL	1212	X'000004BC'
MQIACF_SYSP_TIMESTAMP	1213	X'000004BD'
MQIACF_SYSP_UNIT_ADDRESS	1214	X'000004BE'
MQIACF_SYSP_UNIT_STATUS	1215	X'000004BF'
MQIACF_SYSP_LOG_COPY	1216	X'000004C0'
MQIACF_SYSP_LOG_USED	1217	X'000004C1'
MQIACF_SYSP_LOG_SUSPEND	1218	X'000004C2'
MQIACF_SYSP_OFFLOAD_STATUS	1219	X'000004C3'
MQIACF_SYSP_TOTAL_LOGS	1220	X'000004C4'
MQIACF_SYSP_FULL_LOGS	1221	X'000004C5'
MQIACF_LISTENER_ATTRS	1222	X'000004C6'
MQIACF_LISTENER_STATUS_ATTRS	1223	X'000004C7'
MQIACF_SERVICE_ATTRS	1224	X'000004C8'
MQIACF_SERVICE_STATUS_ATTRS	1225	X'000004C9'
MQIACF_Q_TIME_INDICATOR	1226	X'000004CA'

MQIACF_OLDEST_MSG_AGE	1227	X'000004CB'
MQIACF_AUTH_OPTIONS	1228	X'000004CC'
MQIACF_Q_MGR_STATUS_ATTRS	1229	X'000004CD'
MQIACF_CONNECTION_COUNT	1230	X'000004CE'
MQIACF_Q_MGR_FACILITY	1231	X'000004CF'
MQIACF_CHINIT_STATUS	1232	X'000004D0'
MQIACF_CMD_SERVER_STATUS	1233	X'000004D1'
MQIACF_ROUTE_DETAIL	1234	X'000004D2'
MQIACF_RECORDED_ACTIVITIES	1235	X'000004D3'
MQIACF_MAX_ACTIVITIES	1236	X'000004D4'
MQIACF_DISCONTINUITY_COUNT	1237	X'000004D5'
MQIACF_ROUTE_ACCUMULATION	1238	X'000004D6'
MQIACF_ROUTE_DELIVERY	1239	X'000004D7'
MQIACF_OPERATION_TYPE	1240	X'000004D8'
MQIACF_BACKOUT_COUNT	1241	X'000004D9'
MQIACF_COMP_CODE	1242	X'000004DA'
MQIACF_ENCODING	1243	X'000004DB'
MQIACF_EXPIRY	1244	X'000004DC'
MQIACF_FEEDBACK	1245	X'000004DD'
MQIACF_MSG_FLAGS	1247	X'000004DF'
MQIACF_MSG_LENGTH	1248	X'000004E0'
MQIACF_MSG_TYPE	1249	X'000004E1'
MQIACF_OFFSET	1250	X'000004E2'
MQIACF_ORIGINAL_LENGTH	1251	X'000004E3'
MQIACF_PERSISTENCE	1252	X'000004E4'
MQIACF_PRIORITY	1253	X'000004E5'
MQIACF_REASON_CODE	1254	X'000004E6'
MQIACF_REPORT	1255	X'000004E7'
MQIACF_VERSION	1256	X'000004E8'
MQIACF_UNRECORDED_ACTIVITIES	1257	X'000004E9'
MQIACF_MONITORING	1258	X'000004EA'
MQIACF_ROUTE_FORWARDING	1259	X'000004EB'
MQIACF_SERVICE_STATUS	1260	X'000004EC'
MQIACF_Q_TYPES	1261	X'000004ED'
MQIACF_USER_ID_SUPPORT	1262	X'000004EE'
MQIACF_INTERFACE_VERSION	1263	X'000004EF'
MQIACF_AUTH_SERVICE_ATTRS	1264	X'000004F0'
MQIACF_USAGE_EXPAND_TYPE	1265	X'000004F1'
MQIACF_SYSP_CLUSTER_CACHE	1266	X'000004F2'
MQIACF_SYSP_Db2_BLOB_TASKS	1267	X'000004F3'
MQIACF_SYSP_WLM_INT_UNITS	1268	X'000004F4'
MQIACF_TOPIC_ATTRS	1269	X'000004F5'

MQIACF_PUBSUB_PROPERTIES	1271	X'000004F7'
MQIACF_DESTINATION_CLASS	1273	X'000004F9'
MQIACF_DURABLE_SUBSCRIPTION	1274	X'000004FA'
MQIACF_SUBSCRIPTION_SCOPE	1275	X'000004FB'
MQIACF_VARIABLE_USER_ID	1277	X'000004FD'
MQIACF_REQUEST_ONLY	1280	X'00000500'
MQIACF_PUB_PRIORITY	1283	X'00000503'
MQIACF_SUB_ATTRS	1287	X'00000507'
MQIACF_WILDCARD_SCHEMA	1288	X'00000508'
MQIACF_SUB_TYPE	1289	X'00000509'
MQIACF_MESSAGE_COUNT	1290	X'0000050A'
MQIACF_Q_MGR_PUBSUB	1291	X'0000050B'
MQIACF_Q_MGR_VERSION	1292	X'0000050C'
MQIACF_SUB_STATUS_ATTRS	1294	X'0000050E'
MQIACF_TOPIC_STATUS	1295	X'0000050F'
MQIACF_TOPIC_SUB	1296	X'00000510'
MQIACF_TOPIC_PUB	1297	X'00000511'
MQIACF_RETAINED_PUBLICATION	1300	X'00000514'
MQIACF_TOPIC_STATUS_ATTRS	1301	X'00000515'
MQIACF_TOPIC_STATUS_TYPE	1302	X'00000516'
MQIACF_SUB_OPTIONS	1303	X'00000517'
MQIACF_PUBLISH_COUNT	1304	X'00000518'
MQIACF_CLEAR_TYPE	1305	X'00000519'
MQIACF_CLEAR_SCOPE	1306	X'0000051A'
MQIACF_SUB_LEVEL	1307	X'0000051B'
MQIACF_ASYNC_STATE	1308	X'0000051C'
MQIACF_SUB_SUMMARY	1309	X'0000051D'
MQIACF_OBSOLETE_MSGS	1310	X'0000051E'
MQIACF_PUBSUB_STATUS	1311	X'0000051F'
MQIACF_PS_STATUS_TYPE	1314	X'00000522'
MQIACF_PUBSUB_STATUS_ATTRS	1318	X'00000526'
MQIACF_SELECTOR_TYPE	1321	X'00000529'
MQIACF_MCAST_REL_INDICATOR	1351	X'00000547'
MQIACF_CHLAUTH_TYPE	1352	X'00000548'
MQXR_DIAGNOSTICS_TYPE	1354	X'0000054A'
MQIACF_CHLAUTH_ATTRS	1355	X'0000054B'
MQIACF_OPERATION_ID	1356	X'0000054C'
MQIACF_API_CALLER_TYPE	1357	X'0000054D'
MQIACF_API_ENVIRONMENT	1358	X'0000054E'
MQIACF_TRACE_DETAIL	1359	X'0000054F'
MQIACF_HOBJ	1360	X'00000550'
MQIACF_CALL_TYPE	1361	X'00000551'

MQIACF_MQCB_OPERATION	1362	X'00000552'
MQIACF_MQCB_TYPE	1363	X'00000553'
MQIACF_MQCB_OPTIONS	1364	X'00000554'
MQIACF_CLOSE_OPTIONS	1365	X'00000555'
MQIACF_CTL_OPERATION	1366	X'00000556'
MQIACF_GET_OPTIONS	1367	X'00000557'
MQIACF_RECS_PRESENT	1368	X'00000558'
MQIACF_KNOWN_DEST_COUNT	1369	X'00000559'
MQIACF_UNKNOWN_DEST_COUNT	1370	X'0000055A'
MQIACF_INVALID_DEST_COUNT	1371	X'0000055B'
MQIACF_RESOLVED_TYPE	1372	X'0000055C'
MQIACF_PUT_OPTIONS	1373	X'0000055D'
MQIACF_BUFFER_LENGTH	1374	X'0000055E'
MQIACF_TRACE_DATA_LENGTH	1375	X'0000055F'
MQIACF_SMDS_EXPANDST	1376	X'00000560'
MQIACF_STRUC_LENGTH	1377	X'00000561'
MQIACF_ITEM_COUNT	1378	X'00000562'
MQIACF_EXPIRY_TIME	1379	X'00000563'
MQIACF_CONNECT_TIME	1380	X'00000564'
MQIACF_DISCONNECT_TIME	1381	X'00000565'
MQIACF_HSUB	1382	X'00000566'
MQIACF_SUBRQ_OPTIONS	1383	X'00000567'
MQIACF_XA_RMID	1384	X'00000568'
MQIACF_XA_FLAGS	1385	X'00000569'
MQIACF_XA_RETCODE	1386	X'0000056A'
MQIACF_XA_HANDLE	1387	X'0000056B'
MQIACF_XA_RETVAL	1388	X'0000056C'
MQIACF_STATUS_TYPE	1389	X'0000056D'
MQIACF_XA_COUNT	1390	X'0000056E'
MQIACF_SELECTOR_COUNT	1391	X'0000056F'
MQIACF_SELECTORS	1392	X'00000570'
MQIACF_INTATTR_COUNT	1393	X'00000571'
MQIACF_INTATTRS	1394	X'00000572'
MQIACF_SUBRQ_ACTION	1395	X'00000573'
MQIACF_NUM_PUBS	1396	X'00000574'
MQIACF_POINTER_SIZE	1397	X'00000575'
MQIACF_REMOVE_AUTHREC	1398	X'00000576'
MQIACF_XR_ATTRS	1399	X'00000577'
MQIACF_APPL_FUNCTION_TYPE	1400	X'00000578'
V 9.0.0 MQIACF_AMQP_ATTRS	1401	X'00000579'
MQIACF_EXPORT_TYPE	1402	X'0000057A'

MQIACF_EXPORT_ATTRS	1403	X'0000057B'
MQIACF_SYSTEM_OBJECTS	1404	X'0000057C'
MQIACF_CONNECTION_SWAP	1405	X'0000057D'
V 9.0.0 MQIACF_AMQP_DIAGNOSTICS_TYPE	1406	X'0000057E'
MQIACF_BUFFER_POOL_LOCATION	1408	X'00000580'
MQIACF_LDAP_CONNECTION_STATUS	1409	X'00000581'
MQIACF_SYSP_MAX_ACE_POOL	1410	X'00000582'
MQIACF_PAGECLAS	1411	X'00000583'
MQIACF_AUTH_REC_TYPE	1412	X'00000584'
MQIACF_LAST_USED	1412	X'00000584'

MQIACH_* (Command format Integer Channel Types):

MQIACH_FIRST		1501	X'000005DD'
MQIACH_XMIT_PROTOCOL_TYPE		1501	X'000005DD'
MQIACH_BATCH_SIZE		1502	X'000005DE'
MQIACH_DISC_INTERVAL		1503	X'000005DF'
MQIACH_SHORT_TIMER		1504	X'000005E0'
MQIACH_SHORT_RETRY		1505	X'000005E1'
MQIACH_LONG_TIMER		1506	X'000005E2'
MQIACH_LONG_RETRY		1507	X'000005E3'
MQIACH_PUT_AUTHORITY		1508	X'000005E4'
MQIACH_SEQUENCE_NUMBER_WRAP		1509	X'000005E5'
MQIACH_MAX_MSG_LENGTH		1510	X'000005E6'
MQIACH_CHANNEL_TYPE		1511	X'000005E7'
MQIACH_DATA_COUNT		1512	X'000005E8'
MQIACH_NAME_COUNT		1513	X'000005E9'
MQIACH_MSG_SEQUENCE_NUMBER		1514	X'000005EA'
MQIACH_DATA_CONVERSION		1515	X'000005EB'
MQIACH_IN_DOUBT		1516	X'000005EC'
MQIACH_MCA_TYPE		1517	X'000005ED'
MQIACH_SESSION_COUNT		1518	X'000005EE'
MQIACH_ADAPTER		1519	X'000005EF'
MQIACH_COMMAND_COUNT		1520	X'000005F0'
MQIACH_SOCKET		1521	X'000005F1'
MQIACH_PORT		1522	X'000005F2'
MQIACH_CHANNEL_INSTANCE_TYPE		1523	X'000005F3'
MQIACH_CHANNEL_INSTANCE_ATTRS		1524	X'000005F4'
MQIACH_CHANNEL_ERROR_DATA		1525	X'000005F5'
MQIACH_CHANNEL_TABLE		1526	X'000005F6'
MQIACH_CHANNEL_STATUS		1527	X'000005F7'
MQIACH_INDOUBT_STATUS		1528	X'000005F8'

MQIACH_LAST_SEQ_NUMBER		1529	X'000005F9'
MQIACH_LAST_SEQUENCE_NUMBER		1529	X'000005F9'
MQIACH_CURRENT_MSGS		1531	X'000005FB'
MQIACH_CURRENT_SEQ_NUMBER		1532	X'000005FC'
MQIACH_CURRENT_SEQUENCE_NUMBER		1532	X'000005FC'
MQIACH_SSL_RETURN_CODE		1533	X'000005FD'
MQIACH_MSGS		1534	X'000005FE'
MQIACH_BYTES_SENT		1535	X'000005FF'
MQIACH_BYTES_RCVD		1536	X'00000600'
MQIACH_BYTES_RECEIVED		1536	X'00000600'
MQIACH_BATCHES		1537	X'00000601'
MQIACH_BUFFERS_SENT		1538	X'00000602'
MQIACH_BUFFERS_RCVD		1539	X'00000603'
MQIACH_BUFFERS_RECEIVED		1539	X'00000603'
MQIACH_LONG_RETRIES_LEFT		1540	X'00000604'
MQIACH_SHORT_RETRIES_LEFT		1541	X'00000605'
MQIACH_MCA_STATUS		1542	X'00000606'
MQIACH_STOP_REQUESTED		1543	X'00000607'
MQIACH_MR_COUNT		1544	X'00000608'
MQIACH_MR_INTERVAL		1545	X'00000609'
These rows intentionally left blank			
MQIACH_NPM_SPEED		1562	X'0000061A'
MQIACH_HB_INTERVAL		1563	X'0000061B'
MQIACH_BATCH_INTERVAL		1564	X'0000061C'
MQIACH_NETWORK_PRIORITY		1565	X'0000061D'
MQIACH_KEEP_ALIVE_INTERVAL		1566	X'0000061E'
MQIACH_BATCH_HB		1567	X'0000061F'
MQIACH_SSL_CLIENT_AUTH		1568	X'00000620'
This row intentionally left blank			
MQIACH_ALLOC_RETRY		1570	X'00000622'
MQIACH_ALLOC_FAST_TIMER		1571	X'00000623'
MQIACH_ALLOC_SLOW_TIMER		1572	X'00000624'
MQIACH_DISC_RETRY		1573	X'00000625'
MQIACH_PORT_NUMBER		1574	X'00000626'
MQIACH_HDR_COMPRESSION		1575	X'00000627'
MQIACH_MSG_COMPRESSION		1576	X'00000628'
MQIACH_CLWL_CHANNEL_RANK		1577	X'00000629'
MQIACH_CLWL_CHANNEL_PRIORITY		1578	X'0000062A'
MQIACH_CLWL_CHANNEL_WEIGHT		1579	X'0000062B'
MQIACH_CHANNEL_DISP		1580	X'0000062C'
MQIACH_INBOUND_DISP		1581	X'0000062D'
MQIACH_CHANNEL_TYPES		1582	X'0000062E'

MQIACH_ADAPS_STARTED		1583	X'0000062F'
MQIACH_ADAPS_MAX		1584	X'00000630'
MQIACH_DISPS_STARTED		1585	X'00000631'
MQIACH_DISPS_MAX		1586	X'00000632'
MQIACH_SSLTASKS_STARTED		1587	X'00000633'
MQIACH_SSLTASKS_MAX		1588	X'00000634'
MQIACH_CURRENT_CHL		1589	X'00000635'
MQIACH_CURRENT_CHL_MAX		1590	X'00000636'
MQIACH_CURRENT_CHL_TCP		1591	X'00000637'
MQIACH_CURRENT_CHL_LU62		1592	X'00000638'
MQIACH_ACTIVE_CHL		1593	X'00000639'
MQIACH_ACTIVE_CHL_MAX		1594	X'0000063A'
MQIACH_ACTIVE_CHL_PAUSED		1595	X'0000063B'
MQIACH_ACTIVE_CHL_STARTED		1596	X'0000063C'
MQIACH_ACTIVE_CHL_STOPPED		1597	X'0000063D'
MQIACH_ACTIVE_CHL_RETRY		1598	X'0000063E'
MQIACH_LISTENER_STATUS		1599	X'0000063F'
MQIACH_SHARED_CHL_RESTART		1600	X'00000640'
MQIACH_LISTENER_CONTROL		1601	X'00000641'
MQIACH_BACKLOG		1602	X'00000642'
MQIACH_XMITQ_TIME_INDICATOR		1604	X'00000644'
MQIACH_NETWORK_TIME_INDICATOR		1605	X'00000645'
MQIACH_EXIT_TIME_INDICATOR		1606	X'00000646'
MQIACH_BATCH_SIZE_INDICATOR		1607	X'00000647'
MQIACH_XMITQ_MSGS_AVAILABLE		1608	X'00000648'
MQIACH_CHANNEL_SUBSTATE		1609	X'00000649'
MQIACH_SSL_KEY_RESETS		1610	X'0000064A'
MQIACH_COMPRESSION_RATE		1611	X'0000064B'
MQIACH_COMPRESSION_TIME		1612	X'0000064C'
MQIACH_MAX_XMIT_SIZE		1613	X'0000064D'
MQIACH_DEF_CHANNEL_DISP		1614	X'0000064E'
MQIACH_SHARING_CONVERSATIONS		1615	X'0000064F'
MQIACH_MAX_SHARING_CONVS		1616	X'00000650'
MQIACH_CURRENT_SHARING_CONVS		1617	X'00000651'
MQIACH_MAX_INSTANCES		1618	X'00000652'
MQIACH_MAX_INSTS_PER_CLIENT		1619	X'00000653'
MQIACH_CLIENT_CHANNEL_WEIGHT		1620	X'00000654'
MQIACH_CONNECTION_AFFINITY		1621	X'00000655'
This row intentionally left blank			
MQIACH_RESET_REQUESTED		1623	X'00000657'
MQIACH_BATCH_DATA_LIMIT		1624	X'00000658'
MQIACH_MSG_HISTORY		1625	X'00000659'

MQIACH_MULTICAST_PROPERTIES		1626	X'0000065A'
MQIACH_NEW_SUBSCRIBER_HISTORY		1627	X'0000065B'
MQIACH_MC_HB_INTERVAL		1628	X'0000065C'
MQIACH_USE_CLIENT_ID		1629	X'0000065D'
MQIACH_MQTT_KEEP_ALIVE		1630	X'0000065E'
MQIACH_IN_DOUBT_IN		1631	X'0000065F'
MQIACH_IN_DOUBT_OUT		1632	X'00000660'
MQIACH_MSGS_SENT<		1633	X'00000661'
MQIACH_MSGS_RECEIVED		1634	X'00000662'
MQIACH_MSGS_RCVD		1634	X'00000662'
MQIACH_PENDING_OUT		1635	X'00000663'
MQIACH_AVAILABLE_CIPHERSPECS		1636	X'00000664'
MQIACH_MATCH		1637	X'00000665'
MQIACH_USER_SOURCE		1638	X'00000666'
MQIACH_WARNING		1639	X'00000667'
MQIACH_DEF_RECONNECT		1640	X'00000668'
This row intentionally left blank			
MQIACH_CHANNEL_SUMMARY_ATTRS		1642	X'0000066A'
MQIACH_PROTOCOL		1643	X'0000066B'
 MQIACH_AMQPKEEPALIVE		1644	X'0000066C'
MQIACH_SECURITY_PROTOCOL		1645	X'0000066D'
MQIACH_LAST_USED		1645	X'0000066D'

MQIAMO_* (Command format Integer Monitoring Parameter Types):

MQIAMO_FIRST	701	X'000002BD'
MQIAMO_AVG_BATCH_SIZE	702	X'000002BE'
MQIAMO_AVG_Q_TIME	703	X'000002BF'
MQIAMO_BACKOUTS	704	X'000002C0'
MQIAMO_BROWSES	705	X'000002C1'
MQIAMO_BROWSE_MAX_BYTES	706	X'000002C2'
MQIAMO_BROWSE_MIN_BYTES	707	X'000002C3'
MQIAMO_BROWSES_FAILED	708	X'000002C4'
MQIAMO_CLOSES	709	X'000002C5'
MQIAMO_COMMITS	710	X'000002C6'
MQIAMO_COMMITS_FAILED	711	X'000002C7'
MQIAMO_CONNS	712	X'000002C8'
MQIAMO_CONNS_MAX	713	X'000002C9'
MQIAMO_DISCS	714	X'000002CA'
MQIAMO_DISCS_IMPLICIT	715	X'000002CB'
MQIAMO_DISC_TYPE	716	X'000002CC'
MQIAMO_EXIT_TIME_AVG	717	X'000002CD'

MQIAMO_EXIT_TIME_MAX	718	X'000002CE'
MQIAMO_EXIT_TIME_MIN	719	X'000002CF'
MQIAMO_FULL_BATCHES	720	X'000002D0'
MQIAMO_GENERATED_MSGS	721	X'000002D1'
MQIAMO_GETS	722	X'000002D2'
MQIAMO_GET_MAX_BYTES	723	X'000002D3'
MQIAMO_GET_MIN_BYTES	724	X'000002D4'
MQIAMO_GETS_FAILED	725	X'000002D5'
MQIAMO_INCOMPLETE_BATCHES	726	X'000002D6'
MQIAMO_INQS	727	X'000002D7'
MQIAMO_MSGS	728	X'000002D8'
MQIAMO_NET_TIME_AVG	729	X'000002D9'
MQIAMO_NET_TIME_MAX	730	X'000002DA'
MQIAMO_NET_TIME_MIN	731	X'000002DB'
MQIAMO_OBJECT_COUNT	732	X'000002DC'
MQIAMO_OPENS	733	X'000002DD'
MQIAMO_PUT1S	734	X'000002DE'
MQIAMO_PUTS	735	X'000002DF'
MQIAMO_PUT_MAX_BYTES	736	X'000002E0'
MQIAMO_PUT_MIN_BYTES	737	X'000002E1'
MQIAMO_PUT_RETRIES	738	X'000002E2'
MQIAMO_Q_MAX_DEPTH	739	X'000002E3'
MQIAMO_Q_MIN_DEPTH	740	X'000002E4'
MQIAMO_Q_TIME_AVG	741	X'000002E5'
MQIAMO_Q_TIME_MAX	742	X'000002E6'
MQIAMO_Q_TIME_MIN	743	X'000002E7'
MQIAMO_SETS	744	X'000002E8'
MQIAMO_CONNS_FAILED	749	X'000002ED'
MQIAMO_OPENS_FAILED	751	X'000002EF'
MQIAMO_INQS_FAILED	752	X'000002F0'
MQIAMO_SETS_FAILED	753	X'000002F1'
MQIAMO_PUTS_FAILED	754	X'000002F2'
MQIAMO_PUT1S_FAILED	755	X'000002F3'
MQIAMO_CLOSES_FAILED	757	X'000002F5'
MQIAMO_MSGS_EXPIRED	758	X'000002F6'
MQIAMO_MSGS_NOT_QUEUED	759	X'000002F7'
MQIAMO_MSGS_PURGED	760	X'000002F8'
MQIAMO_SUBS_DUR	764	X'000002FC'
MQIAMO_SUBS_NDUR	765	X'000002FD'
MQIAMO_SUBS_FAILED	766	X'000002FE'
MQIAMO_SUBRQS	767	X'000002FF'
MQIAMO_SUBRQS_FAILED	768	X'00000300'

MQIAMO_CBS	769	X'00000301'
MQIAMO_CBS_FAILED	770	X'00000302'
MQIAMO_CTLS	771	X'00000303'
MQIAMO_CTLS_FAILED	772	X'00000304'
MQIAMO_STATS	773	X'00000305'
MQIAMO_STATS_FAILED	774	X'00000306'
MQIAMO_SUB_DUR_HIGHWATER	775	X'00000307'
MQIAMO_SUB_DUR_LOWWATER	776	X'00000308'
MQIAMO_SUB_NDUR_HIGHWATER	777	X'00000309'
MQIAMO_SUB_NDUR_LOWWATER	778	X'0000030A'
MQIAMO_TOPIC_PUTS	779	X'0000030B'
MQIAMO_TOPIC_PUTS_FAILED	780	X'0000030C'
MQIAMO_TOPIC_PUT1S	781	X'0000030D'
MQIAMO_TOPIC_PUT1S_FAILED	782	X'0000030E'
MQIAMO_PUBLISH_MSG_COUNT	784	X'00000310'
MQIAMO_UNSUBS_DUR	786	X'00000312'
MQIAMO_UNSUBS_NDUR	787	X'00000313'
MQIAMO_UNSUBS_FAILED	788	X'00000314'
MQIAMO_INTERVAL	789	X'00000315'
MQIAMO_MSGS_SENT	790	X'00000316'
MQIAMO_BYTES_SENT	791	X'00000317'
MQIAMO_REPAIR_BYTES	792	X'00000318'
MQIAMO_FEEDBACK_MODE	793	X'00000319'
MQIAMO_RELIABILITY_TYPE	794	X'0000031A'
MQIAMO_LATE_JOIN_MARK	795	X'0000031B'
MQIAMO_NACKS_RCVD	796	X'0000031C'
MQIAMO_REPAIR_PKTS	797	X'0000031D'
MQIAMO_HISTORY_PKTS	798	X'0000031E'
MQIAMO_PENDING_PKTS	799	X'0000031F'
MQIAMO_PKT_RATE	800	X'00000320'
MQIAMO_MCAST_XMIT_RATE	801	X'00000321'
MQIAMO_MCAST_BATCH_TIME	802	X'00000322'
MQIAMO_MCAST_HEARTBEAT	803	X'00000323'
MQIAMO_DEST_DATA_PORT	804	X'00000324'
MQIAMO_DEST_REPAIR_PORT	805	X'00000325'
MQIAMO_ACKS_RCVD	806	X'00000326'
MQIAMO_ACTIVE_ACKERS	807	X'00000327'
MQIAMO_PKTS_SENT	808	X'00000328'
MQIAMO_TOTAL_REPAIR_PKTS	809	X'00000329'
MQIAMO_TOTAL_PKTS_SENT	810	X'0000032A'
MQIAMO_TOTAL_MSGS_SENT	811	X'0000032B'
MQIAMO_TOTAL_BYTES_SENT	812	X'0000032C'

MQIAMO_NUM_STREAMS	813	X'0000032D'
MQIAMO_ACK_FEEDBACK	814	X'0000032E'
MQIAMO_NACK_FEEDBACK	815	X'0000032F'
MQIAMO_PKTS_LOST	816	X'00000330'
MQIAMO_MSGS_RCVD	817	X'00000331'
MQIAMO_MSG_BYTES_RCVD	818	X'00000332'
MQIAMO_MSGS_DELIVERED	819	X'00000333'
MQIAMO_PKTS_PROCESSED	820	X'00000334'
MQIAMO_PKTS_DLVD	821	X'00000335'
MQIAMO_PKTS_DROPPED	822	X'00000336'
MQIAMO_PKTS_DUPLICATED	823	X'00000337'
MQIAMO_NACKS_CREATED	824	X'00000338'
MQIAMO_NACK_PKTS_SENT	825	X'00000339'
MQIAMO_REPAIR_PKTS_RQSTD	826	X'0000033A'
MQIAMO_REPAIR_PKTS_RCVD	827	X'0000033B'
MQIAMO_PKTS_REPAIRED	828	X'0000033C'
MQIAMO_TOTAL_MSGS_RCVD	829	X'0000033D'
MQIAMO_TOTAL_MSGS_BYTES_RCVD	830	X'0000033E'
MQIAMO_TOTAL_REPAIR_PKTS_RCVD	831	X'0000033F'
MQIAMO_TOTAL_REPAIR_PKTS_RQSTD	832	X'00000340'
MQIAMO_TOTAL_MSGS_PROCESSED	833	X'00000341'
MQIAMO_TOTAL_MSGS_SELECTED	834	X'00000342'
MQIAMO_TOTAL_MSGS_EXPIRED	835	X'00000343'
MQIAMO_TOTAL_MSGS_DELIVERED	836	X'00000344'
MQIAMO_TOTAL_MSGS_RETURNED	837	X'00000345'
MQIAMO_LAST_USED	837	X'00000345'

MQIAMO64_* (Command format 64-bit Integer Monitoring Parameter Types):

MQIAMO64_AVG_Q_TIME	703	X'000002BF'
MQIAMO64_Q_TIME_AVG	741	X'000002E5'
MQIAMO64_Q_TIME_MAX	742	X'000002E6'
MQIAMO64_Q_TIME_MIN	743	X'000002E7'
MQIAMO64_BROWSE_BYTES	745	X'000002E9'
MQIAMO64_BYTES	746	X'000002EA'
MQIAMO64_GET_BYTES	747	X'000002EB'
MQIAMO64_PUT_BYTES	748	X'000002EC'
MQIAMO64_TOPIC_PUT_BYTES	783	X'0000030F'
MQIAMO64_PUBLISH_MSG_BYTES	785	X'00000311'

MQIASY_* (Integer System Selectors):

MQIASY_FIRST	-1	X'FFFFFFFF'
MQIASY_CODED_CHAR_SET_ID	-1	X'FFFFFFFF'
MQIASY_TYPE	-2	X'FFFFFFFE'
MQIASY_COMMAND	-3	X'FFFFFFFD'
MQIASY_MSG_SEQ_NUMBER	-4	X'FFFFFFFC'
MQIASY_CONTROL	-5	X'FFFFFFFB'
MQIASY_COMP_CODE	-6	X'FFFFFFFA'
MQIASY_REASON	-7	X'FFFFFFF9'
MQIASY_BAG_OPTIONS	-8	X'FFFFFFF8'
MQIASY_VERSION	-9	X'FFFFFFF7'
MQIASY_LAST_USED	-9	X'FFFFFFF7'
MQIASY_LAST	-2000	X'FFFFF830'

MQIAUT_* (IMS information header Authenticator):

MQIAUT_NONE	"bbbbbbbb"
MQIAUT_NONE_ARRAY	'b','b','b','b','b','b','b','b'

MQIAV_* (Integer Attribute Values):

MQIAV_NOT_APPLICABLE	-1	X'FFFFFFFF'
MQIAV_UNDEFINED	-2	X'FFFFFFFE'

MQICM_* (IMS information header Commit Modes):

MQICM_COMMIT_THEN_SEND	'0'	
MQICM_SEND_THEN_COMMIT	'1'	

MQIDO_* (Command format Indoubt Options):

MQIDO_COMMIT	1	X'00000001'
MQIDO_BACKOUT	2	X'00000002'

MQIEP_* (Interface entry points):

MQIEP_STRUC_ID	"IEP "	
MQIEP_STRUC_ID_ARRAY	'I','E','P',' '	
MQIEP_VERSION_1	1	X'00000001'
MQDXP_CURRENT_VERSION	1	X'00000001'

MQIGQ_* (Intra-Group queuing):

MQIGQ_DISABLED	0	X'00000000'
MQIGQ_ENABLED	1	X'00000001'

MQIGQPA_* (Intra-Group queuing Put Authority):

MQIGQPA_DEFAULT	1	X'00000001'
MQIGQPA_CONTEXT	2	X'00000002'
MQIGQPA_ONLY_IGQ	3	X'00000003'
MQIGQPA_ALTERNATE_OR_IGQ	4	X'00000004'

MQIIH_* (IMS information header structure and Flags):
IMS information header structure

MQIIH_STRUC_ID	"IIHb"	
MQIIH_STRUC_ID_ARRAY	'I','I','H','b'	
MQIIH_VERSION_1	1	X'00000001'
MQIIH_CURRENT_VERSION	1	X'00000001'
MQIIH_LENGTH_1	84	X'00000054'

IMS information header Flags

MQIIH_NONE	0	X'00000000'
MQIIH_PASS_EXPIRATION	1	X'00000001'
MQIIH_UNLIMITED_EXPIRATION	0	X'00000000'
MQIIH_REPLY_FORMAT_NONE	8	X'00000008'
MQIIH_IGNORE_PURG	16	X'00000010'
MQIIH_CM0_REQUEST_RESPONSE	32	X'00000020'

MQIMPO_* (Inquire message property options and structure):
Inquire message property options structure

MQIMPO_STRUC_ID	"IMPO"	
MQIMPO_STRUC_ID_ARRAY	'I','M','P','O'	
MQIMPO_VERSION_1	1	X'00000001'
MQIMPO_CURRENT_VERSION	1	X'00000001'

Inquire Message Property Options

MQIMPO_CONVERT_TYPE	2	X'00000002'
MQIMPO_QUERY_LENGTH	4	X'00000004'
MQIMPO_INQ_FIRST	0	X'00000000'
MQIMPO_INQ_NEXT	8	X'00000008'
MQIMPO_INQ_PROP_UNDER_CURSOR	16	X'00000010'
MQIMPO_CONVERT_VALUE	32	X'00000020'
MQIMPO_NONE	0	X'00000000'

MQINBD_* (Command format Inbound Dispositions):

MQINBD_Q_MGR	0	X'00000000'
MQINBD_GROUP	3	X'00000003'

MQIND_* (Special Index Values):

MQIND_NONE	-1	X'FFFFFFFF'
MQIND_ALL	-2	X'FFFFFFFE'

MQIPADDR_* (IP Address Versions):

MQIPADDR_IPv4	0	X'00000000'
MQIPADDR_IPv6	1	X'00000001'

MQISS_* (IMS information header Security Scopes):

MQISS_CHECK	'C'	
MQISS_FULL	'F'	

MQIT_* (Index Types):

MQIT_NONE	0	X'00000000'
MQIT_MSG_ID	1	X'00000001'
MQIT_CORREL_ID	2	X'00000002'
MQIT_MSG_TOKEN	4	X'00000004'
MQIT_GROUP_ID	5	X'00000005'

MQITEM_* (Item Type for mqInquireItemInfo):

MQITEM_INTEGER	1	X'00000001'
MQITEM_STRING	2	X'00000002'
MQITEM_BAG	3	X'00000003'
MQITEM_BYTE_STRING	4	X'00000004'
MQITEM_INTEGER_FILTER	5	X'00000005'
MQITEM_STRING_FILTER	6	X'00000006'
MQITEM_INTEGER64	7	X'00000007'
MQITEM_BYTE_STRING_FILTER	8	X'00000008'

MQITII_* (IMS information header Transaction Instance Identifier):

MQITII_NONE	X'00...00'	(16 nulls)
MQITII_NONE_ARRAY	'\0','\0',...	(16 nulls)

MQITS_* (IMS information header Transaction States):

MQITS_IN_CONVERSATION	'C'	
MQITS_NOT_IN_CONVERSATION	'b'	
MQITS_ARCHITECTED	'A'	

MQKAI_* (KeepAlive Interval):

MQKAI_AUTO	-1	X'FFFFFFFF'

MQMASTER_* (Master administration):

MQMASTER_NO	0	X'00000000'
MQMASTER_YES	1	X'00000001'

MQMCAS_* (Command format Message Channel Agent Status):

MQMCAS_STOPPED	0	X'00000000'
MQMCAS_RUNNING	3	X'00000003'

MQMCAT_* (MCA Types):

MQMCAT_PROCESS	1	X'00000001'
MQMCAT_THREAD	2	X'00000002'

MQMCD_* (Publish/Subscribe Options Tag Information):
Publish/Subscribe Options Tag Message Content Descriptor (mcd) Tags

MQMCD_FOLDER_VERSION	1	X'00000001'

Publish/Subscribe Options Tag Tag names

MQMCD_MSG_DOMAIN	"Msd"	
MQMCD_MSG_SET	"Set"	
MQMCD_MSG_TYPE	"Type"	
MQMCD_MSG_FORMAT	"Fmt"	

Publish/Subscribe Options Tag XML tag names

MQMCD_MSG_DOMAIN_B	"<Msd>"	
MQMCD_MSG_DOMAIN_E	"</Msd>"	
MQMCD_MSG_SET_B	"<Set>"	
MQMCD_MSG_SET_E	"</Set>"	
MQMCD_MSG_TYPE_B	"<Type>"	
MQMCD_MSG_TYPE_E	"</Type>"	
MQMCD_MSG_FORMAT_B	"<Fmt>"	
MQMCD_MSG_FORMAT_E	"</Fmt>"	

Publish/Subscribe Options Tag Tag values

MQMCD_DOMAIN_NONE	"none"	
MQMCD_DOMAIN_NEON	"neon"	
MQMCD_DOMAIN_MRM	"mrm"	
MQMCD_DOMAIN_JMS_NONE	"jms_none"	
MQMCD_DOMAIN_JMS_TEXT	"jms_text"	
MQMCD_DOMAIN_JMS_OBJECT	"jms_object"	
MQMCD_DOMAIN_JMS_MAP	"jms_map"	
MQMCD_DOMAIN_JMS_STREAM	"jms_stream"	
MQMCD_DOMAIN_JMS_BYTES	"jms_bytes"	

MQMD_* (Message descriptor structure):

MQMD_STRUC_ID	"MDbb"	
MQMD_STRUC_ID_ARRAY	'M','D','b','b'	
MQMD_VERSION_1	1	X'00000001'
MQMD_VERSION_2	2	X'00000002'
MQMD_CURRENT_VERSION	2	X'00000002'

MQMDE_* (Message descriptor extension structure):

MQMDE_STRUC_ID	"MDEb"	
MQMDE_STRUC_ID_ARRAY	'M','D','E','b'	
MQMDE_VERSION_2	2	X'00000002'
MQMDE_CURRENT_VERSION	2	X'00000002'
MQMDE_LENGTH_2	72	X'00000048'

MQMDEF_* (Message descriptor extension Flags):

MQMDEF_NONE	0	X'00000000'

MQMDS_* (Message Delivery Sequence):

MQMDS_PRIORITY	0	X'00000000'
MQMDS_FIFO	1	X'00000001'

MQMF_* (Message Flags):

MQMF_SEGMENTATION_INHIBITED	0	X'00000000'
MQMF_SEGMENTATION_ALLOWED	1	X'00000001'
MQMF_MSG_IN_GROUP	8	X'00000008'
MQMF_LAST_MSG_IN_GROUP	16	X'00000010'
MQMF_SEGMENT	2	X'00000002'
MQMF_LAST_SEGMENT	4	X'00000004'
MQMF_NONE	0	X'00000000'

MQMHBO_* (Message handle to buffer options and structure):
Message handle to buffer options structure

MQMHBO_STRUC_ID	"MHBO"	
MQMHBO_STRUC_ID_ARRAY	'M','H','B','O'	
MQMHBO_VERSION_1	1	X'00000001'
MQMHBO_CURRENT_VERSION	1	X'00000001'

Message Handle To Buffer Options

MQMHBO_PROPERTIES_IN_MQRFH2	1	X'00000001'
MQMHBO_DELETE_PROPERTIES	2	X'00000002'
MQMHBO_NONE	0	X'00000000'

MQMI_* (Message Identifier):

MQMI_NONE	X'00...00'	(24 nulls)
MQMI_NONE_ARRAY	'\0','\0',...	(24 nulls)

MQMMBI_* (Message Mark-Browse Interval):

MQMMBI_UNLIMITED	-1	X'FFFFFFFF'

MQMO_* (Match Options):

MQMO_MATCH_MSG_ID	1	X'00000001'
MQMO_MATCH_CORREL_ID	2	X'00000002'
MQMO_MATCH_GROUP_ID	4	X'00000004'
MQMO_MATCH_MSG_SEQ_NUMBER	8	X'00000008'
MQMO_MATCH_OFFSET	16	X'00000010'
MQMO_MATCH_MSG_TOKEN	32	X'00000020'
MQMO_NONE	0	X'00000000'

MQMODE_* (Command format Mode Options):

MQMODE_FORCE	0	X'00000000'
MQMODE_QUIESCE	1	X'00000001'
MQMODE_TERMINATE	2	X'00000002'

MQMON_* (Monitoring Values):

MQMON_NOT_AVAILABLE	-1	X'FFFFFFFF'
MQMON_NONE	-1	X'FFFFFFFF'
MQMON_Q_MGR	-3	X'FFFFFFFD'
MQMON_OFF	0	X'00000000'
MQMON_ON	1	X'00000001'
MQMON_DISABLED	0	X'00000000'
MQMON_ENABLED	1	X'00000001'
MQMON_LOW	17	X'00000011'
MQMON_MEDIUM	33	X'00000021'
MQMON_HIGH	65	X'00000041'

MQMT_* (Message Types):

MQMT_SYSTEM_FIRST	1	X'00000001'
MQMT_REQUEST	1	X'00000001'
MQMT_REPLY	2	X'00000002'
MQMT_DATAGRAM	8	X'00000008'
MQMT_REPORT	4	X'00000004'
MQMT_MQE_FIELDS_FROM_MQE	112	X'00000070'
MQMT_MQE_FIELDS	113	X'00000071'
MQMT_SYSTEM_LAST	65535	X'0000FFFF'
MQMT_APPL_FIRST	65536	X'00010000'
MQMT_APPL_LAST	999999999	X'3B9AC9FF'

MQMTOK_* (Message Token):

MQMTOK_NONE	X'00...00'	(16 nulls)
MQMTOK_NONE_ARRAY	'\0','\0',...	(16 nulls)

MQNC_* (Name Count):

MQNC_MAX_NAMELIST_NAME_COUNT	256	X'00000100'

MQNPM_* (Nonpersistent Message Class):

MQNPM_CLASS_NORMAL	0	X'00000000'
MQNPM_CLASS_HIGH	10	X'0000000A'

MQNPMS_* (NonPersistent-Message Speeds):

MQNPMS_NORMAL	1	X'00000001'
MQNPMS_FAST	2	X'00000002'

MQNT_* (Namelist Types):

MQNT_NONE	0	X'00000000'
MQNT_Q	1	X'00000001'
MQNT_CLUSTER	2	X'00000002'
MQNT_AUTH_INFO	4	X'00000004'
MQNT_ALL	1001	X'000003E9'

MQNVS_* (Names for Name/Value String):

MQNVS_APPL_TYPE	"OPT_APP_GRPb"	
MQNVS_MSG_TYPE	"OPT_MSG_TYPEb"	

MQOA_* (Limits for Selectors for Object Attributes):

MQOA_FIRST	1	X'00000001'
MQOA_LAST	9000	X'00002328'

MQOD_* (Object descriptor structure):

MQOD_STRUC_ID	"ODbb"	
MQOD_STRUC_ID_ARRAY	'O','D','b','b'	
MQOD_VERSION_1	1	X'00000001'
MQOD_VERSION_2	2	X'00000002'
MQOD_VERSION_3	3	X'00000003'
MQOD_VERSION_4	4	X'00000004'
MQOD_CURRENT_VERSION	4	X'00000004'
MQOD_CURRENT_LENGTH	(value differs by platform or version)	

MQOII_* (Object Instance Identifier):

MQOII_NONE	X'00...00'	(24 nulls)
MQOII_NONE_ARRAY	'\0','\0',...	(24 nulls)

MQOL_* (Original Length):

MQOL_UNDEFINED	-1	X'FFFFFFFF'

MQOM_* (Obsolete Db2 Messages options on Inquire Group):

MQOM_NO	0	X'00000000'
MQOM_YES	1	X'00000001'

MQOO_* (Open Options):

MQOO_BIND_AS_Q_DEF	0	X'00000000'
MQOO_READ_AHEAD_AS_Q_DEF	0	X'00000000'
MQOO_INPUT_AS_Q_DEF	1	X'00000001'
MQOO_INPUT_SHARED	2	X'00000002'
MQOO_INPUT_EXCLUSIVE	4	X'00000004'
MQOO_BROWSE	8	X'00000008'
MQOO_OUTPUT	16	X'00000010'
MQOO_INQUIRE	32	X'00000020'
MQOO_SET	64	X'00000040'

MQOO_SAVE_ALL_CONTEXT	128	X'00000080'
MQOO_PASS_IDENTITY_CONTEXT	256	X'00000100'
MQOO_PASS_ALL_CONTEXT	512	X'00000200'
MQOO_SET_IDENTITY_CONTEXT	1024	X'00000400'
MQOO_SET_ALL_CONTEXT	2048	X'00000800'
MQOO_ALTERNATE_USER_AUTHORITY	4096	X'00001000'
MQOO_FAIL_IF_QUIESCING	8192	X'00002000'
MQOO_BIND_ON_OPEN	16384	X'00004000'
MQOO_BIND_NOT_FIXED	32768	X'00008000'
MQOO_CO_OP	131072	X'00020000'
MQOO_RESOLVE_LOCAL_TOPIC	262144	X'00040000'
MQOO_NO_READ_AHEAD	524288	X'00080000'
MQOO_READ_AHEAD	1048576	X'00100000'
MQOO_BIND_ON_GROUP	4194304	X'00400000'

MQOO_* (Following used in C++ only):

MQOO_RESOLVE_NAMES	65536	X'00010000'
MQOO_RESOLVE_LOCAL_Q	262144	X'00040000'

MQOP_* (Operation codes for MQCTL and MQCB):
Operation codes for MQCTL

MQOP_START	1	X'00000001'
MQOP_START_WAIT	2	X'00000002'
MQOP_STOP	4	X'00000004'

Operation codes for MQCB

MQOP_REGISTER	256	X'00000100'
MQOP_DEREGISTER	512	X'00000200'

Operation codes for MQCTL and MQCB

MQOP_SUSPEND	65536	X'00010000'
MQOP_RESUME	131072	X'00020000'

MQOPEN_* (Values related to MQOPEN_PRIV structure):

MQOPEN_PRIV_VERSION_1	1	X'00000001'
MQOPEN_PRIV_CURRENT_VERSION	1	X'00000001'

MQOPER_* (Activity Operations):

MQOPER_SYSTEM_FIRST	0	X'00000000'
MQOPER_UNKNOWN	0	X'00000000'
MQOPER_BROWSE	1	X'00000001'
MQOPER_DISCARD	2	X'00000002'
MQOPER_GET	3	X'00000003'
MQOPER_PUT	4	X'00000004'
MQOPER_PUT_REPLY	5	X'00000005'
MQOPER_PUT_REPORT	6	X'00000006'
MQOPER_RECEIVE	7	X'00000007'
MQOPER_SEND	8	X'00000008'
MQOPER_TRANSFORM	9	X'00000009'
MQOPER_PUBLISH	10	X'0000000A'
MQOPER_EXCLUDED_PUBLISH	11	X'0000000B'
MQOPER_DISCARDED_PUBLISH	12	X'0000000C'
MQOPER_SYSTEM_LAST	65535	X'0000FFFF'
MQOPER_APPL_FIRST	65536	X'00010000'
MQOPER_APPL_LAST	999999999	X'3B9AC9FF'

MQOT_* (Object Types and Extended Object Types):
Object Types

MQOT_NONE	0	X'00000000'
MQOT_Q	1	X'00000001'
MQOT_NAMELIST	2	X'00000002'
MQOT_PROCESS	3	X'00000003'
MQOT_STORAGE_CLASS	4	X'00000004'
MQOT_Q_MGR	5	X'00000005'
MQOT_CHANNEL	6	X'00000006'
MQOT_AUTH_INFO	7	X'00000007'
MQOT_TOPIC	8	X'00000008'
MQOT_CF_STRUC	10	X'0000000A'
MQOT_LISTENER	11	X'0000000B'
MQOT_SERVICE	12	X'0000000C'
MQOT_RESERVED_1	999	X'000003E7'

Extended Object Types

MQOT_ALL	1001	X'000003E9'
MQOT_ALIAS_Q	1002	X'000003EA'
MQOT_MODEL_Q	1003	X'000003EB'
MQOT_LOCAL_Q	1004	X'000003EC'
MQOT_REMOTE_Q	1005	X'000003ED'
MQOT_SENDER_CHANNEL	1007	X'000003EF'
MQOT_SERVER_CHANNEL	1008	X'000003F0'
MQOT_REQUESTER_CHANNEL	1009	X'000003F1'
MQOT_RECEIVER_CHANNEL	1010	X'000003F2'
MQOT_CURRENT_CHANNEL	1011	X'000003F3'
MQOT_SAVED_CHANNEL	1012	X'000003F4'
MQOT_SVRCONN_CHANNEL	1013	X'000003F5'
MQOT_CLNTCONN_CHANNEL	1014	X'000003F6'
MQOT_SHORT_CHANNEL	1015	X'000003F7'
MQOT_CHLAUTH	1016	X'000003F8'
MQOT_REMOTE_Q_MGR_NAME	1017	X'000003F9'
MQOT_PROT_POLICY	1019	X'000003FB'
MQOT_TT_CHANNEL	1020	X'000003FC'
V 9.0.0 MQOT_AMQP_CHANNEL	1021	X'000003FD'
MQOT_AUTH_REC	1022	X'000003FE'

MQPA_* (Put Authority):

MQPA_DEFAULT	1	X'00000001'
MQPA_CONTEXT	2	X'00000002'
MQPA_ONLY_MCA	3	X'00000003'
MQPA_ALTERNATE_OR_MCA	4	X'00000004'

MQPD_* (Property descriptor, support and context):
Property descriptor structure

MQPD_STRUC_ID	"PDbb"	
MQPD_STRUC_ID_ARRAY	'P','D','b','b'	
MQPD_VERSION_1	1	X'00000001'
MQPD_CURRENT_VERSION	1	X'00000001'

Property Descriptor Options

MQPD_NONE	0	X'00000000'

Property Support Options

MQPD_SUPPORT_OPTIONAL	1	X'00000001'
MQPD_SUPPORT_REQUIRED	1048576	X'00100000'
MQPD_SUPPORT_REQUIRED_IF_LOCAL	1024	X'00000400'
MQPD_REJECT_UNSUP_MASK	-1048576	X'FFF00000'
MQPD_ACCEPT_UNSUP_IF_XMIT_MASK	1047552	X'000FFC00'
MQPD_ACCEPT_UNSUP_MASK	1023	X'000003FF'

Property Context

MQPD_NO_CONTEXT	0	X'00000000'
MQPD_USER_CONTEXT	1	X'00000001'

MQPER_* (Persistence Values):

MQPER_PERSISTENCE_AS_PARENT	-1	X'FFFFFFFF'
MQPER_NOT_PERSISTENT	0	X'00000000'
MQPER_PERSISTENT	1	X'00000001'
MQPER_PERSISTENCE_AS_Q_DEF	2	X'00000002'
MQPER_PERSISTENCE_AS_TOPIC_DEF	2	X'00000002'

MQPL_* (Platforms):

MQPL_MVS	1	X'00000001'
MQPL_OS390	1	X'00000001'
MQPL_ZOS	1	X'00000001'
MQPL_OS2	2	X'00000002'
MQPL_AIX	3	X'00000003'
MQPL_UNIX	3	X'00000003'
MQPL_OS400	4	X'00000004'
MQPL_WINDOWS	5	X'00000005'
MQPL_WINDOWS_NT	11	X'0000000B'
MQPL_VMS	12	X'0000000C'
MQPL_NSK	13	X'0000000D'
 MQPL_NSS	13	X'0000000D'
MQPL_OPEN_TP1	15	X'0000000F'
MQPL_VM	18	X'00000012'
MQPL_TPF	23	X'00000017'
MQPL_VSE	27	X'0000001B'
MQPL_APPLIANCE	28	X'0000001C'

MQPL_NATIVE	1	X'00000001'

MQPMO_* (Put message options and structure for publish mask):
Put message options structure

MQPMO_STRUC_ID	"PMOb"	
MQPMO_STRUC_ID_ARRAY	'P','M','O','b'	
MQPMO_VERSION_1	1	X'00000001'
MQPMO_VERSION_2	2	X'00000002'
MQPMO_VERSION_3	3	X'00000003'
MQPMO_CURRENT_VERSION	3	X'00000003'
MQPMO_CURRENT_LENGTH	(value differs by platform or version)	

Put Message Options

MQPMO_SYNCPOINT	2	X'00000002'
MQPMO_NO_SYNCPOINT	4	X'00000004'
MQPMO_DEFAULT_CONTEXT	32	X'00000020'
MQPMO_NEW_MSG_ID	64	X'00000040'
MQPMO_NEW_CORREL_ID	128	X'00000080'
MQPMO_PASS_IDENTITY_CONTEXT	256	X'00000100'
MQPMO_PASS_ALL_CONTEXT	512	X'00000200'
MQPMO_SET_IDENTITY_CONTEXT	1024	X'00000400'
MQPMO_SET_ALL_CONTEXT	2048	X'00000800'
MQPMO_ALTERNATE_USER_AUTHORITY	4096	X'00001000'
MQPMO_FAIL_IF_QUIESCING	8192	X'00002000'
MQPMO_NO_CONTEXT	16384	X'00004000'
MQPMO_LOGICAL_ORDER	32768	X'00008000'
MQPMO_ASYNC_RESPONSE	65536	X'00010000'
MQPMO_SYNC_RESPONSE	131072	X'00020000'
MQPMO_RESOLVE_LOCAL_Q	262144	X'00040000'
MQPMO_RETAIN	2097152	X'00200000'
MQPMO_MD_FOR_OUTPUT_ONLY	8388608	X'00800000'
MQPMO_SCOPE_QMGR	67108864	X'04000000'
MQPMO_SUPPRESS_REPLYTO	134217728	X'08000000'
MQPMO_NOT_OWN_SUBS	268435456	X'10000000'
MQPMO_RESPONSE_AS_Q_DEF	0	X'00000000'
MQPMO_RESPONSE_AS_TOPIC_DEF	0	X'00000000'
MQPMO_NONE	0	X'00000000'

Put Message Options for publish mask

MQPMO_PUB_OPTIONS_MASK	2097152	X'00200000'

MQPMRF_* (Put Message Record Fields):

MQPMRF_MSG_ID	1	X'00000001'
MQPMRF_CORREL_ID	2	X'00000002'
MQPMRF_GROUP_ID	4	X'00000004'
MQPMRF_FEEDBACK	8	X'00000008'
MQPMRF_ACCOUNTING_TOKEN	16	X'00000010'
MQPMRF_NONE	0	X'00000000'

MQPO_* (Command format Purge Options):

MQPO_YES	1	X'00000001'
MQPO_NO	0	X'00000000'

MQPRI_* (Priority):

MQPRI_PRIORITY_AS_Q_DEF	-1	X'FFFFFFFF'
MQPRI_PRIORITY_AS_PARENT	-2	X'FFFFFFFE'
MQPRI_PRIORITY_AS_PUBLISHED	-3	X'FFFFFFFD'
MQPRI_PRIORITY_AS_TOPIC_DEF	-1	X'FFFFFFFF'

MQPROP_* (Queue and Channel Property Control Values and Maximum Properties Length):
Queue and Channel Property Control Values

MQPROP_COMPATIBILITY	0	X'00000000'
MQPROP_NONE	1	X'00000001'
MQPROP_ALL	2	X'00000002'
MQPROP_FORCE_MQRFH2	3	X'00000003'

Maximum Properties Length

MQPROP_UNRESTRICTED_LENGTH	-1	X'FFFFFFFF'

MQPRT_* (Put Response Values):

MQPRT_RESPONSE_AS_PARENT	0	X'00000000'
MQPRT_SYNC_RESPONSE	1	X'00000001'
MQPRT_ASYNC_RESPONSE	2	X'00000002'

MQPS_* (Publish/Subscribe):
Command format Publish/Subscribe Status

MQPS_STATUS_INACTIVE	0	X'00000000'
MQPS_STATUS_STARTING	1	X'00000001'
MQPS_STATUS_STOPPING	2	X'00000002'
MQPS_STATUS_ACTIVE	3	X'00000003'
MQPS_STATUS_COMPAT	4	X'00000004'
MQPS_STATUS_ERROR	5	X'00000005'
MQPS_STATUS_REFUSED	6	X'00000006'

Publish/Subscribe Tags as strings

MQPS_COMMAND	"MQPSCommand"	
MQPS_COMP_CODE	"MQPSCompCode"	
MQPS_CORREL_ID	"MQPSCorrelId"	
MQPS_DELETE_OPTIONS	"MQPSDelOpts"	
MQPS_ERROR_ID	"MQPSErrorId"	
MQPS_ERROR_POS	"MQPSErrorPos"	
MQPS_INTEGER_DATA	"MQPSIntData"	
MQPS_PARAMETER_ID	"MQPSParmId"	
MQPS_PUBLICATION_OPTIONS	"MQPSPubOpts"	
MQPS_PUBLISH_TIMESTAMP	"MQPSPubTime"	
MQPS_Q_MGR_NAME	"MQPSQMgrName"	
MQPS_Q_NAME	"MQPSQName"	
MQPS_REASON	"MQPSReason"	
MQPS_REASON_TEXT	"MQPSReasonText"	
MQPS_REGISTRATION_OPTIONS	"MQPSRegOpts"	
MQPS_SEQUENCE_NUMBER	"MQPSSeqNum"	
MQPS_STREAM_NAME	"MQPSStreamName"	
MQPS_STRING_DATA	"MQPSStringData"	
MQPS_SUBSCRIPTION_IDENTITY	"MQPSSubIdentity"	
MQPS_SUBSCRIPTION_NAME	"MQPSSubName"	
MQPS_SUBSCRIPTION_USER_DATA	"MQPSSubUserData"	
MQPS_TOPIC	"MQPSTopic"	
MQPS_USER_ID	"MQPSUserId"	

Publish/Subscribe Tags as blank-enclosed strings

MQPS_COMMAND_B	"bMQPSCommandb"	
MQPS_COMP_CODE_B	"bMQPSCompCodeb"	
MQPS_CORREL_ID_B	"bMQPSCorrelIdb"	
MQPS_DELETE_OPTIONS_B	"bMQPSDelOptsb"	
MQPS_ERROR_ID_B	"bMQPSErrorIdb"	
MQPS_ERROR_POS_B	"bMQPSErrorPosb"	
MQPS_INTEGER_DATA_B	"bMQPSIntDatab"	
MQPS_PARAMETER_ID_B	"bMQPSParmIdb"	
MQPS_PUBLICATION_OPTIONS_B	"bMQPSPubOptsb"	
MQPS_PUBLISH_TIMESTAMP_B	"bMQPSPubTimeb"	
MQPS_Q_MGR_NAME_B	"bMQPSQMgrNameb"	
MQPS_Q_NAME_B	"bMQPSQNameb"	
MQPS_REASON_B	"bMQPSReasonb"	
MQPS_REASON_TEXT_B	"bMQPSReasonTextb"	
MQPS_REGISTRATION_OPTIONS_B	"bMQPSRegOptsb"	
MQPS_SEQUENCE_NUMBER_B	"bMQPSSeqNumb"	
MQPS_STREAM_NAME_B	"bMQPSStreamNameb"	
MQPS_STRING_DATA_B	"bMQPSStringDatab"	
MQPS_SUBSCRIPTION_IDENTITY_B	"bMQPSSubIdentityb"	
MQPS_SUBSCRIPTION_NAME_B	"bMQPSSubNameb"	
MQPS_SUBSCRIPTION_USER_DATA_B	"bMQPSSubUserDatab"	
MQPS_TOPIC_B	"bMQPSTopicb"	
MQPS_USER_ID_B	"bMQPSUserIdb"	

Publish/Subscribe Command Tag Values as strings

MQPS_DELETE_PUBLICATION	"DeletePub"	
MQPS_DEREGISTER_PUBLISHER	"DeregPub"	
MQPS_DEREGISTER_SUBSCRIBER	"DeregSub"	
MQPS_PUBLISH	"Publish"	
MQPS_REGISTER_PUBLISHER	"RegPub"	
MQPS_REGISTER_SUBSCRIBER	"RegSub"	
MQPS_REQUEST_UPDATE	"ReqUpdate"	

Publish/Subscribe Command Tag Values as blank-enclosed strings

MQPS_DELETE_PUBLICATION_B	"bDeletePubb"	
MQPS_DEREGISTER_PUBLISHER_B	"bDeregPubb"	
MQPS_DEREGISTER_SUBSCRIBER_B	"bDeregSubb"	
MQPS_PUBLISH_B	"bPublishb"	
MQPS_REGISTER_PUBLISHER_B	"bRegPubb"	
MQPS_REGISTER_SUBSCRIBER_B	"bRegSubb"	
MQPS_REQUEST_UPDATE_B	"bReqUpdateb"	

Publish/Subscribe Options Tag Values as strings

MQPS_ADD_NAME	"AddName"	
MQPS_ANONYMOUS	"Anon"	
MQPS_CORREL_ID_AS_IDENTITY	"CorrelAsId"	
MQPS_DEREGISTER_ALL	"DeregAll"	
MQPS_DIRECT_REQUESTS	"DirectReq"	
MQPS_DUPLICATES_OK	"DupsOK"	
MQPS_FULL_RESPONSE	"FullResp"	
MQPS_INCLUDE_STREAM_NAME	"InclStreamName"	
MQPS_INFORM_IF_RETAINED	"InformIfRet"	
MQPS_IS_RETAINED_PUBLICATION	"IsRetainedPub"	
MQPS_JOIN_EXCLUSIVE	"JoinExcl"	
MQPS_JOIN_SHARED	"JoinShared"	
MQPS_LEAVE_ONLY	"LeaveOnly"	
MQPS_LOCAL	"Local"	
MQPS_LOCKED	"Locked"	
MQPS_NEW_PUBLICATIONS_ONLY	"NewPubsOnly"	
MQPS_NO_ALTERATION	"NoAlter"	
MQPS_NO_REGISTRATION	"NoReg"	
MQPS_NON_PERSISTENT	"NonPers"	
MQPS_NONE	"None"	
MQPS_OTHER_SUBSCRIBERS_ONLY	"OtherSubsOnly"	
MQPS_PERSISTENT	"Pers"	
MQPS_PERSISTENT_AS_PUBLISH	"PersAsPub"	
MQPS_PERSISTENT_AS_Q	"PersAsQueue"	
MQPS_PUBLISH_ON_REQUEST_ONLY	"PubOnReqOnly"	
MQPS_RETAIN_PUBLICATION	"RetainPub"	
MQPS_VARIABLE_USER_ID	"VariableUserId"	

Publish/Subscribe Options Tag Values as blank-enclosed strings

MQPS_ADD_NAME_B	"bAddNameb"	
MQPS_ANONYMOUS_B	"bAnonb"	
MQPS_CORREL_ID_AS_IDENTITY_B	"bCorrelAsIdb"	
MQPS_DEREGISTER_ALL_B	"bDeregAllb"	
MQPS_DIRECT_REQUESTS_B	"bDirectReqb"	
MQPS_DUPLICATES_OK_B	"bDupsOKb"	
MQPS_FULL_RESPONSE_B	"bFullRespb"	
MQPS_INCLUDE_STREAM_NAME_B	"bInclStreamNameb"	
MQPS_INFORM_IF_RETAINED_B	"bInformIfRetb"	
MQPS_IS_RETAINED_PUBLICATION_B	"bIsRetainedPubb"	
MQPS_JOIN_EXCLUSIVE_B	"bJoinExclb"	
MQPS_JOIN_SHARED_B	"bJoinSharedb"	
MQPS_LEAVE_ONLY_B	"bLeaveOnlyb"	
MQPS_LOCAL_B	"bLocalb"	
MQPS_LOCKED_B	"bLockedb"	
MQPS_NEW_PUBLICATIONS_ONLY_B	"bNewPubsOnlyb"	
MQPS_NO_ALTERATION_B	"bNoAlterb"	
MQPS_NO_REGISTRATION_B	"bNoRegb"	
MQPS_NON_PERSISTENT_B	"bNonPersb"	
MQPS_NONE_B	"bNoneb"	
MQPS_OTHER_SUBSCRIBERS_ONLY_B	"bOtherSubsOnlyb"	
MQPS_PERSISTENT_B	"bPersb"	
MQPS_PERSISTENT_AS_PUBLISH_B	"bPersAsPubb"	
MQPS_PERSISTENT_AS_Q_B	"bPersAsQueueb"	
MQPS_PUBLISH_ON_REQUEST_ONLY_B	"bPubOnReqOnlyb"	
MQPS_RETAIN_PUBLICATION_B	"bRetainPubb"	
MQPS_VARIABLE_USER_ID_B	"bVariableUserIdb"	

MQPSC_* (Publish/Subscribe Options Tag Publish/Subscribe Command Folder (psc) Tags):

MQPSC_FOLDER_VERSION	1	X'00000001'

MQPSC_* (Publish/Subscribe Options Tag Tag names):

MQPSC_COMMAND	"Command"	
MQPSC_REGISTRATION_OPTION	"RegOpt"	
MQPSC_PUBLICATION_OPTION	"PubOpt"	
MQPSC_DELETE_OPTION	"DelOpt"	
MQPSC_TOPIC	"Topic"	
MQPSC_SUBSCRIPTION_POINT	"SubPoint"	
MQPSC_FILTER	"Filter"	

MQPSC_Q_MGR_NAME	"QMgrName"	
MQPSC_Q_NAME	"QName"	
MQPSC_PUBLISH_TIMESTAMP	"PubTime"	
MQPSC_SEQUENCE_NUMBER	"SeqNum"	
MQPSC_SUBSCRIPTION_NAME	"SubName"	
MQPSC_SUBSCRIPTION_IDENTITY	"SubIdentity"	
MQPSC_SUBSCRIPTION_USER_DATA	"SubUserData"	
MQPSC_CORREL_ID	"CorrelId"	

MQPSC_* (Publish/Subscribe Options Tag XML tag names):

MQPSC_COMMAND_B	"<Command>"	
MQPSC_COMMAND_E	"</Command>"	
MQPSC_REGISTRATION_OPTION_B	"<RegOpt>"	
MQPSC_REGISTRATION_OPTION_E	"</RegOpt>"	
MQPSC_PUBLICATION_OPTION_B	"<PubOpt>"	
MQPSC_PUBLICATION_OPTION_E	"</PubOpt>"	
MQPSC_DELETE_OPTION_B	"<DelOpt>"	
MQPSC_DELETE_OPTION_E	"</DelOpt>"	
MQPSC_TOPIC_B	"<Topic>"	
MQPSC_TOPIC_E	"</Topic>"	
MQPSC_SUBSCRIPTION_POINT_B	"<SubPoint>"	
MQPSC_SUBSCRIPTION_POINT_E	"</SubPoint>"	
MQPSC_FILTER_B	"<Filter>"	
MQPSC_FILTER_E	"</Filter>"	
MQPSC_Q_MGR_NAME_B	"<QMgrName>"	
MQPSC_Q_MGR_NAME_E	"</QMgrName>"	
MQPSC_Q_NAME_B	"<QName>"	
MQPSC_Q_NAME_E	"</QName>"	
MQPSC_PUBLISH_TIMESTAMP_B	"<PubTime>"	
MQPSC_PUBLISH_TIMESTAMP_E	"</PubTime>"	
MQPSC_SEQUENCE_NUMBER_B	"<SeqNum>"	
MQPSC_SEQUENCE_NUMBER_E	"</SeqNum>"	
MQPSC_SUBSCRIPTION_NAME_B	"<SubName>"	
MQPSC_SUBSCRIPTION_NAME_E	"</SubName>"	
MQPSC_SUBSCRIPTION_IDENTITY_B	"<SubIdentity>"	
MQPSC_SUBSCRIPTION_IDENTITY_E	"</SubIdentity>"	
MQPSC_SUBSCRIPTION_USER_DATA_B	"<SubUserData>"	
MQPSC_SUBSCRIPTION_USER_DATA_E	"</SubUserData>"	
MQPSC_CORREL_ID_B	"<CorrelId>"	

MQPSC_CORREL_ID_E	"</CorrelId>"	

MQPSC_* (Publish/Subscribe Options Tag Values as strings):

MQPSC_DELETE_PUBLICATION	"DeletePub"	
MQPSC_DEREGISTER_SUBSCRIBER	"DeregSub"	
MQPSC_PUBLISH	"Publish"	
MQPSC_REGISTER_SUBSCRIBER	"RegSub"	
MQPSC_REQUEST_UPDATE	"ReqUpdate"	

MQPSC_* (Publish/Subscribe Options Tag Values as strings):

MQPSC_ADD_NAME	"AddName"	
MQPSC_CORREL_ID_AS_IDENTITY	"CorrelAsId"	
MQPSC_DEREGISTER_ALL	"DeregAll"	
MQPSC_DUPLICATES_OK	"DupsOK"	
MQPSC_FULL_RESPONSE	"FullResp"	
MQPSC_INFORM_IF_RETAINED	"InformIfRet"	
MQPSC_IS_RETAINED_PUB	"IsRetainedPub"	
MQPSC_JOIN_SHARED	"JoinShared"	
MQPSC_JOIN_EXCLUSIVE	"JoinExcl"	
MQPSC_LEAVE_ONLY	"LeaveOnly"	
MQPSC_LOCAL	"Local"	
MQPSC_LOCKED	"Locked"	
MQPSC_NEW_PUBS_ONLY	"NewPubsOnly"	
MQPSC_NO_ALTERATION	"NoAlter"	
MQPSC_NON_PERSISTENT	"NonPers"	
MQPSC_OTHER_SUBS_ONLY	"OtherSubsOnly"	
MQPSC_PERSISTENT	"Pers"	
MQPSC_PERSISTENT_AS_PUBLISH	"PersAsPub"	
MQPSC_PERSISTENT_AS_Q	"PersAsQueue"	
MQPSC_NONE	"None"	
MQPSC_PUB_ON_REQUEST_ONLY	"PubOnReqOnly"	
MQPSC_RETAIN_PUB	"RetainPub"	
MQPSC_VARIABLE_USER_ID	"VariableUserId"	

MQPSCR_* (Publish/Subscribe Options):
Publish/Subscribe Options Tag Publish/Subscribe Response Folder (pscr) Tags

MQPSCR_FOLDER_VERSION	1	X'00000001'

Publish/Subscribe Options Tag Tag names

MQPSCR_COMPLETION	"Completion"	
MQPSCR_RESPONSE	"Response"	
MQPSCR_REASON	"Reason"	

Publish/Subscribe Options Tag XML tag names

MQPSCR_COMPLETION_B	"<Completion>"	
MQPSCR_COMPLETION_E	"</Completion>"	
MQPSCR_RESPONSE_B	"<Response>"	
MQPSCR_RESPONSE_E	"</Response>"	
MQPSCR_REASON_B	"<Reason>"	
MQPSCR_REASON_E	"</Reason>"	

Publish/Subscribe Options Tag Tag values

MQPSCR_OK	"ok"	
MQPSCR_WARNING	"warning"	
MQPSCR_ERROR	"error"	

MQPSM_* (Pub/Sub Mode):

MQPSM_DISABLED	0	X'00000000'
MQPSM_COMPAT	1	X'00000001'
MQPSM_ENABLED	2	X'00000002'

MQPSPROP_* (Pub/Sub Message Properties):

MQPSPROP_NONE	0	X'00000000'
MQPSPROP_COMPAT	1	X'00000001'
MQPSPROP_RFH2	2	X'00000002'
MQPSPROP_MSGPROP	3	X'00000003'

MQPSST_* (Command format Pub/Sub Status Type):

MQPSST_ALL	0	X'00000000'
MQPSST_LOCAL	1	X'00000001'
MQPSST_PARENT	2	X'00000002'
MQPSST_CHILD	3	X'00000003'

MQPUBO_* (Publish/Subscribe Publication Options):

MQPUBO_NONE	0	X'00000000'
MQPUBO_CORREL_ID_AS_IDENTITY	1	X'00000001'
MQPUBO_RETAIN_PUBLICATION	2	X'00000002'
MQPUBO_OTHER_SUBSCRIBERS_ONLY	4	X'00000004'
MQPUBO_NO_REGISTRATION	8	X'00000008'
MQPUBO_IS_RETAINED_PUBLICATION	16	X'00000010'

MQPXP_* (Publish/subscribe routing exit parameter structure):

MQPXP_STRUC_ID	"PXPb"	
MQPXP_STRUC_ID_ARRAY	'P','X','P','b'	
MQPXP_VERSION_1	1	X'00000001'
MQPXP_CURRENT_VERSION	1	X'00000001'

MQQA_* (Queue attributes):
Inhibit Get Values

MQQA_GET_INHIBITED	1	X'00000001'
MQQA_GET_ALLOWED	0	X'00000000'

Inhibit Put Values

MQQA_PUT_INHIBITED	1	X'00000001'
MQQA_PUT_ALLOWED	0	X'00000000'

Queue Shareability

MQQA_SHAREABLE	1	X'00000001'
MQQA_NOT_SHAREABLE	0	X'00000000'

Back-Out Hardening

MQQA_BACKOUT_HARDENED	1	X'00000001'
MQQA_BACKOUT_NOT_HARDENED	0	X'00000000'

MQQDT_* (Queue Definition Types):

MQQDT_PREDEFINED	1	X'00000001'
MQQDT_PERMANENT_DYNAMIC	2	X'00000002'
MQQDT_TEMPORARY_DYNAMIC	3	X'00000003'
MQQDT_SHARED_DYNAMIC	4	X'00000004'

MQQF_* (Queue Flags):

MQQF_LOCAL_Q	1	X'00000001'
MQQF_CLWL_USEQ_ANY	64	X'00000040'
MQQF_CLWL_USEQ_LOCAL	128	X'00000080'

MQQMDT_* (Command format Queue Manager Definition Types):

MQQMDT_EXPLICIT_CLUSTER_SENDER	1	X'00000001'
MQQMDT_AUTO_CLUSTER_SENDER	2	X'00000002'
MQQMDT_AUTO_EXP_CLUSTER_SENDER	4	X'00000004'
MQQMDT_CLUSTER_RECEIVER	3	X'00000003'

MQQMF_* (Queue Manager Flags):

MQQMF_REPOSITORY_Q_MGR	2	X'00000002'
MQQMF_CLUSSDR_USER_DEFINED	8	X'00000008'
MQQMF_CLUSSDR_AUTO_DEFINED	16	X'00000010'
MQQMF_AVAILABLE	32	X'00000020'

MQQMFAC_* (Command format Queue Manager Facility):

MQQMFAC_IMS_BRIDGE	1	X'00000001'
MQQMFAC_DB2	2	X'00000002'

MQQMSTA_* (Command format Queue Manager Status):

MQQMSTA_STARTING	1	X'00000001'
MQQMSTA_RUNNING	2	X'00000002'
MQQMSTA_QUIESCING	3	X'00000003'

MQQMT_* (Command format Queue Manager Types):

MQQMT_NORMAL	0	X'00000000'
MQQMT_REPOSITORY	1	X'00000001'

MQQO_* (Command format Quiesce Options):

MQQO_YES	1	X'00000001'
MQQO_NO	0	X'00000000'

MQQSGD_* (Queue-sharing group dispositions):

MQQSGD_ALL	-1	X'FFFFFFFF'
MQQSGD_Q_MGR	0	X'00000000'
MQQSGD_COPY	1	X'00000001'
MQQSGD_SHARED	2	X'00000002'
MQQSGD_GROUP	3	X'00000003'
MQQSGD_PRIVATE	4	X'00000004'
MQQSGD_LIVE	6	X'00000006'

MQQSGS_* (Command format queue-sharing group status):

MQQSGS_UNKNOWN	0	X'00000000'
MQQSGS_CREATED	1	X'00000001'
MQQSGS_ACTIVE	2	X'00000002'
MQQSGS_INACTIVE	3	X'00000003'
MQQSGS_FAILED	4	X'00000004'
MQQSGS_PENDING	5	X'00000005'

MQQSIE_* (Command format Queue Service-Interval Events):

MQQSIE_NONE	0	X'00000000'
MQQSIE_HIGH	1	X'00000001'
MQQSIE_OK	2	X'00000002'

MQQSO_* (Command format Queue Status Open Options for SET, BROWSE, INPUT):

MQQSO_NO	0	X'00000000'
MQQSO_YES	1	X'00000001'
MQQSO_SHARED	1	X'00000001'
MQQSO_EXCLUSIVE	2	X'00000002'

MQQSOT_* (Command format Queue Status Open Types):

MQQSOT_ALL	1	X'00000001'
MQQSOT_INPUT	2	X'00000002'
MQQSOT_OUTPUT	3	X'00000003'

MQQSUM_* (Command format Queue Status Uncommitted Messages):

MQQSUM_YES	1	X'00000001'
MQQSUM_NO	0	X'00000000'

MQQT_* (Queue Types and Extended Queue Types):
Queue Types

MQQT_LOCAL	1	X'00000001'
MQQT_MODEL	2	X'00000002'
MQQT_ALIAS	3	X'00000003'
MQQT_REMOTE	6	X'00000006'
MQQT_CLUSTER	7	X'00000007'

Extended Queue Types

MQQT_ALL	1001	X'000003E9'

MQRC_* (reason codes):

MQRC_NONE	0	X'00000000'
MQRC_APPL_FIRST	900	X'00000384'
MQRC_APPL_LAST	999	X'000003E7'
MQRC_ALIAS_BASE_Q_TYPE_ERROR	2001	X'000007D1'
MQRC_ALREADY_CONNECTED	2002	X'000007D2'
MQRC_BACKED_OUT	2003	X'000007D3'
MQRC_BUFFER_ERROR	2004	X'000007D4'
MQRC_BUFFER_LENGTH_ERROR	2005	X'000007D5'
MQRC_CHAR_ATTR_LENGTH_ERROR	2006	X'000007D6'
MQRC_CHAR_ATTRS_ERROR	2007	X'000007D7'
MQRC_CHAR_ATTRS_TOO_SHORT	2008	X'000007D8'
MQRC_CONNECTION_BROKEN	2009	X'000007D9'
MQRC_DATA_LENGTH_ERROR	2010	X'000007DA'
MQRC_DYNAMIC_Q_NAME_ERROR	2011	X'000007DB'
MQRC_ENVIRONMENT_ERROR	2012	X'000007DC'
MQRC_EXPIRY_ERROR	2013	X'000007DD'
MQRC_FEEDBACK_ERROR	2014	X'000007DE'
MQRC_GET_INHIBITED	2016	X'000007E0'
MQRC_HANDLE_NOT_AVAILABLE	2017	X'000007E1'
MQRC_HCONN_ERROR	2018	X'000007E2'

MQRC_HOBJ_ERROR	2019	X'000007E3'
MQRC_INHIBIT_VALUE_ERROR	2020	X'000007E4'
MQRC_INT_ATTR_COUNT_ERROR	2021	X'000007E5'
MQRC_INT_ATTR_COUNT_TOO_SMALL	2022	X'000007E6'
MQRC_INT_ATTRS_ARRAY_ERROR	2023	X'000007E7'
MQRC_SYNCPOINT_LIMIT_REACHED	2024	X'000007E8'
MQRC_MAX_CONNS_LIMIT_REACHED	2025	X'000007E9'
MQRC_MD_ERROR	2026	X'000007EA'
MQRC_MISSING_REPLY_TO_Q	2027	X'000007EB'
MQRC_MSG_TYPE_ERROR	2029	X'000007ED'
MQRC_MSG_TOO_BIG_FOR_Q	2030	X'000007EE'
MQRC_MSG_TOO_BIG_FOR_Q_MGR	2031	X'000007EF'
MQRC_NO_MSG_AVAILABLE	2033	X'000007F1'
MQRC_NO_MSG_UNDER_CURSOR	2034	X'000007F2'
MQRC_NOT_AUTHORIZED	2035	X'000007F3'
MQRC_NOT_OPEN_FOR_BROWSE	2036	X'000007F4'
MQRC_NOT_OPEN_FOR_INPUT	2037	X'000007F5'
MQRC_NOT_OPEN_FOR_INQUIRE	2038	X'000007F6'
MQRC_NOT_OPEN_FOR_OUTPUT	2039	X'000007F7'
MQRC_NOT_OPEN_FOR_SET	2040	X'000007F8'
MQRC_OBJECT_CHANGED	2041	X'000007F9'
MQRC_OBJECT_IN_USE	2042	X'000007FA'
MQRC_OBJECT_TYPE_ERROR	2043	X'000007FB'
MQRC_OD_ERROR	2044	X'000007FC'
MQRC_OPTION_NOT_VALID_FOR_TYPE	2045	X'000007FD'
MQRC_OPTIONS_ERROR	2046	X'000007FE'
MQRC_PERSISTENCE_ERROR	2047	X'000007FF'
MQRC_PERSISTENT_NOT_ALLOWED	2048	X'00000800'
MQRC_PRIORITY_EXCEEDS_MAXIMUM	2049	X'00000801'
MQRC_PRIORITY_ERROR	2050	X'00000802'
MQRC_PUT_INHIBITED	2051	X'00000803'
MQRC_Q_DELETED	2052	X'00000804'
MQRC_Q_FULL	2053	X'00000805'
MQRC_Q_NOT_EMPTY	2055	X'00000807'
MQRC_Q_SPACE_NOT_AVAILABLE	2056	X'00000808'
MQRC_Q_TYPE_ERROR	2057	X'00000809'
MQRC_Q_MGR_NAME_ERROR	2058	X'0000080A'
MQRC_Q_MGR_NOT_AVAILABLE	2059	X'0000080B'
MQRC_REPORT_OPTIONS_ERROR	2061	X'0000080D'
MQRC_SECOND_MARK_NOT_ALLOWED	2062	X'0000080E'
MQRC_SECURITY_ERROR	2063	X'0000080F'
MQRC_SELECTOR_COUNT_ERROR	2065	X'00000811'

MQRC_SELECTOR_LIMIT_EXCEEDED	2066	X'00000812'
MQRC_SELECTOR_ERROR	2067	X'00000813'
MQRC_SELECTOR_NOT_FOR_TYPE	2068	X'00000814'
MQRC_SIGNAL_OUTSTANDING	2069	X'00000815'
MQRC_SIGNAL_REQUEST_ACCEPTED	2070	X'00000816'
MQRC_STORAGE_NOT_AVAILABLE	2071	X'00000817'
MQRC_SYNCPOINT_NOT_AVAILABLE	2072	X'00000818'
MQRC_TRIGGER_CONTROL_ERROR	2075	X'0000081B'
MQRC_TRIGGER_DEPTH_ERROR	2076	X'0000081C'
MQRC_TRIGGER_MSG_PRIORITY_ERR	2077	X'0000081D'
MQRC_TRIGGER_TYPE_ERROR	2078	X'0000081E'
MQRC_TRUNCATED_MSG_ACCEPTED	2079	X'0000081F'
MQRC_TRUNCATED_MSG_FAILED	2080	X'00000820'
MQRC_UNKNOWN_ALIAS_BASE_Q	2082	X'00000822'
MQRC_UNKNOWN_OBJECT_NAME	2085	X'00000825'
MQRC_UNKNOWN_OBJECT_Q_MGR	2086	X'00000826'
MQRC_UNKNOWN_REMOTE_Q_MGR	2087	X'00000827'
MQRC_WAIT_INTERVAL_ERROR	2090	X'0000082A'
MQRC_XMIT_Q_TYPE_ERROR	2091	X'0000082B'
MQRC_XMIT_Q_USAGE_ERROR	2092	X'0000082C'
MQRC_NOT_OPEN_FOR_PASS_ALL	2093	X'0000082D'
MQRC_NOT_OPEN_FOR_PASS_IDENT	2094	X'0000082E'
MQRC_NOT_OPEN_FOR_SET_ALL	2095	X'0000082F'
MQRC_NOT_OPEN_FOR_SET_IDENT	2096	X'00000830'
MQRC_CONTEXT_HANDLE_ERROR	2097	X'00000831'
MQRC_CONTEXT_NOT_AVAILABLE	2098	X'00000832'
MQRC_SIGNAL1_ERROR	2099	X'00000833'
MQRC_OBJECT_ALREADY_EXISTS	2100	X'00000834'
MQRC_OBJECT_DAMAGED	2101	X'00000835'
MQRC_RESOURCE_PROBLEM	2102	X'00000836'
MQRC_ANOTHER_Q_MGR_CONNECTED	2103	X'00000837'
MQRC_UNKNOWN_REPORT_OPTION	2104	X'00000838'
MQRC_STORAGE_CLASS_ERROR	2105	X'00000839'
MQRC_COD_NOT_VALID_FOR_XCF_Q	2106	X'0000083A'
MQRC_XWAIT_CANCELED	2107	X'0000083B'
MQRC_XWAIT_ERROR	2108	X'0000083C'
MQRC_SUPPRESSED_BY_EXIT	2109	X'0000083D'
MQRC_FORMAT_ERROR	2110	X'0000083E'
MQRC_SOURCE_CCSID_ERROR	2111	X'0000083F'
MQRC_SOURCE_INTEGER_ENC_ERROR	2112	X'00000840'
MQRC_SOURCE_DECIMAL_ENC_ERROR	2113	X'00000841'
MQRC_SOURCE_FLOAT_ENC_ERROR	2114	X'00000842'

MQRC_TARGET_CCSID_ERROR	2115	X'00000843'
MQRC_TARGET_INTEGER_ENC_ERROR	2116	X'00000844'
MQRC_TARGET_DECIMAL_ENC_ERROR	2117	X'00000845'
MQRC_TARGET_FLOAT_ENC_ERROR	2118	X'00000846'
MQRC_NOT_CONVERTED	2119	X'00000847'
MQRC_CONVERTED_MSG_TOO_BIG	2120	X'00000848'
MQRC_TRUNCATED	2120	X'00000848'
MQRC_NO_EXTERNAL_PARTICIPANTS	2121	X'00000849'
MQRC_PARTICIPANT_NOT_AVAILABLE	2122	X'0000084A'
MQRC_OUTCOME_MIXED	2123	X'0000084B'
MQRC_OUTCOME_PENDING	2124	X'0000084C'
MQRC_BRIDGE_STARTED	2125	X'0000084D'
MQRC_BRIDGE_STOPPED	2126	X'0000084E'
MQRC_ADAPTER_STORAGE_SHORTAGE	2127	X'0000084F'
MQRC_UOW_IN_PROGRESS	2128	X'00000850'
MQRC_ADAPTER_CONN_LOAD_ERROR	2129	X'00000851'
MQRC_ADAPTER_SERV_LOAD_ERROR	2130	X'00000852'
MQRC_ADAPTER_DEFS_ERROR	2131	X'00000853'
MQRC_ADAPTER_DEFS_LOAD_ERROR	2132	X'00000854'
MQRC_ADAPTER_CONV_LOAD_ERROR	2133	X'00000855'
MQRC_BO_ERROR	2134	X'00000856'
MQRC_DH_ERROR	2135	X'00000857'
MQRC_MULTIPLE_REASONS	2136	X'00000858'
MQRC_OPEN_FAILED	2137	X'00000859'
MQRC_ADAPTER_DISC_LOAD_ERROR	2138	X'0000085A'
MQRC_CNO_ERROR	2139	X'0000085B'
MQRC_CICS_WAIT_FAILED	2140	X'0000085C'
MQRC_DLH_ERROR	2141	X'0000085D'
MQRC_HEADER_ERROR	2142	X'0000085E'
MQRC_SOURCE_LENGTH_ERROR	2143	X'0000085F'
MQRC_TARGET_LENGTH_ERROR	2144	X'00000860'
MQRC_SOURCE_BUFFER_ERROR	2145	X'00000861'
MQRC_TARGET_BUFFER_ERROR	2146	X'00000862'
MQRC_IIH_ERROR	2148	X'00000864'
MQRC_PCF_ERROR	2149	X'00000865'
MQRC_DBCS_ERROR	2150	X'00000866'
MQRC_OBJECT_NAME_ERROR	2152	X'00000868'
MQRC_OBJECT_Q_MGR_NAME_ERROR	2153	X'00000869'
MQRC_RECS_PRESENT_ERROR	2154	X'0000086A'
MQRC_OBJECT_RECORDS_ERROR	2155	X'0000086B'
MQRC_RESPONSE_RECORDS_ERROR	2156	X'0000086C'
MQRC_ASID_MISMATCH	2157	X'0000086D'

MQRC_PMO_RECORD_FLAGS_ERROR	2158	X'0000086E'
MQRC_PUT_MSG_RECORDS_ERROR	2159	X'0000086F'
MQRC_CONN_ID_IN_USE	2160	X'00000870'
MQRC_Q_MGR_QUIESCING	2161	X'00000871'
MQRC_Q_MGR_STOPPING	2162	X'00000872'
MQRC_DUPLICATE_RECOV_COORD	2163	X'00000873'
MQRC_PMO_ERROR	2173	X'0000087D'
MQRC_API_EXIT_NOT_FOUND	2182	X'00000886'
MQRC_API_EXIT_LOAD_ERROR	2183	X'00000887'
MQRC_REMOTE_Q_NAME_ERROR	2184	X'00000888'
MQRC_INCONSISTENT_PERSISTENCE	2185	X'00000889'
MQRC_GMO_ERROR	2186	X'0000088A'
MQRC_CICS_BRIDGE_RESTRICTION	2187	X'0000088B'
MQRC_STOPPED_BY_CLUSTER_EXIT	2188	X'0000088C'
MQRC_CLUSTER_RESOLUTION_ERROR	2189	X'0000088D'
MQRC_CONVERTED_STRING_TOO_BIG	2190	X'0000088E'
MQRC_TMC_ERROR	2191	X'0000088F'
MQRC_PAGESET_FULL	2192	X'00000890'
MQRC_STORAGE_MEDIUM_FULL	2192	X'00000890'
MQRC_PAGESET_ERROR	2193	X'00000891'
MQRC_NAME_NOT_VALID_FOR_TYPE	2194	X'00000892'
MQRC_UNEXPECTED_ERROR	2195	X'00000893'
MQRC_UNKNOWN_XMIT_Q	2196	X'00000894'
MQRC_UNKNOWN_DEF_XMIT_Q	2197	X'00000895'
MQRC_DEF_XMIT_Q_TYPE_ERROR	2198	X'00000896'
MQRC_DEF_XMIT_Q_USAGE_ERROR	2199	X'00000897'
MQRC_MSG_MARKED_BROWSE_CO_OP	2200	X'00000898'
MQRC_NAME_IN_USE	2201	X'00000899'
MQRC_CONNECTION_QUIESCING	2202	X'0000089A'
MQRC_CONNECTION_STOPPING	2203	X'0000089B'
MQRC_ADAPTER_NOT_AVAILABLE	2204	X'0000089C'
MQRC_MSG_ID_ERROR	2206	X'0000089E'
MQRC_CORREL_ID_ERROR	2207	X'0000089F'
MQRC_FILE_SYSTEM_ERROR	2208	X'000008A0'
MQRC_NO_MSG_LOCKED	2209	X'000008A1'
MQRC_SOAP_DOTNET_ERROR	2210	X'000008A2'
MQRC_SOAP_AXIS_ERROR	2211	X'000008A3'
MQRC_SOAP_URL_ERROR	2212	X'000008A4'
MQRC_FILE_NOT_AUDITED	2216	X'000008A8'
MQRC_CONNECTION_NOT_AUTHORIZED	2217	X'000008A9'
MQRC_MSG_TOO_BIG_FOR_CHANNEL	2218	X'000008AA'
MQRC_CALL_IN_PROGRESS	2219	X'000008AB'

MQRC_RMH_ERROR	2220	X'000008AC'
MQRC_Q_MGR_ACTIVE	2222	X'000008AE'
MQRC_Q_MGR_NOT_ACTIVE	2223	X'000008AF'
MQRC_Q_DEPTH_HIGH	2224	X'000008B0'
MQRC_Q_DEPTH_LOW	2225	X'000008B1'
MQRC_Q_SERVICE_INTERVAL_HIGH	2226	X'000008B2'
MQRC_Q_SERVICE_INTERVAL_OK	2227	X'000008B3'
MQRC_RFH_HEADER_FIELD_ERROR	2228	X'000008B4'
MQRC_RAS_PROPERTY_ERROR	2229	X'000008B5'
MQRC_UNIT_OF_WORK_NOT_STARTED	2232	X'000008B8'
MQRC_CHANNEL_AUTO_DEF_OK	2233	X'000008B9'
MQRC_CHANNEL_AUTO_DEF_ERROR	2234	X'000008BA'
MQRC_CFH_ERROR	2235	X'000008BB'
MQRC_CFIL_ERROR	2236	X'000008BC'
MQRC_CFIN_ERROR	2237	X'000008BD'
MQRC_CFSL_ERROR	2238	X'000008BE'
MQRC_CFST_ERROR	2239	X'000008BF'
MQRC_INCOMPLETE_GROUP	2241	X'000008C1'
MQRC_INCOMPLETE_MSG	2242	X'000008C2'
MQRC_INCONSISTENT_CCSIDS	2243	X'000008C3'
MQRC_INCONSISTENT_ENCODINGS	2244	X'000008C4'
MQRC_INCONSISTENT_UOW	2245	X'000008C5'
MQRC_INVALID_MSG_UNDER_CURSOR	2246	X'000008C6'
MQRC_MATCH_OPTIONS_ERROR	2247	X'000008C7'
MQRC_MDE_ERROR	2248	X'000008C8'
MQRC_MSG_FLAGS_ERROR	2249	X'000008C9'
MQRC_MSG_SEQ_NUMBER_ERROR	2250	X'000008CA'
MQRC_OFFSET_ERROR	2251	X'000008CB'
MQRC_ORIGINAL_LENGTH_ERROR	2252	X'000008CC'
MQRC_SEGMENT_LENGTH_ZERO	2253	X'000008CD'
MQRC_UOW_NOT_AVAILABLE	2255	X'000008CF'
MQRC_WRONG_GMO_VERSION	2256	X'000008D0'
MQRC_WRONG_MD_VERSION	2257	X'000008D1'
MQRC_GROUP_ID_ERROR	2258	X'000008D2'
MQRC_INCONSISTENT_BROWSE	2259	X'000008D3'
MQRC_XQH_ERROR	2260	X'000008D4'
MQRC_SRC_ENV_ERROR	2261	X'000008D5'
MQRC_SRC_NAME_ERROR	2262	X'000008D6'
MQRC_DEST_ENV_ERROR	2263	X'000008D7'
MQRC_DEST_NAME_ERROR	2264	X'000008D8'
MQRC_TM_ERROR	2265	X'000008D9'
MQRC_CLUSTER_EXIT_ERROR	2266	X'000008DA'

MQRC_CLUSTER_EXIT_LOAD_ERROR	2267	X'000008DB'
MQRC_CLUSTER_PUT_INHIBITED	2268	X'000008DC'
MQRC_CLUSTER_RESOURCE_ERROR	2269	X'000008DD'
MQRC_NO_DESTINATIONS_AVAILABLE	2270	X'000008DE'
MQRC_CONN_TAG_IN_USE	2271	X'000008DF'
MQRC_PARTIALLY_CONVERTED	2272	X'000008E0'
MQRC_CONNECTION_ERROR	2273	X'000008E1'
MQRC_OPTION_ENVIRONMENT_ERROR	2274	X'000008E2'
MQRC_CD_ERROR	2277	X'000008E5'
MQRC_CLIENT_CONN_ERROR	2278	X'000008E6'
MQRC_CHANNEL_STOPPED_BY_USER	2279	X'000008E7'
MQRC_HCONFIG_ERROR	2280	X'000008E8'
MQRC_FUNCTION_ERROR	2281	X'000008E9'
MQRC_CHANNEL_STARTED	2282	X'000008EA'
MQRC_CHANNEL_STOPPED	2283	X'000008EB'
MQRC_CHANNEL_CONV_ERROR	2284	X'000008EC'
MQRC_SERVICE_NOT_AVAILABLE	2285	X'000008ED'
MQRC_INITIALIZATION_FAILED	2286	X'000008EE'
MQRC_TERMINATION_FAILED	2287	X'000008EF'
MQRC_UNKNOWN_Q_NAME	2288	X'000008F0'
MQRC_SERVICE_ERROR	2289	X'000008F1'
MQRC_Q_ALREADY_EXISTS	2290	X'000008F2'
MQRC_USER_ID_NOT_AVAILABLE	2291	X'000008F3'
MQRC_UNKNOWN_ENTITY	2292	X'000008F4'
MQRC_UNKNOWN_AUTH_ENTITY	2293	X'000008F5'
MQRC_UNKNOWN_REF_OBJECT	2294	X'000008F6'
MQRC_CHANNEL_ACTIVATED	2295	X'000008F7'
MQRC_CHANNEL_NOT_ACTIVATED	2296	X'000008F8'
MQRC_UOW_CANCELED	2297	X'000008F9'
MQRC_FUNCTION_NOT_SUPPORTED	2298	X'000008FA'
MQRC_SELECTOR_TYPE_ERROR	2299	X'000008FB'
MQRC_COMMAND_TYPE_ERROR	2300	X'000008FC'
MQRC_MULTIPLE_INSTANCE_ERROR	2301	X'000008FD'
MQRC_SYSTEM_ITEM_NOT_ALTERABLE	2302	X'000008FE'
MQRC_BAG_CONVERSION_ERROR	2303	X'000008FF'
MQRC_SELECTOR_OUT_OF_RANGE	2304	X'00000900'
MQRC_SELECTOR_NOT_UNIQUE	2305	X'00000901'
MQRC_INDEX_NOT_PRESENT	2306	X'00000902'
MQRC_STRING_ERROR	2307	X'00000903'
MQRC_ENCODING_NOT_SUPPORTED	2308	X'00000904'
MQRC_SELECTOR_NOT_PRESENT	2309	X'00000905'
MQRC_OUT_SELECTOR_ERROR	2310	X'00000906'

MQRC_STRING_TRUNCATED	2311	X'00000907'
MQRC_SELECTOR_WRONG_TYPE	2312	X'00000908'
MQRC_INCONSISTENT_ITEM_TYPE	2313	X'00000909'
MQRC_INDEX_ERROR	2314	X'0000090A'
MQRC_SYSTEM_BAG_NOT_ALTERABLE	2315	X'0000090B'
MQRC_ITEM_COUNT_ERROR	2316	X'0000090C'
MQRC_FORMAT_NOT_SUPPORTED	2317	X'0000090D'
MQRC_SELECTOR_NOT_SUPPORTED	2318	X'0000090E'
MQRC_ITEM_VALUE_ERROR	2319	X'0000090F'
MQRC_HBAG_ERROR	2320	X'00000910'
MQRC_PARAMETER_MISSING	2321	X'00000911'
MQRC_CMD_SERVER_NOT_AVAILABLE	2322	X'00000912'
MQRC_STRING_LENGTH_ERROR	2323	X'00000913'
MQRC_INQUIRY_COMMAND_ERROR	2324	X'00000914'
MQRC_NESTED_BAG_NOT_SUPPORTED	2325	X'00000915'
MQRC_BAG_WRONG_TYPE	2326	X'00000916'
MQRC_ITEM_TYPE_ERROR	2327	X'00000917'
MQRC_SYSTEM_BAG_NOT_DELETABLE	2328	X'00000918'
MQRC_SYSTEM_ITEM_NOT_DELETABLE	2329	X'00000919'
MQRC_CODED_CHAR_SET_ID_ERROR	2330	X'0000091A'
MQRC_MSG_TOKEN_ERROR	2331	X'0000091B'
MQRC_MISSING_WIH	2332	X'0000091C'
MQRC_WIH_ERROR	2333	X'0000091D'
MQRC_RFH_ERROR	2334	X'0000091E'
MQRC_RFH_STRING_ERROR	2335	X'0000091F'
MQRC_RFH_COMMAND_ERROR	2336	X'00000920'
MQRC_RFH_PARM_ERROR	2337	X'00000921'
MQRC_RFH_DUPLICATE_PARM	2338	X'00000922'
MQRC_RFH_PARM_MISSING	2339	X'00000923'
MQRC_CHAR_CONVERSION_ERROR	2340	X'00000924'
MQRC_UCS2_CONVERSION_ERROR	2341	X'00000925'
MQRC_DB2_NOT_AVAILABLE	2342	X'00000926'
MQRC_OBJECT_NOT_UNIQUE	2343	X'00000927'
MQRC_CONN_TAG_NOT_RELEASED	2344	X'00000928'
MQRC_CF_NOT_AVAILABLE	2345	X'00000929'
MQRC_CF_STRUC_IN_USE	2346	X'0000092A'
MQRC_CF_STRUC_LIST_HDR_IN_USE	2347	X'0000092B'
MQRC_CF_STRUC_AUTH_FAILED	2348	X'0000092C'
MQRC_CF_STRUC_ERROR	2349	X'0000092D'
MQRC_CONN_TAG_NOT_USABLE	2350	X'0000092E'
MQRC_GLOBAL_UOW_CONFLICT	2351	X'0000092F'
MQRC_LOCAL_UOW_CONFLICT	2352	X'00000930'

MQRC_HANDLE_IN_USE_FOR_UOW	2353	X'00000931'
MQRC_UOW_ENLISTMENT_ERROR	2354	X'00000932'
MQRC_UOW_MIX_NOT_SUPPORTED	2355	X'00000933'
MQRC_WXP_ERROR	2356	X'00000934'
MQRC_CURRENT_RECORD_ERROR	2357	X'00000935'
MQRC_NEXT_OFFSET_ERROR	2358	X'00000936'
MQRC_NO_RECORD_AVAILABLE	2359	X'00000937'
MQRC_OBJECT_LEVEL_INCOMPATIBLE	2360	X'00000938'
MQRC_NEXT_RECORD_ERROR	2361	X'00000939'
MQRC_BACKOUT_THRESHOLD_REACHED	2362	X'0000093A'
MQRC_MSG_NOT_MATCHED	2363	X'0000093B'
MQRC_JMS_FORMAT_ERROR	2364	X'0000093C'
MQRC_SEGMENTS_NOT_SUPPORTED	2365	X'0000093D'
MQRC_WRONG_CF_LEVEL	2366	X'0000093E'
MQRC_CONFIG_CREATE_OBJECT	2367	X'0000093F'
MQRC_CONFIG_CHANGE_OBJECT	2368	X'00000940'
MQRC_CONFIG_DELETE_OBJECT	2369	X'00000941'
MQRC_CONFIG_REFRESH_OBJECT	2370	X'00000942'
MQRC_CHANNEL_SSL_ERROR	2371	X'00000943'
MQRC_PARTICIPANT_NOT_DEFINED	2372	X'00000944'
MQRC_CF_STRUC_FAILED	2373	X'00000945'
MQRC_API_EXIT_ERROR	2374	X'00000946'
MQRC_API_EXIT_INIT_ERROR	2375	X'00000947'
MQRC_API_EXIT_TERM_ERROR	2376	X'00000948'
MQRC_EXIT_REASON_ERROR	2377	X'00000949'
MQRC_RESERVED_VALUE_ERROR	2378	X'0000094A'
MQRC_NO_DATA_AVAILABLE	2379	X'0000094B'
MQRC_SCO_ERROR	2380	X'0000094C'
MQRC_KEY_REPOSITORY_ERROR	2381	X'0000094D'
MQRC_CRYPTO_HARDWARE_ERROR	2382	X'0000094E'
MQRC_AUTH_INFO_REC_COUNT_ERROR	2383	X'0000094F'
MQRC_AUTH_INFO_REC_ERROR	2384	X'00000950'
MQRC_AIR_ERROR	2385	X'00000951'
MQRC_AUTH_INFO_TYPE_ERROR	2386	X'00000952'
MQRC_AUTH_INFO_CONN_NAME_ERROR	2387	X'00000953'
MQRC_LDAP_USER_NAME_ERROR	2388	X'00000954'
MQRC_LDAP_USER_NAME_LENGTH_ERR	2389	X'00000955'
MQRC_LDAP_PASSWORD_ERROR	2390	X'00000956'
MQRC_SSL_ALREADY_INITIALIZED	2391	X'00000957'
MQRC_SSL_CONFIG_ERROR	2392	X'00000958'
MQRC_SSL_INITIALIZATION_ERROR	2393	X'00000959'
MQRC_Q_INDEX_TYPE_ERROR	2394	X'0000095A'

MQRC_CFBS_ERROR	2395	X'0000095B'
MQRC_SSL_NOT_ALLOWED	2396	X'0000095C'
MQRC_JSSE_ERROR	2397	X'0000095D'
MQRC_SSL_PEER_NAME_MISMATCH	2398	X'0000095E'
MQRC_SSL_PEER_NAME_ERROR	2399	X'0000095F'
MQRC_UNSUPPORTED_CIPHER_SUITE	2400	X'00000960'
MQRC_SSL_CERTIFICATE_REVOKED	2401	X'00000961'
MQRC_SSL_CERT_STORE_ERROR	2402	X'00000962'
MQRC_CLIENT_EXIT_LOAD_ERROR	2406	X'00000966'
MQRC_CLIENT_EXIT_ERROR	2407	X'00000967'
MQRC_UOW_COMMITTED	2408	X'00000968'
MQRC_SSL_KEY_RESET_ERROR	2409	X'00000969'
MQRC_UNKNOWN_COMPONENT_NAME	2410	X'0000096A'
MQRC_LOGGER_STATUS	2411	X'0000096B'
MQRC_COMMAND_MQSC	2412	X'0000096C'
MQRC_COMMAND_PCF	2413	X'0000096D'
MQRC_CFIF_ERROR	2414	X'0000096E'
MQRC_CFSF_ERROR	2415	X'0000096F'
MQRC_CFGR_ERROR	2416	X'00000970'
MQRC_MSG_NOT_ALLOWED_IN_GROUP	2417	X'00000971'
MQRC_FILTER_OPERATOR_ERROR	2418	X'00000972'
MQRC_NESTED_SELECTOR_ERROR	2419	X'00000973'
MQRC_EPH_ERROR	2420	X'00000974'
MQRC_RFH_FORMAT_ERROR	2421	X'00000975'
MQRC_CFBF_ERROR	2422	X'00000976'
MQRC_CLIENT_CHANNEL_CONFLICT	2423	X'00000977'
MQRC_SD_ERROR	2424	X'00000978'
MQRC_TOPIC_STRING_ERROR	2425	X'00000979'
MQRC_STS_ERROR	2426	X'0000097A'
MQRC_NO_SUBSCRIPTION	2428	X'0000097C'
MQRC_SUBSCRIPTION_IN_USE	2429	X'0000097D'
MQRC_STAT_TYPE_ERROR	2430	X'0000097E'
MQRC_SUB_USER_DATA_ERROR	2431	X'0000097F'
MQRC_SUB_ALREADY_EXISTS	2432	X'00000980'
MQRC_IDENTITY_MISMATCH	2434	X'00000982'
MQRC_ALTER_SUB_ERROR	2435	X'00000983'
MQRC_DURABILITY_NOT_ALLOWED	2436	X'00000984'
MQRC_NO_RETAINED_MSG	2437	X'00000985'
MQRC_SRO_ERROR	2438	X'00000986'
MQRC_SUB_NAME_ERROR	2440	X'00000988'
MQRC_OBJECT_STRING_ERROR	2441	X'00000989'
MQRC_PROPERTY_NAME_ERROR	2442	X'0000098A'

MQRC_SEGMENTATION_NOT_ALLOWED	2443	X'0000098B'
MQRC_CBD_ERROR	2444	X'0000098C'
MQRC_CTLO_ERROR	2445	X'0000098D'
MQRC_NO_CALLBACKS_ACTIVE	2446	X'0000098E'
MQRC_CALLBACK_NOT_REGISTERED	2448	X'00000990'
MQRC_OPTIONS_CHANGED	2457	X'00000999'
MQRC_READ_AHEAD_MSGS	2458	X'0000099A'
MQRC_SELECTOR_SYNTAX_ERROR	2459	X'0000099B'
MQRC_HMSG_ERROR	2460	X'0000099C'
MQRC_CMHO_ERROR	2461	X'0000099D'
MQRC_DMHO_ERROR	2462	X'0000099E'
MQRC_SMPO_ERROR	2463	X'0000099F'
MQRC_IMPO_ERROR	2464	X'000009A0'
MQRC_PROPERTY_NAME_TOO_BIG	2465	X'000009A1'
MQRC_PROP_VALUE_NOT_CONVERTED	2466	X'000009A2'
MQRC_PROP_TYPE_NOT_SUPPORTED	2467	X'000009A3'
MQRC_PROPERTY_VALUE_TOO_BIG	2469	X'000009A5'
MQRC_PROP_CONV_NOT_SUPPORTED	2470	X'000009A6'
MQRC_PROPERTY_NOT_AVAILABLE	2471	X'000009A7'
MQRC_PROP_NUMBER_FORMAT_ERROR	2472	X'000009A8'
MQRC_PROPERTY_TYPE_ERROR	2473	X'000009A9'
MQRC_PROPERTIES_TOO_BIG	2478	X'000009AE'
MQRC_PUT_NOT_RETAINED	2479	X'000009AF'
MQRC_ALIAS_TARGTYPE_CHANGED	2480	X'000009B0'
MQRC_DMPO_ERROR	2481	X'000009B1'
MQRC_PD_ERROR	2482	X'000009B2'
MQRC_CALLBACK_TYPE_ERROR	2483	X'000009B3'
MQRC_CBD_OPTIONS_ERROR	2484	X'000009B4'
MQRC_MAX_MSG_LENGTH_ERROR	2485	X'000009B5'
MQRC_CALLBACK_ROUTINE_ERROR	2486	X'000009B6'
MQRC_CALLBACK_LINK_ERROR	2487	X'000009B7'
MQRC_OPERATION_ERROR	2488	X'000009B8'
MQRC_BMHO_ERROR	2489	X'000009B9'
MQRC_UNSUPPORTED_PROPERTY	2490	X'000009BA'
MQRC_PROP_NAME_NOT_CONVERTED	2492	X'000009BC'
MQRC_GET_ENABLED	2494	X'000009BE'
MQRC_MODULE_NOT_FOUND	2495	X'000009BF'
MQRC_MODULE_INVALID	2496	X'000009C0'
MQRC_MODULE_ENTRY_NOT_FOUND	2497	X'000009C1'
MQRC_MIXED_CONTENT_NOT_ALLOWED	2498	X'000009C2'
MQRC_MSG_HANDLE_IN_USE	2499	X'000009C3'
MQRC_HCONN_ASYNC_ACTIVE	2500	X'000009C4'

MQRC_MHBO_ERROR	2501	X'000009C5'
MQRC_PUBLICATION_FAILURE	2502	X'000009C6'
MQRC_SUB_INHIBITED	2503	X'000009C7'
MQRC_SELECTOR_ALWAYS_FALSE	2504	X'000009C8'
MQRC_XEPO_ERROR	2507	X'000009CB'
MQRC_DURABILITY_NOT_ALTERABLE	2509	X'000009CD'
MQRC_TOPIC_NOT_ALTERABLE	2510	X'000009CE'
MQRC_SUBLEVEL_NOT_ALTERABLE	2512	X'000009D0'
MQRC_PROPERTY_NAME_LENGTH_ERR	2513	X'000009D1'
MQRC_DUPLICATE_GROUP_SUB	2514	X'000009D2'
MQRC_GROUPING_NOT_ALTERABLE	2515	X'000009D3'
MQRC_SELECTOR_INVALID_FOR_TYPE	2516	X'000009D4'
MQRC_HOBJ_QUIESCED	2517	X'000009D5'
MQRC_HOBJ_QUIESCED_NO_MSGS	2518	X'000009D6'
MQRC_SELECTION_STRING_ERROR	2519	X'000009D7'
MQRC_RES_OBJECT_STRING_ERROR	2520	X'000009D8'
MQRC_CONNECTION_SUSPENDED	2521	X'000009D9'
MQRC_INVALID_DESTINATION	2522	X'000009DA'
MQRC_INVALID_SUBSCRIPTION	2523	X'000009DB'
MQRC_SELECTOR_NOT_ALTERABLE	2524	X'000009DC'
MQRC_RETAINED_MSG_Q_ERROR	2525	X'000009DD'
MQRC_RETAINED_NOT_DELIVERED	2526	X'000009DE'
MQRC_RFH_RESTRICTED_FORMAT_ERR	2527	X'000009DF'
MQRC_CONNECTION_STOPPED	2528	X'000009E0'
MQRC_ASYNC_UOW_CONFLICT	2529	X'000009E1'
MQRC_ASYNC_XA_CONFLICT	2530	X'000009E2'
MQRC_PUBSUB_INHIBITED	2531	X'000009E3'
MQRC_MSG_HANDLE_COPY_FAILURE	2532	X'000009E4'
MQRC_DEST_CLASS_NOT_ALTERABLE	2533	X'000009E5'
MQRC_OPERATION_NOT_ALLOWED	2534	X'000009E6'
MQRC_ACTION_ERROR	2535	X'000009E7'
MQRC_CHANNEL_NOT_AVAILABLE	2537	X'000009E9'
MQRC_HOST_NOT_AVAILABLE	2538	X'000009EA'
MQRC_CHANNEL_CONFIG_ERROR	2539	X'000009EB'
MQRC_UNKNOWN_CHANNEL_NAME	2540	X'000009EC'
MQRC_LOOPING_PUBLICATION	2541	X'000009ED'
MQRC_ALREADY_JOINED	2542	X'000009EE'
MQRC_CHANNEL_SSL_WARNING	2552	X'000009F8'
MQRC_OCSP_URL_ERROR	2553	X'000009F9'
MQRC_CIPHER_SPEC_NOT_SUITE_B	2591	X'00000A1F'
MQRC_SUITE_B_ERROR	2592	X'00000A20'
MQRC_PASSWORD_PROTECTION_ERROR	2594	X'00000A22'

MQRC_REOPEN_EXCL_INPUT_ERROR	6100	X'000017D4'
MQRC_REOPEN_INQUIRE_ERROR	6101	X'000017D5'
MQRC_REOPEN_SAVED_CONTEXT_ERR	6102	X'000017D6'
MQRC_REOPEN_TEMPORARY_Q_ERROR	6103	X'000017D7'
MQRC_ATTRIBUTE_LOCKED	6104	X'000017D8'
MQRC_CURSOR_NOT_VALID	6105	X'000017D9'
MQRC_ENCODING_ERROR	6106	X'000017DA'
MQRC_STRUC_ID_ERROR	6107	X'000017DB'
MQRC_NULL_POINTER	6108	X'000017DC'
MQRC_NO_CONNECTION_REFERENCE	6109	X'000017DD'
MQRC_NO_BUFFER	6110	X'000017DE'
MQRC_BINARY_DATA_LENGTH_ERROR	6111	X'000017DF'
MQRC_BUFFER_NOT_AUTOMATIC	6112	X'000017E0'
MQRC_INSUFFICIENT_BUFFER	6113	X'000017E1'
MQRC_INSUFFICIENT_DATA	6114	X'000017E2'
MQRC_DATA_TRUNCATED	6115	X'000017E3'
MQRC_ZERO_LENGTH	6116	X'000017E4'
MQRC_NEGATIVE_LENGTH	6117	X'000017E5'
MQRC_NEGATIVE_OFFSET	6118	X'000017E6'
MQRC_INCONSISTENT_FORMAT	6119	X'000017E7'
MQRC_INCONSISTENT_OBJECT_STATE	6120	X'000017E8'
MQRC_CONTEXT_OBJECT_NOT_VALID	6121	X'000017E9'
MQRC_CONTEXT_OPEN_ERROR	6122	X'000017EA'
MQRC_STRUC_LENGTH_ERROR	6123	X'000017EB'
MQRC_NOT_CONNECTED	6124	X'000017EC'
MQRC_NOT_OPEN	6125	X'000017ED'
MQRC_DISTRIBUTION_LIST_EMPTY	6126	X'000017EE'
MQRC_INCONSISTENT_OPEN_OPTIONS	6127	X'000017EF'
MQRC_WRONG_VERSION	6128	X'000017F0'
MQRC_REFERENCE_ERROR	6129	X'000017F1'

MQRCCF_* (Command format header reason codes):

MQRCCF_CFH_TYPE_ERROR	3001	X'00000BB9'
MQRCCF_CFH_LENGTH_ERROR	3002	X'00000BBA'
MQRCCF_CFH_VERSION_ERROR	3003	X'00000BBB'
MQRCCF_CFH_MSG_SEQ_NUMBER_ERR	3004	X'00000BBC'
MQRCCF_CFH_CONTROL_ERROR	3005	X'00000BBD'
MQRCCF_CFH_PARM_COUNT_ERROR	3006	X'00000BBE'
MQRCCF_CFH_COMMAND_ERROR	3007	X'00000BBF'
MQRCCF_COMMAND_FAILED	3008	X'00000BC0'
MQRCCF_CFIN_LENGTH_ERROR	3009	X'00000BC1'

MQRCCF_CFST_LENGTH_ERROR	3010	X'00000BC2'
MQRCCF_CFST_STRING_LENGTH_ERR	3011	X'00000BC3'
MQRCCF_FORCE_VALUE_ERROR	3012	X'00000BC4'
MQRCCF_STRUCTURE_TYPE_ERROR	3013	X'00000BC5'
MQRCCF_CFIN_PARM_ID_ERROR	3014	X'00000BC6'
MQRCCF_CFST_PARM_ID_ERROR	3015	X'00000BC7'
MQRCCF_MSG_LENGTH_ERROR	3016	X'00000BC8'
MQRCCF_CFIN_DUPLICATE_PARM	3017	X'00000BC9'
MQRCCF_CFST_DUPLICATE_PARM	3018	X'00000BCA'
MQRCCF_PARM_COUNT_TOO_SMALL	3019	X'00000BCB'
MQRCCF_PARM_COUNT_TOO_BIG	3020	X'00000BCC'
MQRCCF_Q_ALREADY_IN_CELL	3021	X'00000BCD'
MQRCCF_Q_TYPE_ERROR	3022	X'00000BCE'
MQRCCF_MD_FORMAT_ERROR	3023	X'00000BCF'
MQRCCF_CFSL_LENGTH_ERROR	3024	X'00000BD0'
MQRCCF_REPLACE_VALUE_ERROR	3025	X'00000BD1'
MQRCCF_CFIL_DUPLICATE_VALUE	3026	X'00000BD2'
MQRCCF_CFIL_COUNT_ERROR	3027	X'00000BD3'
MQRCCF_CFIL_LENGTH_ERROR	3028	X'00000BD4'
MQRCCF_QUIESCE_VALUE_ERROR	3029	X'00000BD5'
MQRCCF_MODE_VALUE_ERROR	3029	X'00000BD5'
MQRCCF_MSG_SEQ_NUMBER_ERROR	3030	X'00000BD6'
MQRCCF_PING_DATA_COUNT_ERROR	3031	X'00000BD7'
MQRCCF_PING_DATA_COMPARE_ERROR	3032	X'00000BD8'
MQRCCF_CFSL_PARM_ID_ERROR	3033	X'00000BD9'
MQRCCF_CHANNEL_TYPE_ERROR	3034	X'00000BDA'
MQRCCF_PARM_SEQUENCE_ERROR	3035	X'00000BDB'
MQRCCF_XMIT_PROTOCOL_TYPE_ERR	3036	X'00000BDC'
MQRCCF_BATCH_SIZE_ERROR	3037	X'00000BDD'
MQRCCF_DISC_INT_ERROR	3038	X'00000BDE'
MQRCCF_SHORT_RETRY_ERROR	3039	X'00000BDF'
MQRCCF_SHORT_TIMER_ERROR	3040	X'00000BE0'
MQRCCF_LONG_RETRY_ERROR	3041	X'00000BE1'
MQRCCF_LONG_TIMER_ERROR	3042	X'00000BE2'
MQRCCF_SEQ_NUMBER_WRAP_ERROR	3043	X'00000BE3'
MQRCCF_MAX_MSG_LENGTH_ERROR	3044	X'00000BE4'
MQRCCF_PUT_AUTH_ERROR	3045	X'00000BE5'
MQRCCF_PURGE_VALUE_ERROR	3046	X'00000BE6'
MQRCCF_CFIL_PARM_ID_ERROR	3047	X'00000BE7'
MQRCCF_MSG_TRUNCATED	3048	X'00000BE8'
MQRCCF_CCSID_ERROR	3049	X'00000BE9'
MQRCCF_ENCODING_ERROR	3050	X'00000BEA'

MQRCCF_QUEUES_VALUE_ERROR	3051	X'00000BEB'
MQRCCF_DATA_CONV_VALUE_ERROR	3052	X'00000BEC'
MQRCCF_INDOUBT_VALUE_ERROR	3053	X'00000BED'
MQRCCF_ESCAPE_TYPE_ERROR	3054	X'00000BEE'
MQRCCF_REPOS_VALUE_ERROR	3055	X'00000BEF'
MQRCCF_CHANNEL_TABLE_ERROR	3062	X'00000BF6'
MQRCCF_MCA_TYPE_ERROR	3063	X'00000BF7'
MQRCCF_CHL_INST_TYPE_ERROR	3064	X'00000BF8'
MQRCCF_CHL_STATUS_NOT_FOUND	3065	X'00000BF9'
MQRCCF_CFSL_DUPLICATE_PARM	3066	X'00000BFA'
MQRCCF_CFSL_TOTAL_LENGTH_ERROR	3067	X'00000BFB'
MQRCCF_CFSL_COUNT_ERROR	3068	X'00000BFC'
MQRCCF_CFSL_STRING_LENGTH_ERR	3069	X'00000BFD'
MQRCCF_BROKER_DELETED	3070	X'00000BFE'
MQRCCF_STREAM_ERROR	3071	X'00000BFF'
MQRCCF_TOPIC_ERROR	3072	X'00000C00'
MQRCCF_NOT_REGISTERED	3073	X'00000C01'
MQRCCF_Q_MGR_NAME_ERROR	3074	X'00000C02'
MQRCCF_INCORRECT_STREAM	3075	X'00000C03'
MQRCCF_Q_NAME_ERROR	3076	X'00000C04'
MQRCCF_NO_RETAINED_MSG	3077	X'00000C05'
MQRCCF_DUPLICATE_IDENTITY	3078	X'00000C06'
MQRCCF_INCORRECT_Q	3079	X'00000C07'
MQRCCF_CORREL_ID_ERROR	3080	X'00000C08'
MQRCCF_NOT_AUTHORIZED	3081	X'00000C09'
MQRCCF_UNKNOWN_STREAM	3082	X'00000C0A'
MQRCCF_REG_OPTIONS_ERROR	3083	X'00000C0B'
MQRCCF_PUB_OPTIONS_ERROR	3084	X'00000C0C'
MQRCCF_UNKNOWN_BROKER	3085	X'00000C0D'
MQRCCF_Q_MGR_CCSID_ERROR	3086	X'00000C0E'
MQRCCF_DEL_OPTIONS_ERROR	3087	X'00000C0F'
MQRCCF_CLUSTER_NAME_CONFLICT	3088	X'00000C10'
MQRCCF_REPOS_NAME_CONFLICT	3089	X'00000C11'
MQRCCF_CLUSTER_Q_USAGE_ERROR	3090	X'00000C12'
MQRCCF_ACTION_VALUE_ERROR	3091	X'00000C13'
MQRCCF_COMMS_LIBRARY_ERROR	3092	X'00000C14'
MQRCCF_NETBIOS_NAME_ERROR	3093	X'00000C15'
MQRCCF_BROKER_COMMAND_FAILED	3094	X'00000C16'
MQRCCF_CFST_CONFLICTING_PARM	3095	X'00000C17'
MQRCCF_PATH_NOT_VALID	3096	X'00000C18'
MQRCCF_PARM_SYNTAX_ERROR	3097	X'00000C19'
MQRCCF_PWD_LENGTH_ERROR	3098	X'00000C1A'

MQRCCF_FILTER_ERROR	3150	X'00000C4E'
MQRCCF_WRONG_USER	3151	X'00000C4F'
MQRCCF_DUPLICATE_SUBSCRIPTION	3152	X'00000C50'
MQRCCF_SUB_NAME_ERROR	3153	X'00000C51'
MQRCCF_SUB_IDENTITY_ERROR	3154	X'00000C52'
MQRCCF_SUBSCRIPTION_IN_USE	3155	X'00000C53'
MQRCCF_SUBSCRIPTION_LOCKED	3156	X'00000C54'
MQRCCF_ALREADY_JOINED	3157	X'00000C55'
MQRCCF_OBJECT_IN_USE	3160	X'00000C58'
MQRCCF_UNKNOWN_FILE_NAME	3161	X'00000C59'
MQRCCF_FILE_NOT_AVAILABLE	3162	X'00000C5A'
MQRCCF_DISC_RETRY_ERROR	3163	X'00000C5B'
MQRCCF_ALLOC_RETRY_ERROR	3164	X'00000C5C'
MQRCCF_ALLOC_SLOW_TIMER_ERROR	3165	X'00000C5D'
MQRCCF_ALLOC_FAST_TIMER_ERROR	3166	X'00000C5E'
MQRCCF_PORT_NUMBER_ERROR	3167	X'00000C5F'
MQRCCF_CHL_SYSTEM_NOT_ACTIVE	3168	X'00000C60'
MQRCCF_ENTITY_NAME_MISSING	3169	X'00000C61'
MQRCCF_PROFILE_NAME_ERROR	3170	X'00000C62'
MQRCCF_AUTH_VALUE_ERROR	3171	X'00000C63'
MQRCCF_AUTH_VALUE_MISSING	3172	X'00000C64'
MQRCCF_OBJECT_TYPE_MISSING	3173	X'00000C65'
MQRCCF_CONNECTION_ID_ERROR	3174	X'00000C66'
MQRCCF_LOG_TYPE_ERROR	3175	X'00000C67'
MQRCCF_PROGRAM_NOT_AVAILABLE	3176	X'00000C68'
MQRCCF_PROGRAM_AUTH_FAILED	3177	X'00000C69'
MQRCCF_NONE_FOUND	3200	X'00000C80'
MQRCCF_SECURITY_SWITCH_OFF	3201	X'00000C81'
MQRCCF_SECURITY_REFRESH_FAILED	3202	X'00000C82'
MQRCCF_PARM_CONFLICT	3203	X'00000C83'
MQRCCF_COMMAND_INHIBITED	3204	X'00000C84'
MQRCCF_OBJECT_BEING_DELETED	3205	X'00000C85'
MQRCCF_STORAGE_CLASS_IN_USE	3207	X'00000C87'
MQRCCF_OBJECT_NAME_RESTRICTED	3208	X'00000C88'
MQRCCF_OBJECT_LIMIT_EXCEEDED	3209	X'00000C89'
MQRCCF_OBJECT_OPEN_FORCE	3210	X'00000C8A'
MQRCCF_DISPOSITION_CONFLICT	3211	X'00000C8B'
MQRCCF_Q_MGR_NOT_IN_QSG	3212	X'00000C8C'
MQRCCF_ATTR_VALUE_FIXED	3213	X'00000C8D'
MQRCCF_NAMELIST_ERROR	3215	X'00000C8F'
MQRCCF_NO_CHANNEL_INITIATOR	3217	X'00000C91'
MQRCCF_CHANNEL_INITIATOR_ERROR	3218	X'00000C92'

MQRCCF_COMMAND_LEVEL_CONFLICT	3222	X'00000C96'
MQRCCF_Q_ATTR_CONFLICT	3223	X'00000C97'
MQRCCF_EVENTS_DISABLED	3224	X'00000C98'
MQRCCF_COMMAND_SCOPE_ERROR	3225	X'00000C99'
MQRCCF_COMMAND_REPLY_ERROR	3226	X'00000C9A'
MQRCCF_FUNCTION_RESTRICTED	3227	X'00000C9B'
MQRCCF_PARM_MISSING	3228	X'00000C9C'
MQRCCF_PARM_VALUE_ERROR	3229	X'00000C9D'
MQRCCF_COMMAND_LENGTH_ERROR	3230	X'00000C9E'
MQRCCF_COMMAND_ORIGIN_ERROR	3231	X'00000C9F'
MQRCCF_LISTENER_CONFLICT	3232	X'00000CA0'
MQRCCF_LISTENER_STARTED	3233	X'00000CA1'
MQRCCF_LISTENER_STOPPED	3234	X'00000CA2'
MQRCCF_CHANNEL_ERROR	3235	X'00000CA3'
MQRCCF_CF_STRUC_ERROR	3236	X'00000CA4'
MQRCCF_UNKNOWN_USER_ID	3237	X'00000CA5'
MQRCCF_UNEXPECTED_ERROR	3238	X'00000CA6'
MQRCCF_NO_XCF_PARTNER	3239	X'00000CA7'
MQRCCF_CFGR_PARM_ID_ERROR	3240	X'00000CA8'
MQRCCF_CFIF_LENGTH_ERROR	3241	X'00000CA9'
MQRCCF_CFIF_OPERATOR_ERROR	3242	X'00000CAA'
MQRCCF_CFIF_PARM_ID_ERROR	3243	X'00000CAB'
MQRCCF_CFSF_FILTER_VAL_LEN_ERR	3244	X'00000CAC'
MQRCCF_CFSF_LENGTH_ERROR	3245	X'00000CAD'
MQRCCF_CFSF_OPERATOR_ERROR	3246	X'00000CAE'
MQRCCF_CFSF_PARM_ID_ERROR	3247	X'00000CAF'
MQRCCF_TOO_MANY_FILTERS	3248	X'00000CB0'
MQRCCF_LISTENER_RUNNING	3249	X'00000CB1'
MQRCCF_LSTR_STATUS_NOT_FOUND	3250	X'00000CB2'
MQRCCF_SERVICE_RUNNING	3251	X'00000CB3'
MQRCCF_SERV_STATUS_NOT_FOUND	3252	X'00000CB4'
MQRCCF_SERVICE_STOPPED	3253	X'00000CB5'
MQRCCF_CFBS_DUPLICATE_PARM	3254	X'00000CB6'
MQRCCF_CFBS_LENGTH_ERROR	3255	X'00000CB7'
MQRCCF_CFBS_PARM_ID_ERROR	3256	X'00000CB8'
MQRCCF_CFBS_STRING_LENGTH_ERR	3257	X'00000CB9'
MQRCCF_CFGR_LENGTH_ERROR	3258	X'00000CBA'
MQRCCF_CFGR_PARM_COUNT_ERROR	3259	X'00000CBB'
MQRCCF_CONN_NOT_STOPPED	3260	X'00000CBC'
MQRCCF_SERVICE_REQUEST_PENDING	3261	X'00000CBD'
MQRCCF_NO_START_CMD	3262	X'00000CBE'
MQRCCF_NO_STOP_CMD	3263	X'00000CBF'

MQRCCF_CFBF_LENGTH_ERROR	3264	X'00000CC0'
MQRCCF_CFBF_PARM_ID_ERROR	3265	X'00000CC1'
MQRCCF_CFBF_OPERATOR_ERROR	3266	X'00000CC2'
MQRCCF_CFBF_FILTER_VAL_LEN_ERR	3267	X'00000CC3'
MQRCCF_LISTENER_STILL_ACTIVE	3268	X'00000CC4'
MQRCCF_DEF_XMIT_Q_CLUS_ERROR	3269	X'0000CC5'
MQRCCF_TOPICSTR_ALREADY_EXISTS	3300	X'00000CE4'
MQRCCF_SHARING_CONVS_ERROR	3301	X'00000CE5'
MQRCCF_SHARING_CONVS_TYPE	3302	X'00000CE6'
MQRCCF_SECURITY_CASE_CONFLICT	3303	X'00000CE7'
MQRCCF_TOPIC_TYPE_ERROR	3305	X'00000CE9'
MQRCCF_MAX_INSTANCES_ERROR	3306	X'00000CEA'
MQRCCF_MAX_INSTS_PER_CLNT_ERR	3307	X'00000CEB'
MQRCCF_TOPIC_STRING_NOT_FOUND	3308	X'00000CEC'
MQRCCF_SUBSCRIPTION_POINT_ERR	3309	X'00000CED'
MQRCCF_SUB_ALREADY_EXISTS	3311	X'00000CEF'
MQRCCF_UNKNOWN_OBJECT_NAME	3312	X'00000CF0'
MQRCCF_REMOTE_Q_NAME_ERROR	3313	X'00000CF1'
MQRCCF_DURABILITY_NOT_ALLOWED	3314	X'00000CF2'
MQRCCF_HOBJ_ERROR	3315	X'00000CF3'
MQRCCF_DEST_NAME_ERROR	3316	X'00000CF4'
MQRCCF_INVALID_DESTINATION	3317	X'00000CF5'
MQRCCF_PUBSUB_INHIBITED	3318	X'00000CF6'
MQRCCF_CHLAUTH_TYPE_ERROR	3326	X'00000CFE'
MQRCCF_CHLAUTH_ACTION_ERROR	3327	X'00000CFF'
MQRCCF_CHLAUTH_USERSRC_ERROR	3335	X'00000D07'
MQRCCF_WRONG_CHLAUTH_TYPE	3336	X'00000D08'
MQRCCF_CHLAUTH_ALREADY_EXISTS	3337	X'00000D09'
MQRCCF_CHLAUTH_NOT_FOUND	3338	X'00000D0A'
MQRCCF_WRONG_CHLAUTH_ACTION	3339	X'00000D0B'
MQRCCF_WRONG_CHLAUTH_USERSRC	3340	X'00000D0C'
MQRCCF_CHLAUTH_WARN_ERROR	3341	X'00000D0D'
MQRCCF_WRONG_CHLAUTH_MATCH	3342	X'00000D0E'
MQRCCF_IPADDR_RANGE_CONFLICT	3343	X'00000D0F'
MQRCCF_CHLAUTH_MAX_EXCEEDED	3344	X'00000D10'
MQRCCF_IPADDR_ERROR	3345	X'00000D11'
MQRCCF_IPADDR_RANGE_ERROR	3346	X'00000D12'
MQRCCF_PROFILE_NAME_MISSING	3347	X'00000D13'
MQRCCF_CHLAUTH_CLNTUSER_ERROR	3348	X'00000D14'
MQRCCF_CHLAUTH_NAME_ERROR	3349	X'00000D15'
MQRCCF_SUITE_B_ERROR	3353	X'00000D19'
MQRCCF_PSCLUS_DISABLED_TOPDEF	3359	X'00000D1F'

MQRCCF_PSCLUS_TOPIC_EXISTS	3360	X'00000D20'
MQRCCF_INVALID_PROTOCOL	3365	X'00000D25'
V 9.0.4 MQRCCF_ACCESS_BLOCKED	3382	X'00000D36'
MQRCCF_OBJECT_ALREADY_EXISTS	4001	X'00000FA1'
MQRCCF_OBJECT_WRONG_TYPE	4002	X'00000FA2'
MQRCCF_LIKE_OBJECT_WRONG_TYPE	4003	X'00000FA3'
MQRCCF_OBJECT_OPEN	4004	X'00000FA4'
MQRCCF_ATTR_VALUE_ERROR	4005	X'00000FA5'
MQRCCF_UNKNOWN_Q_MGR	4006	X'00000FA6'
MQRCCF_Q_WRONG_TYPE	4007	X'00000FA7'
MQRCCF_OBJECT_NAME_ERROR	4008	X'00000FA8'
MQRCCF_ALLOCATE_FAILED	4009	X'00000FA9'
MQRCCF_HOST_NOT_AVAILABLE	4010	X'00000FAA'
MQRCCF_CONFIGURATION_ERROR	4011	X'00000FAB'
MQRCCF_CONNECTION_REFUSED	4012	X'00000FAC'
MQRCCF_ENTRY_ERROR	4013	X'00000FAD'
MQRCCF_SEND_FAILED	4014	X'00000FAE'
MQRCCF_RECEIVED_DATA_ERROR	4015	X'00000FAF'
MQRCCF_RECEIVE_FAILED	4016	X'00000FB0'
MQRCCF_CONNECTION_CLOSED	4017	X'00000FB1'
MQRCCF_NO_STORAGE	4018	X'00000FB2'
MQRCCF_NO_COMMS_MANAGER	4019	X'00000FB3'
MQRCCF_LISTENER_NOT_STARTED	4020	X'00000FB4'
MQRCCF_BIND_FAILED	4024	X'00000FB8'
MQRCCF_CHANNEL_INDOUBT	4025	X'00000FB9'
MQRCCF_MQCONN_FAILED	4026	X'00000FBA'
MQRCCF_MQOPEN_FAILED	4027	X'00000FBB'
MQRCCF_MQGET_FAILED	4028	X'00000FBC'
MQRCCF_MQPUT_FAILED	4029	X'00000FBD'
MQRCCF_PING_ERROR	4030	X'00000FBE'
MQRCCF_CHANNEL_IN_USE	4031	X'00000FBF'
MQRCCF_CHANNEL_NOT_FOUND	4032	X'00000FC0'
MQRCCF_UNKNOWN_REMOTE_CHANNEL	4033	X'00000FC1'
MQRCCF_REMOTE_QM_UNAVAILABLE	4034	X'00000FC2'
MQRCCF_REMOTE_QM_TERMINATING	4035	X'00000FC3'
MQRCCF_MQINQ_FAILED	4036	X'00000FC4'
MQRCCF_NOT_XMIT_Q	4037	X'00000FC5'
MQRCCF_CHANNEL_DISABLED	4038	X'00000FC6'
MQRCCF_USER_EXIT_NOT_AVAILABLE	4039	X'00000FC7'
MQRCCF_COMMIT_FAILED	4040	X'00000FC8'
MQRCCF_WRONG_CHANNEL_TYPE	4041	X'00000FC9'

MQRCCF_CHANNEL_ALREADY_EXISTS	4042	X'00000FCA'
MQRCCF_DATA_TOO_LARGE	4043	X'00000FCB'
MQRCCF_CHANNEL_NAME_ERROR	4044	X'00000FCC'
MQRCCF_XMIT_Q_NAME_ERROR	4045	X'00000FCD'
MQRCCF_MCA_NAME_ERROR	4047	X'00000FCF'
MQRCCF_SEND_EXIT_NAME_ERROR	4048	X'00000FD0'
MQRCCF_SEC_EXIT_NAME_ERROR	4049	X'00000FD1'
MQRCCF_MSG_EXIT_NAME_ERROR	4050	X'00000FD2'
MQRCCF_RCV_EXIT_NAME_ERROR	4051	X'00000FD3'
MQRCCF_XMIT_Q_NAME_WRONG_TYPE	4052	X'00000FD4'
MQRCCF_MCA_NAME_WRONG_TYPE	4053	X'00000FD5'
MQRCCF_DISC_INT_WRONG_TYPE	4054	X'00000FD6'
MQRCCF_SHORT_RETRY_WRONG_TYPE	4055	X'00000FD7'
MQRCCF_SHORT_TIMER_WRONG_TYPE	4056	X'00000FD8'
MQRCCF_LONG_RETRY_WRONG_TYPE	4057	X'00000FD9'
MQRCCF_LONG_TIMER_WRONG_TYPE	4058	X'00000FDA'
MQRCCF_PUT_AUTH_WRONG_TYPE	4059	X'00000FDB'
MQRCCF_KEEP_ALIVE_INT_ERROR	4060	X'00000FDC'
MQRCCF_MISSING_CONN_NAME	4061	X'00000FDD'
MQRCCF_CONN_NAME_ERROR	4062	X'00000FDE'
MQRCCF_MQSET_FAILED	4063	X'00000FDF'
MQRCCF_CHANNEL_NOT_ACTIVE	4064	X'00000FE0'
MQRCCF_TERMINATED_BY_SEC_EXIT	4065	X'00000FE1'
MQRCCF_DYNAMIC_Q_SCOPE_ERROR	4067	X'00000FE3'
MQRCCF_CELL_DIR_NOT_AVAILABLE	4068	X'00000FE4'
MQRCCF_MR_COUNT_ERROR	4069	X'00000FE5'
MQRCCF_MR_COUNT_WRONG_TYPE	4070	X'00000FE6'
MQRCCF_MR_EXIT_NAME_ERROR	4071	X'00000FE7'
MQRCCF_MR_EXIT_NAME_WRONG_TYPE	4072	X'00000FE8'
MQRCCF_MR_INTERVAL_ERROR	4073	X'00000FE9'
MQRCCF_MR_INTERVAL_WRONG_TYPE	4074	X'00000FEA'
MQRCCF_NPM_SPEED_ERROR	4075	X'00000FEB'
MQRCCF_NPM_SPEED_WRONG_TYPE	4076	X'00000FEC'
MQRCCF_HB_INTERVAL_ERROR	4077	X'00000FED'
MQRCCF_HB_INTERVAL_WRONG_TYPE	4078	X'00000FEE'
MQRCCF_CHAD_ERROR	4079	X'00000FEF'
MQRCCF_CHAD_WRONG_TYPE	4080	X'00000FF0'
MQRCCF_CHAD_EVENT_ERROR	4081	X'00000FF1'
MQRCCF_CHAD_EVENT_WRONG_TYPE	4082	X'00000FF2'
MQRCCF_CHAD_EXIT_ERROR	4083	X'00000FF3'
MQRCCF_CHAD_EXIT_WRONG_TYPE	4084	X'00000FF4'
MQRCCF_SUPPRESSED_BY_EXIT	4085	X'00000FF5'

MQRCCF_BATCH_INT_ERROR	4086	X'00000FF6'
MQRCCF_BATCH_INT_WRONG_TYPE	4087	X'00000FF7'
MQRCCF_NET_PRIORITY_ERROR	4088	X'00000FF8'
MQRCCF_NET_PRIORITY_WRONG_TYPE	4089	X'00000FF9'
MQRCCF_CHANNEL_CLOSED	4090	X'00000FFA'
MQRCCF_Q_STATUS_NOT_FOUND	4091	X'00000FFB'
MQRCCF_SSL_CIPHER_SPEC_ERROR	4092	X'00000FFC'
MQRCCF_SSL_PEER_NAME_ERROR	4093	X'00000FFD'
MQRCCF_SSL_CLIENT_AUTH_ERROR	4094	X'00000FFE'
MQRCCF_RETAINED_NOT_SUPPORTED	4095	X'00000FFF'

MQRCN_* (Client reconnect Constants):

MQRCN_NO	0	X'00000000'
MQRCN_YES	1	X'00000001'
MQRCN_Q_MGR	2	X'00000002'
MQRCN_DISABLED	3	X'00000003'

MQRCVTIME_* (Receive Timeout Types):

MQRCVTIME_MULTIPLY	0	X'00000000'
MQRCVTIME_ADD	1	X'00000001'
MQRCVTIME_EQUAL	2	X'00000002'

MQREADA_* (Read Ahead Values):

MQREADA_NO	0	X'00000000'
MQREADA_YES	1	X'00000001'
MQREADA_DISABLED	2	X'00000002'
MQREADA_INHIBITED	3	X'00000003'
MQREADA_BACKLOG	4	X'00000004'

MQRECORDING_* (Recording Options):

MQRECORDING_DISABLED	0	X'00000000'
MQRECORDING_Q	1	X'00000001'
MQRECORDING_MSG	2	X'00000002'

MQREGO_* (Publish/Subscribe Registration Options):

MQREGO_NONE	0	X'00000000'
MQREGO_CORREL_ID_AS_IDENTITY	1	X'00000001'
MQREGO_ANONYMOUS	2	X'00000002'
MQREGO_LOCAL	4	X'00000004'
MQREGO_DIRECT_REQUESTS	8	X'00000008'
MQREGO_NEW_PUBLICATIONS_ONLY	16	X'00000010'
MQREGO_PUBLISH_ON_REQUEST_ONLY	32	X'00000020'
MQREGO_DEREGISTER_ALL	64	X'00000040'
MQREGO_INCLUDE_STREAM_NAME	128	X'00000080'
MQREGO_INFORM_IF_RETAINED	256	X'00000100'
MQREGO_DUPLICATES_OK	512	X'00000200'
MQREGO_NON_PERSISTENT	1024	X'00000400'
MQREGO_PERSISTENT	2048	X'00000800'
MQREGO_PERSISTENT_AS_PUBLISH	4096	X'00001000'
MQREGO_PERSISTENT_AS_Q	8192	X'00002000'
MQREGO_ADD_NAME	16384	X'00004000'
MQREGO_NO_ALTERATION	32768	X'00008000'
MQREGO_FULL_RESPONSE	65536	X'00010000'
MQREGO_JOIN_SHARED	131072	X'00020000'
MQREGO_JOIN_EXCLUSIVE	262144	X'00040000'
MQREGO_LEAVE_ONLY	524288	X'00080000'
MQREGO_VARIABLE_USER_ID	1048576	X'00100000'
MQREGO_LOCKED	2097152	X'00200000'

MQRFH_* (Rules and formatting header structure and Flags):

Rules and formatting header structure

MQRFH_STRUC_ID	"RFHb"	
MQRFH_STRUC_ID_ARRAY	'R','F','H','b'	
MQRFH_VERSION_1	1	X'00000001'
MQRFH_VERSION_2	2	X'00000002'
MQRFH_STRUC_LENGTH_FIXED	32	X'00000020'
MQRFH_STRUC_LENGTH_FIXED_2	36	X'00000024'

Rules and formatting header Flags

MQRFH_NONE	0	X'00000000'
MQRFH_NO_FLAGS	0	X'00000000'

MQRFH2_* (Publish/Subscribe Options Tag RFH2 Top-level folder Tags):

MQRFH2_NAME_VALUE_VERSION	1	X'00000001'

MQRFH2_* (Publish/Subscribe Options Tag Tag names):

MQRFH2_PUBSUB_CMD_FOLDER	"psc"	
MQRFH2_PUBSUB_RESP_FOLDER	"pscr"	
MQRFH2_MSG_CONTENT_FOLDER	"mcd"	
MQRFH2_USER_FOLDER	"usr"	

MQRFH2_* (Publish/Subscribe Options Tag XML tag names):

MQRFH2_PUBSUB_CMD_FOLDER_B	"<psc>"	
MQRFH2_PUBSUB_CMD_FOLDER_E	"</psc>"	
MQRFH2_PUBSUB_RESP_FOLDER_B	"<pscr>"	
MQRFH2_PUBSUB_RESP_FOLDER_E	"</pscr>"	
MQRFH2_MSG_CONTENT_FOLDER_B	"<mcd>"	
MQRFH2_MSG_CONTENT_FOLDER_E	"</mcd>"	
MQRFH2_USER_FOLDER_B	"<usr>"	
MQRFH2_USER_FOLDER_E	"</usr>"	

MQRL_* (Returned Length):

MQRL_UNDEFINED	-1	X'FFFFFFFF'

MQRMH_* (Reference message header structure):

MQRMH_STRUC_ID	"RMHb"	
MQRMH_STRUC_ID_ARRAY	'R','M','H','b'	
MQRMH_VERSION_1	1	X'00000001'
MQRMH_CURRENT_VERSION	1	X'00000001'

MQRMHF_* (Reference message header Flags):

MQRMHF_LAST	1	X'00000001'
MQRMHF_NOT_LAST	0	X'00000000'

MQRO_* (Report Options):

MQRO_EXCEPTION	16777216	X'01000000'
MQRO_EXCEPTION_WITH_DATA	50331648	X'03000000'
MQRO_EXCEPTION_WITH_FULL_DATA	117440512	X'07000000'
MQRO_EXPIRATION	2097152	X'00200000'
MQRO_EXPIRATION_WITH_DATA	6291456	X'00600000'
MQRO_EXPIRATION_WITH_FULL_DATA	14680064	X'00E00000'
MQRO_COA	256	X'00000100'
MQRO_COA_WITH_DATA	768	X'00000300'
MQRO_COA_WITH_FULL_DATA	1792	X'00000700'
MQRO_COD	2048	X'00000800'
MQRO_COD_WITH_DATA	6144	X'00001800'
MQRO_COD_WITH_FULL_DATA	14336	X'00003800'
MQRO_PAN	1	X'00000001'
MQRO_NAN	2	X'00000002'
MQRO_ACTIVITY	4	X'00000004'
MQRO_NEW_MSG_ID	0	X'00000000'
MQRO_PASS_MSG_ID	128	X'00000080'
MQRO_COPY_MSG_ID_TO_CORREL_ID	0	X'00000000'
MQRO_PASS_CORREL_ID	64	X'00000040'
MQRO_DEAD_LETTER_Q	0	X'00000000'
MQRO_DISCARD_MSG	134217728	X'08000000'
MQRO_PASS_DISCARD_AND_EXPIRY	16384	X'00004000'
MQRO_NONE	0	X'00000000'

MQRO_* (Report Options Masks):

MQRO_REJECT_UNSUP_MASK	270270464	X'101C0000'
MQRO_ACCEPT_UNSUP_MASK	-270532353	X'EFE000FF'
MQRO_ACCEPT_UNSUP_IF_XMIT_MASK	261888	X'0003FF00'

MQROUTE_* (Trace-route):
Trace-route Max Activities (MQIACF_MAX_ACTIVITIES)

MQROUTE_UNLIMITED_ACTIVITIES	0	X'00000000'

Trace-route Detail (MQIACF_ROUTE_DETAIL)

MQROUTE_DETAIL_LOW	2	X'00000002'
MQROUTE_DETAIL_MEDIUM	8	X'00000008'
MQROUTE_DETAIL_HIGH	32	X'00000020'

Trace-route Forwarding (MQIACF_ROUTE_FORWARDING)

MQROUTE_FORWARD_ALL	256	X'00000100'
MQROUTE_FORWARD_IF_SUPPORTED	512	X'00000200'
MQROUTE_FORWARD_REJ_UNSUP_MASK	-65536	X'FFFF0000'

Trace-route Delivery (MQIACF_ROUTE_DELIVERY)

MQROUTE_DELIVER_YES	4096	X'00001000'
MQROUTE_DELIVER_NO	8192	X'00002000'
MQROUTE_DELIVER_REJ_UNSUP_MASK	-65536	X'FFFF0000'

Trace-route Accumulation (MQIACF_ROUTE_ACCUMULATION)

MQROUTE_ACCUMULATE_NONE	65539	X'00010003'
MQROUTE_ACCUMULATE_IN_MSG	65540	X'00010004'
MQROUTE_ACCUMULATE_AND_REPLY	65541	X'00010005'

MQRP_* (Command format Replace Options):

MQRP_YES	1	X'00000001'
MQRP_NO	0	X'00000000'

MQRQ_* (Command format Reason Qualifiers):

MQRQ_CONN_NOT_AUTHORIZED	1	X'00000001'
MQRQ_OPEN_NOT_AUTHORIZED	2	X'00000002'
MQRQ_CLOSE_NOT_AUTHORIZED	3	X'00000003'
MQRQ_CMD_NOT_AUTHORIZED	4	X'00000004'
MQRQ_Q_MGR_STOPPING	5	X'00000005'
MQRQ_Q_MGR_QUIESCING	6	X'00000006'
MQRQ_CHANNEL_STOPPED_OK	7	X'00000007'
MQRQ_CHANNEL_STOPPED_ERROR	8	X'00000008'
MQRQ_CHANNEL_STOPPED_RETRY	9	X'00000009'
MQRQ_CHANNEL_STOPPED_DISABLED	10	X'0000000A'
MQRQ_BRIDGE_STOPPED_OK	11	X'0000000B'
MQRQ_BRIDGE_STOPPED_ERROR	12	X'0000000C'
MQRQ_SSL_HANDSHAKE_ERROR	13	X'0000000D'
MQRQ_SSL_CIPHER_SPEC_ERROR	14	X'0000000E'
MQRQ_SSL_CLIENT_AUTH_ERROR	15	X'0000000F'

MQRQ_SSL_PEER_NAME_ERROR	16	X'00000010'
MQRQ_SUB_NOT_AUTHORIZED	17	X'00000011'
MQRQ_SUB_DEST_NOT_AUTHORIZED	18	X'00000012'
MQRQ_SSL_UNKNOWN_REVOCATION	19	X'00000013'
MQRQ_SYS_CONN_NOT_AUTHORIZED	20	X'00000014'
MQRQ_CHANNEL_BLOCKED_ADDRESS	21	X'00000015'
MQRQ_CHANNEL_BLOCKED_USERID	22	X'00000016'
MQRQ_CHANNEL_BLOCKED_NOACCESS	23	X'00000017'
MQRQ_MAX_ACTIVE_CHANNELS	24	X'00000018'
MQRQ_MAX_CHANNELS	25	X'00000019'
MQRQ_SVRCONN_INST_LIMIT	26	X'0000001A'
MQRQ_CLIENT_INST_LIMIT!	27	X'0000001B'
MQRQ_CAF_NOT_INSTALLED	28	X'0000001C'

MQRT_* (Command format Refresh Types):

MQRT_CONFIGURATION	1	X'00000001'
MQRT_EXPIRY	2	X'00000002'
MQRT_NSPROC	3	X'00000003'
MQRT_PROXYSUB	4	X'00000004'

MQRU_* (Request Only):

MQRU_PUBLISH_ON_REQUEST	1	X'00000001'
MQRU_PUBLISH_ALL	2	X'00000002'

MQSCA_* (TLS Client Authentication):

MQSCA_REQUIRED	0	X'00000000'
MQSCA_OPTIONAL	1	X'00000001'

MQSCO_* (TLS configuration options):
TLS configuration options structure

MQSCO_STRUC_ID	"SCOb"	
MQSCO_STRUC_ID_ARRAY	'S','C','O','b'	
MQSCO_VERSION_1	1	X'00000001'
MQSCO_VERSION_2	2	X'00000002'
MQSCO_VERSION_3	3	X'00000003'
MQSCO_VERSION_4	4	X'00000004'
MQSCO_CURRENT_VERSION	4	X'00000004'

TLS configuration options Key Reset Count

MQSCO_RESET_COUNT_DEFAULT	0	X'00000000'

Command format Queue Definition Scope

MQSCO_Q_MGR	1	X'00000001'
MQSCO_CELL	2	X'00000002'

MQSCOPE_* (Publish scope):

MQSCOPE_ALL	0	X'00000000'
MQSCOPE_AS_PARENT	1	X'00000001'
MQSCOPE_QMGR	4	X'00000004'

MQSCYC_* (Security Case):

MQSCYC_UPPER	0	X'00000000'
MQSCYC_MIXED	1	X'00000001'

MQSD_* (Object descriptor structure):

MQSD_STRUC_ID	"SDbb"	
MQSD_STRUC_ID_ARRAY	'S','D','b','b'	
MQSD_VERSION_1	1	X'00000001'
MQSD_CURRENT_VERSION	1	X'00000001'

MQSECITEM_* (Command format Security Items):

MQSECITEM_ALL	0	X'00000000'
MQSECITEM_MQADMIN	1	X'00000001'
MQSECITEM_MQNLIST	2	X'00000002'
MQSECITEM_MQPROC	3	X'00000003'
MQSECITEM_MQQUEUE	4	X'00000004'
MQSECITEM_MQCONN	5	X'00000005'
MQSECITEM_MQCMDS	6	X'00000006'
MQSECITEM_MXADMIN	7	X'00000007'
MQSECITEM_MXNLIST	8	X'00000008'
MQSECITEM_MXPROC	9	X'00000009'
MQSECITEM_MXQUEUE	10	X'0000000A'
MQSECITEM_MXTOPIC	11	X'0000000B'

MQSECPROT_* (Security Protocol Types):

MQSECPROT_NONE	0	X'00000000'
MQSECPROT_SSLV30	1	X'00000001'
MQSECPROT_TLSV10	2	X'00000002'
MQSECPROT_TLSV12	4	X'00000004'

MQSECSW_* (Command format Security Switches and Switch States):

Command format Security Switches

MQSECSW_PROCESS	1	X'00000001'
MQSECSW_NAMELIST	2	X'00000002'
MQSECSW_Q	3	X'00000003'
MQSECSW_TOPIC	4	X'00000004'
MQSECSW_CONTEXT	6	X'00000006'
MQSECSW_ALTERNATE_USER	7	X'00000007'
MQSECSW_COMMAND	8	X'00000008'
MQSECSW_CONNECTION	9	X'00000009'
MQSECSW_SUBSYSTEM	10	X'0000000A'
MQSECSW_COMMAND_RESOURCES	11	X'0000000B'
MQSECSW_Q_MGR	15	X'0000000F'
MQSECSW_QSG	16	X'00000010'

Command format Security Switch States

MQSECSW_OFF_FOUND	21	X'00000015'
MQSECSW_ON_FOUND	22	X'00000016'
MQSECSW_OFF_NOT_FOUND	23	X'00000017'
MQSECSW_ON_NOT_FOUND	24	X'00000018'
MQSECSW_OFF_ERROR	25	X'00000019'
MQSECSW_ON_OVERRIDDEN	26	X'0000001A'

MQSECTYPE_* (Command format Security Types):

MQSECTYPE_AUTHSERV	1	X'00000001'
MQSECTYPE_SSL	2	X'00000002'
MQSECTYPE_CLASSES	3	X'00000003'

MQSEG_* (Segmentation):

MQSEG_INHIBITED	'b'	
MQSEG_ALLOWED	'A'	

MQSEL_* (Special Selector Values):

MQSEL_ANY_SELECTOR	-30001	X'FFFF8ACF'
MQSEL_ANY_USER_SELECTOR	-30002	X'FFFF8ACE'
MQSEL_ANY_SYSTEM_SELECTOR	-30003	X'FFFF8ACD'
MQSEL_ALL_SELECTORS	-30001	X'FFFF8ACF'
MQSEL_ALL_USER_SELECTORS	-30002	X'FFFF8ACE'
MQSEL_ALL_SYSTEM_SELECTORS	-30003	X'FFFF8ACD'

MQSELTYPE_* (Selector Types):

MQSELTYPE_NONE	0	X'00000000'
MQSELTYPE_STANDARD	1	X'00000001'
MQSELTYPE_EXTENDED	2	X'00000002'

MQSID_* (Security Identifier):

MQSID_NONE	X'00...00'	(40 nulls)
MQSID_NONE_ARRAY	'\0','\0',...	(40 nulls)

MQSIDT_* (Security Identifier Types):

MQSIDT_NONE	X'00'	
MQSIDT_NT_SECURITY_ID	X'01'	
MQSIDT_WAS_SECURITY_ID	X'02'	

MQSMPO_* (Set message property options and structure):
Set message property options structure

MQSMPO_STRUC_ID	"SMPO"	
MQSMPO_STRUC_ID_ARRAY	'S','M','P','O'	
MQSMPO_VERSION_1	1	X'00000001'
MQSMPO_CURRENT_VERSION	1	X'00000001'

Set Message Property Options

MQSMPO_SET_FIRST	0	X'00000000'
MQSMPO_SET_PROP_UNDER_CURSOR	1	X'00000001'
MQSMPO_SET_PROP_AFTER_CURSOR	2	X'00000002'
MQSMPO_APPEND_PROPERTY	4	X'00000004'
MQSMPO_SET_PROP_BEFORE_CURSOR	8	X'00000008'
MQSMPO_NONE	0	X'00000000'

MQSO_* (Subscribe Options):

MQSO_NONE	0	X'00000000'
MQSO_NON_DURABLE	0	X'00000000'
MQSO_READ_AHEAD_AS_Q_DEF	0	X'00000000'
MQSO_ALTER	1	X'00000001'
MQSO_CREATE	2	X'00000002'
MQSO_RESUME	4	X'00000004'
MQSO_DURABLE	8	X'00000008'
MQSO_GROUP_SUB	16	X'00000010'
MQSO_MANAGED	32	X'00000020'
MQSO_SET_IDENTITY_CONTEXT	64	X'00000040'
MQSO_FIXED_USERID	256	X'00000100'
MQSO_ANY_USERID	512	X'00000200'
MQSO_PUBLICATIONS_ON_REQUEST	2048	X'00000800'
MQSO_NEW_PUBLICATIONS_ONLY	4096	X'00001000'
MQSO_FAIL_IF_QUIESCING	8192	X'00002000'
MQSO_ALTERNATE_USER_AUTHORITY	262144	X'00040000'
MQSO_WILDCARD_CHAR	1048576	X'00100000'
MQSO_WILDCARD_TOPIC	2097152	X'00200000'
MQSO_SET_CORREL_ID	4194304	X'00400000'
MQSO_SCOPE_QMGR	67108864	X'04000000'
MQSO_NO_READ_AHEAD	134217728	X'08000000'
MQSO_READ_AHEAD	268435456	X'10000000'

MQSP_* (Sync point Availability):

MQSP_AVAILABLE	1	X'00000001'
MQSP_NOT_AVAILABLE	0	X'00000000'

MQSQQM_* (Shared Queue Queue Manager Name):

MQSQQM_USE	0	X'00000000'
MQSQQM_IGNORE	1	X'00000001'

MQSR_* (Action):

MQSR_ACTION_PUBLICATION	1	X'00000001'

MQSRO_* (Subscription request options structure):

MQSRO_STRUC_ID	"SROb"	
MQSRO_STRUC_ID_ARRAY	'S','R','O','b'	
MQSRO_VERSION_1	1	X'00000001'
MQSRO_CURRENT_VERSION	1	X'00000001'
MQSRO_NONE	0	X'00000000'
MQSRO_FAIL_IF_QUIESCING	8192	X'00002000'

MQSS_* (Segment Status):

MQSS_NOT_A_SEGMENT	'b'	
MQSS_SEGMENT	'S'	
MQSS_LAST_SEGMENT	'L'	

MQSSL_* (TLS FIPS Requirements):

MQSSL_FIPS_NO	0	X'00000000'
MQSSL_FIPS_YES	1	X'00000001'

MQSTAT_* (Stat Options):

MQSTAT_TYPE_ASYNC_ERROR	0	X'00000000'
MQSTAT_TYPE_RECONNECTION	0	X'00000000'
MQSTAT_TYPE_RECONNECTION_ERROR	0	X'00000000'

MQSTS_* (Status reporting structure structure):

MQSTS_STRUC_ID	"STAT"	
MQSTS_STRUC_ID_ARRAY	'S','T','A','T'	
MQSTS_VERSION_1	1	X'00000001'
MQSTS_CURRENT_VERSION	1	X'00000001'

MQSUB_* (Durable subscriptions):
Durable subscriptions

MQSUB_DURABLE_AS_PARENT	0	X'00000000'
MQSUB_DURABLE_ALLOWED	1	X'00000001'
MQSUB_DURABLE_INHIBITED	2	X'00000002'

Durable Subscriptions

MQSUB_DURABLE_ALL	-1	X'FFFFFFFF'
MQSUB_DURABLE_YES	1	X'00000001'
MQSUB_DURABLE_NO	2	X'00000002'

MQSUBTYPE_* (Command format Subscription Types):

MQSUBTYPE_API	1	X'00000001'
MQSUBTYPE_ADMIN	2	X'00000002'
MQSUBTYPE_PROXY	3	X'00000003'
MQSUBTYPE_ALL	-1	X'FFFFFFFF'
MQSUBTYPE_USER	-2	X'FFFFFFFE'

MQSUS_* (Command format Suspend Status):

MQSUS_YES	1	X'00000001'
MQSUS_NO	0	X'00000000'

MQSVC_* (Service):
Service Types

MQSVC_TYPE_COMMAND	0	X'00000000'
MQSVC_TYPE_SERVER	1	X'00000001'

Service Controls

MQSVC_CONTROL_Q_MGR	0	X'00000000'
MQSVC_CONTROL_Q_MGR_START	1	X'00000001'
MQSVC_CONTROL_MANUAL	2	X'00000002'

Service Status

MQSVC_STATUS_STOPPED	0	X'00000000'
MQSVC_STATUS_STARTING	1	X'00000001'
MQSVC_STATUS_RUNNING	2	X'00000002'
MQSVC_STATUS_STOPPING	3	X'00000003'
MQSVC_STATUS_RETRYING	4	X'00000004'

MQSYNCPOINT_* (Command format Syncpoint values for Pub/Sub migration):

MQSYNCPOINT_YES	0	X'00000000'
MQSYNCPOINT_IFPER	1	X'00000001'

MQSYSP_* (Command format System Parameter Values):

MQSYSP_NO	0	X'00000000'
MQSYSP_YES	1	X'00000001'
MQSYSP_EXTENDED	2	X'00000002'
MQSYSP_TYPE_INITIAL	10	X'0000000A'
MQSYSP_TYPE_SET	11	X'0000000B'
MQSYSP_TYPE_LOG_COPY	12	X'0000000C'
MQSYSP_TYPE_LOG_STATUS	13	X'0000000D'
MQSYSP_TYPE_ARCHIVE_TAPE	14	X'0000000E'
MQSYSP_ALLOC_BLK	20	X'00000014'
MQSYSP_ALLOC_TRK	21	X'00000015'
MQSYSP_ALLOC_CYL	22	X'00000016'
MQSYSP_STATUS_BUSY	30	X'0000001E'
MQSYSP_STATUS_PREMOUNT	31	X'0000001F'
MQSYSP_STATUS_AVAILABLE	32	X'00000020'
MQSYSP_STATUS_UNKNOWN	33	X'00000021'
MQSYSP_STATUS_ALLOC_ARCHIVE	34	X'00000022'
MQSYSP_STATUS_COPYING_BSDS	35	X'00000023'
MQSYSP_STATUS_COPYING_LOG	36	X'00000024'

MQTA_* (Topic attributes):
Wildcards

MQTA_BLOCK	1	X'00000001'
MQTA_PASSTHRU	2	X'00000002'

Subscriptions Allowed

MQTA_SUB_AS_PARENT	0	X'00000000'
MQTA_SUB_INHIBITED	1	X'00000001'
MQTA_SUB_ALLOWED	2	X'00000002'

Proxy Sub Propagation

MQTA_PROXY_SUB_FORCE	1	X'00000001'
MQTA_PROXY_SUB_FIRSTUSE	2	X'00000002'

Publications Allowed

MQTA_PUB_AS_PARENT	0	X'00000000'
MQTA_PUB_INHIBITED	1	X'00000001'
MQTA_PUB_ALLOWED	2	X'00000002'

MQTC_* (Trigger Controls):

MQTC_OFF	0	X'00000000'
MQTC_ON	1	X'00000001'

MQTCPKEEP_* (TCP Keepalive):

MQTCPKEEP_NO	0	X'00000000'
MQTCPKEEP_YES	1	X'00000001'

MQTCPSTACK_* (TCP Stack Types):

MQTCPSTACK_SINGLE	0	X'00000000'
MQTCPSTACK_MULTIPLE	1	X'00000001'

MQTIME_* (Command format Time units):

MQTIME_UNIT_MINS	0	X'00000000'
MQTIME_UNIT_SECS	1	X'00000001'

MQTM_* (Trigger message structure):

MQTM_STRUC_ID	"TMbb"	
MQTM_STRUC_ID_ARRAY	'T','M','b','b'	
MQTM_VERSION_1	1	X'00000001'
MQTM_CURRENT_VERSION	1	X'00000001'

MQTMC_* (Trigger message character format structure):

MQTMC_STRUC_ID	"TMCb"	
MQTMC_STRUC_ID_ARRAY	'T','M','C','b'	
MQTMC_VERSION_1	"bbb1"	
MQTMC_VERSION_2	"bbb2"	
MQTMC_CURRENT_VERSION	"bbb2"	
MQTMC_VERSION_1_ARRAY	'b','b','b','1'	
MQTMC_VERSION_2_ARRAY	'b','b','b','2'	

MQTMC_CURRENT_VERSION_ARRAY	'b','b','b','2'	

MQTOPT_* (Topic Type):

MQTOPT_LOCAL	0	X'00000000'
MQTOPT_CLUSTER	1	X'00000001'
MQTOPT_ALL	2	X'00000002'

MQTRAXSTR_* (Channel Initiator Trace Autostart):

MQTRAXSTR_NO	0	X'00000000'
MQTRAXSTR_YES	1	X'00000001'

MQTSCOPE_* (Subscription Scope):

MQTSCOPE_QMGR	1	X'00000001'
MQTSCOPE_ALL	2	X'00000002'

MQTT_* (Trigger Types):

MQTT_NONE	0	X'00000000'
MQTT_FIRST	1	X'00000001'
MQTT_EVERY	2	X'00000002'
MQTT_DEPTH	3	X'00000003'

MQTYPE_* (Property data types):

MQTYPE_AS_SET	0	X'00000000'
MQTYPE_NULL	2	X'00000002'
MQTYPE_BOOLEAN	4	X'00000004'
MQTYPE_BYTE_STRING	8	X'00000008'
MQTYPE_INT8	16	X'00000010'
MQTYPE_INT16	32	X'00000020'
MQTYPE_INT32	64	X'00000040'
MQTYPE_LONG	64	X'00000040'
MQTYPE_INT64	128	X'00000080'
MQTYPE_FLOAT32	256	X'00000100'
MQTYPE_FLOAT64	512	X'00000200'
MQTYPE_STRING	1024	X'00000400'

MQUA_* (Publish/Subscribe User Attribute Selectors):

MQUA_FIRST	65536	X'00010000'
MQUA_LAST	999999999	X'3B9AC9FF'

MQUIDSUPP_* (Command format User ID Support):

MQUIDSUPP_NO	0	X'00000000'
MQUIDSUPP_YES	1	X'00000001'

MQUNDELIVERED_* (Command format Undelivered values for Pub/Sub migration):

MQUNDELIVERED_NORMAL	0	X'00000000'
MQUNDELIVERED_SAFE	1	X'00000001'
MQUNDELIVERED_DISCARD	2	X'00000002'
MQUNDELIVERED_KEEP	3	X'00000003'

MQUOWST_* (Command format UOW States):

MQUOWST_NONE	0	X'00000000'
MQUOWST_ACTIVE	1	X'00000001'
MQUOWST_PREPARED	2	X'00000002'
MQUOWST_UNRESOLVED	3	X'00000003'

MQUOWT_* (Command format UOW Types):

MQUOWT_Q_MGR	0	X'00000000'
MQUOWT_CICS	1	X'00000001'
MQUOWT_RRS	2	X'00000002'
MQUOWT_IMS	3	X'00000003'
MQUOWT_XA	4	X'00000004'

MQUS_* (Queue Usages):

MQUS_NORMAL	0	X'00000000'
MQUS_TRANSMISSION	1	X'00000001'

MQUSAGE_* (Command format Page Set Usage Values and Data Set Usage Values):
Command format Page Set Usage Values

MQUSAGE_PS_AVAILABLE	0	X'00000000'
MQUSAGE_PS_DEFINED	1	X'00000001'
MQUSAGE_PS_OFFLINE	2	X'00000002'
MQUSAGE_PS_NOT_DEFINED	3	X'00000003'
MQUSAGE_PS_SUSPENDED	4	X'00000004'
MQUSAGE_EXPAND_USER	1	X'00000001'
MQUSAGE_EXPAND_SYSTEM	2	X'00000002'
MQUSAGE_EXPAND_NONE	3	X'00000003'

Command format Data Set Usage Values

MQUSAGE_DS_OLDEST_ACTIVE_UOW	10	X'0000000A'
MQUSAGE_DS_OLDEST_PS_RECOVERY	11	X'0000000B'
MQUSAGE_DS_OLDEST_CF_RECOVERY	12	X'0000000C'

MQVL_* (Value Length):

MQVL_NULL_TERMINATED	-1	X'FFFFFFFF'
MQVL_EMPTY_STRING	0	X'00000000'

MQVU_* (Variable User ID):

MQVU_FIXED_USER	1	X'00000001'
MQVU_ANY_USER	2	X'00000002'

MQWDR_* (Cluster workload exit destination record structure):

MQWDR_STRUC_ID	"WDRb"	
MQWDR_STRUC_ID_ARRAY	'W','D','R','b'	
MQWDR_VERSION_1	1	X'00000001'
MQWDR_VERSION_2	2	X'00000002'
MQWDR_CURRENT_VERSION	2	X'00000002'
MQWDR_LENGTH_1	124	X'0000007C'
MQWDR_LENGTH_2	136	X'00000088'
MQWDR_CURRENT_LENGTH	136	X'00000088'

MQWI_* (Wait Interval):

MQWI_UNLIMITED	-1	X'FFFFFFFF'

MQWIH_* (Workload information header structure and Flags):
Workload information header structure

MQWIH_STRUC_ID	"WIHb"	
MQWIH_STRUC_ID_ARRAY	'W','I','H','b'	
MQWIH_VERSION_1	1	X'00000001'
MQWIH_CURRENT_VERSION	1	X'00000001'
MQWIH_LENGTH_1	120	X'00000078'
MQWIH_CURRENT_LENGTH	120	X'00000078'

Workload information header Flags

MQWIH_NONE	0	X'00000000'

MQWQR_* (Cluster workload exit queue record structure):

MQWQR_STRUC_ID	"WQRb"	
MQWQR_STRUC_ID_ARRAY	'W','Q','R','b'	
MQWQR_VERSION_1	1	X'00000001'
MQWQR_VERSION_2	2	X'00000002'
MQWQR_VERSION_3	3	X'00000003'
MQWQR_CURRENT_VERSION	3	X'00000003'
MQWQR_LENGTH_1	200	X'000000C8'
MQWQR_LENGTH_2	208	X'000000D0'
MQWQR_LENGTH_3	212	X'000000D4'
MQWQR_CURRENT_LENGTH	212	X'000000D4'

MQWS_* (Wildcard Schema):

MQWS_DEFAULT	0	X'00000000'
MQWS_CHAR	1	X'00000001'
MQWS_TOPIC	2	X'00000002'

MQWXP_* (Cluster workload exit parameter structure):
MQWXP_* (Cluster workload exit parameter structure)

MQWXP_STRUC_ID	"WXPb"	
MQWXP_STRUC_ID_ARRAY	'W','X','P','b'	
MQWXP_VERSION_1	1	X'00000001'
MQWXP_VERSION_2	2	X'00000002'
MQWXP_VERSION_3	3	X'00000003'
MQWXP_VERSION_4	4	X'00000004'
MQWXP_CURRENT_VERSION	4	X'00000004'

MQWXP_* (Cluster Workload Flags)

MQWXP_PUT_BY_CLUSTER_CHL	2	X'00000002'

Related reference:
"Fields in MQWXP - Cluster workload exit parameter structure" on page 3634
Description of the fields in the MQWXP - Cluster workload exit parameter structure

MQXACT_* (API Caller Types):

MQXACT_EXTERNAL	1	X'00000001'
MQXACT_INTERNAL	2	X'00000002'

MQXC_* (Exit Commands):

MQXC_MQOPEN	1	X'00000001'
MQXC_MQCLOSE	2	X'00000002'
MQXC_MQGET	3	X'00000003'
MQXC_MQPUT	4	X'00000004'
MQXC_MQPUT1	5	X'00000005'
MQXC_MQINQ	6	X'00000006'
MQXC_MQSET	8	X'00000008'
MQXC_MQBACK	9	X'00000009'
MQXC_MQCMIT	10	X'0000000A'

MQXCC_* (Exit Responses):

MQXCC_OK	0	X'00000000'
MQXCC_SUPPRESS_FUNCTION	-1	X'FFFFFFFF'
MQXCC_SKIP_FUNCTION	-2	X'FFFFFFFE'
MQXCC_SEND_AND_REQUEST_SEC_MSG	-3	X'FFFFFFFD'
MQXCC_SEND_SEC_MSG	-4	X'FFFFFFFC'
MQXCC_SUPPRESS_EXIT	-5	X'FFFFFFFB'
MQXCC_CLOSE_CHANNEL	-6	X'FFFFFFFA'
MQXCC_REQUEST_ACK	-7	X'FFFFFFF9'
MQXCC_FAILED	-8	X'FFFFFFF8'

MQXDR_* (Exit Response):

MQXDR_OK	0	X'00000000'
MQXDR_CONVERSION_FAILED	1	X'00000001'

MQXE_* (Environments):

MQXE_OTHER	0	X'00000000'
MQXE_MCA	1	X'00000001'
MQXE_MCA_SVRCONN	2	X'00000002'
MQXE_COMMAND_SERVER	3	X'00000003'
MQXE_MQSC	4	X'00000004'

MQXEPO_* (Register Entry Point Options structure and Exit Options):
Register Entry Point Options structure

MQXEPO_STRUC_ID	"XEPO"	
MQXEPO_STRUC_ID_ARRAY	'X','E','P','O'	
MQXEPO_VERSION_1	1	X'00000001'
MQXEPO_CURRENT_VERSION	1	X'00000001'

Exit Options

MQXEPO_NONE	0	X'00000000'

MQXF_* (API Function Identifiers):

MQXF_INIT	1	X'00000001'
MQXF_TERM	2	X'00000002'
MQXF_CONN	3	X'00000003'
MQXF_CONNX	4	X'00000004'
MQXF_DISC	5	X'00000005'
MQXF_OPEN	6	X'00000006'
MQXF_CLOSE	7	X'00000007'
MQXF_PUT1	8	X'00000008'
MQXF_PUT	9	X'00000009'
MQXF_GET	10	X'0000000A'
MQXF_DATA_CONV_ON_GET	11	X'0000000B'
MQXF_INQ	12	X'0000000C'
MQXF_SET	13	X'0000000D'
MQXF_BEGIN	14	X'0000000E'
MQXF_CMIT	15	X'0000000F'
MQXF_BACK	16	X'00000010'
MQXF_STAT	18	X'00000012'
MQXF_CB	19	X'00000013'

MQXF_CTL	20	X'00000014'
MQXF_CALLBACK	21	X'00000015'
MQXF_SUB	22	X'00000016'
MQXF_SUBRQ	23	X'00000017'
MQXF_XACLOSE	24	X'00000018'
MQXF_XACOMMIT	25	X'00000019'
MQXF_XACOMPLETE	26	X'0000001A'
MQXF_XAEND	27	X'0000001B'
MQXF_XAFORGET	28	X'0000001C'
MQXF_XAOPEN	29	X'0000001D'
MQXF_XAPREPARE	30	X'0000001E'
MQXF_XARECOVER	31	X'0000001F'
MQXF_XAROLLBACK	32	X'00000020'
MQXF_XASTART	33	X'00000021'
MQXF_AXREG	34	X'00000022'
MQXF_AXUNREG	35	X'00000023'

MQXP_* (API crossing exit parameter structure):

MQXP_STRUC_ID	"XPbb"	
MQXP_STRUC_ID_ARRAY	'X','P','b','b'	
MQXP_VERSION_1	1	X'00000001'

MQXPDA_* (Problem Determination Area):

MQXPDA_NONE	X'00...00'	(48 nulls)
MQXPDA_NONE_ARRAY	'\0','\0',...	(48 nulls)

MQXPT_* (Transport Types):

MQXPT_ALL	-1	X'FFFFFFFF'
MQXPT_LOCAL	0	X'00000000'
MQXPT_LU62	1	X'00000001'
MQXPT_TCP	2	X'00000002'
MQXPT_NETBIOS	3	X'00000003'
MQXPT_SPX	4	X'00000004'
MQXPT_DECNET	5	X'00000005'
MQXPT_UDP	6	X'00000006'

MQXQH_* (Transmission queue header structure):

MQXQH_STRUC_ID	"XQHb"	
MQXQH_STRUC_ID_ARRAY	'X','Q','H','b'	
MQXQH_VERSION_1	1	X'00000001'
MQXQH_CURRENT_VERSION	1	X'00000001'

MQXR_* (Exit Reasons):

MQXR_BEFORE	1	X'00000001'
MQXR_AFTER	2	X'00000002'
MQXR_CONNECTION	3	X'00000003'
MQXR_INIT	11	X'0000000B'
MQXR_TERM	12	X'0000000C'
MQXR_MSG	13	X'0000000D'
MQXR_XMIT	14	X'0000000E'
MQXR_SEC_MSG	15	X'0000000F'
MQXR_INIT_SEC	16	X'00000010'
MQXR_RETRY	17	X'00000011'
MQXR_AUTO_CLUSSDR	18	X'00000012'
MQXR_AUTO_RECEIVER	19	X'00000013'
MQXR_CLWL_OPEN	20	X'00000014'
MQXR_CLWL_PUT	21	X'00000015'
MQXR_CLWL_MOVE	22	X'00000016'
MQXR_CLWL_REPOS	23	X'00000017'
MQXR_CLWL_REPOS_MOVE	24	X'00000018'
MQXR_END_BATCH	25	X'00000019'
MQXR_ACK_RECEIVED	26	X'0000001A'
MQXR_AUTO_SVRCONN	27	X'0000001B'
MQXR_AUTO_CLUSRCVR	28	X'0000001C'
MQXR_SEC_PARMS	29	X'0000001D'

MQXR2_* (Exit Response 2):

MQXR2_PUT_WITH_DEF_ACTION	0	X'00000000'
MQXR2_PUT_WITH_DEF_USERID	1	X'00000001'
MQXR2_PUT_WITH_MSG_USERID	2	X'00000002'
MQXR2_USE_AGENT_BUFFER	0	X'00000000'
MQXR2_USE_EXIT_BUFFER	4	X'00000004'
MQXR2_DEFAULT_CONTINUATION	0	X'00000000'
MQXR2_CONTINUE_CHAIN	8	X'00000008'
MQXR2_SUPPRESS_CHAIN	16	X'00000010'
MQXR2_STATIC_CACHE	0	X'00000000'
MQXR2_DYNAMIC_CACHE	32	X'00000020'

MQXT_* (Exit Identifiers):

MQXT_API_CROSSING_EXIT	1	X'00000001'
MQXT_API_EXIT	2	X'00000002'
MQXT_CHANNEL_SEC_EXIT	11	X'0000000B'
MQXT_CHANNEL_MSG_EXIT	12	X'0000000C'
MQXT_CHANNEL_SEND_EXIT	13	X'0000000D'
MQXT_CHANNEL_RCV_EXIT	14	X'0000000E'
MQXT_CHANNEL_MSG_RETRY_EXIT	15	X'0000000F'
MQXT_CHANNEL_AUTO_DEF_EXIT	16	X'00000010'
MQXT_CLUSTER_WORKLOAD_EXIT	20	X'00000014'
MQXT_PUBSUB_ROUTING_EXIT	21	X'00000015'

MQXUA_* (Exit User Area Value):

MQXUA_NONE	X'00...00'	(16 nulls)
MQXUA_NONE_ARRAY	'\0','\0',...	(16 nulls)

MQXWD_* (Exit wait descriptor structure):

MQXWD_STRUC_ID	"XWDb"	
MQXWD_STRUC_ID_ARRAY	'X','W','D','b'	
MQXWD_VERSION_1	1	X'00000001'

MQZAC_* (Application context structure):

MQZAC_STRUC_ID	"ZACb"	
MQZAC_STRUC_ID_ARRAY	'Z','A','C','b'	
MQZAC_VERSION_1	1	X'00000001'
MQZAC_CURRENT_VERSION	1	X'00000001'

MQZAD_* (Authority data structure):

MQZAD_STRUC_ID	"ZADb"	
MQZAD_STRUC_ID_ARRAY	'Z','A','D','b'	
MQZAD_VERSION_1	1	X'00000001'
MQZAD_VERSION_2	2	X'00000002'
MQZAD_CURRENT_VERSION	2	X'00000002'

MQZAET_* (Installable Services Entity Types):

MQZAET_NONE	0	X'00000000'
MQZAET_PRINCIPAL	1	X'00000001'
MQZAET_GROUP	2	X'00000002'
MQZAET_UNKNOWN	3	X'00000003'

MQZAO_* (Installable Services Authorizations):

MQZAO_CONNECT	1	X'00000001'
MQZAO_BROWSE	2	X'00000002'
MQZAO_INPUT	4	X'00000004'
MQZAO_OUTPUT	8	X'00000008'
MQZAO_INQUIRE	16	X'00000010'
MQZAO_SET	32	X'00000020'
MQZAO_PASS_IDENTITY_CONTEXT	64	X'00000040'
MQZAO_PASS_ALL_CONTEXT	128	X'00000080'
MQZAO_SET_IDENTITY_CONTEXT	256	X'00000100'
MQZAO_SET_ALL_CONTEXT	512	X'00000200'
MQZAO_ALTERNATE_USER_AUTHORITY	1024	X'00000400'
MQZAO_PUBLISH	2048	X'00000800'
MQZAO_SUBSCRIBE	4096	X'00001000'
MQZAO_RESUME	8192	X'00002000'
MQZAO_ALL_MQI	16383	X'00003FFF'
MQZAO_CREATE	65536	X'00010000'
MQZAO_DELETE	131072	X'00020000'
MQZAO_DISPLAY	262144	X'00040000'
MQZAO_CHANGE	524288	X'00080000'
MQZAO_CLEAR	1048576	X'00100000'
MQZAO_CONTROL	2097152	X'00200000'
MQZAO_CONTROL_EXTENDED	4194304	X'00400000'
MQZAO_AUTHORIZE	8388608	X'00800000'
MQZAO_ALL_ADMIN	16646144	X'00FE0000'
MQZAO_ALL	16662527	X'00FE3FFF'
MQZAO_REMOVE	16777216	X'01000000'
MQZAO_NONE	0	X'00000000'

MQZAS_* (Installable Services Service Interface Version):

MQZAS_VERSION_1	1	X'00000001'
MQZAS_VERSION_2	2	X'00000002'
MQZAS_VERSION_3	3	X'00000003'
MQZAS_VERSION_4	4	X'00000004'
MQZAS_VERSION_5	5	X'00000005'
MQZAS_VERSION_6	6	X'00000006'

MQZAT_* (Authentication Types):

MQZAT_INITIAL_CONTEXT	0	X'00000000'
MQZAT_CHANGE_CONTEXT	1	X'00000001'

MQZCI_* (Installable Services Continuation Indicator):

MQZCI_DEFAULT	0	X'00000000'
MQZCI_CONTINUE	0	X'00000000'
MQZCI_STOP	1	X'00000001'

MQZED_* (Entity data structure):

MQZED_STRUC_ID	"ZEDb"	
MQZED_STRUC_ID_ARRAY	'Z','E','D','b'	
MQZED_VERSION_1	1	X'00000001'
MQZED_VERSION_2	2	X'00000002'
MQZED_CURRENT_VERSION	2	X'00000002'

MQZFP_* (Free parameters structure):

MQZFP_STRUC_ID	"ZFPb"	
MQZFP_STRUC_ID_ARRAY	'Z','F','P','b'	
MQZFP_VERSION_1	1	X'00000001'
MQZFP_CURRENT_VERSION	1	X'00000001'

MQZIC_* (Identity context structure):

MQZIC_STRUC_ID	"ZICb"	
MQZIC_STRUC_ID_ARRAY	'Z','I','C','b'	
MQZIC_VERSION_1	1	X'00000001'
MQZIC_CURRENT_VERSION	1	X'00000001'

MQZID_* (Function ids for services):
Function ids common to all services

MQZID_INIT	0	X'00000000'
MQZID_TERM	1	X'00000001'

Function ids for Authority service

MQZID_INIT_AUTHORITY	0	X'00000000'
MQZID_TERM_AUTHORITY	1	X'00000001'
MQZID_CHECK_AUTHORITY	2	X'00000002'
MQZID_COPY_ALL_AUTHORITY	3	X'00000003'
MQZID_DELETE_AUTHORITY	4	X'00000004'
MQZID_SET_AUTHORITY	5	X'00000005'
MQZID_GET_AUTHORITY	6	X'00000006'
MQZID_GET_EXPLICIT_AUTHORITY	7	X'00000007'
MQZID_REFRESH_CACHE	8	X'00000008'
MQZID_ENUMERATE_AUTHORITY_DATA	9	X'00000009'
MQZID_AUTHENTICATE_USER	10	X'0000000A'
MQZID_FREE_USER	11	X'0000000B'
MQZID_INQUIRE	12	X'0000000C'
MQZID_CHECK_PRIVILEGED	13	X'0000000D'

Function ids for Name service

MQZID_INIT_NAME	0	X'00000000'
MQZID_TERM_NAME	1	X'00000001'
MQZID_LOOKUP_NAME	2	X'00000002'
MQZID_INSERT_NAME	3	X'00000003'
MQZID_DELETE_NAME	4	X'00000004'

Function ids for Userid service

MQZID_INIT_USERID	0	X'00000000'
MQZID_TERM_USERID	1	X'00000001'
MQZID_FIND_USERID	2	X'00000002'

MQZIO_* (Installable Services Initialization Options):

MQZIO_PRIMARY	0	X'00000000'
MQZIO_SECONDARY	1	X'00000001'

MQZNS_* (Name Service Interface Version):

MQZNS_VERSION_1	1	X'00000001'

MQZSE_* (Installable Services Start-Enumeration Indicator):

MQZSE_START	1	X'00000001'
MQZSE_CONTINUE	0	X'00000000'

MQZSL_* (Installable Services Selector Indicator):

MQZSL_NOT_RETURNED	0	X'00000000'
MQZSL_RETURNED	1	X'00000001'

MQZTO_* (Installable Services Termination Options):

MQZTO_PRIMARY	0	X'00000000'
MQZTO_SECONDARY	1	X'00000001'

MQZUS_* (Userid Service Interface Version):

MQZUS_VERSION_1	1	X'00000001'

Data types used in the MQI

Information on the data types that can be used in the Message Queue Interface (MQI). Descriptions, fields, and language declarations for relevant languages with each data type.

Introducing data types used in the MQI:

This section introduces the data types used in the MQI, and gives you some guidance on using them in the supported programming languages.

Elementary data types:

This section contains information about data types used in the MQI (or in exit functions). These are described in detail, followed by examples showing how to declare the elementary data types in the supported programming languages in the following topics.

The data types used in the MQI (or in exit functions) are either:

- Elementary data types, or
- Aggregates of elementary data types (arrays or structures)

The following elementary data types are used in the MQI (or in exit functions):

Elementary data type name	Data type	Description
MQBOOL	Boolean	The MQBOOL data type represents a boolean value. The value 0 represents false. Any other value represents true. An MQBOOL must be aligned as for the MQLONG data type.

Elementary data type name	Data type	Description
MQBYTE	Byte	The MQBYTE data type represents a single byte of data. No particular interpretation is placed on the byte; it is treated as a string of bits, and not as a binary number or character. No special alignment is required. When MQBYTE data is sent between queue managers that use different character sets or encodings, the MQBYTE data is not converted in any way. The *MsgId* and *CorrelId* fields in the MQMD structure are like this. An array of MQBYTE is sometimes used to represent an area of main storage that is not known to the queue manager. For example, the area might contain application message data or a structure. The boundary alignment of this area must be compatible with the nature of the data contained within it. In the C programming language, any data type can be used for function parameters that are shown as arrays of MQBYTE. This is because such parameters are always passed by address, and in C the function parameter is declared as a pointer-to-void.
MQBYTEn	String of *n* bytes	Each MQBYTEn data type represents a string of *n* bytes, where *n* can take any of the following values: 8, 16, 24, 32, 40, or 128. Each byte is described by the MQBYTE data type. No special alignment is required. If the data in the byte string is shorter than the defined length of the string, the data must be padded with nulls to fill the string. When the queue manager returns byte strings to the application (for example, on the MQGET call), the queue manager pads with nulls to the defined length of the string. Named constants are available to define the lengths of byte string fields. These are listed in “Constants” on page 2048

Elementary data type name	Data type	Description
MQCHAR	Character	The MQCHAR data type represents a single-byte character, or one byte of a double-byte or multi-byte character. No special alignment is required. When MQCHAR data is sent between queue managers that use different character sets or encodings, the MQCHAR data usually requires conversion in order for the data to be interpreted correctly. The queue manager does this automatically for MQCHAR data in the MQMD structure. Conversion of MQCHAR data in the application message data is controlled by the MQGMO_CONVERT option specified on the MQGET call; see the description of this option in "MQGMO - Get-message options" on page 2330 for further details.
MQCHARn	String of *n* characters	Each MQCHARn data type represents a string of *n* characters, where *n* can take any of the following values: 4, 8, 12, 20, 28, 32, 48, 64, 128, or 256. Each character is described by the MQCHAR data type. No special alignment is required. If the data in the string is shorter than the defined length of the string, the data must be padded with blanks to fill the string. In some cases a null character can be used to end the string prematurely, instead of padding with blanks; the null character and characters following it are treated as blanks, up to the defined length of the string. The places where a null can be used are identified in the call and data type descriptions. When the queue manager returns character strings to the application (for example, on the MQGET call), the queue manager always pads with blanks to the defined length of the string; the queue manager does not use the null character to delimit the string. Named constants are available that define the lengths of character string fields and are listed in "Constants" on page 2048.

Elementary data type name	Data type	Description
MQFLOAT32	32-bit floating point number	The MQFLOAT32 data type is a 32-bit floating-point number represented using the standard IEEE floating-point format. An MQFLOAT32 must be aligned on a 4-byte boundary. The use of MQFLOAT32 in C on z/OS requires the use of the FLOAT(IEEE) compiler flag. The use of MQFLOAT32 in COBOL is limited to compilers that support floating-point numbers in IEEE format. This might require the use of the FLOAT(NATIVE) compiler flag.
MQFLOAT64	64-bit floating point number	The MQFLOAT64 data type is a 64-bit floating-point number represented using the standard IEEE floating-point format. An MQFLOAT64 must be aligned on an 8-byte boundary. The use of MQFLOAT64 in C on z/OS requires the use of the FLOAT(IEEE) compiler flag. The use of MQFLOAT64 in COBOL is limited to compilers that support floating-point numbers in IEEE format. This might require the use of the FLOAT(NATIVE) compiler flag.
MQHCONFIG	Configuration handle	The MQHCONFIG data type represents a configuration handle, that is, the component that is being configured for a particular installable service. A configuration handle must be aligned on its natural boundary. Applications must not rely on the format of the data stored inside this handle. If valid, its value is intended to be usable in further MQI calls, but is not intended to have any meaning besides that purpose.
MQHCONN	Connection handle	The MQHCONN data type represents a connection handle, that is, the connection to a particular queue manager. A connection handle must be aligned on a 4-byte boundary. Applications must not rely on the format of the data stored inside this handle. If valid, its value is intended to be usable in further MQI calls, but is not intended to have any meaning besides that purpose.

Elementary data type name	Data type	Description
MQHMSG	Message handle	The MQHMSG data type represents a message handle that gives access to a message. A message handle must be aligned on an 8-byte boundary. Applications must not rely on the format of the data stored inside this handle. If valid, its value is intended to be usable in further MQI calls, but is not intended to have any meaning besides that purpose.
MQHOBJ	Object handle	The MQHOBJ data type represents an object handle that gives access to an object. An object handle must be aligned on a 4-byte boundary. Applications must not rely on the format of the data stored inside this handle. If valid, its value is intended to be usable in further MQI calls, but is not intended to have any meaning besides that purpose.
MQINT8	8-bit signed integer	The MQINT8 data type is an 8-bit signed integer that can take any value in the range -128 to +127, unless otherwise restricted by the context.
MQINT16	16-bit signed integer	The MQINT16 data type is a 16-bit signed integer that can take any value in the range -32 768 to +32 767, unless otherwise restricted by the context. An MQINT16 must be aligned on a 2-byte boundary.
MQINT32	32-bit signed integer	The MQINT32 data type is a 32-bit signed binary integer that can take any value in the range -2 147 483 648 through +2 147 483 647, unless otherwise restricted by the context. See the definition of MQLONG.
MQINT64	64-bit signed integer	The MQINT64 data type is a 64-bit signed integer that can take any value in the range -9 223 372 036 854 775 808 through +9 223 372 036 854 775 807, unless otherwise restricted by the context. For COBOL, the valid range is limited to -999 999 999 999 999 999 through +999 999 999 999 999 999. A 64-bit integer must be aligned on an 8-byte boundary.

Elementary data type name	Data type	Description
MQLONG	32-bit signed integer	The MQLONG data type is a 32-bit signed binary integer that can take any value in the range -2 147 483 648 through +2 147 483 647, unless otherwise restricted by the context. For COBOL, the valid range is limited to -999 999 999 through +999 999 999. An MQLONG must be aligned on a 4-byte boundary.
MQPID	Process identifier	The IBM MQ process identifier. This is the same identifier used in MQ trace and FFST™™ dumps, but might be different from the operating system process identifier.
MQPTR	Pointer	The MQPTR data type is the address of data of any type. A pointer must be aligned on its natural boundary; this is a 16-byte boundary on IBM i, and an 8-byte boundary on other platforms. Some programming languages support typed pointers; the MQI also uses these in a few cases (for example, PMQCHAR and PMQLONG in the C programming language).
MQTID	Thread identifier	The IBM MQ thread identifier. This is the same identifier used in MQ trace and FFST™ dumps, but might be different from the operating system thread identifier.
MQUINT8	8-bit unsigned integer	The MQUINT8 data type is an 8-bit unsigned integer that can take any value in the range 0 to +255, unless otherwise restricted by the context.
MQUINT16	16-bit unsigned integer	The MQUINT16 data type is a 16-bit unsigned integer that can take any value in the range 0 through +65 535, unless otherwise restricted by the context. An MQUINT16 must be aligned on a 2-byte boundary.
MQUINT32	32-bit unsigned integer	The MQUINT32 data type is a 32-bit unsigned binary integer. See the definition of MQULONG.

Elementary data type name	Data type	Description
MQUINT64	64-bit unsigned integer	The MQINT64 data type is a 64-bit unsigned integer that can take any value in the range 0 through +18 446 744 073 709 551 615, unless otherwise restricted by the context. For COBOL, the valid range is limited to 0 through +999 999 999 999 999 999. A 64-bit integer must be aligned on an 8-byte boundary.
MQULONG	32-bit unsigned integer	The MQULONG data type is a 32-bit unsigned binary integer that can take any value in the range 0 through +4 294 967 294, unless otherwise restricted by the context. For COBOL, the valid range is limited to 0 through +999 999 999. An MQULONG must be aligned on a 4-byte boundary.
PMQACH	Pointer	Pointer to a data structure of type MQACH
PMQAIR	Pointer	Pointer to a data structure of type MQAIR
PMQAXC	Pointer	Pointer to a data structure of type MQAXC
PMQAXP	Pointer	Pointer to a data structure of type MQAXP
PMQBMHO	Pointer	Pointer to a data structure of type MQBMHO
PMQBO	Pointer	Pointer to a data structure of type MQBO
PMQBOOL	Pointer	Pointer to data of type MQBOOL
PMQBYTE	Pointer	Pointer to data of type MQBYTE
PMQBYTEn	Pointer	Pointer to data of type MQBYTEn, where n can be 8, 16, 24, 32, 40, 128
PMQCBC	Pointer	Pointer to a data structure of type MQCBC
PMQCBD	Pointer	Pointer to a data structure of type MQCBD
PMQCHAR	Pointer	Pointer to data of type MQCHAR
PMQCHARN	Pointer	Pointer to a data type of MQCHARN, where n can be 4, 8, 12, 20, 28, 32, 48, 64, 128, 256, 264
PMQCHARV	Pointer	Pointer to a data structure of type MQCHARV
PMQCIH	Pointer	Pointer to a data structure of type MQCIH
PMQCMHO	Pointer	Pointer to a data structure of type MQCMHO

Elementary data type name	Data type	Description
PMQCNO	Pointer	Pointer to a data structure of type MQCNO
PMQCSP	Pointer	Pointer to a data structure of type MQCSP
PMQCTLO	Pointer	Pointer to a data structure of type MQCTLO
PMQDH	Pointer	Pointer to a data structure of type MQDH
PMQDHO	Pointer	Pointer to a data structure of type MQDHO
PMQDLH	Pointer	Pointer to a data structure of type MQDLH
PMQDMHO	Pointer	Pointer to a data structure of type MQDMHO
PMQDMPO	Pointer	Pointer to a data structure of type MQDMPO
PMQEPH	Pointer	Pointer to a data structure of type MQEPH
PMQFLOAT32	Pointer	Pointer to a data structure of type MQFLOAT32
PMQFLOAT64	Pointer	Pointer to a data structure of type MQFLOAT64
PMQFUNC	Pointer	Pointer to a function
PMQGMO	Pointer	Pointer to a data structure of type MQGMO
PMQHCONFIG	Pointer	Pointer to data of type MQHCONFIG
PMQHCONN	Pointer	Pointer to data of type MQHCONN
PMQHMSG	Pointer	Pointer to data of type MQHMSG
PMQHOBJ	Pointer	Pointer to data of type MQHOBJ
PMQIIH	Pointer	Pointer to a data structure of type MQIIH
PMQIMPO	Pointer	Pointer to a data structure of type MQIMPO
PMQINT8	Pointer	Pointer to data of type MQINT8
PMQINT16	Pointer	Pointer to data of type MQINT16
PMQINT32	Pointer	Pointer to data of type MQINT32
PMQINT64	Pointer	Pointer to data of type MQINT64
PMQLONG	Pointer	Pointer to data of type MQLONG
PMQMD	Pointer	Pointer to structure of type MQMD
PMQMDE	Pointer	Pointer to a data structure of type MQMDE
PMQMD1	Pointer	Pointer to a data structure of type MQMD1
PMQMD2	Pointer	Pointer to a data structure of type MQMD2

Elementary data type name	Data type	Description
PMQMHBO	Pointer	Pointer to a data structure of type MQMHBO
PMQOD	Pointer	Pointer to a data structure of type MQOD
PMQOR	Pointer	Pointer to a data structure of type MQOR
PMQPD	Pointer	Pointer to a data structure of type MQPD
PMQPID	Pointer	Pointer to a process identifier
PMQMD	Pointer	Pointer to a data structure of type MQMD
PMQPMO	Pointer	Pointer to a data structure of type MQPMO
PMQPTR	Pointer	Pointer to data of type MQPTR
PMQRFH	Pointer	Pointer to a data structure of type MQRFH
PMQRFH2	Pointer	Pointer to a data structure of type MQRFH2
PMQRMH	Pointer	Pointer to a data structure of type MQRMH
PMQRR	Pointer	Pointer to a data structure of type MQRR
PMQSCO	Pointer	Pointer to a data structure of type MQSCO
PMQSD	Pointer	Pointer to a data structure of type MQSD
PMQSMPO	Pointer	Pointer to a data structure of type MQSMPO
PMQSRO	Pointer	Pointer to a data structure of type MQSRO
PMSSTS	Pointer	Pointer to a data structure of type MQSTS
PMQTID	Pointer	Pointer to a thread ID
PMQTM	Pointer	Pointer to a data structure of type MQTM
PMQTMC2	Pointer	Pointer to a data structure of type MQTMC2
PMQUINT8	Pointer	Pointer to a data type of MQUINT8
PMQUINT16	Pointer	Pointer to a data type of MQUINT16
PMQUINT32	Pointer	Pointer to a data type of MQUINT32
PMQUINT64	Pointer	Pointer to a data type of MQUINT64
PMQULONG	Pointer	Pointer to a data type of MQULONG
PMQVOID	Pointer	
PMQWIH	Pointer	Pointer to a data structure of type MQWIH
PMQXQH	Pointer	Pointer to a data structure of type MQXQH

C declarations:

Data type	Representation
MQBOOL	`typedef MQLONG MQBOOL;`
MQBYTE	`typedef unsigned char MQBYTE;`
MQBYTE8	`typedef MQBYTE MQBYTE8[8];`
MQBYTE16	`typedef MQBYTE MQBYTE16[16];`
MQBYTE24	`typedef MQBYTE MQBYTE24[24];`
MQBYTE32	`typedef MQBYTE MQBYTE32[32];`
MQBYTE40	`typedef MQBYTE MQBYTE40[40];`
MQCHAR	`typedef char MQCHAR;`
MQCHAR4	`typedef MQCHAR MQCHAR4[4];`
MQCHAR8	`typedef MQCHAR MQCHAR8[8];`
MQCHAR12	`typedef MQCHAR MQCHAR12[12];`
MQCHAR20	`typedef MQCHAR MQCHAR20[20];`
MQCHAR28	`typedef MQCHAR MQCHAR28[28];`
MQCHAR32	`typedef MQCHAR MQCHAR32[32];`
MQCHAR48	`typedef MQCHAR MQCHAR48[48];`
MQCHAR64	`typedef MQCHAR MQCHAR64[64];`
MQCHAR128	`typedef MQCHAR MQCHAR128[128];`
MQCHAR256	`typedef MQCHAR MQCHAR256[256];`
MQFLOAT32	`typedef float MQFLOAT32;`
MQFLOAT64	`typedef double MQFLOAT64;`
MQHCONFIG	`typedef void MQPOINTER MQHCONFIG;`
MQHCONN	`typedef MQLONG MQHCONN;`
MQHOBJ	`typedef MQLONG MQHOBJ;`
MQINT8	`typedef signed char MQINT8;`
MQINT16	`typedef short MQINT16;`
MQINT64	On 64-bit UNIX: `typedef long;` On 32-bit AIX, Solaris, and HP-UX: `typedef int64_t;` On IBM i, Linux, and z/OS: `typedef long long;` On Windows: `typedef _int64;`

Data type	Representation
MQLONG	On IBM i: `typedef long MQLONG;` other platforms: `if defined(MQ_64_BIT)` `  typedef int MQLONG;` `else` `  typedef long MQLONG;`
MQPID	`typedef MQLONG MQPID;`
MQPTR	`typedef void MQPOINTER MQPTR;`
MQTID	`typedef MQLONG MQTID;`
MQUINT8	`typedef unsigned char MQUINT8;`
MQUINT16	`typedef unsigned short MQUINT16;`
MQUINT64	On 64-bit UNIX: `typedef unsigned long;` On 32-bit AIX, Solaris, and HP-UX: `typedef uint64_t;` On IBM i, Linux, and z/OS: `typedef unsigned long long;` On Windows: `typedef unsigned _int64;`
MQULONG	On IBM i: `typedef unsigned long MQULONG;` other platforms: `if defined(MQ_64_BIT)` `  typedef unsigned int MQULONG;` `else` `  typedef unsigned long MQULONG;`
PMQBO	`typedef MQBO MQPOINTER PMQBO;`
PMQBOOL	`typedef MQBOOL MQPOINTER PMQBOOL;`
PMQBYTE	`typedef MQBYTE MQPOINTER PMQBYTE;`
PMQBYTE8	`typedef MQBYTE8[8] MQPOINTER PMQBYTE8[8];`
PMQBYTE16	`typedef MQBYTE16[16] MQPOINTER PMQBYTE16[16];`
PMQBYTE24	`typedef MQBYTE24[24] MQPOINTER PMQBYTE24[24];`
PMQBYTE32	`typedef MQBYTE32[32] MQPOINTER PMQBYTE32[32];`
PMQBYTE40	`typedef MQBYTE40[40] MQPOINTER PMQBYTE40[40];`
PMQBYTE128	`typedef MQBYTE128[128] MQPOINTER PMQBYTE128[128];`
PMQCHAR	`typedef MQCHAR MQPOINTER PMQCHAR;`
PMQCHAR4	`typedef MQCHAR4[4] MQPOINTER PMQCHAR4[4];`
PMQCHAR8	`typedef MQCHAR8[8] MQPOINTER PMQCHAR8[8];`
PMQCHAR12	`typedef MQCHAR12[12] MQPOINTER PMQCHAR12[12];`
PMQCHAR20	`typedef MQCHAR20[20] MQPOINTER PMQCHAR20[20];`
PMQCHAR28	`typedef MQCHAR28[28] MQPOINTER PMQCHAR28[28];`

Data type	Representation
PMQCHAR32	typedef MQCHAR32[32] MQPOINTER PMQCHAR32[32];
PMQCHAR48	typedef MQCHAR48[48] MQPOINTER PMQCHAR48[48];
PMQCHAR64	typedef MQCHAR64[64] MQPOINTER PMQCHAR64[64];
PMQCHAR128	typedef MQCHAR128[128] MQPOINTER PMQCHAR128[128];
PMQCHAR256	typedef MQCHAR256[256] MQPOINTER PMQCHAR256[256];
PMQCHAR264	typedef MQCHAR264[264] MQPOINTER PMQCHAR264[264];
PMQCIH	typedef MQCIH MQPOINTER PMQCIH;
PMQCNO	typedef MQCNO MQPOINTER PMQCNO;
PMQDLH	typedef MQDLH MQPOINTER PMQDLH;
PMQFUNC	typedef void MQPOINTER PMQFUNC;
PMQFLOAT32	typedef MQFLOAT32 MQPOINTER PMQFLOAT32;
PMQFLOAT64	typedef MQFLOAT64 MQPOINTER PMQFLOAT64;
PMQGMO	typedef MQGMO MQPOINTER PMQGMO;
PMQHCONFIG	typedef MQHCONFIG MQPOINTER PMQHCONFIG;
PMQHCONN	typedef MQHCONN MQPOINTER PMQHCONN;
PMQHOBJ	typedef MQHOBJ MQPOINTER PMQHOBJ;
PMQIIH	typedef MQIIH MQPOINTER PMQIIH;
PMQINT8	typedef MQINT8 MQPOINTER PMQINT8;
PMQINT16	typedef MQINT16 MQPOINTER PMQINT16;
PMQLONG	typedef MQLONG MQPOINTER PMQLONG;
PMQMD	typedef MQMD MQPOINTER PMQMD;
PMQMD1	typedef MQMD1[1] MQPOINTER PMQMD1[1];
PMQMDE	typedef MQMDE MQPOINTER PMQMDE;
PMQOD	typedef MQOD MQPOINTER PMQOD;
PMQPMO	typedef MQPMO MQPOINTER PMQPMO;
PMQPTR	typedef MQPTR MQPOINTER PMQPTR;
PMQRFH	typedef MQRFH MQPOINTER PMQRFH;
PMQRFH2	typedef MQRFH2[2] MQPOINTER PMQRFH2[2];
PMQRMH	typedef MQRMH MQPOINTER PMQRMH;
PMQTM	typedef MQTM MQPOINTER PMQTM;
PMQTMC2	typedef MQTMC2[2] MQPOINTER PMQTMC2[2];
PMQUINT8	typedef MQUINT8 MQPOINTER PMQUINT8;
PMQUINT16	typedef MQUINT16 MQPOINTER PMQUINT16;
PMQULONG	typedef MQULONG MQPOINTER PMQULONG;
PMQVOID	typedef void MQPOINTER PMQVOID;
PMQWIH	typedef MQWIH MQPOINTER PMQWIH;
PMQXQH	typedef MQXQH MQPOINTER PMQXQH;
PPMQBO	typedef PMQBO MQPOINTER PPMQBO;
PPMQBYTE	typedef PMQBYTE MQPOINTER PPMQBYTE;
PPMQCHAR	typedef PMQCHAR MQPOINTER PPMQCHAR;
PPMQCNO	typedef PMQCNO MQPOINTER PPMQCNO;

Data type	Representation
PPMQGMO	`typedef PMQGMO MQPOINTER PPMQGMO;`
PPMQHCONN	`typedef PMQHCONN MQPOINTER PPMQHCONN;`
PPMQHOBJ	`typedef PMQHOBJ MQPOINTER PPMQHOBJ;`
PPMQLONG	`typedef PMQLONG MQPOINTER PPMQLONG;`
PPMQMD	`typedef PMQMD MQPOINTER PPMQMD;`
PPMQOD	`typedef PMQOD MQPOINTER PPMQOD;`
PPMQPMO	`typedef PMQPMO MQPOINTER PPMQPMO;`
PPMQULONG	`typedef PMQULONG MQPOINTER PPMQULONG;`
PPMQVOID	`typedef PMQVOID MQPOINTER PPMQVOID;`
Where `defined(MQ_64_BIT)` means a 64 bit platform.	

See "Data types" on page 2215 for a description of the MQPOINTER macro variable.

COBOL declarations:

Data type	Representation
MQBOOL	`PIC S9(9) BINARY`
MQBYTE	`PIC X`
MQBYTE8	`PIC X(8)`
MQBYTE16	`PIC X(16)`
MQBYTE24	`PIC X(24)`
MQBYTE32	`PIC X(32)`
MQBYTE40	`PIC X(40)`
MQCHAR	`PIC X`
MQCHAR4	`PIC X(4)`
MQCHAR8	`PIC X(8)`
MQCHAR12	`PIC X(12)`
MQCHAR20	`PIC X(20)`
MQCHAR28	`PIC X(28)`
MQCHAR32	`PIC X(32)`
MQCHAR48	`PIC X(48)`
MQCHAR64	`PIC X(64)`
MQCHAR128	`PIC X(128)`
MQCHAR256	`PIC X(256)`
MQFLOAT32	`USAGE COMP-1`
MQFLOAT64	`USAGE COMP-2`
MQHCONN	`PIC S9(9) BINARY`
MQHOBJ	`PIC S9(9) BINARY`
MQINT8	`PIC S9(2) BINARY`
MQINT16	`PIC S9(4) BINARY`
MQINT64	`PIC S9(18) BINARY`
MQLONG	`PIC S9(9) BINARY`

Data type	Representation
MQPTR	POINTER
MQUINT8	PIC 9(2) BINARY
MQUINT16	PIC 9(4) BINARY
MQUINT64	PIC 9(18) BINARY
MQULONG	PIC 9(9) BINARY

PL/I declarations:

PL/I is supported on z/OS.

Data type	Representation
MQBOOL	fixed bin(31)
MQBYTE	char(1)
MQBYTE8	char(8)
MQBYTE16	char(16)
MQBYTE24	char(24)
MQBYTE32	char(32)
MQBYTE40	char(40)
MQCHAR	char(1)
MQCHAR4	char(4)
MQCHAR8	char(8)
MQCHAR12	char(12)
MQCHAR20	char(20)
MQCHAR28	char(28)
MQCHAR32	char(32)
MQCHAR48	char(48)
MQCHAR64	char(64)
MQCHAR128	char(128)
MQCHAR256	char(256)
MQFLOAT32	binary float(21) ieee
MQFLOAT64	binary float(52) ieee
MQHCONN	fixed bin(31)
MQHOBJ	fixed bin(31)
MQINT8	fixed bin(7)
MQINT16	fixed bin(15)
MQINT64	fixed bin(63)
MQLONG	fixed bin(31)
MQPTR	pointer
MQUINT8	fixed bin(8)
MQUINT16	fixed bin(16)
MQUINT64	fixed bin(64)
MQULONG	fixed bin(32)

System/390 assembler declarations:

System/390 assembler is supported on z/OS only.

Data type	Representation
MQBOOL	`DS F`
MQBYTE	`DS XL1`
MQBYTE8	`DS XL8`
MQBYTE16	`DS XL16`
MQBYTE24	`DS XL24`
MQBYTE32	`DS XL32`
MQBYTE40	`DS XL40`
MQCHAR	`DS CL1`
MQCHAR4	`DS CL4`
MQCHAR8	`DS CL8`
MQCHAR12	`DS CL12`
MQCHAR20	`DS CL20`
MQCHAR28	`DS CL28`
MQCHAR32	`DS CL32`
MQCHAR48	`DS CL48`
MQCHAR64	`DS CL64`
MQCHAR128	`DS CL128`
MQCHAR256	`DS CL256`
MQFLOAT32	`DS EB`
MQFLOAT64	`DS DB`
MQHCONN	`DS F`
MQHOBJ	`DS F`
MQINT8	`DS XL1`
MQINT16	`DS H`
MQINT64	`DS D`
MQLONG	`DS F`
MQPTR	`DS F`
MQUINT8	`DS XL1`
MQUINT16	`DS H`
MQUINT64	`DS D`
MQULONG	`DS F`

Structure data types - introduction:

This section introduces the structure data types used in the MQI. The structure data types themselves are described in subsequent sections.

Summary:

The following tables summarize the structure data types used in the MQI.

Table 165. Structure data types used on MQI calls (or exit functions):

Structure	Description	Calls where used
MQACH	API exit chain header	
MQAIR	Authentication information record	MQCONNX
MQAXC	API exit context	
MQAXP	API exit parameter	
MQBMHO	Buffer to message handle options	MQBUFMH
MQBO	Begin options	MQBEGIN
MQCBD	Callback descriptor	MQCB
MQCBO	Create-bag options	mqCreateBag
MQCHARV	Variable length string	MQINQMP
MQCNO	Connect options	MQCONNX
MQCSP	Security parameters	MQCONNX
MQCTLO	Callback options	MQCTL
MQDMPO	Delete message property options	MQDLTMP
MQGMO	Get-message options	MQGET
MQIMPO	Inquire message property options	MQINQMP
MQMD	Message descriptor	MQBUFMH, MQMHBUF, MQCB, MQGET, MQPUT, MQPUT1
MQMHBO	Message handle to buffer options	MQMHBUF
MQOD	Object descriptor	MQOPEN, MQPUT1
MQOR	Object record	MQOPEN, MQPUT1
MQPD	Property descriptor	MQSETMP
MQPMO	Put-message options	MQPUT, MQPUT1
MQPMR	Put-message record	MQPUT, MQPUT1
MQRR	Response record	MQOPEN, MQPUT, MQPUT1
MQSCO	TLS configuration options	MQCONNX
MQSD	Subscription descriptor	MQSUB
MQSMPO	Set message property option	MQSETMP
MQSRO	Subscription request options	MQSUBRQ
MQSTS	Status reporting structure	MQSTAT

Table 166. Structure data types used in message data:

Structure	Description
MQCIH	CICS information header
MQCFH	PCF header
MQEPH	Embedded PCF header
MQDH	Distribution header
MQDLH	Dead letter (undelivered message) header
MQIIH	IMS information header
MQMDE	Message descriptor extension
MQRFH	Rules and formatting header
MQRFH2	Rules and formatting header 2
MQRMH	Reference message header
MQTM	Trigger message
MQTMC2	Trigger message (character format 2)
MQWIH	Work information header
MQXQH	Transmission queue header

Note: The MQDXP structure (data conversion exit parameter) is described in "Data conversion" on page 2909, together with the associated data conversion calls.

Rules for structure data types:

Programming languages vary in their level of support for structures, and certain rules and conventions are adopted to map the MQI structures consistently in each programming language:

1. Structures must be aligned on their natural boundaries.
   - Most MQI structures require 4-byte alignment.
   - On IBM i, structures containing pointers require 16-byte alignment; these are: MQCNO, MQOD, MQPMO.
2. Each field in a structure must be aligned on its natural boundary.
   - Fields with data types that equate to MQLONG must be aligned on 4-byte boundaries.
   - Fields with data types that equate to MQPTR must be aligned on 16-byte boundaries on IBM i, and 4-byte boundaries in other environments.
   - Other fields are aligned on 1-byte boundaries.
3. The length of a structure must be a multiple of its boundary alignment.
   - Most MQI structures have lengths that are multiples of 4 bytes.
   - On IBM i, structures containing pointers have lengths that are multiples of 16 bytes.
4. Where necessary, padding bytes or fields must be added to ensure compliance with the preceding rules.

Conventions used in the descriptions:

The description of each structure data type includes:

- An overview of the purpose and use of the structure
- Descriptions of the fields in the structure, in a form that is independent of the programming language
- Examples of how the structure is declared in each of the supported programming languages

The description of each structure data type contains the following sections:

Structure name
The name of the structure, followed by a summary of the fields in the structure.

Overview
A brief description of the purpose and use of the structure.

Fields Descriptions of the fields. For each field, the name of the field is followed by its elementary data type in parentheses ( ). In text, field names are shown using an italic typeface; for example *Version*.

There is also a description of the purpose of the field, together with a list of any values that the field can take. Names of constants are shown in uppercase; for example MQGMO_STRUC_ID. A set of constants having the same prefix is shown using the * character, for example: MQIA_*.

In the descriptions of the fields, the following terms are used:

input You supply information in the field when you make a call.

output
The queue manager returns information in the field when the call completes or fails.

input/output
You supply information in the field when you make a call, and the queue manager changes the information when the call completes or fails.

Initial values
A table showing the initial values for each field in the data definition files supplied with the MQI.

C declaration
Typical declaration of the structure in C.

COBOL declaration
Typical declaration of the structure in COBOL.

PL/I declaration
Typical declaration of the structure in PL/I.

System/390 assembler declaration
Typical declaration of the structure in System/390 assembler language.

Visual Basic declaration
Typical declaration of the structure in Visual Basic.

C programming:

This section contains information to help you use the MQI from the C programming language.

Header files:

Header files are provided to help you write C application programs that use the MQI.

These header files are summarized in Table 167.

Table 167. C header files

File	Contents
CMQC	Function prototypes, data types, and named constants for the main MQI
CMQXC	Function prototypes, data types, and named constants for the data conversion exit
CMQEC	Function prototypes, data types, and named constants for the main MQI, data conversion exit and Interface Entry Points structure (CMQEC includes CMQXC and CMQC.)
CMQSTRC	Functions that convert MQI constant definitions to the text equivalent. **Attention:** z/OS Not applicable to z/OS.

To improve the portability of applications, code the name of the header file in lowercase on the `#include` preprocessor directive:

```
#include "cmqec.h"
```

Functions:

You do not need to specify all parameters that are passed by address every time you invoke a function.

- Pass parameters that are *input-only* and of type MQHCONN, MQHOBJ, or MQLONG by value.
- Pass all other parameters by address.

Where a particular parameter is not required, use a null pointer as the parameter on the function invocation, in place of the address of the parameter data. Parameters for which this is possible are identified in the call descriptions.

No parameter is returned as the value of the function; in C terminology, this means that all functions return `void`.

The attributes of the function are defined by the MQENTRY macro variable; the value of this macro variable depends on the environment.

Parameters with undefined data type:

The **Buffer** parameter on the MQGET, MQPUT, and MQPUT1 functions has an undefined data type. This parameter is used to send and receive the application's message data.

Parameters of this sort are shown in the C examples as arrays of MQBYTE. You can declare the parameters in this way, but it is usually more convenient to declare them as the particular structure that describes the layout of the data in the message. Declare the actual function parameter as a pointer-to-void, and specify the address of any sort of data as the parameter on the function invocation.

Data types:

Define all data types using the C `typedef` statement. For each data type, also define the corresponding pointer data type. The name of the pointer data type is the name of the elementary or structure data type prefixed with the letter P to denote a pointer. Define the attributes of the pointer using the MQPOINTER macro variable; the value of this macro variable depends on the environment. The following illustrates how to declare pointer data types:

```
#define MQPOINTER *               /* depends on environment */
...
typedef MQLONG MQPOINTER PMQLONG;  /* pointer to MQLONG */
typedef MQMD   MQPOINTER PMQMD;    /* pointer to MQMD */
```

Manipulating binary strings:

Declare strings of binary data as one of the MQBYTEn data types.

Whenever you copy, compare, or set fields of this type, use the C functions **memcpy**, **memcmp**, or **memset** ; for example:

```
#include <string.h>
#include "cmqc.h"

MQMD MyMsgDesc;

memcpy(MyMsgDesc.MsgId,           /* set "MsgId" field to nulls    */
       MQMI_NONE,                 /* ...using named constant       */
       sizeof(MyMsgDesc.MsgId));

memset(MyMsgDesc.CorrelId,        /* set "CorrelId" field to nulls */
       0x00,                      /* ...using a different method   */
       sizeof(MQBYTE24));
```

Do not use the string functions **strcpy**, **strcmp**, **strncpy**, or **strncmp**, because these do not work correctly for data declared with the MQBYTEn data types.

Manipulating character strings:

When the queue manager returns character data to the application, the queue manager always pads the character data with blanks to the defined length of the field; the queue manager *does not* return null-terminated strings.

Therefore, when copying, comparing, or concatenating such strings, use the string functions **strncpy**, **strncmp**, or **strncat**.

Do not use the string functions that require the string to be terminated by a null (**strcpy**, **strcmp**, **strcat**). Also, do not use the function **strlen** to determine the length of the string; use instead the **sizeof** function to determine the length of the field.

Initial values for structures:

The header files define various macro variables that you can use to provide initial values for the MQ structures when you declare instances of those structures.

These macro variables have names of the form MQxxx_DEFAULT, where MQxxx represents the name of the structure. They are used in the following way:

```
MQMD   MyMsgDesc = {MQMD_DEFAULT};
MQPMO  MyPutOpts = {MQPMO_DEFAULT};
```

For some character fields (for example, the *StrucId* fields that occur in most structures, or the *Format* field that occurs in MQMD), the MQI defines particular values that are valid. For each of the valid values, *two* macro variables are provided:

- One macro variable defines the value as a string with a length, excluding the implied null matches, exactly the defined length of the field. For example, for the *Format* field in MQMD the following macro variable is provided (¬ represents a blank character):

  ```
  #define MQFMT_STRING "MQSTR¬¬¬"
  ```

  Use this form with the memcpy and memcmp functions.
- The other macro variable defines the value as an array of characters; the name of this macro variable is the name of the string form suffixed with _ARRAY. For example:

  ```
  #define MQFMT_STRING_ARRAY 'M','Q','S','T','R','⌂','⌂','⌂'
  ```

  Use this form to initialize the field when you declare an instance of the structure with values different from those provided by the MQMD_DEFAULT macro variable. (This is not always necessary; in some environments you can use the string form of the value in both situations. However, you can use the array form for declarations, because this is required for compatibility with the C++ programming language.)

Initial values for dynamic structures:

When a variable number of instances of a structure is required, the instances are typically created in main storage obtained dynamically using the `calloc` or `malloc` functions. To initialize the fields in such structures, consider the following technique:

1. Declare an instance of the structure using the appropriate MQxxx_DEFAULT macro variable to initialize the structure. This instance becomes the model for other instances:

   ```
   MQMD Model = {MQMD_DEFAULT}; /* declare model instance */
   ```

   The `static` or `auto` keywords can be coded on the declaration in order to give the model instance static or dynamic lifetime, as required.
2. Use the `calloc` or `malloc` functions to obtain storage for a dynamic instance of the structure:

   ```
   PMQMD Instance;
   Instance = malloc(sizeof(MQMD)); /* get storage for dynamic instance */
   ```
3. Use the `memcpy` function to copy the model instance to the dynamic instance:

   ```
   memcpy(Instance,&Model,sizeof(MQMD)); /* initialize dynamic instance */
   ```

Use from C++:

For the C++ programming language, the header files contain the following additional statements that are included only when you use a C++ compiler:

```
#ifdef __cplusplus
  extern "C" {
#endif

/* rest of header file */

#ifdef __cplusplus
  }
#endif
```

Notational conventions:

This information shows how to invoke the functions and declare parameters.

In some cases, the parameters are arrays with a size that is not fixed. For these, a lowercase n is used to represent a numeric constant. When you code the declaration for that parameter, replace the n with the numeric value required.

COBOL programming:

This section contains information to help you use the MQI from the COBOL programming language.

COPY files:

Various COPY files are provided to help you write COBOL application programs that use the MQI. There are two files containing named constants, and two files for each of the structures.

Each structure is provided in two forms: a form with initial values, and a form without:

- Use the structures with initial values in the WORKING-STORAGE SECTION of a COBOL program; they are contained in COPY files with names suffixed with the letter V (for Values).
- Use the structures without initial values in the `LINKAGE SECTION` of a COBOL program; they are contained in COPY files with names suffixed with the letter L (for Linkage).

The COPY files are summarized in Table 168. Not all the files listed are available in all environments.

Table 168. COBOL COPY files

File (with initial values)	File (without initial values)	Contents
CMQAIRV	CMQAIRL	Authentication information record
CMQBOV	CMQBOL	Begin options structure
CMQCIHV	CMQCIHL	CICS information header structure
CMQCNOV	CMQCNOL	Connect options structure
CMQDHV	CMQDHL	Distribution header structure
CMQDLHV	CMQDLHL	Dead letter header structure
CMQDXPV	CMQDXPL	Data conversion exit parameter structure
CMQGMOV	CMQGMOL	Get message options structure
CMQIIHV	CMQIIHL	IMS information header structure
CMQMDV	CMQMDL	Message descriptor structure
CMQMDEV	CMQMDEL	Message descriptor extension structure
CMQMD1V	CMQMD1L	Message descriptor structure version 1
CMQODV	CMQODL	Object descriptor structure
CMQORV	CMQORL	Object record structure
CMQPMOV	CMQPMOL	Put message options structure
CMQRFHV	CMQRFHL	Rules and formatting header structure
CMQRFH2V	CMQRFH2L	Rules and formatting header structure version 2
CMQRMHV	CMQRMHL	Reference message header structure
CMQRRV	CMQRRL	Response record structure
CMQSCOV	CMQSCOL	TLS configuration options
CMQTMV	CMQTML	Trigger message structure
CMQTMCV	CMQTMCL	Trigger message structure (character format)
CMQTMC2V	CMQTMC2L	Trigger message structure (character format) version 2
CMQWIHV	CMQWIHL	Work information header structure
CMQXQHV	CMQXQHL	Transmission queue header structure
CMQV	-	Named constants for main MQI
CMQXV	-	Named constants for data conversion exit
CMQMD2V	CMQMD2L	Message descriptor structure version 2

Structures:

In the COPY file, each structure declaration begins with a level-10 item; this enables you to declare several instances of the structure by coding the level-01 declaration and then using the `COPY` statement to copy in the remainder of the structure declaration. To refer to the appropriate instance, use the `IN` keyword:

```
* Declare two instances of MQMD
 01  MY-MQMD.
     COPY CMQMDV.
 01  MY-OTHER-MQMD.
     COPY CMQMDV.
*
* Set MSGTYPE field in MY-OTHER-MQMD
     MOVE MQMT-REQUEST TO MQMD-MSGTYPE IN MY-OTHER-MQMD.
```

Align the structures on appropriate boundaries. If you use the `COPY` statement to include a structure following an item that is not the level-01 item, ensure that the structure begins at the appropriate offset from the start of the level-01 item. Most MQI structures require 4-byte alignment; the exceptions to this are MQCNO, MQOD, and MQPMO, which require 16-byte alignment on IBM i.

In this section, the names of fields in structures are shown without a prefix. In COBOL, the field names are prefixed with the name of the structure followed by a hyphen. However, if the structure name ends with a numeric digit, indicating that the structure is a second or later version of the original structure, the numeric digit is omitted from the prefix. Field names in COBOL are shown in uppercase (although lowercase or mixed case can be used if required). For example, the field *MsgType* described in "MQMD - Message descriptor" on page 2387 becomes MQMD-MSGTYPE in COBOL.

The V-suffix structures are declared with initial values for all the fields; you need to set only those fields where you want a value that is different from the supplied initial value.

Pointers:

Some structures need to address optional data that might be discontiguous with the structure, such as the MQOR and MQRR records addressed by the MQOD structure.

To address this optional data, the structures contain fields that are declared with the pointer data type. However, COBOL does not support the pointer data type in all environments. Because of this, the optional data can also be addressed using fields that contain the offset of the data from the start of the structure.

If you want to port an application between environments, ascertain whether the pointer data type is available in all the intended environments. If it is not, the application must address the optional data using the offset fields instead of the pointer fields.

In those environments where pointers are not supported, declare the pointer fields as byte strings of the appropriate length, with the initial value being the all-null byte string. Do not alter this initial value if you are using the offset fields.

Named constants:

In this section, the names of constants are shown containing the underscore character (_) as part of the name. In COBOL, use the hyphen character (-) in place of the underscore.

Constants that have character-string values use the single quotation mark as the string delimiter ('). In some environments, you might have to specify an appropriate compiler option to cause the compiler to accept the single quotation mark as the string delimiter in place of the double quotation mark.

The named constants are declared in the COPY files as level-10 items. To use the constants, declare the level-01 item explicitly, and then use the `COPY` statement to copy in the declarations of the constants:

```
* Declare a structure to hold the constants
 01  MY-MQ-CONSTANTS.
     COPY CMQV.
```

The preceding method causes the constants to occupy storage in the program even if they are not referenced. If you include the constants in many separate programs within the same run unit, multiple copies of the constants exist, consuming main storage unnecessarily. Avoid this effect by using one of the following techniques:

- Add the `GLOBAL` clause to the level-01 declaration:

  ```
  * Declare a global structure to hold the constants
   01  MY-MQ-CONSTANTS GLOBAL.
       COPY CMQV.
  ```

  This causes allocates storage for only one set of constants within the run unit. The constants, however, can be referenced by any program within the run unit, not just the program that contains the level-01 declaration.

  Note: The `GLOBAL` clause is not supported in all environments.
- Manually copy into each program only those constants that are referenced by that program. Do not use the `COPY` statement to copy all the constants into the program.

Notational conventions:

The latter topics in this section show how to invoke the calls and declare parameters. In some cases, the parameters are tables or character strings the size of which is not fixed. For these, a lowercase n is used to represent a numeric constant. When you code the declaration for that parameter, replace the n with the numeric value required.

System/390 assembler programming:

This section contains information to help to you use the MQI from the System/390 Assembler programming language.

Macros:

Various macros are provided to help you to write assembler application programs that use the MQI.

There are two macros for named constants, and one macro for each of the structures. These files are summarized in Table 169.

Table 169. Assembler macros

File	Contents
CMQA	Named constants (equates) for main MQI
CMQCIHA	CICS information header structure
CMQCNOA	Connect options structure
CMQDLHA	Dead letter header structure
CMQDXPA	Data conversion exit parameter structure
CMQGMOA	Get message options structure
CMQIIHA	IMS information header structure
CMQMDA	Message descriptor structure
CMQMDEA	Message descriptor extension structure
CMQODA	Object descriptor structure
CMQPMOA	Put message options structure
CMQRFHA	Rules and formatting header structure
CMQRFH2A	Rules and formatting header structure version 2
CMQRMHA	Reference message header structure
CMQTMA	Trigger message structure
CMQTMC2A	Trigger message structure (character format) version 2
CMQVERA	Structure version control
CMQWIHA	Work information header structure
CMQXA	Named constants for data conversion exit
CMQXPA	API crossing exit parameter structure
CMQXQHA	Transmission queue header structure

Structures:

The structures are generated by macros that have various parameters to control the action of the macro. These parameters are described in the following sections.

From time to time new versions of the MQ structures are introduced. The additional fields in a new version can cause a structure that previously was smaller than 256 bytes to become larger than 256 bytes. Because of this, write assembler instructions that are intended to copy an MQ structure, or to set an MQ structure to nulls, to work correctly with structures that might be larger than 256 bytes. Alternatively, use the `DCLVER` macro parameter or CMQVERA macro with the `VERSION` parameter to declare a specific version of the structure.

Specifying the name of the structure:

To declare more than one instance of a structure, the macro prefixes the name of each field in the structure with a user-specifiable string and an underscore.

The string used is the label specified on the invocation of the macro. If no label is specified, the name of the structure is used to construct the prefix:

```
* Declare two object descriptors
         CMQODA  ,        Prefix used="MQOD_" (the default)
MY_MQOD  CMQODA  ,        Prefix used="MY_MQOD_"
```

The structure declarations shown in this section use the default prefix.

Specifying the form of the structure:

Structure declarations can be generated by the macro in one of two forms, controlled by the DSECT parameter:

DSECT=YES
: An assembler DSECT instruction is used to start a new data section; the structure definition immediately follows the DSECT statement. The label on the macro invocation is used as the name of the data section; if no label is specified, the name of the structure is used.

DSECT=NO
: Assembler DC instructions are used to define the structure at the current position in the routine. The fields are initialized with values, which can be specified by coding the relevant parameters on the macro invocation. Fields for which no values are specified on the macro invocation are initialized with default values.

The value specified must be uppercase. If the DSECT parameter is not specified, DSECT=NO is assumed.

Controlling the version of the structure:

By default, the macros always declare the most recent version of each structure.

Although you can use the VERSION macro parameter to specify a value for the *Version* field in the structure, that parameter defines the initial value for the *Version* field, and does not control the version of the structure actually declared. To control the version of the structure that is declared, use the DCLVER parameter:

DCLVER=CURRENT
: The version declared is the current (most recent) version.

DCLVER=SPECIFIED
: The version declared is the version specified by the VERSION parameter. If you omit the VERSION parameter, the default is version 1.

  If you specify the VERSION parameter, the value must be a self-defining numeric constant, or the named constant for the version required (for example, MQCNO_VERSION_3). If you specify some other value, the structure is declared as if DCLVER=CURRENT had been specified, even if the value of VERSION resolves to a valid value.

The value specified must be uppercase. If you omit the DCLVER parameter, the value used is taken from the MQDCLVER global macro variable. You can set this variable using the CMQVERA macro.

Declaring one structure embedded within another:

To declare one structure as a component of another structure, use the NESTED parameter:

NESTED=YES
The structure declaration is nested within another.

NESTED=NO
The structure declaration is not nested within another.

The value specified must be uppercase. If you omit the NESTED parameter, NESTED=NO is assumed.

Specifying initial values for fields:

Specify the value to be used to initialize a field in a structure by coding the name of that field (without the prefix) as a parameter on the macro invocation, accompanied by the value required.

For example, to declare a message-descriptor structure with the *MsgType* field initialized with MQMT_REQUEST, and the *ReplyToQ* field initialized with the string "MY_REPLY_TO_QUEUE", use the following:

```
MY_MQMD  CMQMDA  MSGTYPE=MQMT_REQUEST,                              X
               REPLYTOQ=MY_REPLY_TO_QUEUE
```

If you specify a named constant (equate) as a value on the macro invocation, use the CMQA macro to define the named constant. Do not enclose character string values in single quotation marks.

Controlling the listing:

Control the appearance of the structure declaration in the assembler listing using the LIST parameter:

LIST=YES
The structure declaration appears in the assembler listing.

LIST=NO
The structure declaration does not appear in the assembler listing.

The value specified must be uppercase. If you omit the LIST parameter, LIST=NO is assumed.

CMQVERA macro:

This macro allows you to set the default value to be used for the DCLVER parameter on the structure macros. The value specified by CMQVERA is used by the structure macro only if you omit the DCLVER parameter from the invocation of the structure macro. The default value is set by coding the CMQVERA macro with the DCLVER parameter:

DCLVER=CURRENT
The default version is set to the current (most recent) version.

DCLVER=SPECIFIED
The default version is set to the version specified by the VERSION parameter.

You must specify the **DCLVER** parameter, and the value must be uppercase. The value set by CMQVERA remains the default value until the next invocation of CMQVERA, or the end of the assembly. If you omit CMQVERA, the default is DCLVER=CURRENT.

Notational conventions:

Later sections show how to invoke the calls and declare parameters. In some cases, the parameters are arrays or character strings with a size that is not fixed for which, a lowercase n is used to represent a numeric constant. When you code the declaration for that parameter, replace the n with the numeric value required.

MQAIR - Authentication information record:

The MQAIR structure represents the authentication information record.

The following table summarizes the fields in the structure.

Table 170. Fields in MQAIR

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
AuthInfoType	Type of authentication information	AuthInfoType
AuthInfoConnName	Connection name of LDAP CRL server	AuthInfoConnName
LDAPUserNamePtr	Address of LDAP user name	LDAPUserNamePtr
LDAPUserNameOffset	Offset of LDAP user name from start of MQSCO	LDAPUserNameOffset
LDAPUserNameLength	Length of LDAP user name	LDAPUserNameLength
LDAPPassword	Password to access LDAP server	LDAPPassword
Note: The remaining fields are ignored if *Version* is less than MQAIR_VERSION_2.		
OCSPResponderURL	URL at which the OCSP responder can be contacted	OCSPResponderURL

Overview for MQAIR:

The MQAIR structure allows an application running as an IBM MQ MQI client to specify information about an authenticator that is to be used for the client connection. The structure is an input parameter on the MQCONNX call.

Availability: AIX, HP-UX, Solaris, Linux and Windows clients.

Character set and encoding: Data in MQAIR must be in the character set and encoding of the local queue manager; these are given by the **CodedCharSetId** queue manager attribute and MQENC_NATIVE.

Fields for MQAIR:

The MQAIR structure contains the following fields; the fields are described in **alphabetical order**:

AuthInfoConnName (MQCHAR264):

This is either the host name or the network address of a host on which the LDAP server is running. This can be followed by an optional port number, enclosed in parentheses. The default port number is 389.

If the value is shorter than the length of the field, terminate the value with a null character, or pad it with blanks to the length of the field. If the value is not valid, the call fails with reason code MQRC_AUTH_INFO_CONN_NAME_ERROR.

This is an input field. The length of this field is given by MQ_AUTH_INFO_CONN_NAME_LENGTH. The initial value of this field is the null string in C, and blank characters in other programming languages.

AuthInfoType (MQLONG):

This is the type of authentication information contained in the record.

The value can be one of the two following parameters:

MQAIT_CRL_LDAP
Certificate revocation checking using LDAP server.

MQAIT_OCSP
Certificate revocation checking using OCSP.

If the value is not valid, the call fails with reason code MQRC_AUTH_INFO_TYPE_ERROR.

This is an input field. The initial value of this field is MQAIT_CRL_LDAP.

LDAPPassword (MQCHAR32):

This is the password needed to access the LDAP CRL server. If the value is shorter than the length of the field, terminate the value with a null character, or pad it with blanks to the length of the field.

If the LDAP server does not require a password, or you omit the LDAP user name, *LDAPPassword* must be null or blank. If you omit the LDAP user name and *LDAPPassword* is not null or blank, the call fails with reason code MQRC_LDAP_PASSWORD_ERROR.

This is an input field. The length of this field is given by MQ_LDAP_PASSWORD_LENGTH. The initial value of this field is the null string in C, and blank characters in other programming languages.

LDAPUserNameLength (MQLONG):

This is the length in bytes of the LDAP user name addressed by the *LDAPUserNamePtr* or *LDAPUserNameOffset* field. The value must be in the range zero through MQ_DISTINGUISHED_NAME_LENGTH. If the value is not valid, the call fails with reason code MQRC_LDAP_USER_NAME_LENGTH_ERR.

If the LDAP server involved does not require a user name, set this field to zero.

This is an input field. The initial value of this field is 0.

LDAPUserNameOffset (MQLONG):

This is the offset in bytes of the LDAP user name from the start of the MQAIR structure.

The offset can be positive or negative. The field is ignored if *LDAPUserNameLength* is zero.

You can use either *LDAPUserNamePtr* or *LDAPUserNameOffset* to specify the LDAP user name, but not both; see the description of the *LDAPUserNamePtr* field for details.

This is an input field. The initial value of this field is 0.

LDAPUserNamePtr (PMQCHAR):

This is the LDAP user name.

It consists of the Distinguished Name of the user who is attempting to access the LDAP CRL server. If the value is shorter than the length specified by *LDAPUserNameLength*, terminate the value with a null character, or pad it with blanks to the length *LDAPUserNameLength*. The field is ignored if *LDAPUserNameLength* is zero.

You can supply the LDAP user name in one of two ways:

- By using the pointer field *LDAPUserNamePtr*

  In this case, the application can declare a string that is separate from the MQAIR structure, and set *LDAPUserNamePtr* to the address of the string.

  Consider using *LDAPUserNamePtr* for programming languages that support the pointer data type in a fashion that is portable to different environments (for example, the C programming language).
- By using the offset field *LDAPUserNameOffset*

  In this case, the application must declare a compound structure containing the MQSCO structure followed by the array of MQAIR records followed by the LDAP user name strings, and set *LDAPUserNameOffset* to the offset of the appropriate name string from the start of the MQAIR structure. Ensure that this value is correct, and has a value that can be accommodated within an MQLONG (the most restrictive programming language is COBOL, for which the valid range is -999 999 999 through +999 999 999).

  Consider using *LDAPUserNameOffset* for programming languages that do not support the pointer data type, or that implement the pointer data type in a fashion that might not be portable to different environments (for example, the COBOL programming language).

Whichever technique is chosen, use only one of *LDAPUserNamePtr* and *LDAPUserNameOffset* ; the call fails with reason code MQRC_LDAP_USER_NAME_ERROR if both are nonzero.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

OCSPResponderURL (MQCHAR256):

For an MQAIR structure that represents connection details for an OCSP responder, this field contains the URL at which the responder can be contacted.

The value of this field is an HTTP URL. This field takes priority over a URL in an AuthorityInfoAccess (AIA) certificate extension.

The value is ignored unless both the following statements are true:

- The MQAIR structure is Version 2 or later (the Version field is set to MQAIR_VERSION_2 or greater).
- The AuthInfoType field is set to MQAIT_OCSP.

If the field does not contain an HTTP URL in the correct format (and is not being ignored), the MQCONNX call fails with reason code MQRC_OCSP_URL_ERROR.

This field is case-sensitive. It must start with the string http:// in lowercase. The rest of the URL might be case-sensitive, depending on the OCSP server implementation.

This field is not subject to data conversion.

StrucId (MQCHAR4):

The value must be:

MQAIR_STRUC_ID
Identifier for the authentication information record.

For the C programming language, the constant MQAIR_STRUC_ID_ARRAY is also defined; this has the same value as MQAIR_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQAIR_STRUC_ID.

Version (MQLONG):

The version number of the MQAIR structure.

The value must be one of the following:

MQAIR_VERSION_1
Version-1 authentication information record.

MQAIR_VERSION_2
Version-2 authentication information record.

The following constant specifies the version number of the current version:

MQAIR_CURRENT_VERSION
Current version of authentication information record.

This is always an input field. The initial value of this field is MQAIR_VERSION_1.

Initial values and language declarations for MQAIR:

Table 171. Initial values of fields in MQAIR

Field name	Name of constant	Value of constant
StrucId	MQAIR_STRUC_ID	'AIR¬'
Version	MQAIR_VERSION_1	1
AuthInfoType	MQAIT_CRL_LDAP	1
AuthInfoConnName	None	Null string or blanks
LDAPUserNamePtr	None	Null pointer or null bytes
LDAPUserNameOffset	None	0
LDAPUserNameLength	None	0
LDAPPassword	None	Null string or blanks
OCSPResponderURL	None	Null string or blanks

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQAIR_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure:

   ```
   MQAIR MyAIR = {MQAIR_DEFAULT};
   ```

C declaration for MQAIR:

```
typedef struct tagMQAIR MQAIR;
struct tagMQAIR {
  MQCHAR4    StrucId;             /* Structure identifier */
  MQLONG     Version;             /* Structure version number */
  MQLONG     AuthInfoType;        /* Type of authentication
                                     information */
  MQCHAR264  AuthInfoConnName;    /* Connection name of CRL LDAP
                                     server */
  PMQCHAR    LDAPUserNamePtr;     /* Address of LDAP user name */
  MQLONG     LDAPUserNameOffset;  /* Offset of LDAP user name from start
                                     of MQAIR structure */
  MQLONG     LDAPUserNameLength;  /* Length of LDAP user name */
  MQCHAR32   LDAPPassword;        /* Password to access LDAP server */
  MQCHAR256  OCSPResponderURL;    /* URL of OCSP responder */

};
```

COBOL declaration for MQAIR:

```
**   MQAIR structure
  10 MQAIR.
**    Structure identifier
   15 MQAIR-STRUCID            PIC X(4).
**    Structure version number
   15 MQAIR-VERSION            PIC S9(9) BINARY.
**    Type of authentication information
   15 MQAIR-AUTHINFOTYPE       PIC S9(9) BINARY.
**    Connection name of CRL LDAP server
   15 MQAIR-AUTHINFOCONNNAME   PIC X(264).
**    Address of LDAP user name
   15 MQAIR-LDAPUSERNAMEPTR    POINTER.
**    Offset of LDAP user name from start of MQAIR structure
   15 MQAIR-LDAPUSERNAMEOFFSET PIC S9(9) BINARY.
**    Length of LDAP user name
   15 MQAIR-LDAPUSERNAMELENGTH PIC S9(9) BINARY.
**    Password to access LDAP server
   15 MQAIR-LDAPPASSWORD       PIC X(32).
**    URL of OCSP responder
   15 MQAIR-OCSPRESPONDERURL   PIC X(256).
```

Visual Basic declaration for MQAIR:

```
Type MQAIR
  StrucId            As String*4   'Structure identifier'
  Version            As Long       'Structure version number'
  AuthInfoType       As Long       'Type of authentication information'
  AuthInfoConnName   As String*264 'Connection name of CRL LDAP server'
  LDAPUserNamePtr    As MQPTR      'Address of LDAP user name'
  LDAPUserNameOffset As Long       'Offset of LDAP user name from start'
                                   'of MQAIR structure'
  LDAPUserNameLength As Long       'Length of LDAP user name'
  LDAPPassword       As String*32  'Password to access LDAP server'
End Type
```

MQBMHO - Buffer to message handle options:

The following table summarizes the fields in the structure. MQBMHO structure - buffer to message handle options

Table 172. Fields in MQBMHO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options controlling the action of MQBMHO	Options

Overview for MQBMHO:

Availability: All. Buffer to message handle options structure - overview

Purpose: The MQBMHO structure allows applications to specify options that control how message handles are produced from buffers. The structure is an input parameter on the MQBUFMH call.

Character set and encoding: Data in MQBMHO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields for MQBMHO:

Buffer to message handle options structure - fields

The MQBMHO structure contains the following fields; the fields are described in **alphabetical order**:

Options (MQLONG):

Buffer to message handle structure - Options field

The value can be:

MQBMHO_DELETE_PROPERTIES
Properties that are added to the message handle are deleted from the buffer. If the call fails no properties are deleted.

Default options: If you do not need the option described, use the following option:

MQBMHO_NONE
No options specified.

This is always an input field. The initial value of this field is MQBMHO_DELETE_PROPERTIES.

StrucId (MQCHAR4):

Buffer to message handle structure - StrucId field

This is the structure identifier. The value must be:

MQBMHO_STRUC_ID
Identifier for buffer to message handle structure.

For the C programming language, the constant MQBMHO_STRUC_ID_ARRAY is also defined; this has the same value as MQBMHO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQBMHO_STRUC_ID.

Version (MQLONG):

Buffer to message handle structure - Version field

This is the structure version number. The value must be:

MQBMHO_VERSION_1
Version number for buffer to message handle structure.

The following constant specifies the version number of the current version:

MQBMHO_CURRENT_VERSION
Current version of buffer to message handle structure.

This is always an input field. The initial value of this field is MQBMHO_VERSION_1.

Initial values and language declarations for MQBMHO:

Buffer to message handle structure - Initial values

Table 173. Initial values of fields in MQBMHO

Field name	Name of constant	Value of constant
StrucId	MQBMHO_STRUC_ID	'BMHO'
Version	MQBMHO_VERSION_1	1
Options	MQBMHO_NONE	0

Notes:

1. In the C programming language, the macro variable MQBMHO_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure:

```
MQBMHO MyBMHO = {MQBMHO_DEFAULT};
```

C declaration for MQBMHO:

Buffer to message handle structure - C language declaration

```
typedef struct tagMQBMHO MQBMHO;
struct tagMQBMHO {
  MQCHAR4  StrucId;        /* Structure identifier */
  MQLONG   Version;        /* Structure version number */
  MQLONG   Options;        /* Options that control the action of
                              MQBUFMH */
};
```

COBOL declaration for MQBMHO:

Buffer to message handle structure - COBOL language declaration

```
**   MQBMHO structure
  10 MQBMHO.
**    Structure identifier
   15 MQBMHO-STRUCID               PIC X(4).
**    Structure version number
   15 MQBMHO-VERSION               PIC S9(9) BINARY.
**    Options that control the action of MQBUFMH
   15 MQBMHO-OPTIONS               PIC S9(9) BINARY.
```

PL/I declaration for MQBMHO:

Buffer to message handle structure - PL/I language declaration

```
Dcl
 1 MQBMHO based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 Options        fixed bin(31), /* Options that control the action
                                     of MQBUFMH */
```

High Level Assembler declaration for MQBMHO:

Buffer to message handle structure - Assembler language declaration

```
MQBMHO                  DSECT
MQBMHO_STRUCID          DS   CL4  Structure identifier
MQBMHO_VERSION          DS   F    Structure version number
MQBMHO_OPTIONS          DS   F    Options that control the
*                                  action of MQBUFMH
MQBMHO_LENGTH           EQU  *-MQBMHO
MQBMHO_AREA             DS   CL(MQBMHO_LENGTH)
```

MQBO - Begin options:

The following table summarizes the fields in the structure.

Table 174. Fields in MQBO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options that control the action of MQBEGIN	Options

Overview for MQBO:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows ; not available for IBM MQ MQI clients.

Purpose: The MQBO structure allows the application to specify options relating to the creation of a unit of work. The structure is an input/output parameter on the MQBEGIN call.

Character set and encoding: Data in MQBO must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Fields for MQBO:

The MQBO structure contains the following fields; the fields are described in **alphabetical order**:

Options (MQLONG):

This field is always an input field. Its initial value is MQBO_NONE.

The value must be:

MQBO_NONE
No options specified.

StrucId (MQCHAR4):

This field is always an input field. Its initial value is MQBO_STRUC_ID.

The value must be:

MQBO_STRUC_ID
Identifier for begin-options structure.

For the C programming language, the constant MQBO_STRUC_ID_ARRAY is also defined; this has the same value as MQBO_STRUC_ID, but is an array of characters instead of a string.

Version (MQLONG):

This field is always an input field. Its initial value is MQBO_VERSION_1.

The value must be:

MQBO_VERSION_1
Version number for begin-options structure.

The following constant specifies the version number of the current version:

MQBO_CURRENT_VERSION
Current version of begin-options structure.

Initial values and language declarations for MQBO:

Table 175. Initial values of fields in MQBO for MQBO

Field name	Name of constant	Value of constant
StrucId	MQBO_STRUC_ID	'BO¬¬'
Version	MQBO_VERSION_1	1
Options	MQBO_NONE	0

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQBO_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure:

   ```
   MQBO MyBO = {MQBO_DEFAULT};
   ```

C declaration for MQBO:

```
typedef struct tagMQBO MQBO;
struct tagMQBO {
  MQCHAR4  StrucId;  /* Structure identifier */
  MQLONG   Version;  /* Structure version number */
  MQLONG   Options;  /* Options that control the action of MQBEGIN */
};
```

COBOL declaration for MQBO:

```
**   MQBO structure
  10 MQBO.
**    Structure identifier
   15 MQBO-STRUCID PIC X(4).
**    Structure version number
   15 MQBO-VERSION PIC S9(9) BINARY.
**    Options that control the action of MQBEGIN
   15 MQBO-OPTIONS PIC S9(9) BINARY.
```

PL/I declaration for MQBO:

```
dcl
 1 MQBO based,
  3 StrucId char(4),       /* Structure identifier */
  3 Version fixed bin(31), /* Structure version number */
  3 Options fixed bin(31); /* Options that control the action of
                              MQBEGIN */
```

Visual Basic declaration for MQBO:

```
Type MQBO
  StrucId As String*4 'Structure identifier'
  Version As Long     'Structure version number'
  Options As Long     'Options that control the action of MQBEGIN'
End Type
```

MQCBC - Callback context:

The following table summarizes the fields in the structure. Structure describing the callback routine.

Table 176. Fields in MQCBC

Field	Description	Topic
StrucID	Structure identifier	StrucID
Version	Structure version number	Version
CallType	Why function has been called	CallType
Hobj	Object handle	Hobj
CallbackArea	Field for callback function to use	CallbackArea
ConnectionArea	Field for callback function to use	ConnectionArea
CompCode	Completion code	CompCode
Reason	Reason code	Reason
State	Indication of the state of the current consumer	State
DataLength	Message length	DataLength
BufferLength	Length of message buffer in bytes	BufferLength
Flags	General flags	Flags
Note: The remaining field is ignored if Version is less than MQCBC_VERSION_2		
ReconnectDelay	Number of milliseconds before reconnection attempt	ReconnectDelay

Overview for MQCBC:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, z/OS, plus IBM MQ MQI clients connected to these systems.

Purpose: The MQCBC structure is used to specify context information that is passed to a callback function.

The structure is an input/output parameter on the call to a message consumer routine.

Version: The current version of MQCBC is MQCBC_VERSION_2.

Character set and encoding: Data in MQCBC must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure will be in the character set and encoding of the client.

Fields for MQCBC:

Alphabetic list of fields for the MQCBC structure.

The MQCBC structure contains the following fields; the fields are described in alphabetical order:

BufferLength (MQLONG):

This field is the length in bytes of the message buffer that has been passed to this function.

The buffer can be larger than both the MaxMsgLength value defined for the consumer and the ReturnedLength value in the MQGMO.

The actual message length is supplied in DataLength field.

The application can use the entire buffer for its own purposes for the duration of the callback function.

This is an input field to the message consumer function; it is not relevant to an exception handler function.

CallbackArea (MQPTR):

This field is available for the callback function to use.

The queue manager makes no decisions based on the contents of this field and it is passed unchanged from the CallbackArea field in the MQCBD structure, which is a parameter on the MQCB call used to define the callback function.

Changes to the *CallbackArea* are preserved across the invocations of the callback function for an *HObj*. This field is not shared with callback functions for other handles.

This is an input/output field to the callback function. The initial value of this field is a null pointer or null bytes.

CallType (MQLONG):

Field containing information about why this function has been called; the following values are defined.

Message delivery call types: These call types contain information about a message. The **DataLength** and **BufferLength** parameters are valid for these call types.

MQCBCT_MSG_REMOVED
: The message consumer function has been invoked with a message that has been destructively removed from the object handle.

  If the value of *CompCode* is MQCC_WARNING, the value of the *Reason* field is MQRC_TRUNCATED_MSG_ACCEPTED or one of the codes indicating a data conversion problem.

MQCBCT_MSG_NOT_REMOVED
: The message consumer function has been invoked with a message that has not yet been destructively removed from the object handle. The message can be destructively removed from the object handle using the *MsgToken*.

  The message might not have been removed because:

  - The MQGMO options requested a browse operation, MQGMO_BROWSE_*
  - The message is larger than the available buffer and the MQGMO options do not specify MQGMO_ACCEPT_TRUNCATED_MSG

If the value of *CompCode* is MQCC_WARNING, the value of the *Reason* field is MQRC_TRUNCATED_MSG_FAILED or one of the codes indicating a data conversion problem.

Callback control call types: These call types contain information about the control of the callback and do not contain details about a message. These call types are requested using Options in the MQCBD structure.

The **DataLength** and **BufferLength** parameters are not valid for these call types.

MQCBCT_REGISTER_CALL
The purpose of this call type is to allow the callback function to perform some initial setup.

The callback function is invoked is immediately after the callback is registered, that is, upon return from an MQCB call using a value for the *Operation* field of MQOP_REGISTER.

This call type is used both for message consumers and event handlers.

If requested, this is the first invocation of the callback function.

The value of the *Reason* field is MQRC_NONE.

MQCBCT_START_CALL
The purpose of this call type is to allow the callback function to perform some setup when it is started, for example, reinstating resources that were cleaned up when it was previously stopped.

The callback function is invoked when the connection is started using either MQOP_START or MQOP_START_WAIT.

If a callback function is registered within another callback function, this call type is invoked when the callback returns.

This call type is used for message consumers only.

The value of the *Reason* field is MQRC_NONE.

MQCBCT_STOP_CALL
The purpose of this call type is to allow the callback function to perform some cleanup when it is stopped for a while, for example, cleaning up additional resources that have been acquired during the consuming of messages.

The callback function is invoked when an MQCTL call is issued using a value for the *Operation* field of MQOP_STOP.

This call type is used for message consumers only.

The value of the *Reason* field is set to indicate the reason for stopping.

MQCBCT_DEREGISTER_CALL
The purpose of this call type is to allow the callback function to perform final cleanup at the end of the consume process. The callback function is invoked when the:

- Callback function is deregistered using an MQCB call with MQOP_DEREGISTER.
- Queue is closed, causing an implicit deregister. In this instance the callback function is passed MQHO_UNUSABLE_HOBJ as the object handle.
- MQDISC call completes - causing an implicit close and, therefore, a deregister. In this case the connection is not disconnected immediately, and any ongoing transaction is not yet committed.

If any of these actions are taken inside the callback function itself, the action is invoked once the callback returns.

This call type is used both for message consumers and event handlers.

If requested, this is the last invocation of the callback function.

The value of the *Reason* field is set to indicate the reason for stopping.

MQCBCT_EVENT_CALL
Event handler function

The event handler function has been invoked without a message when the queue manager or connection stops or quiesces.

This call can be used to take appropriate action for all callback functions. **Message consumer function**

The message consumer function has been invoked without a message when an error (*CompCode* = MQCC_FAILED) has been detected that is specific to the object handle; for example *Reason* code = MQRC_GET_INHIBITED.

The value of the *Reason* field is set to indicate the reason for the call.

MQCBCT_MC_EVENT_CALL

The event handler function has been invoked for multicast events; The event handler is sent IBM MQ Multicast events instead of 'normal' IBM MQ events.

For more information about MQCBCT_MC_EVENT_CALL, see Multicast exception reporting.

CompCode (MQLONG):

This field is the completion code. It indicates whether there were any problems consuming the message.

The value is one of the following:

MQCC_OK
Successful completion

MQCC_WARNING
Warning (partial completion)

MQCC_FAILED
Call failed

This is an input field. The initial value of this field is MQCC_OK.

ConnectionArea (MQPTR):

This field is available for the callback function to use.

The queue manager makes no decisions based on the contents of this field and it is passed unchanged from the ConnectionArea field in the MQCTLO structure, which is a parameter on the MQCTL call used to control the callback function.

Any changes made to this field by the callback functions are preserved across the invocations of the callback function. This area can be used to pass information that is to be shared by all callback functions. Unlike *CallbackArea*, this area is common across all callbacks for a connection handle.

This is an input and output field. The initial value of this field is a null pointer or null bytes.

DataLength (MQLONG):

This is the length in bytes of the application data in the message. If the value is zero, it means that the message contains no application data.

The DataLength field contains the length of the message but not necessarily the length of the message data passed to the consumer. It could be that the message was truncated. Use the ReturnedLength field in the MQGMO to determine how much data has actually been passed to the consumer.

If the reason code indicates the message has been truncated, you can use the DataLength field to determine how large the actual message is. This allows you to determine the size of the buffer required to accommodate the message data, and then issue an MQCB call to update the MaxMsgLength with an appropriate value.

If the MQGMO_CONVERT option is specified, the converted message could be larger than the value returned for DataLength. In such cases, the application probably needs to issue an MQCB call to update the MaxMsgLength to to be greater than the value returned by the queue manager for DataLength.

To avoid message truncation problems, specify MaxMsgLength as MQCBD_FULL_MSG_LENGTH. This causes the queue manager to allocate a buffer for the full message length after data conversion. Be aware, however, that even if this option is specified, it is still possible that sufficient storage is not available to correctly process the request. Applications should always check the returned reason code. For example, if it is not possible to allocate sufficient storage to convert the message, the messages is returned to the application unconverted.

This is an input field to the message consumer function; it is not relevant to an event handler function.

Flags (MQLONG):

Flags containing information about this consumer.

The following option is defined:

MQCBCF_READA_BUFFER_EMPTY
This flag can be returned if a previous MQCLOSE call using the MQCO_QUIESCE option failed with a reason code of MQRC_READ_AHEAD_MSGS.

This code indicated that the last read ahead message is being returned and that the buffer is now empty. If the application issues another MQCLOSE call using the MQCO_QUIESCE) option, it succeeds.

Note, that an application is not guaranteed to be given a message with this flag set, as there might still be messages in the read-ahead buffer that do not match the current selection criteria. In this instance, the consumer function is invoked with the reason code MQRC_HOBJ_QUIESCED.

If the read ahead buffer is completely empty, the consumer is invoked with the MQCBCF_READA_BUFFER_EMPTY flag and the reason code MQRC_HOBJ_QUIESCED_NO_MSGS.

This is an input field to the message consumer function; it is not relevant to an event handler function.

Hobj (MQHOBJ):

This is the object handle for calls to the message consumer.

For an event handler, this value is MQHO_NONE

The application can use this handle and the message token in the Get Message Options block to get the message if a message has not been removed from the queue.

This is always an input field. The initial value of this field is MQHO_UNUSABLE_HOBJ

Reason (MQLONG):

This is the reason code qualifying the *CompCode*.

This is an input field. The initial value of this field is MQRC_NONE.

State (MQLONG):

An indication as to the state of the current consumer. This field is of most value to an application when a nonzero reason code is passed to the consumer function.

You can use this field to simplify application programming because you do not need to code behavior for each reason code.

This is an input field. The initial value of this field is MQCS_NONE

State	**Queue manager action**	**Value of constant**
MQCS_NONE This reason code represents a normal call with no additional reason information	None; this is the normal operation.	0
MQCS_SUSPENDED_TEMPORARY These reason codes represent temporary conditions.	The callback routine is called to report the condition and then suspended. After a period of time the system might attempt the operation again, which can lead to the same condition being raised again.	1
MQCS_SUSPENDED_USER_ACTION These reason codes represent conditions where the callback needs to take action to resolve the condition.	The consumer is suspended and the callback routine is called to report the condition. The callback routine should resolve the condition if possible and either RESUME or close down the connection.	2
MQCS_SUSPENDED These reason codes represent failures that prevent further message callbacks.	The queue manager automatically suspends the callback function. If the callback function is resumed it is likely to receive the same reason code again.	3
MQCS_STOPPED These reason codes represent the end of message consumption.	Delivered to the exception handler and to callbacks that specified MQCBDO_STOP_CALL. No further messages can be consumed.	4

StrucId (MQCHAR4):

The value in this field is the structure identifier.

The value must be:

MQCBC_STRUC_ID
Identifier for callback context structure.

For the C programming language, the constant MQCBC_STRUC_ID_ARRAY is also defined; this has the same value as MQCBC_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQCBC_STRUC_ID.

Version (MQLONG):

The value in this field is the structure version number.

The value must be:

MQCBC_VERSION_1
Version-1 callback context structure.

The following constant specifies the version number of the current version:

MQCBC_CURRENT_VERSION
Current version of the callback context structure.

This is always an input field. The initial value of this field is MQCBC_VERSION_1.

The callback function is always passed the latest version of the structure.

ReconnectDelay (MQLONG):

ReconnectDelay indicates how long the queue manager will wait before trying to reconnect. The field can be modified by an event handler to change the delay or stop reconnection altogether.

Use the `ReconnectDelay` field only if the value of the `Reason` field in the Callback Context is `MQRC_RECONNECTING`.

On entry to the event handler the value of `ReconnectDelay` is the number of milliseconds the queue manager is going to wait before making a reconnection attempt. Table 177 lists the values that you can set to modify the behavior of the queue manager on return from the event handler.

Table 177. `ReconnectDelay` values

Name	Value	Description
`MQRD_NO_RECONNECT`	`-1`	Make no more reconnection attempts. An error is returned to the application.
`MQRD_NO_DELAY`	`0`	Try to reconnect immediately.
`Milliseconds`	`>0`	Wait for this many milliseconds before retrying the connection.

Initial values and language declarations for MQCBC:

Callback context structure - initial values

There are no initial values for the **MQCBC** structure. The structure is passed as a parameter to a callback routine. The queue manager initializes the structure; applications never initialize it.

C declaration for MQCBC:

Callback context structure - C language declaration

```
typedef struct tagMQCBC MQCBC;
struct tagMQCBC {
   MQCHAR4   StrucId;                /* Structure identifier */
   MQLONG    Version;                /* Structure version number */
   MQLONG    CallType;               /* Why Function was called */
   MQHOBJ    Hobj;                   /* Object Handle */
   MQPTR     CallbackArea;           /* Callback data passed to the function */
   MQPTR     ConnectionArea;         /* MQCTL data area passed to the function */
   MQLONG    CompCode;               /* Completion Code */
   MQLONG    Reason;                 /* Reason Code */
   MQLONG    State;                  /* Consumer State */
   MQLONG    DataLength;             /* Message Data Length */
   MQLONG    BufferLength;           /* Buffer Length */
   MQLONG    Flags;                  /* Flags containing information about
                                        this consumer */
  /* Ver:1 */
   MQLONG    ReconnectDelay;         /* Number of milliseconds before */
  /* Ver:2 */  };                   /* reconnect attempt */
```

COBOL declaration for MQCBC:

```
** MQCBC structure
 10  MQCBC.
** Structure Identifier
 15  MQCBC-STRUCID                    PIC X(4).
** Structure Version
 15  MQCBC-VERSION                    PIC S9(9) BINARY.
 ** Call Type
 15  MQCBC-CALLTYPE                   PIC S9(9) BINARY.
** Object Handle
 15  MQCBC-HOBJ                       PIC S9(9) BINARY.
** Callback User Area
 15  MQCBC-CALLBACKAREA               POINTER
** Connection Area
 15  MQCBC-CONNECTIONAREA             POINTER
** Completion Code
 15  MQCBC-COMPCODE                   PIC S9(9) BINARY.
** Reason Code
 15  MQCBC-REASON                     PIC S9(9) BINARY.
** Consumer State
 15  MQCBC-STATE                      PIC S9(9) BINARY.
** Data Length
 15  MQCBC-DATALENGTH                 PIC S9(9) BINARY.
** Buffer Length
 15  MQCBC-BUFFERLENGTH               PIC S9(9) BINARY.
** Flags
 15  MQCBC-FLAGS                      PIC S9(9) BINARY.
** Ver:1 **
** Number of milliseconds before reconnect attempt
 15  MQCBC-RECONNECTDELAY PIC S9(9) BINARY.
** Ver:2 **
```

PL/I declaration for MQCBC:

```
dcl
 1 MQCBC based,
  3 StrucId             char(4),       /* Structure identifier */
  3 Version             fixed bin(31), /* Structure version */
  3 CallType            fixed bin(31), /* Callback type */
  3 Hobj                fixed bin(31), /* Object Handle */
  3 CallbackArea        pointer,       /* User area passed to the function */
  3 ConnectionArea      pointer,       /* Connection User Area */
  3 CompCode            fixed bin(31); /* Completion Code */
  3 Reason              fixed bin(31); /* Reason Code */
  3 State               fixed bin(31); /* Consumer State */
  3 DataLength          fixed bin(31); /* Message Data Length */
  3 BufferLength        fixed bin(31); /* Message Buffer length */
  3 Flags               fixed bin(31); /* Consumer Flags */
/* Ver:1 */
  3 ReconnectDelay      fixed bin(31); /* Number of milliseconds before */
/* Ver:2 */                           /* reconnect attempt */
```

High Level Assembler declaration for MQCBC:

```
    MQCBC                 DSECT
    MQCBC                 DS  0F    Force fullword alignment
    MQCBC_STRUCID         DS  CL4   Structure identifier
    MQCBC_VERSION         DS  F     Structure version number
    MQCBC_CALLTYPE        DS  F     Why Function was called
    MQCBC_HOBJ            DS  F     Object Handle
    MQCBC_CALLBACKAREA    DS  A     Callback data passed to the function
    MQCBC_CONNECTIONAREA  DS  A     MQCTL Data area passed to the function
    MQCBC_COMPCODE        DS  F     Completion Code
    MQCBC_REASON          DS  F     Reason Code
    MQCBC_STATE           DS  F     Consumer State
    MQCBC_DATALENGTH      DS  F     Message Data Length
    MQCBC_BUFFERLENGTH    DS  F     Buffer Length
    MQCBC_FLAGS           DS  F     Flags containing information about this consumer
    MQCBC_RECONNECTDELAY  DS  F     Number of milliseconds before reconnect
    MQCBC_LENGTH          EQU *-MQCBC
                          ORG       MQCBC
    MQCBC_AREA            DS  CL(MQCBC_LENGTH)
```

MQCBD - Callback descriptor:

The following table summarizes the fields in the structure. Structure specifying the callback function.

Table 178. Fields in MQCBD

Field	Description	Topic
StrucID	Structure identifier	StrucID
Version	Structure version number	Version
CallbackType	Type of callback function	CallbackType
Options	Options controlling message consumption	Options
Callback Area	Field for callback function to use	CallbackArea
CallbackFunction	Whether the function is invoked as an API call	CallbackFunction
CallbackName	Whether the function is invoked as a dynamically-linked program	CallbackName
MaxMsgLength	Length of longest message that can be read	MaxMsgLength

Overview for MQCBD:
Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, z/OS, and IBM MQ MQI clients connected to these systems.

Purpose: The MQCBD structure is used to specify a callback function and the options controlling its use by the queue manager.

The structure is an input parameter on the MQCB call.

Version: The current version of MQCBD is MQCBD_VERSION_1.

Character set and encoding: Data in MQCBD must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Fields for MQCBD:

Alphabetic list of fields for the MQCBD structure.

The MQCBD structure contains the following fields; the fields are described in alphabetical order:

CallbackArea (MQPTR):

Callback descriptor structure - CallbackArea field

This is a field that is available for the callback function to use.

The queue manager makes no decisions based on the contents of this field and it is passed unchanged from the CallbackArea field in the MQCBC structure, which is a parameter on the callback function declaration.

The value is used only on an *Operation* having a value MQOP_REGISTER, with no currently defined callback, it does not replace a previous definition.

This is an input and output field to the callback function. The initial value of this field is a null pointer or null bytes.

CallbackFunction (MQPTR):

Callback descriptor structure - CallbackFunction field

The callback function is invoked as a function call.

Use this field to specify a pointer to the callback function.

You must specify either *CallbackFunction* or *CallbackName*. If you specify both, the reason code MQRC_CALLBACK_ROUTINE_ERROR is returned.

If neither *CallbackName* nor *CallbackFunction* is set, the call fails with the reason code MQRC_CALLBACK_ROUTINE_ERROR.

This option is not supported in the following environment: Programming languages and compilers that do not support function-pointer references. In such situations, the call fails with the reason code MQRC_CALLBACK_ROUTINE_ERROR.

z/OS On z/OS, the function must expect to be called with OS linkage conventions. For example, in the C programming language, specify:

```
#pragma linkage(MQCB_FUNCTION,OS)
```

This is an input field. The initial value of this field is a null pointer or null bytes.

Note: When using CICS with IBM WebSphere MQ Version 7.0.1, asynchronous consumption is supported if:

- Apar PK66866 is applied to CICS TS 3.2
- Apar PK89844 is applied to CICS TS 4.1

CallbackName (MQCHAR128):

Callback descriptor structure - CallbackName field

The callback function is invoked as a dynamically linked program.

You must specify either *CallbackFunction* or *CallbackName*. If you specify both, the reason code MQRC_CALLBACK_ROUTINE_ERROR is returned.

If neither *CallbackName* nor *CallbackFunction* is not set, the call fails with the reason code MQRC_CALLBACK_ROUTINE_ERROR.

The module is loaded when the first callback routine to use is registered, and unloaded when the last callback routine to use it deregisters.

Except where noted in the following text, the name is left-justified within the field, with no embedded blanks; the name itself is padded with blanks to the length of the field. In the descriptions that follow, square brackets ([ ]) denote optional information:

IBM i The callback name can be one of the following formats:

- Library "/" Program
- Library "/" ServiceProgram "("FunctionName")"

For example, `MyLibrary/MyProgram(MyFunction)`.

The library name can be *LIBL. Both the library and program names are limited to a maximum of 10 characters.

UNIX The callback name is the name of a dynamically-loadable module or library, suffixed with the name of a function residing in that library. The function name must be enclosed in parentheses. The library name can optionally be prefixed with a directory path:

```
[path]library(function)
```

If the path is not specified the system search path is used.

The name is limited to a maximum of 128 characters.

Windows

The callback name is the name of a dynamic-link library, suffixed with the name of a function residing in that library. The function name must be enclosed in parentheses .The library name can optionally be prefixed with a directory path and drive:

```
[d:][path]library(function)
```

If the drive and path are not specified the system search path is used.

The name is limited to a maximum of 128 characters.

z/OS The callback name is the name of a load module that is valid for specification on the EP parameter of the LINK or LOAD macro.

The name is limited to a maximum of 8 characters.

z/OS CICS
The callback name is the name of a load module that is valid for specification on the PROGRAM parameter of the EXEC CICS LINK command macro.

The name is limited to a maximum of 8 characters.

The program can be defined as remote using the REMOTESYTEM option of the installed PROGRAM definition or by the dynamic routing program.

The remote CICS region must be connected to IBM MQ if the program is to use IBM MQ API calls. Note, however, that the Hobj field in the MQCBC structure is not valid in a remote system.

If a failure occurs trying to load *CallbackName*, one of the following error codes is returned to the application:

- MQRC_MODULE_NOT_FOUND
- MQRC_MODULE_INVALID
- MQRC_MODULE_ENTRY_NOT_FOUND

A message is also written to the error log containing the name of the module for which the load was attempted, and the failing reason code from the operating system.

This is an input field. The initial value of this field is a null string or blanks.

CallbackType (MQLONG):

Callback descriptor structure - CallbackType field

This is the type of the callback function. The value must be one of:

MQCBT_MESSAGE_CONSUMER
Defines this callback as a message consumer function.

A message consumer callback function is called when a message, meeting the selection criteria specified, is available on an object handle and the connection is started.

MQCBT_EVENT_HANDLER
Defines this callback as the asynchronous event routine; it is not driven to consume messages for a handle.

Hobj is not required on the MQCB call defining the event handler and is ignored if specified.

The event handler is called for conditions that affect the whole message consumer environment. The consumer function is invoked without a message when an event, for example, a queue manager or connection stopping, or quiescing, occurs. It is not called for conditions that are specific to a single message consumer, for example, MQRC_GET_INHIBITED.

Events are delivered to the application, regardless of whether the connection is started or stopped, except in the following environments:

- CICS on z/OS environment
- nonthreaded applications

If the caller does not pass one of these values, the call fails with a *Reason* code of MQRC_CALLBACK_TYPE_ERROR

This is always an input field. The initial value of this field is MQCBT_MESSAGE_CONSUMER.

MaxMsgLength (MQLONG):

This is the length in bytes of the longest message that can be read from the handle and given to the callback routine. Callback descriptor structure - MaxMsgLength field

If a message has a longer length, the callback routine receives *MaxMsgLength* bytes of the message, and reason code:

- MQRC_TRUNCATED_MSG_FAILED or
- MQRC_TRUNCATED_MSG_ACCEPTED if you specified MQGMO_ACCEPT_TRUNCATED_MSG.

The actual message length is supplied in the DataLength field of the MQCBC structure.

The following special value is defined:

MQCBD_FULL_MSG_LENGTH
The buffer length is adjusted by the system to return messages without truncation.

If insufficient memory is available to allocate a buffer to receive the message, the system calls the callback function with an MQRC_STORAGE_NOT_AVAILABLE reason code.

If, for example, you request data conversion, and there is insufficient memory available to convert the message data, the unconverted message is passed to the callback function.

This is an input field. The initial value of the *MaxMsgLength* field is MQCBD_FULL_MSG_LENGTH.

Options (MQLONG):

Callback descriptor structure - Options field

You can specify one or more of these options. To specify more than one option, either add the values together (do not add the same constant more than once), or combine the values using the bitwise OR operation (if the programming language supports bit operations).

MQCBDO_FAIL_IF_QUIESCING
The MQCB call fails if the queue manager is in the quiescing state.

On z/OS, this option also forces the MQCB call to fail if the connection (for a CICS or IMS application) is in the quiescing state.

Specify MQGMO_FAIL_IF_QUIESCING, in the MQGMO options passed on the MQCB call, to cause notification to message consumers when they are quiescing.

Control options: The following options control whether the callback function is called, without a message, when the state of the consumer changes:

MQCBDO_REGISTER_CALL
The callback function is invoked with call type MQCBCT_REGISTER_CALL.

MQCBDO_START_CALL
The callback function is invoked with call type MQCBCT_START_CALL.

MQCBDO_STOP_CALL
The callback function is invoked with call type MQCBCT_STOP_CALL.

MQCBDO_DEREGISTER_CALL
The callback function is invoked with call type MQCBCT_DEREGISTER_CALL.

MQCBDO_EVENT_CALL
The callback function is invoked with call type MQCBCT_EVENT_CALL.

MQCBDO_MC_EVENT_CALL
The callback function is invoked with call type MQCBCT__MC_EVENT_CALL.

See CallType for further details about these call types.

Default option: If you do not need any of the options described, use the following option:

MQCBDO_NONE
Use this value to indicate that no other options have been specified; all options assume their default values.

MQCBDO_NONE is defined to aid program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

This is an input field. The initial value of the *Options* field is MQCBDO_NONE.

StrucId (MQCHAR4):

Callback descriptor structure - StrucId field

This is the structure identifier; the value must be:

MQCBD_STRUC_ID
Identifier for callback descriptor structure.

For the C programming language, the constant MQCBD_STRUC_ID_ARRAY is also defined; this has the same value as MQCBD_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQCBD_STRUC_ID.

Version (MQLONG):

Callback descriptor structure - Version field

This is the structure version number; the value must be:

MQCBD_VERSION_1
Version-1 callback descriptor structure.

The following constant specifies the version number of the current version:

MQCBD_CURRENT_VERSION
Current version of callback descriptor structure.

This is always an input field. The initial value of this field is MQCBD_VERSION_1.

Initial values and language declarations for MQCBD:

Callback descriptor structure - Initial values

Table 179. Initial values of fields in MQCBD

Field name	Name of constant	Value of constant
StrucId	MQCBD_STRUC_ID	'CBD¬'
Version	MQCBD_VERSION_1	1
CallBackType	MQCBT_MESSAGE_CONSUMER	1
Options	MQCBDO_NONE	0
CallbackArea	None	Null pointer or null blanks
CallbackFunction	None	Null pointer or null blanks
CallbackName	None	Null string or blanks

Table 179. Initial values of fields in MQCBD (continued)

Field name	Name of constant	Value of constant
MaxMsgLength	MQCBD_FULL_MSG_LENGTH	-1

Notes:

1. The symbol ¬ represents a single blank character.
2. The value Null string or blanks denotes the null sting in the C programming language, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQCBD_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure:

   ```
   MQCBD MyCBD = {MQCBD_DEFAULT};
   ```

C declaration for MQCBD:

Callback descriptor structure - C language declaration

```
typedef struct tagMQCBD MQCBD;
 struct tagMQCBD {
   MQCHAR4    StrucId;                /* Structure identifier */
   MQLONG     Version;                /* Structure version number */
   MQLONG     CallBackType;           /* Callback function type */
   MQLONG     Options;                /* Options controlling message
                                         consumption */
   MQPTR      CallbackArea;           /* User data passed to the function */
   MQPTR      CallbackFunction;       /* Callback function pointer */
   MQCHAR128  CallbackName;           /* Callback name */
   MQLONG     MaxMsgLength;           /* Maximum message length */
 };
```

COBOL declaration for MQCBD:

```
** MQCBCD structure
 10  MQCBD.
** Structure Identifier
 15  MQCBD-STRUCID                    PIC X(4).
** Structure Version
 15  MQCBD-VERSION                    PIC S9(9) BINARY.
 ** Callback Type
 15  MQCBD-CALLBACKTYPE               PIC S9(9) BINARY.
** Options
 15  MQCBD-OPTIONS                    PIC S9(9) BINARY.
** Callback User Area
 15  MQCBD-CALLBACKAREA               POINTER
** Callback Function Pointer
 15  MQCBD-CALLBACKFUNCTION           FUNCTION-POINTER
** Callback Program Name
 15  MQCBD-CALLBACKNAME               PIC X(128)
** Maximum Message Length
 15  MQCDB-MAXMSGLENGTH               PIC S9(9) BINARY.
```

PL/I declaration for MQCBD:

```
dcl
 1 MQCBD based,
  3 StrucId             char(4),       /* Structure identifier*/
  3 Version             fixed bin(31), /* Structure version*/
  3 CallbackType        fixed bin(31), /* Callback function type */
  3 Options             fixed bin(31), /* Options */
  3 CallbackArea        pointer,       /* User area passed to the function */
  3 CallbackFunction    pointer,       /* Callback Function Pointer */
  3 CallbackName        char(128),     /* Callback Program Name */
  3 MaxMsgLength        fixed bin(31); /* Maximum Message Length */
```

MQCHARV - Variable Length String:

The following table summarizes the fields in the structure.

Field	Description	Topic
VSPtr	Pointer to the variable length string	VSPtr
VSOffset	Offset in bytes of the variable length string from the start of the structure that contains this MQCHARV structure	VSOffset
VSLength	The length in bytes of the variable length string addressed by the VSPtr or VSOffset field.	VSLength
VSBufSize	The size in bytes of the buffer addressed by the VSPtr or VSOffset field.	VSBufSize
VSCCSID	The character set identifier of the variable length string addressed by the VSPtr or VSOffset field.	VSCCSID

Overview for MQCHARV:

Availability: AIX, HP-UX, Solaris, Linux, IBM i, Windows, plus IBM MQ MQI clients connected to these systems.

Purpose: Use the MQCHARV structure to describe a variable length string.

Character set and encoding: Data in the MQCHARV must be in the encoding of the local queue manager that is given by MQENC_NATIVE and the character set of the VSCCSID field within the structure. If the application is running as an MQ client, the structure must be in the encoding of the client. Some character sets have a representation that depends on the encoding. If VSCCSID is one of these character sets, the encoding used is the same encoding as that of the other fields in the MQCHARV. The character set identified by VSCCSID can be a double-byte character set (DBCS).

Usage: The MQCHARV structure addresses data that might be discontiguous with the structure containing it. To address this data, fields declared with the pointer data type can be used. Be aware that COBOL does not support the pointer data type in all environments. Because of this, the data can also be addressed using fields that contain the offset of the data from the start of the structure containing the MQCHARV.

COBOL programming

If you want to port an application between environments, you must ascertain whether the pointer data type is available in all the intended environments. If not, the application must address the data using the offset fields instead of the pointer fields.

In those environments where pointers are not supported, you can declare the pointer fields as byte strings of the appropriate length, with the initial value being the all-null byte string. Do not alter this initial value if you are using the offset fields. One way to do this without changing the supplied copy books is to use the following:

```
COPY CMQCHRVV REPLACING POINTER BY ==BINARY PIC S9(9)==.
```

where CMQCHRVV can be exchanged for the copy book to be used.

Fields for MQCHARV:

The MQCHARV structure contains the following fields; the fields are described in **alphabetical order**:

VSBufSize (MQLONG):

This is the size in bytes of the buffer addressed by the VSPtr or VSOffset field.

When the MQCHARV structure is used as an output field on a function call, this field must be initialised with the length of the buffer provided. If the value of VSLength is greater than VSBufSize then only VSBufSize bytes of data are returned to the caller in the buffer.

This value must be a value greater than or equal to zero, or the following special value which is recognized:

MQVS_USE_VSLENGTH

When specified, the length of the buffer is taken from the VSLength field in the MQCHARV structure. Do not use this value when using the structure as an output field and a buffer is provided.

This is the initial value of this field.

VSCCSID (MQLONG):

This is the character set identifier of the variable length string addressed by the VSPtr or VSOffset field.

The initial value of this field is MQCCSI_APPL which is defined by MQ to indicate that it should be changed to the true character set identifier of the current process. As a result, the value MQCCSI_APPL is never associated with a variable length string. The initial value of this field can be changed by defining a different value for the constant MQCCSI_APPL for your compile unit by the appropriate means for your application's programming language.

VSLength (MQLONG):

The length in bytes of the variable length string addressed by the VSPtr or VSOffset field.

The initial value of this field is 0. The value must be either greater than or equal to zero or the following special value which is recognized:

MQVS_NULL_TERMINATED

If MQVS_NULL_TERMINATED is not specified, VSLength bytes are included as part of the string. If null characters are present they do not delimit the string.

If MQVS_NULL_TERMINATED is specified, the string is delimited by the first null encountered in the string. The null itself is not included as part of that string.

Note: The null character used to terminate a string if MQVS_NULL_TERMINATED is specified is a null from the codeset specified by VSCCSID.

For example, in UTF-16 V 9.0.0 (CCSIDs 1200, 13488, and 17584), this is the two byte Unicode encoding where a null is represented by a 16-bit number of all zeros. In UTF-16 it is common to find single bytes set to all zero which are part of characters (7-bit ASCII characters for instance), but the strings will only be null terminated when two 'zero' bytes are found on an even byte boundary. It is possible to get two 'zero' bytes on an odd boundary when they are each part of valid characters. For example x'01' x'00 x'00' x'30' represents two valid Unicode characters and does not null terminate the string.

VSOffset (MQLONG):

The offset can be positive or negative. You can use either the VSPtr or VSOffset field to specify the variable length string, but not both. The offset in bytes of the variable length string from the start of the MQCHARV, or the structure containing it.

When the MQCHARV structure is embedded within another structure, this value is the offset in bytes of the variable length string from the start of the structure that contains this MQCHARV structure. When the MQCHARV structure is not embedded within another structure, for example, if it is specified as a parameter on a function call, the offset is relative to the start of the MQCHARV structure.

The initial value of this field is 0.

VSPtr (MQPTR):

This is a pointer to the variable length string.

You can use either the VSPtr or VSOffset field to specify the variable length string, but not both.

The initial value of this field is a null pointer or null bytes.

Initial values and language declarations for MQCHARV:

Initial values of fields in MQCHARV

Field name	Name of constant	Value of constant
VSPtr	None	Null pointer or null bytes.
VSOffset	None	0
VSBufSize	MQVS_USE_VSLENGTH	0
VSLength	None	0
VSCCSID	MQCCSI_APPL	-3
Note: In the C programming language, the macro variable MQCHARV_DEFAULT contains the values that are listed in the table. It can be used in the following way to provide initial values for the fields in the structure: `MQCHARV MyVarStr = {MQCHARV_DEFAULT};`		

C declaration for MQCHARV:

```
typedef struct tagMQCHARV MQCHARV;
struct tagMQCHARV {
  MQPTR     VSPtr;               /* Address of variable length string */
  MQLONG    VSOffset;            /* Offset of variable length string */
  MQLONG    VSBufSize;           /* Size of buffer */
  MQLONG    VSLength;            /* Length of variable length string */
  MQLONG    VSCCSID;             /* CCSID of variable length string */
};
```

COBOL declaration for MQCHARV:

```
** MQCHARV structure
 10  MQCHARV.
** Address of variable length string
 15  MQCHARV-VSPTR        POINTER.
** Offset of variable length string
 15  MQCHARV-VSOFFSET     PIC S9(9) BINARY.
** Size of buffer
 15  MQCHARV-VSBUFSIZE    PIC S9(9) BINARY.
** Length of variable length string
 15  MQCHARV-VSLENGTH     PIC S9(9) BINARY.
** CCSID of variable length string
 15  MQCHARV-VSCCSID      PIC S9(9) BINARY.
```

PL/I declaration for MQCHARV:

```
dcl
 1 MQCHARV based,
  3 VSPtr       pointer,       /* Address of variable length string */
  3 VSOffset    fixed bin(31), /* Offset of variable length string */
  3 VSBufSize   fixed bin(31), /* Size of buffer */
  3 VSLength    fixed bin(31), /* Length of variable length string */
  3 VSCCSID     fixed bin(31); /* CCSID of variable length string */
```

High Level Assembler declaration for MQCHARV:

```
MQCHARV             DSECT
MQCHARV_VSPTR       DS   F     Address of variable length string
MQCHARV_VSOFFSET    DS   F     Offset of variable length string
MQCHARV_VSBUFSIZE   DS   F     Size of buffer
MQCHARV_VSLENGTH    DS   F     Length of variable length string
MQCHARV_VSCCSID     DS   F     CCSID of variable length string
*
MQCHARV_LENGTH      EQU  *-MQCHARV
                    ORG  MQCHARV
MQCHARV_AREA        DS   CL(MQCHARV_LENGTH)
```

Redefinition of MQCCSI_APPL:

The following examples show how you can override the value of MQCCSI_APPL in various programming languages. You can change the value of MQCCSI_APPL, removing the need to set the VSCCSID for each variable length string separately.

In these examples the CCSID is set to 1208; change this to the value you require. This becomes the default value, which you can override by setting the VSCCSID in any specific instance of MQCHARV.

C usage

```
#define MQCCSI_APPL 1208
#include <cmqc.h>
```

COBOL usage

```
COPY CMQXYZV REPLACING -3 BY 1208.
```

PL/I usage

```
%MQCCSI_APPL = '1208';
%include syslib(cmqp);
```

System/390 assembler usage

```
MQCCSI_APPL EQU 1208
CMQA LIST=NO
```

MQCIH - CICS bridge header:

All the CICS versions supported by IBM MQ Version 9.0.0, and later, use the CICS supplied version of the bridge.

For more information about configuring the IBM MQ CICS adapter, and the IBM MQ CICS bridge components, see the Configuring connections to MQ section of the CICS documentation.

Table 180. Fields in MQCIH

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
StrucLength	Length of MQCIH structure	StrucLength
Encoding	Reserved	Encoding
CodedCharSetId	Reserved	CodedCharSetId
Format	MQ format name of data that follows MQCIH	Format
Flags	Flags	Flags
ReturnCode	Return code from bridge	ReturnCode
CompCode	MQ completion code or CICS EIBRESP	CompCode
Reason	MQ reason or feedback code, or CICS EIBRESP2	Reason
UOWControl	Unit-of-work control	UOWControl
GetWaitInterval	Wait interval for MQGET call issued by bridge task	GetWaitInterval
LinkType	Link type	LinkType
OutputDataLength	Output COMMAREA data length	OutputDataLength
FacilityKeepTime	Bridge facility release time	FacilityKeepTime
ADSDescriptor	Send/receive ADS descriptor	ADSDescriptor

Table 180. Fields in MQCIH (continued)

Field	Description	Topic
ConversationalTask	Whether task can be conversational	ConversationalTask
TaskEndStatus	Status at end of task	TaskEndStatus
Facility	Bridge facility token	Facility
Function	MQ call name or CICS EIBFN function	Function
AbendCode	Abend code	AbendCode
Authenticator	Password or passticket	Authenticator
Reserved1	Reserved	Reserved1
ReplyToFormat	MQ format name of reply message	ReplyToFormat
RemoteSysId	Remote CICS system Id to use	RemoteSysId
RemoteTransId	CICS RTRANSID to use	RemoteTransId
TransactionId	Transaction to attach	TransactionId
FacilityLike	Terminal emulated attributes	FacilityLike
AttentionId	AID key	AttentionId
StartCode	Transaction start code	StartCode
CancelCode	Abend transaction code	CancelCode
NextTransactionId	Next transaction to attach	NextTransactionId
Reserved2	Reserved	Reserved2
Reserved3	Reserved	Reserved3
Note: The remaining fields are not present if *Version* is less than MQCIH_VERSION_2.		
CursorPosition	Cursor position	CursorPosition
ErrorOffset	Offset of error in message	ErrorOffset
InputItem	Reserved	InputItem
Reserved4	Reserved	Reserved4

Overview for MQCIH:

The MQCIH structure describes the header information for a message sent to CICS across the CICS bridge. For any IBM MQ supported platform you can create and transmit a message that includes the MQCIH structure, but only an IBM MQ for z/OS queue manager can use the CICS bridge. Therefore, for the message to get to CICS from a non-z/OS queue manager, your queue manager network must include at least one z/OS queue manager through which the message can be routed.

Availability: AIX, HP-UX, z/OS, Solaris, Linux, Windows, plus IBM MQ MQI clients connected to these systems.

Purpose: The MQCIH structure describes the information that can be present at the start of a message sent to the CICS bridge through IBM MQ for z/OS.

Format name: MQFMT_CICS.

Version: The current version of MQCIH is MQCIH_VERSION_2. Fields that exist only in the more-recent version of the structure are identified as such in the descriptions that follow.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQCIH, with the initial value of the *Version* field set to MQCIH_VERSION_2.

Character set and encoding: Special conditions apply to the character set and encoding used for the MQCIH structure and application message data:

- Applications that connect to the queue manager that owns the CICS bridge queue must provide an MQCIH structure that is in the character set and encoding of the queue manager. This is because data conversion of the MQCIH structure is not performed in this case.
- Applications that connect to other queue managers can provide an MQCIH structure that is in any of the supported character sets and encodings; the receiving message channel agent connected to the queue manager that owns the CICS bridge queue converts the MQCIH structure.
- The application message data following the MQCIH structure must be in the same character set and encoding as the MQCIH structure. You cannot use the *CodedCharSetId* and *Encoding* fields in the MQCIH structure to specify the character set and encoding of the application message data.

  You must provide a data-conversion exit to convert the application message data if the data is not one of the built-in formats supported by the queue manager.

Usage: If the application requires values that are the same as the initial values shown in Table 182 on page 2268, and the bridge is running with AUTH=LOCAL or AUTH=IDENTIFY, you can omit the MQCIH structure from the message. In all other cases, the structure must be present.

The bridge accepts either a version-1 or a version-2 MQCIH structure, but for 3270 transactions, you must use a version-2 structure.

The application must ensure that fields documented as request fields have appropriate values in the message sent to the bridge; these fields are input to the bridge.

Fields documented as response fields are set by the CICS bridge in the reply message that the bridge sends to the application. Error information is returned in the *ReturnCode*, *Function*, *CompCode*, *Reason*, and *AbendCode* fields, but not all of them are set in all cases. Table 181 shows which fields are set for different values of *ReturnCode*.

Table 181. Contents of error information fields in MQCIH structure for MQCIH

ReturnCode	*Function*	*CompCode*	*Reason*	*AbendCode*
MQCRC_OK	-	-	-	-
MQCRC_BRIDGE_ERROR	-	-	MQFB_CICS_*	-
MQCRC_MQ_API_ERROR MQCRC_BRIDGE_TIMEOUT	MQ call name	MQ *CompCode*	MQ *Reason*	-
MQCRC_CICS_EXEC_ERROR MQCRC_SECURITY_ERROR MQCRC_PROGRAM_NOT_AVAILABLE MQCRC_TRANSID_NOT_AVAILABLE	CICS EIBFN	CICS EIBRESP	CICS EIBRESP2	-
MQCRC_BRIDGE_ABEND MQCRC_APPLICATION_ABEND	-	-	-	CICS ABCODE

Fields for MQCIH:

The MQCIH structure contains the following fields; the fields are described in **alphabetical order**:

AbendCode (MQCHAR4):

AbendCode is a response field. The length of this field is given by MQ_ABEND_CODE_LENGTH. The initial value of this field is 4 blank characters.

The value returned in this field is significant only if the *ReturnCode* field has the value MQCRC_APPLICATION_ABEND or MQCRC_BRIDGE_ABEND. If it does, *AbendCode* contains the CICS ABCODE value.

ADSDescriptor (MQLONG):

This field is an indicator specifying whether to send ADS descriptors on SEND and RECEIVE BMS requests.

The following values are defined:

MQCADSD_NONE
Do not send or receive ADS descriptors.

MQCADSD_SEND
Send ADS descriptors.

MQCADSD_RECV
Receive ADS descriptors.

MQCADSD_MSGFORMAT
Use message format for the ADS descriptors.

This sends or receives the ADS descriptors using the long form of the ADS descriptor. The long form has fields that are aligned on 4-byte boundaries.

Set the *ADSDescriptor* field as follows:

- If you are not using ADS descriptors, set the field to MQCADSD_NONE.
- If you are using ADS descriptors with the *same* CCSID in each environment, set the field to the sum of MQCADSD_SEND and MQCADSD_RECV.
- If you are using ADS descriptors with *different* CCSIDs in each environment, set the field to the sum of MQCADSD_SEND, MQCADSD_RECV, and MQCADSD_MSGFORMAT.

This is a request field used only for 3270 transactions. The initial value of this field is MQCADSD_NONE.

AttentionId (MQCHAR4):

The value in this field determines the initial value of the AID key when the transaction is started. It is a 1 byte value, left-aligned.

AttentionId is a request field used only for 3270 transactions. The length of this field is given by MQ_ATTENTION_ID_LENGTH. The initial value of this field is four blanks.

Authenticator (MQCHAR8):

The value of this field is the password or passticket.

If user-identifier authentication is active for the CICS bridge, *Authenticator* is used with the user identifier in the MQMD identity context to authenticate the sender of the message.

This is a request field. The length of this field is given by MQ_AUTHENTICATOR_LENGTH. The initial value of this field is 8 blanks.

CancelCode (MQCHAR4):

The value in this field is the abend code to be used to terminate the transaction (normally a conversational transaction that is requesting more data). Otherwise this field is set to blanks.

This field is a request field used only for 3270 transactions. The length of this field is given by MQ_CANCEL_CODE_LENGTH. The initial value of this field is four blanks.

CodedCharSetId (MQLONG):

CodedCharSetId is a reserved field; its value is not significant. The initial value of this field is 0.

The Character Set ID for supported structures which follow an MQCIH structure is the same as the Character Set ID of the MQCIH structure itself and is taken from any preceding IBM MQ header.

CompCode (MQLONG):

This field is a response field. Its initial value is MQCC_OK

The value returned in this field depends on *ReturnCode* ; see Table 181 on page 2256.

ConversationalTask (MQLONG):

This field is an indicator specifying whether to allow the task to issue requests for more information, or to stop the task and issue an abend message.

The value must be one of the following options:

MQCCT_YES
The task is conversational.

MQCCT_NO
The task is not conversational.

This field is a request field used only for 3270 transactions. The initial value of this field is MQCCT_NO.

CursorPosition (MQLONG):

The value in this field shows the initial cursor position when the transaction is started. For conversational transactions, the cursor position is in the RECEIVE vector.

This field is a request field used only for 3270 transactions. The initial value of this field is 0. This field is not present if *Version* is less than MQCIH_VERSION_2.

Encoding (MQLONG):

This field is a reserved field; its value is not significant. Its initial value is 0.

The Encoding for supported structures which follow an MQCIH structure is the same as the Encoding of the MQCIH structure itself and taken from any preceding IBM MQ header.

ErrorOffset (MQLONG):

The ErrorOffset field shows the position of invalid data detected by the bridge exit. This field provides the offset from the start of the message to the location of the invalid data.

ErrorOffset is a response field used only for 3270 transactions. The initial value of this field is 0. This field is not present if *Version* is less than MQCIH_VERSION_2.

Facility (MQBYTE8):

This field shows the 8-byte bridge facility token.

A bridge facility token enables multiple transactions in a pseudo-conversation to use the same bridge facility (virtual 3270 terminal). In the first, or only, message in a pseudo-conversation, set a value of MQCFAC_NONE. This value tells CICS to allocate a new bridge facility for this message. A bridge facility token is returned in response messages when a nonzero *FacilityKeepTime* is specified on the input message. Subsequent input messages within a pseudo-conversation must then use the same bridge facility token.

The following special value is defined:

MQCFAC_NONE
No facility token specified.

For the C programming language, the constant MQCFAC_NONE_ARRAY is also defined, and has the same value as MQCFAC_NONE, but is an array of characters instead of a string.

This field is both a request and a response field used only for 3270 transactions. The length of this field is given by MQ_FACILITY_LENGTH. The initial value of this field is MQCFAC_NONE.

FacilityKeepTime (MQLONG):

FacilityKeepTime is the length of time in seconds that the bridge facility is kept after the user transaction ends.

For pseudo-conversational transactions, specify a value that corresponds to the expected duration of a pseudo-conversation; specify zero for the last transaction of a pseudo-conversation, and for other transaction types specify zero.

This field is a request field used only for 3270 transactions. The initial value of this field is 0.

FacilityLike (MQCHAR4):

FacilityLike is the name of an installed terminal that is to be used as a model for the bridge facility.

A value of blanks means that *FacilityLike* is taken from the bridge transaction profile definition, or a default value is used.

This field is a request field used only for 3270 transactions. The length of this field is given by MQ_FACILITY_LIKE_LENGTH. The initial value of this field is four blanks.

Flags (MQLONG):

This field is a request field. The initial value of this field is MQCIH_NONE.

The value must be:

MQCIH_NONE
No flags.

MQCIH_PASS_EXPIRATION
The reply message contains:

- The same expiry report options as the request message.
- The remaining expiry time from the request message with no adjustment made for the processing time of the bridge.

If you omit this value, the expiry time is set to *unlimited*.

MQCIH_REPLY_WITHOUT_NULLS
The reply message length of a CICS DPL program request is adjusted to exclude trailing nulls (X'00') at the end of the COMMAREA returned by the DPL program. If this value is not set, the nulls might be significant, and the full COMMAREA is returned.

MQCIH_SYNC_ON_RETURN
The CICS link for DPL requests uses the SYNCONRETURN option, causing CICS to take a sync point when the program completes if it is shipped to another CICS region. The bridge does not specify to which CICS region to ship the request; that is controlled by the CICS program definition or workload balancing facilities.

Format (MQCHAR8):

This field shows the IBM MQ format name of the data that follows the MQCIH structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as the rules for coding the *Format* field in MQMD.

This format name is also used for the reply message, if the *ReplyToFormat* field has the value MQFMT_NONE.

- For DPL requests, *Format* must be the format name of the COMMAREA.
- For 3270 requests, *Format* must be CSQCBDCI, and the bridge sets the format to CSQCBDCO for Reply messages.

The data-conversion exits for these formats must be installed on the queue manager where they are to run.

If the request message generates an error reply message, the error reply message has a format name of MQFMT_STRING.

This field is a request field. The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

Function (MQCHAR4):

This field is a response field. The length of this field is given by MQ_FUNCTION_LENGTH. The initial value of this field is MQCFUNC_NONE.

The value returned in this field depends on *ReturnCode* ; see Table 181 on page 2256. The following values are possible when *Function* contains an IBM MQ call name:

MQCFUNC_MQCONN
MQCONN call.

MQCFUNC_MQGET
MQGET call.

MQCFUNC_MQINQ
MQINQ call.

MQCFUNC_MQOPEN
MQOPEN call.

MQCFUNC_MQPUT
MQPUT call.

MQCFUNC_MQPUT1
MQPUT1 call.

MQCFUNC_NONE
No call.

In all cases, for the C programming language the constants MQCFUNC_*_ARRAY are also defined; these constants have the same values as the corresponding MQCFUNC_* constants, but are arrays of characters instead of strings.

GetWaitInterval (MQLONG):

This field is a request field. Its initial value is MQCGWI_DEFAULT.

This field applies only when *UOWControl* has the value MQCUOWC_FIRST. It enables the sending application to specify the approximate time in milliseconds that the MQGET calls issued by the bridge will wait for second and subsequent request messages for the unit of work started by this message. This facility overrides the default wait interval used by the bridge. You can use the following special values:

MQCGWI_DEFAULT
Default wait interval.

This value causes the CICS bridge to wait for the time specified when the bridge was started.

MQWI_UNLIMITED
Unlimited wait interval.

InputItem (MQLONG):

This field is a reserved field. The value must be 0.

This field is not present if *Version* is less than MQCIH_VERSION_2.

LinkType (MQLONG):

This field is a request field. Its initial value is MQCLT_PROGRAM.

This value indicates the type of object that the bridge tries to link. It must be one of the following values:

MQCLT_PROGRAM
DPL program.

MQCLT_TRANSACTION
3270 transaction.

NextTransactionId (MQCHAR4):

This value is the name of the next transaction returned by the user transaction (usually by EXEC CICS RETURN TRANSID). If there is no next transaction, this field is set to blanks.

This field is a response field used only for 3270 transactions. The length of this field is given by MQ_TRANSACTION_ID_LENGTH. The initial value of this field is four blanks.

OutputDataLength (MQLONG):

This field is a request field used only for DPL programs. Its initial value is MQCODL_AS_INPUT.

This value is the length of the user data to be returned to the client in a reply message. This length includes the 8-byte program name. The length of the COMMAREA passed to the linked program is the maximum of this field and the length of the user data in the request message, minus 8.

Note: The length of the user data in a message is the length of the message excluding the MQCIH structure.

If the length of the user data in the request message is smaller than *OutputDataLength*, the DATALENGTH option of the LINK command is used, enabling the LINK to be function-shipped efficiently to another CICS region.

You can use the following special value:

MQCODL_AS_INPUT
Output length is same as input length.

This value might be needed even if no reply is requested, in order to ensure that the COMMAREA passed to the linked program is of sufficient size.

Reason (MQLONG):

This field is a response field. Its initial value is MQRC_NONE.

The value returned in this field depends on *ReturnCode* ; see Table 181 on page 2256.

RemoteSysId (MQCHAR4):

This field shows the CICS system identifier of the CICS system processing the request.

If this field is blank, the CICS system request is processed on the same CICS system as the bridge monitor. The SYSID used is returned in the Reply message.

For a 3270 pseudo-conversation, all subsequent messages in the conversation must specify the remote SYSID returned in the initial reply. If specified, the SYSID must:

- Be active.
- Have access to the IBM MQ Request queue.
- Be accessible by the CICS ISC links from the CICS system of the bridge monitor.

RemoteTransId (MQCHAR4):

This field is an optional Request field. The length of this field is given by MQ_TRANSACTION_ID_LENGTH.

If specified, the field is used as the RTRANSID value of CICS START.

ReplyToFormat (MQCHAR8):

The value of this field is the IBM MQ format name of the reply message that is sent in response to the current message.

The rules for coding this field are the same as those rules for coding the *Format* field in MQMD.

This field is a request field used only for DPL programs. The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

Reserved1 (MQCHAR8):

This field is a reserved field. The value must be 8 blanks.

Reserved2 (MQCHAR8):

This field is a reserved field. The value must be 8 blanks.

Reserved3 (MQCHAR8):

This field is a reserved field. The value must be 8 blanks.

Reserved4 (MQLONG):

This field is a reserved field. The value must be 0.

This field is not present if *Version* is less than MQCIH_VERSION_2.

ReturnCode (MQLONG):

The value of this field is the return code from the CICS bridge describing the outcome of the processing performed by the bridge. This field is a response field, with an initial value of MQCRC_OK.

The *Function*, *CompCode*, *Reason*, and *AbendCode* fields might contain additional information (see Table 181 on page 2256). The value is one of the following:

MQCRC_APPLICATION_ABEND
(5, X'005') Application ended abnormally.

MQCRC_BRIDGE_ABEND
(4, X'004') CICS bridge ended abnormally.

MQCRC_BRIDGE_ERROR
(3, X'003') CICS bridge detected an error.

MQCRC_BRIDGE_TIMEOUT
(8, X'008') Second or later message within current unit of work not received within specified time.

MQCRC_CICS_EXEC_ERROR
(1, X'001') EXEC CICS statement detected an error.

MQCRC_MQ_API_ERROR
(2, X'002') MQ call detected an error.

MQCRC_OK
(0, X'000') No error.

MQCRC_PROGRAM_NOT_AVAILABLE
(7, X'007') Program not available.

MQCRC_SECURITY_ERROR
(6, X'006') Security error occurred.

MQCRC_TRANSID_NOT_AVAILABLE
(9, X'009') Transaction not available.

StartCode (MQCHAR4):

The value of this field is an indicator specifying whether the bridge emulates a terminal transaction or a transaction initiated with START.

The value must be one of the following:

MQCSC_START
Start.

MQCSC_STARTDATA
Start data.

MQCSC_TERMINPUT
Terminal input.

MQCSC_NONE
None.

In all cases, for the C programming language the constants MQCSC_*_ARRAY are also defined; these constants have the same values as the corresponding MQCSC_* constants, but are arrays of characters instead of strings.

In the response from the bridge, this field is set to the start code appropriate to the next transaction ID contained in the *NextTransactionId* field. The following start codes are possible in the response:

- MQCSC_START
- MQCSC_STARTDATA
- MQCSC_TERMINPUT

For CICS Transaction Server Version 1.2, this field is a request field only; its value in the response is undefined.

For CICS Transaction Server Version 1.3 and subsequent releases, this field is both a request and a response field.

This field is used only for 3270 transactions. The length of this field is given by MQ_START_CODE_LENGTH. The initial value of this field is MQCSC_NONE.

StrucId (MQCHAR4):

This field is a request field, with an initial value of MQCIH_STRUC_ID.

The value must be:

MQCIH_STRUC_ID
Identifier for CICS information header structure.

For the C programming language, the constant MQCIH_STRUC_ID_ARRAY is also defined; this has the same value as MQCIH_STRUC_ID, but is an array of characters instead of a string.

StrucLength (MQLONG):

This field is a request field, with an initial value of MQCIH_LENGTH_2.

The value must be one of the following:

MQCIH_LENGTH_1
Length of version-1 CICS information header structure.

MQCIH_LENGTH_2
Length of version-2 CICS information header structure.

The following constant specifies the length of the current version:

MQCIH_CURRENT_LENGTH
Length of current version of CICS information header structure.

TaskEndStatus (MQLONG):

This field is a response field, showing the status of the user transaction at end of task. The field is used only for 3270 transactions, and its initial value is MQCTES_NOSYNC.

One of the following values is returned:

MQCTES_NOSYNC
Not synchronized.

The user transaction has not yet completed and has not syncpointed. The *MsgType* field in MQMD is MQMT_REQUEST in this case.

MQCTES_COMMIT
Commit unit of work.

The user transaction has not yet completed, but has syncpointed the first unit of work. The *MsgType* field in MQMD is MQMT_DATAGRAM in this case.

MQCTES_BACKOUT
Back out unit of work.

The user transaction has not yet completed. The current unit of work is backed out. The *MsgType* field in MQMD is MQMT_DATAGRAM in this case.

MQCTES_ENDTASK
End task.

The user transaction has ended (or abended). The *MsgType* field in MQMD is MQMT_REPLY in this case.

TransactionId (MQCHAR4):

This field is a request field. Its length is given by MQ_TRANSACTION_ID_LENGTH. The initial value of this field is four blanks.

If *LinkType* has the value MQCLT_TRANSACTION, *TransactionId* is the transaction identifier of the user transaction to be run; specify a nonblank value in this case.

If *LinkType* has the value MQCLT_PROGRAM, *TransactionId* is the transaction code under which all programs within the unit of work are to be run. If you specify a blank value, the CICS DPL bridge default transaction code (CKBP) is used. If the value is nonblank, you must have defined it to CICS as a local transaction with an initial program that is CSQCBP00. This field applies only when *UOWControl* has the value MQCUOWC_FIRST or MQCUOWC_ONLY.

UOWControl (MQLONG):

This field is a request field which controls the unit-of-work processing performed by the CICS bridge. The initial value of this field is MQCUOWC_ONLY.

You can request the bridge to run a single transaction, or one or more programs within a unit of work. The field indicates whether the CICS bridge starts a unit of work, performs the requested function within the current unit of work, or ends the unit of work by committing it or backing it out. Various combinations are supported, to optimize the data transmission flows.

The value must be one of the following:

MQCUOWC_ONLY
Start unit of work, perform function, then commit the unit of work.

MQCUOWC_CONTINUE
Additional data for the current unit of work (3270 only).

MQCUOWC_FIRST
Start unit of work and perform function.

MQCUOWC_MIDDLE
Perform function within current unit of work

MQCUOWC_LAST
Perform function, then commit the unit of work.

MQCUOWC_COMMIT
Commit the unit of work (DPL only).

MQCUOWC_BACKOUT
Back out the unit of work (DPL only).

Version (MQLONG):

This field is a request field. Its initial value is MQCIH_VERSION_2.

The value must be one of the following:

MQCIH_VERSION_1
Version-1 CICS information header structure.

MQCIH_VERSION_2
Version-2 CICS information header structure.

Fields that exist only in the more-recent version of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQCIH_CURRENT_VERSION
Current version of CICS information header structure.

Initial values and language declarations for MQCIH:

Table 182. Initial values of fields in MQCIH for MQCIH

Field name	Name of constant	Value of constant
StrucId	MQCIH_STRUC_ID	'CIH¬'
Version	MQCIH_VERSION_2	2
StrucLength	MQCIH_LENGTH_2	180
Encoding	None	0
CodedCharSetId	None	0
Format	MQFMT_NONE	Blanks
Flags	MQCIH_NONE	0
ReturnCode	MQCRC_OK	0
CompCode	MQCC_OK	0
Reason	MQRC_NONE	0
UOWControl	MQCUOWC_ONLY	273
GetWaitInterval	MQCGWI_DEFAULT	-2
LinkType	MQCLT_PROGRAM	1
OutputDataLength	MQCODL_AS_INPUT	-1
FacilityKeepTime	None	0
ADSDescriptor	MQCADSD_NONE	0
ConversationalTask	MQCCT_NO	0
TaskEndStatus	MQCTES_NOSYNC	0
Facility	MQCFAC_NONE	Nulls
Function	MQCFUNC_NONE	Blanks
AbendCode	None	Blanks
Authenticator	None	Blanks
Reserved1	None	Blanks
ReplyToFormat	MQFMT_NONE	Blanks
RemoteSysId	None	Blanks
RemoteTransId	None	Blanks

Table 182. Initial values of fields in MQCIH for MQCIH (continued)

Field name	Name of constant	Value of constant
TransactionId	None	Blanks
FacilityLike	None	Blanks
AttentionId	None	Blanks
StartCode	MQCSC_NONE	Blanks
CancelCode	None	Blanks
NextTransactionId	None	Blanks
Reserved2	None	Blanks
Reserved3	None	Blanks
CursorPosition	None	0
ErrorOffset	None	0
InputItem	None	0
Reserved4	None	0

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQCIH_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure:

   ```
   MQCIH MyCIH = {MQCIH_DEFAULT};
   ```

C declaration for MQCIH:

```
typedef struct tagMQCIH MQCIH;
struct tagMQCIH {
  MQCHAR4  StrucId;             /* Structure identifier */
  MQLONG   Version;             /* Structure version number */
  MQLONG   StrucLength;         /* Length of MQCIH structure */
  MQLONG   Encoding;            /* Reserved */
  MQLONG   CodedCharSetId;      /* Reserved */
  MQCHAR8  Format;              /* MQ format name of data that follows
                                   MQCIH */
  MQLONG   Flags;               /* Flags */
  MQLONG   ReturnCode;          /* Return code from bridge */
  MQLONG   CompCode;            /* MQ completion code or CICS EIBRESP */
  MQLONG   Reason;              /* MQ reason or feedback code, or CICS
                                   EIBRESP2 */
  MQLONG   UOWControl;          /* Unit-of-work control */
  MQLONG   GetWaitInterval;     /* Wait interval for MQGET call issued
                                   by bridge task */
  MQLONG   LinkType;            /* Link type */
  MQLONG   OutputDataLength;    /* Output COMMAREA data length */
  MQLONG   FacilityKeepTime;    /* Bridge facility release time */
  MQLONG   ADSDescriptor;       /* Send/receive ADS descriptor */
  MQLONG   ConversationalTask;  /* Whether task can be conversational */
  MQLONG   TaskEndStatus;       /* Status at end of task */
  MQBYTE8  Facility;            /* Bridge facility token */
  MQCHAR4  Function;            /* MQ call name or CICS EIBFN
                                   function */
  MQCHAR4  AbendCode;           /* Abend code */
  MQCHAR8  Authenticator;       /* Password or passticket */
  MQCHAR8  Reserved1;           /* Reserved */
  MQCHAR8  ReplyToFormat;       /* MQ format name of reply message */
  MQCHAR4  RemoteSysId;         /* Reserved */
  MQCHAR4  RemoteTransId;       /* Reserved */
```

```
  MQCHAR4  TransactionId;       /* Transaction to attach */
  MQCHAR4  FacilityLike;        /* Terminal emulated attributes */
  MQCHAR4  AttentionId;         /* AID key */
  MQCHAR4  StartCode;           /* Transaction start code */
  MQCHAR4  CancelCode;          /* Abend transaction code */
  MQCHAR4  NextTransactionId;   /* Next transaction to attach */
  MQCHAR8  Reserved2;           /* Reserved */
  MQCHAR8  Reserved3;           /* Reserved */
  MQLONG   CursorPosition;      /* Cursor position */
  MQLONG   ErrorOffset;         /* Offset of error in message */
  MQLONG   InputItem;           /* Reserved */
  MQLONG   Reserved4;           /* Reserved */
};
```

COBOL declaration for MQCIH:

```
**   MQCIH structure
  10 MQCIH.
**    Structure identifier
   15 MQCIH-STRUCID            PIC X(4).
**    Structure version number
   15 MQCIH-VERSION            PIC S9(9) BINARY.
**    Length of MQCIH structure
   15 MQCIH-STRUCLENGTH        PIC S9(9) BINARY.
**    Reserved
   15 MQCIH-ENCODING           PIC S9(9) BINARY.
**    Reserved
   15 MQCIH-CODEDCHARSETID     PIC S9(9) BINARY.
**    MQ format name of data that follows MQCIH
   15 MQCIH-FORMAT             PIC X(8).
**    Flags
   15 MQCIH-FLAGS              PIC S9(9) BINARY.
**    Return code from bridge
   15 MQCIH-RETURNCODE         PIC S9(9) BINARY.
**    MQ completion code or CICS EIBRESP
   15 MQCIH-COMPCODE           PIC S9(9) BINARY.
**    MQ reason or feedback code, or CICS EIBRESP2
   15 MQCIH-REASON             PIC S9(9) BINARY.
**    Unit-of-work control
   15 MQCIH-UOWCONTROL         PIC S9(9) BINARY.
**    Wait interval for MQGET call issued by bridge task
   15 MQCIH-GETWAITINTERVAL    PIC S9(9) BINARY.
**    Link type
   15 MQCIH-LINKTYPE           PIC S9(9) BINARY.
**    Output COMMAREA data length
   15 MQCIH-OUTPUTDATALENGTH   PIC S9(9) BINARY.
**    Bridge facility release time
   15 MQCIH-FACILITYKEEPTIME   PIC S9(9) BINARY.
**    Send/receive ADS descriptor
   15 MQCIH-ADSDESCRIPTOR      PIC S9(9) BINARY.
**    Whether task can be conversational
   15 MQCIH-CONVERSATIONALTASK PIC S9(9) BINARY.
**    Status at end of task
   15 MQCIH-TASKENDSTATUS      PIC S9(9) BINARY.
**    Bridge facility token
   15 MQCIH-FACILITY           PIC X(8).
**    MQ call name or CICS EIBFN function
   15 MQCIH-FUNCTION           PIC X(4).
**    Abend code
   15 MQCIH-ABENDCODE          PIC X(4).
**    Password or passticket
   15 MQCIH-AUTHENTICATOR      PIC X(8).
**    Reserved
   15 MQCIH-RESERVED1          PIC X(8).
**    MQ format name of reply message
   15 MQCIH-REPLYTOFORMAT      PIC X(8).
**    Reserved
```

```
   15 MQCIH-REMOTESYSID        PIC X(4).
**    Reserved
   15 MQCIH-REMOTETRANSID      PIC X(4).
**    Transaction to attach
   15 MQCIH-TRANSACTIONID      PIC X(4).
**    Terminal emulated attributes
   15 MQCIH-FACILITYLIKE       PIC X(4).
**    AID key
   15 MQCIH-ATTENTIONID        PIC X(4).
**    Transaction start code
   15 MQCIH-STARTCODE          PIC X(4).
**    Abend transaction code
   15 MQCIH-CANCELCODE         PIC X(4).
**    Next transaction to attach
   15 MQCIH-NEXTTRANSACTIONID  PIC X(4).
**    Reserved
   15 MQCIH-RESERVED2          PIC X(8).
**    Reserved
   15 MQCIH-RESERVED3          PIC X(8).
**    Cursor position
   15 MQCIH-CURSORPOSITION     PIC S9(9) BINARY.
**    Offset of error in message
   15 MQCIH-ERROROFFSET        PIC S9(9) BINARY.
**    Reserved
   15 MQCIH-INPUTITEM          PIC S9(9) BINARY.
**    Reserved
   15 MQCIH-RESERVED4          PIC S9(9) BINARY.
```

PL/I declaration for MQCIH:

```
dcl
 1 MQCIH based,
  3 StrucId            char(4),       /* Structure identifier */
  3 Version            fixed bin(31), /* Structure version number */
  3 StrucLength        fixed bin(31), /* Length of MQCIH structure */
  3 Encoding           fixed bin(31), /* Reserved */
  3 CodedCharSetId     fixed bin(31), /* Reserved */
  3 Format             char(8),       /* MQ format name of data that
                                         follows MQCIH */
  3 Flags              fixed bin(31), /* Flags */
  3 ReturnCode         fixed bin(31), /* Return code from bridge */
  3 CompCode           fixed bin(31), /* MQ completion code or CICS
                                         EIBRESP */
  3 Reason             fixed bin(31), /* MQ reason or feedback code, or
                                         CICS EIBRESP2 */
  3 UOWControl         fixed bin(31), /* Unit-of-work control */
  3 GetWaitInterval    fixed bin(31), /* Wait interval for MQGET call
                                         issued by bridge task */
  3 LinkType           fixed bin(31), /* Link type */
  3 OutputDataLength   fixed bin(31), /* Output COMMAREA data length */
  3 FacilityKeepTime   fixed bin(31), /* Bridge facility release time */
  3 ADSDescriptor      fixed bin(31), /* Send/receive ADS descriptor */
  3 ConversationalTask fixed bin(31), /* Whether task can be
                                         conversational */
  3 TaskEndStatus      fixed bin(31), /* Status at end of task */
  3 Facility           char(8),       /* Bridge facility token */
  3 Function           char(4),       /* MQ call name or CICS EIBFN
                                         function */
  3 AbendCode          char(4),       /* Abend code */
  3 Authenticator      char(8),       /* Password or passticket */
  3 Reserved1          char(8),       /* Reserved */
  3 ReplyToFormat      char(8),       /* MQ format name of reply
                                         message */
  3 RemoteSysId        char(4),       /* Reserved */
  3 RemoteTransId      char(4),       /* Reserved */
  3 TransactionId      char(4),       /* Transaction to attach */
  3 FacilityLike       char(4),       /* Terminal emulated attributes */
```

```
  3 AttentionId        char(4),       /* AID key */
  3 StartCode          char(4),       /* Transaction start code */
  3 CancelCode         char(4),       /* Abend transaction code */
  3 NextTransactionId  char(4),       /* Next transaction to attach */
  3 Reserved2          char(8),       /* Reserved */
  3 Reserved3          char(8),       /* Reserved */
  3 CursorPosition     fixed bin(31), /* Cursor position */
  3 ErrorOffset        fixed bin(31), /* Offset of error in message */
  3 InputItem          fixed bin(31), /* Reserved */
  3 Reserved4          fixed bin(31); /* Reserved */
```

High Level Assembler declaration for MQCIH:

```
MQCIH                   DSECT
MQCIH_STRUCID           DS   CL4  Structure identifier
MQCIH_VERSION           DS   F    Structure version number
MQCIH_STRUCLENGTH       DS   F    Length of MQCIH structure
MQCIH_ENCODING          DS   F    Reserved
MQCIH_CODEDCHARSETID    DS   F    Reserved
MQCIH_FORMAT            DS   CL8  MQ format name of data that follows
*                                 MQCIH
MQCIH_FLAGS             DS   F    Flags
MQCIH_RETURNCODE        DS   F    Return code from bridge
MQCIH_COMPCODE          DS   F    MQ completion code or CICS EIBRESP
MQCIH_REASON            DS   F    MQ reason or feedback code, or CICS
*                                 EIBRESP2
MQCIH_UOWCONTROL        DS   F    Unit-of-work control
MQCIH_GETWAITINTERVAL   DS   F    Wait interval for MQGET call issued
*                                 by bridge task
MQCIH_LINKTYPE          DS   F    Link type
MQCIH_OUTPUTDATALENGTH  DS   F    Output COMMAREA data length
MQCIH_FACILITYKEEPTIME  DS   F    Bridge facility release time
MQCIH_ADSDESCRIPTOR     DS   F    Send/receive ADS descriptor
MQCIH_CONVERSATIONALTASK DS  F    Whether task can be conversational
MQCIH_TASKENDSTATUS     DS   F    Status at end of task
MQCIH_FACILITY          DS   XL8  Bridge facility token
MQCIH_FUNCTION          DS   CL4  MQ call name or CICS EIBFN function
MQCIH_ABENDCODE         DS   CL4  Abend code
MQCIH_AUTHENTICATOR     DS   CL8  Password or passticket
MQCIH_RESERVED1         DS   CL8  Reserved
MQCIH_REPLYTOFORMAT     DS   CL8  MQ format name of reply message
MQCIH_REMOTESYSID       DS   CL4  Reserved
MQCIH_REMOTETRANSID     DS   CL4  Reserved
MQCIH_TRANSACTIONID     DS   CL4  Transaction to attach
MQCIH_FACILITYLIKE      DS   CL4  Terminal emulated attributes
MQCIH_ATTENTIONID       DS   CL4  AID key
MQCIH_STARTCODE         DS   CL4  Transaction start code
MQCIH_CANCELCODE        DS   CL4  Abend transaction code
MQCIH_NEXTTRANSACTIONID DS   CL4  Next transaction to attach
MQCIH_RESERVED2         DS   CL8  Reserved
MQCIH_RESERVED3         DS   CL8  Reserved
MQCIH_CURSORPOSITION    DS   F    Cursor position
MQCIH_ERROROFFSET       DS   F    Offset of error in message
MQCIH_INPUTITEM         DS   F    Reserved
MQCIH_RESERVED4         DS   F    Reserved
*
MQCIH_LENGTH            EQU  *-MQCIH
                        ORG  MQCIH
MQCIH_AREA              DS   CL(MQCIH_LENGTH)
```

Visual Basic declaration for MQCIH:

```
Type MQCIH
  StrucId            As String*4 'Structure identifier'
  Version            As Long     'Structure version number'
  StrucLength        As Long     'Length of MQCIH structure'
  Encoding           As Long     'Reserved'
  CodedCharSetId     As Long     'Reserved'
  Format             As String*8 'MQ format name of data that follows'
                                 'MQCIH'
  Flags              As Long     'Flags'
  ReturnCode         As Long     'Return code from bridge'
  CompCode           As Long     'MQ completion code or CICS EIBRESP'
  Reason             As Long     'MQ reason or feedback code, or CICS'
                                 'EIBRESP2'
  UOWControl         As Long     'Unit-of-work control'
  GetWaitInterval    As Long     'Wait interval for MQGET call issued'
                                 'by bridge task'
  LinkType           As Long     'Link type'
  OutputDataLength   As Long     'Output COMMAREA data length'
  FacilityKeepTime   As Long     'Bridge facility release time'
  ADSDescriptor      As Long     'Send/receive ADS descriptor'
  ConversationalTask As Long     'Whether task can be conversational'
  TaskEndStatus      As Long     'Status at end of task'
  Facility           As MQBYTE8  'Bridge facility token'
  Function           As String*4 'MQ call name or CICS EIBFN function'
  AbendCode          As String*4 'Abend code'
  Authenticator      As String*8 'Password or passticket'
  Reserved1          As String*8 'Reserved'
  ReplyToFormat      As String*8 'MQ format name of reply message'
  RemoteSysId        As String*4 'Reserved'
  RemoteTransId      As String*4 'Reserved'
  TransactionId      As String*4 'Transaction to attach'
  FacilityLike       As String*4 'Terminal emulated attributes'
  AttentionId        As String*4 'AID key'
  StartCode          As String*4 'Transaction start code'
  CancelCode         As String*4 'Abend transaction code'
  NextTransactionId  As String*4 'Next transaction to attach'
  Reserved2          As String*8 'Reserved'
  Reserved3          As String*8 'Reserved'
  CursorPosition     As Long     'Cursor position'
  ErrorOffset        As Long     'Offset of error in message'
  InputItem          As Long     'Reserved'
  Reserved4          As Long     'Reserved'
End Type
```

MQCMHO - Create message handle options:

The following table summarizes the fields in the structure.

Table 183. Fields in MQCMHO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options	Options

Overview for MQCMHO:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, z/OS and IBM MQ clients.

Purpose: The **MQCMHO** structure allows applications to specify options that control how message handles are created. The structure is an input parameter on the **MQCRTMH** call.

Character set and encoding: Data in **MQCMHO** must be in the character set of the application and encoding of the application (**MQENC_NATIVE**).

Fields for MQCMHO:

The MQCMHO structure contains the following fields; the fields are described in **alphabetical order**:

Options (MQLONG):

This field is always an input field. Its initial value is MQCMHO_DEFAULT_VALIDATION.

One of the following options can be specified:

MQCMHO_VALIDATE

When **MQSETMP** is called to set a property in this message handle, the property name is validated to ensure that it:

- contains no invalid characters.
- does not begin JMS or usr.JMS except for the following:
  - JMSCorrelationID
  - JMSReplyTo
  - JMSType
  - JMSXGroupID
  - JMSXGroupSeq

  These names are reserved for JMS properties.
- is not one of the following keywords, in any mixture of uppercase or lowercase:
  - AND
  - BETWEEN
  - ESCAPE
  - FALSE
  - IN
  - IS
  - LIKE
  - NOT
  - NULL
  - OR
  - TRUE
- does not begin Body. or Root. (except for Root.MQMD.).

If the property is MQ-defined (mq.*) and the name is recognized, the property descriptor fields are set to the correct values for the property. If the property is not recognized, the *Support* field of the property descriptor is set to **MQPD_OPTIONAL**.

MQCMHO_DEFAULT_VALIDATION

This value specifies that the default level of validation of property names occur.

The default level of validation is equivalent to the level specified by **MQCMHO_VALIDATE**.

This value is the default value.

MQCMHO_NO_VALIDATION

No validation on the property name occurs. See the description of **MQCMHO_VALIDATE**.

Default option: If none of the preceding options described is required, the following option can be used:

MQCMHO_NONE

All options assume their default values. Use this value to indicate that no other options have been specified. **MQCMHO_NONE** aids program documentation; it is not intended that this option is used with any other, but as its value is zero, such use cannot be detected.

StrucId (MQCHAR4):

This field is always an input field. Its initial value is MQCMHO_STRUC_ID.

This is the structure identifier; the value must be:

MQCMHO_STRUC_ID

Identifier for create message handle options structure.

For the C programming language, the constant **MQCMHO_STRUC_ID_ARRAY** is also defined; this has the same value as **MQCMHO_STRUC_ID**, but is an array of characters instead of a string.

Version (MQLONG):

This field is always an input field. Its initial value is MQCMHO_VERSION_1.

This is the structure version number; the value must be:

MQCMHO_VERSION_1
Version-1 create message handle options structure.

The following constant specifies the version number of the current version:

MQCMHO_CURRENT_VERSION
Current version of create message handle options structure.

Initial values and language declarations for MQCMHO:

Table 184. Initial values of fields in MQCMHO

Field name	Name of constant	Value of constant
StrucId	MQCMHO_STRUC_ID	'CMHO'
Version	MQCMHO_VERSION_1	1
Options	MQCMHO_DEFAULT_VALIDATION	0

Notes:

1. In the C programming language, the macro variable MQCMHO_DEFAULT contains the values that are listed in the table. It can be used in the following way to provide initial values for the fields in the structure:

```
MQCMHO MyCMHO = {MQCMHO_DEFAULT};
```

C declaration for MQCMHO:

```
struct tagMQCMHO {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    Options;         /* Options that control the action of MQCRTMH */
};
```

COBOL declaration for MQCMHO:

```
**   MQCMHO structure
  10 MQCMHO.
**    Structure identifier
   15 MQCMHO-STRUCID        PIC X(4).
**    Structure version number
   15 MQCMHO-VERSION        PIC S9(9) BINARY.
**    Options that control the action of MQCRTMH
   15 MQCMHO-OPTIONS        PIC S9(9) BINARY.
```

PL/I declaration for MQCMHO:

```
dcl
 1 MQCMHO based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 Options        fixed bin(31), /* Options that control the action of MQCRTMH */
```

High Level Assembler declaration for MQCMHO:

```
MQCMHO                DSECT
MQCMHO_STRUCID        DS   CL4   Structure identifier
MQCMHO_VERSION        DS   F     Structure version number
MQCMHO_OPTIONS        DS   F     Options that control the action of
*                                MQCRTMH
MQCMHO_LENGTH         EQU  *-MQCMHO
MQCMHO_AREA           DS   CL(MQCMHO_LENGTH)
```

MQCNO - Connect options:

The following table summarizes the fields in the structure.

Table 185. Fields in MQCNO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options that control the action of MQCONNX	Options
Note: The remaining fields are ignored if *Version* is less than MQCNO_VERSION_2.		
ClientConnOffset	Offset of MQCD structure for client connection	ClientConnOffset
ClientConnPtr	Address of MQCD structure for client connection	ClientConnPtr
Note: The remaining fields are ignored if *Version* is less than MQCNO_VERSION_3.		
ConnTag	Queue manager connection tag	ConnTag
Note: The remaining fields are ignored if *Version* is less than MQCNO_VERSION_4.		
SSLConfigPtr	Address of MQSCO structure for client connection	SSLConfigPtr
SSLConfigOffset	Offset of MQSCO structure for client connection	SSLConfigOffset
Note: The remaining fields are ignored if *Version* is less than MQCNO_VERSION_5.		
ConnectionId	Unique connection ID	ConnectionId
SecurityParmsOffset	Security parameters	SecurityParmsOffset
SecurityParmsPtr	Security parameters	SecurityParmsPtr

Table 185. Fields in MQCNO (continued)

Field	Description	Topic
V 9.0.0 **Note:** The remaining fields are ignored if *Version* is less than MQCNO_VERSION_6.		
V 9.0.0 *CCDTUrlLength*	CCDT URL length	CCDTUrlLength
V 9.0.0 *CCDTUrlOffset*	CCDT URL offset	CCDTUrlOffset
V 9.0.0 *CCDTUrlPtr*	CCDT URL pointer	CCDTUrlPtr

Related information:
Using MQCONNX

Overview for MQCNO:

Availability: All versions except MQCNO_VERSION_4: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ MQI clients connected to these systems.

Purpose: The MQCNO structure allows the application to specify options relating to the connection to the local queue manager. The structure is an input/output parameter on the MQCONNX call. For more information about using shared handles, and the MQCONNX call, see Shared (thread independent) connections with MQCONNX .

Version: The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQCNO, but with the initial value of the *Version* field set to MQCNO_VERSION_1. To use fields that are not present in the version-1 structure, the application must set the *Version* field to the version number of the version required.

Character set and encoding: Data in MQCNO must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an IBM MQ MQI client, the structure must be in the character set and encoding of the client.

Fields for MQCNO:

The MQCNO structure contains the following fields; the fields are described in **alphabetical order**:

CCDTUrlLength (MQLONG): V 9.0.0

CCDTUrlLength is the length of the string identified by either CCDTUrlPtr or CCDTUrlOffset which contains a URL that identifies the location of the client connection channel table to use for the connection. The initial value of the field is zero.

Use *CCDTUrlLength* only when the application issuing the MQCONNX call is running as an IBM MQ MQI client.

This is a programmatic alternative to setting the MQCHLLIB and MQCHLTAB environment variables.

If the application is not running as an client, *CCDTUrlLength* is ignored.

This field is ignored if *Version* is less than MQCNO_VERSION_6.

CCDTUrlOffset (MQLONG): V 9.0.0

CCDTUrlOffset is the offset in bytes, from the start of the MQCNO structure, to a string which contains a URL that identifies the location of the client connection channel table to use for the connection. The offset can be positive or negative and the initial value of the field is zero.

Use *CCDTUrlOffset* only when the application issuing the MQCONNX call is running as an IBM MQ MQI client.

Important: You can use only one of *CCDTUrlPtr* and *CCDTUrlOffset*. The call fails with reason code MQRC_CCDT_URL_ERROR if both fields are nonzero.

This is a programmatic alternative to setting the MQCHLLIB and MQCHLTAB environment variables.

If the application is not running as an client, *CCDTUrlOffset* is ignored.

This field is ignored if *Version* is less than MQCNO_VERSION_6.

CCDTUrlPtr (PMQCHAR): V 9.0.0

CCDTUrlPtr is an optional pointer to a string which contains a URL, to identify the location of the client connection channel table to use for the connection.. This field is an input field, with an initial value of a null pointer in programming languages that support pointers, and an all-null byte string otherwise.

Use *CCDTUrlPtr* only when the application issuing the MQCONNX call is running as an IBM MQ MQI client.

Important: You can use only one of *CCDTUrlPtr* and *CCDTUrlOffset*. The call fails with reason code MQRC_CCDT_URL_ERROR if both fields are nonzero.

This is a programmatic alternative to setting the MQCHLLIB and MQCHLTAB environment variables.

If the application is not running as an client, *CCDTUrlPtr* is ignored.

This field is ignored if *Version* is less than MQCNO_VERSION_6.

ClientConnOffset (MQLONG):

ClientConnOffset is the offset in bytes of an MQCD channel definition structure from the start of the MQCNO structure. The offset can be positive or negative. This field is an input field with an initial value of 0.

Use *ClientConnOffset* only when the application issuing the MQCONNX call is running as an IBM MQ MQI client. For information about how to use this field, see the description of the *ClientConnPtr* field.

This field is ignored if *Version* is less than MQCNO_VERSION_2.

ClientConnPtr (MQPTR):

ClientConnPtr is an input field. Its initial value is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise.

Use *ClientConnOffset* and *ClientConnPtr* only when the application issuing the MQCONNX call is running as an IBM MQ MQI client. By specifying one or other of these fields, the application can control the definition of the client connection channel by providing an MQCD channel definition structure that contains the values required.

If the application is running as an IBM MQ MQI client, but does not provide an MQCD structure, the MQSERVER environment variable is used to select the channel definition. If MQSERVER is not set, the client channel table is used.

If the application is not running as an IBM MQ MQI client, *ClientConnOffset* and *ClientConnPtr* are ignored.

If the application provides an MQCD structure, set the fields listed to the values required; other fields in MQCD are ignored. You can pad character strings with blanks to the length of the field, or terminated them with a null character. See "Fields" on page 3560 for more information about the fields in the MQCD structure.

Field in MQCD	Value
ChannelName	Channel name.
Version	Structure version number. Must not be less than MQCD_VERSION_7.
TransportType	Any supported transport type.
ModeName	LU 6.2 mode name.
TpName	LU 6.2 transaction program name.
SecurityExit	Name of channel security exit.
SendExit	Name of channel send exit.
ReceiveExit	Name of channel receive exit.
MaxMsgLength	Maximum length in bytes of messages that can be sent over the client connection channel.
SecurityUserData	User data for security exit.
SendUserData	User data for send exit.
ReceiveUserData	User data for receive exit.
UserIdentifier	User identifier to be used to establish an LU 6.2 session.
Password	Password to be used to establish an LU 6.2 session.
ConnectionName	Connection name.
HeartbeatInterval	Time in seconds between heartbeat flows.
StrucLength	Length of the MQCD structure.
ExitNameLength	Length of exit names addressed by *SendExitPtr* and *ReceiveExitPtr*. Must be greater than zero if *SendExitPtr* or *ReceiveExitPtr* is set to a value that is not the null pointer.
ExitDataLength	Length of exit data addressed by *SendUserDataPtr* and *ReceiveUserDataPtr*. Must be greater than zero if *SendUserDataPtr* or *ReceiveUserDataPtr* is set to a value that is not the null pointer.
SendExitsDefined	Number of send exits addressed by *SendExitPtr*. If zero, *SendExit* and *SendUserData* provide the exit name and data. If greater than zero, *SendExitPtr* and *SendUserDataPtr* provide the exit names and data, and *SendExit* and *SendUserData* must be blank.
ReceiveExitsDefined	Number of receive exits addressed by *ReceiveExitPtr*. If zero, *ReceiveExit* and *ReceiveUserData* provide the exit name and data. If greater than zero, *ReceiveExitPtr* and *ReceiveUserDataPtr* provide the exit names and data, and *ReceiveExit* and *ReceiveUserData* must be blank.
SendExitPtr	Address of name of first send exit.
SendUserDataPtr	Address of data for first send exit.

Field in MQCD	Value
ReceiveExitPtr	Address of name of first receive exit.
ReceiveUserDataPtr	Address of data for first receive exit.
LongRemoteUserIdLength	Length of long remote user identifier.
LongRemoteUserIdPtr	Address of long remote user identifier.
RemoteSecurityId	Remote security identifier.
SSLCipherSpec	TLS CipherSpec.
SSLPeerNamePtr	Address of TLS peer name.
SSLPeerNameLength	Length of TLS peer name.
KeepAliveInterval	Value passed to the communications stack for keepalive timing for the channel
LocalAddress	The local communications address, including the IP address of the local network adapter to use, and a range of ports to use for outgoing connections.

Provide the channel definition structure in one of two ways:

- By using the offset field *ClientConnOffset*

  In this case, the application must declare a compound structure containing an MQCNO followed by the channel definition structure MQCD, and set *ClientConnOffset* to the offset of the channel definition structure from the start of the MQCNO. Ensure that this offset is correct. *ClientConnPtr* must be set to the null pointer or null bytes.

  Use *ClientConnOffset* for programming languages that do not support the pointer data type, or that implement the pointer data type in a way that is not portable to different environments (for example, the COBOL programming language).

  For the Visual Basic programming language, a compound structure called MQCNOCD is provided in the header file CMQXB.BAS; this structure contains an MQCNO structure followed by an MQCD structure. Initialize MQCNOCD by invoking the MQCNOCD_DEFAULTS subroutine. MQCNOCD is used with the MQCONNXAny variant of the MQCONNX call; see the description of the MQCONNX call for further details.
- By using the pointer field *ClientConnPtr*

  In this case, the application can declare the channel definition structure separately from the MQCNO structure, and set *ClientConnPtr* to the address of the channel definition structure. Set *ClientConnOffset* to zero.

  Use *ClientConnPtr* for programming languages that support the pointer data type in a way that is portable to different environments (for example, the C programming language).

  In the C programming language, you can use the macro variable MQCD_CLIENT_CONN_DEFAULT to provide initial values for the structure that are more suitable for use on the MQCONNX call than the initial values provided by MQCD_DEFAULT.

Whichever technique you choose, you can use only one of *ClientConnOffset* and *ClientConnPtr* ; the call fails with reason code MQRC_CLIENT_CONN_ERROR if both are nonzero.

When the MQCONNX call has completed, the MQCD structure is not referenced again.

This field is ignored if *Version* is less than MQCNO_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

ConnectionId (MQBYTE24):

ConnectionId is a unique 24-byte identifier that allows IBM MQ to reliably identify an application. An application can use this identifier for correlation in PUT and GET calls. This output parameter has an initial value of 24 null bytes in all programming languages.

The queue manager assigns a unique ID to all connections, however they are established. If an MQCONNX establishes the connection with a version 5 MQCNO, the application can determine the ConnectionId from the returned MQCNO. The assigned identifier is guaranteed to be unique among all other identifiers that IBM MQ generates, such as CorrelId, MsgID, and GroupId.

Use the ConnectionId to identify long running units of work using the PCF command Inquire Connection or the MQSC command DISPLAY CONN. The ConnectionId used by MQSC commands (CONN) is derived from the ConnectionId returned here. The PCF Inquire and Stop Connection commands can use the ConnectionId returned here without modification.

You can use the ConnectionId to force the end of a long running unit of work, by specifying the ConnectionId using the PCF command Stop Connection or the MQSC command STOP CONN. See Stop Connection and STOP CONN for more information about using these commands.

This field is not returned if Version is less than MQCNO_VERSION_5.

The length of this field is given by MQ_CONNECTION_ID_LENGTH.

ConnTag (MQBYTE128):

ConnTag is a tag that the queue manager associates with the resources that are affected by the application during this connection. Each application or application instance must use a different value for the tag, so that the queue manager can correctly serialize access to the affected resources. This field is an input field, and its initial value is MQCT_NONE.

See the descriptions of the MQCNO_*_CONN_TAG_* options for further details about values to be used by different applications. The tag ceases to be valid when the application terminates or issues the MQDISC call.

Note: Connection tag values beginning with `MQ` in upper, lower, or mixed case in either ASCII or EBCDIC are reserved for use by IBM products. Do not use connection tag values beginning with these letters.

Use the following special value if you require no tag:

MQCT_NONE
The value is binary zero for the length of the field.

For the C programming language, the constant MQCT_NONE_ARRAY is also defined; this constant has the same value as MQCT_NONE, but is an array of characters instead of a string.

This field is used when connecting to a z/OS queue manager. In other environments, specify the value MQCT_NONE.

The length of this field is given by MQ_CONN_TAG_LENGTH. This field is ignored if *`Version`* is less than MQCNO_VERSION_3.

Options (MQLONG):

Options that control the action of MQCONNX.

Accounting options

The following options control the type of accounting if the **AccountingConnOverride** queue manager attribute is set to MQMON_ENABLED:

MQCNO_ACCOUNTING_MQI_ENABLED
When monitoring data collection is disabled in the queue manager definition by setting the **MQIAccounting** attribute to MQMON_OFF, setting this flag enables MQI accounting data collection.

MQCNO_ACCOUNTING_MQI_DISABLED
When monitoring data collection is disabled in the queue manager definition by setting the **MQIAccounting** attribute to MQMON_OFF, setting this flag stops MQI accounting data collection.

MQCNO_ACCOUNTING_Q_ENABLED
When queue-accounting data collection is disabled in the queue manager definition by setting the **MQIAccounting** attribute to MQMON_OFF, setting this flag enables accounting data collection for those queues that specify a queue manager in the *MQIAccounting* field of their queue definition.

MQCNO_ACCOUNTING_Q_DISABLED
When queue-accounting data collection is disabled in the queue manager definition by setting the **MQIAccounting** attribute to MQMON_OFF, setting this flag switches off accounting data collection for those queues that specify a queue manager in the *MQIAccounting* field of their queue definition.

If none of these flags are defined, the accounting for the connection is as defined in the Queue Manager attributes.

Binding options

The following options control the type of IBM MQ binding to use. Specify only one of these options:

MQCNO_STANDARD_BINDING
The application and the local queue manager agent (the component that manages queuing operations) run in separate units of execution (typically, in separate processes). This arrangement maintains the integrity of the queue manager; that is, it protects the queue manager from errant programs.

If the queue manager supports multiple binding types, and you set MQCNO_STANDARD_BINDING, the queue manager uses the **DefaultBindType** attribute in the *Connection* stanza in the qm.ini file to select the actual type of binding. If this stanza is not defined, or the value cannot be used or is not appropriate for the application, the queue manager selects an appropriate binding type. The queue manager sets the actual binding type used in the connect options.

Use MQCNO_STANDARD_BINDING in situations where the application might not have been fully tested, or might be unreliable or untrustworthy. MQCNO_STANDARD_BINDING is the default.

This option is supported in all environments.

If you are linking to the mqm library, then a standard server connection using the default bind type is attempted first. If the underlying server library failed to load, a client connection is attempted instead.

- If the MQ_CONNECT_TYPE environment variable is specified, one of the following options can be supplied to change the behavior of MQCONN or MQCONNX if MQCNO_STANDARD_BINDING is specified. (The exception to this is if

MQCNO_FASTPATH_BINDING is specified with MQ_CONNECT_TYPE set to `LOCAL` or `STANDARD` to allow fastpath connections to be downgraded by the administrator without a related change to the application:

Value	Meaning
CLIENT	A client connection only is attempted.
FASTPATH	This value was supported in previous releases, but is now ignored if specified.
LOCAL	A server connection only is attempted. Fastpath connections are downgraded to a standard server connection.
STANDARD	Supported for compatibility with previous releases. This value is now treated as LOCAL.

- If the MQ_CONNECTI_TYPE environment variable is not set when MQCONNX is called, a standard server connection using the default bind type is attempted. If the server library fails to load, a client connection is attempted.

MQCNO_FASTPATH_BINDING

The application and the local queue manager agent are part of the same unit of execution. This is in contrast to the typical method of binding, where the application and the local queue manager agent run in separate units of execution.

MQCNO_FASTPATH_BINDING is ignored if the queue manager does not support this type of binding; processing continues as though the option had not been specified.

MQCNO_FASTPATH_BINDING can be of advantage in situations where multiple processes consume more resources than the overall resource used by the application. An application that uses the fastpath binding is known as a *trusted application*.

Consider the following important points when deciding whether to use the fastpath binding:

- Using the MQCNO_FASTPATH_BINDING option does not prevent an application altering or corrupting messages and other data areas belonging to the queue manager. Use this option only in situations where you have fully evaluated these issues.
- The application must not use asynchronous signals or timer interrupts (such as `sigkill`) with MQCNO_FASTPATH_BINDING. There are also restrictions on the use of shared memory segments.
- The application must use the MQDISC call to disconnect from the queue manager.
- The application must finish before you end the queue manager with the `endmqm` command.
- On IBM i, the job must run under a user profile that belongs to the `QMQMADM` group. Also, the program must not stop abnormally, otherwise unpredictable results can occur.
- On UNIX, the `mqm` user identifier must be the effective user identifier, and the `mqm` group identifier must be the effective group identifier. To make the application run in this way, configure the program so that it is owned by the `mqm` user identifier and `mqm` group identifier, and then set the `setuid` and `setgid` permission bits on the program.

  The IBM MQ Object Authority Manager (OAM) still uses the real user ID for authority checking.
- On Windows, the program must be a member of the `mqm` group. Fastpath binding is not supported for 64 bit applications.

The MQCNO_FASTPATH_BINDING option is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, and Windows.

z/OS On z/OS, the option is accepted but ignored.

For more information about the implications of using trusted applications, see Restrictions for trusted applications.

MQCNO_SHARED_BINDING

With MQCNO_SHARED_BINDING, the application and the local queue manager agent share some resources. MQCNO_SHARED_BINDING is ignored if the queue manager does not support this type of binding. Processing continues as though the option had not been specified.

MQCNO_ISOLATED_BINDING

In this case, the application process and the local queue manager agent are isolated from each other in that they do not share resources. MQCNO_ISOLATED_BINDING is ignored if the queue manager does not support this type of binding. Processing continues as though the option had not been specified.

MQCNO_CLIENT_BINDING

Specify this option to make the application attempt a client connection only. This option has the following limitations:

- z/OS MQCNO_CLIENT_BINDING is ignored on z/OS.
- MQCNO_CLIENT_BINDING is rejected with MQRC_OPTIONS_ERROR if it is specified with any MQCNO binding option other than MQCNO_STANDARD_BINDING.
- MQCNO_CLIENT_BINDING is not available for Java or .NET as they have their own mechanisms for choosing the bind type.

MQCNO_LOCAL_BINDING

Specify this option to make the application attempt a server connection. If either MQCNO_FASTPATH_BINDING, MQCNO_ISOLATED_BINDING, or MQCNO_SHARED_BINDING is also specified, then the connection is of that type instead, and is documented in this section. Otherwise a standard server connection is attempted using the default bind type. MQCNO_LOCAL_BINDING has the following limitations:

- z/OS MQCNO_LOCAL_BINDING is ignored on z/OS.
- MQCNO_LOCAL_BINDING is rejected with MQRC_OPTIONS_ERROR if it is specified with any MQCNO reconnect option other than MQCNO_RECONNECT_AS_DEF.
- MQCNO_LOCAL_BINDING is not available for Java or .NET as they have their own mechanisms for choosing the bind type.

On AIX, HP-UX, Solaris, Linux, and Windows, you can use the environment variable `MQ_CONNECT_TYPE` with the bind type specified by the *`Options`* field, to control the type of binding used. If you specify this environment variable, it must have the value `FASTPATH` or `STANDARD` ; if it has a different value, it is ignored. The value of the environment variable is case sensitive; see MQCONNX environment variable for more information.

The environment variable and *`Options`* field interact as follows:

- If you omit the environment variable, or give it a value that is not supported, use of the fastpath binding is determined solely by the *`Options`* field.
- If you give the environment variable a supported value, the fastpath binding is used only if *both* the environment variable and *`Options`* field specify the fastpath binding.

z/OS

Connection-tag options

These options are supported only when connecting to a z/OS queue manager and they control the use of the connection tag *`ConnTag`*. You can specify only one of these options:

MQCNO_SERIALIZE_CONN_TAG_Q_MGR

This option requests exclusive use of the connection tag within the local queue manager. If the connection tag is already in use in the local queue manager, the MQCONNX call fails with reason code MQRC_CONN_TAG_IN_USE. The outcome of the call is not affected by using the connection tag elsewhere in the queue-sharing group to which the local queue manager belongs.

MQCNO_SERIALIZE_CONN_TAG_QSG

This option requests exclusive use of the connection tag within the queue-sharing group to which the local queue manager belongs. If the connection tag is already in use in the queue-sharing group, the MQCONNX call fails with reason code MQRC_CONN_TAG_IN_USE.

MQCNO_RESTRICT_CONN_TAG_Q_MGR

This option requests shared use of the connection tag within the local queue manager. If the connection tag is already in use in the local queue manager, the MQCONNX call can succeed if the requesting application is running in the same processing scope as the existing user of the tag. If this condition is not satisfied, the MQCONNX call fails with reason code MQRC_CONN_TAG_IN_USE. The outcome of the call is not affected by use of the connection tag elsewhere in the queue-sharing group to which the local queue manager belongs.

- Applications must run within the same MVS address space to share the connection tag. If the application using the connection tag is a client application, MQCNO_RESTRICT_CONN_TAG_Q_MGR is not allowed.

MQCNO_RESTRICT_CONN_TAG_QSG

This option requests shared use of the connection tag within the queue-sharing group to which the local queue manager belongs. If the connection tag is already in use in the queue-sharing group, the MQCONNX call can succeed provided the requesting application is running in the same processing scope and is connected to the same queue manager, as the existing user of the tag.

If these conditions are not satisfied, the MQCONNX call fails with reason code MQRC_CONN_TAG_IN_USE.

- Applications must run within the same MVS address space to share the connection tag. If the application using the connection tag is a client application, MQCNO_RESTRICT_CONN_TAG_QSG is not allowed.

If none of these options are specified, *ConnTag* is not used. These options are not valid if *Version* is less than MQCNO_VERSION_3.

Handle-sharing options

These options are supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, and Windows. They control the sharing of handles between different threads (units of parallel processing) within the same process. You can specify only one of these options:

MQCNO_HANDLE_SHARE_NONE

This option indicates that connection and object handles can be used only by the thread that caused the handle to be allocated (that is, the thread that issued the MQCONN, MQCONNX , or MQOPEN call). The handles cannot be used by other threads belonging to the same process.

MQCNO_HANDLE_SHARE_BLOCK

This option indicates that connection and object handles allocated by one thread of a process can be used by other threads belonging to the same process. However, only one thread at a time can use any particular handle; that is, only serial use of a handle is permitted. If a thread tries to use a handle that is already in use by another thread, the call blocks (waits) until the handle becomes available.

MQCNO_HANDLE_SHARE_NO_BLOCK

This is the same as MQCNO_HANDLE_SHARE_BLOCK, except that if the handle is in use by another thread, the call completes immediately with MQCC_FAILED and MQRC_CALL_IN_PROGRESS instead of blocking until the handle becomes available.

A thread can have zero or one non-shared handles:

- Each MQCONN or MQCONNX call that specifies MQCNO_HANDLE_SHARE_NONE returns a new nonshared handle on the first call, and the same non-shared handle on the second and later calls (assuming no intervening MQDISC call). The reason code is MQRC_ALREADY_CONNECTED for the second and later calls.
- Each MQCONNX call that specifies MQCNO_HANDLE_SHARE_BLOCK or MQCNO_HANDLE_SHARE_NO_BLOCK returns a new shared handle on each call.

Object handles inherit the same sharing properties as the connection handle specified on the MQOPEN call that created the object handle. Also, units of work inherit the same sharing properties as the connection handle used to start the unit of work; if the unit of work is started in one thread using a shared handle, the unit of work can be updated in another thread using the same handle.

If you do not specify a handle-sharing option, the default is determined by the environment:

- In the Microsoft Transaction Server (MTS) environment, the default is the same as MQCNO_HANDLE_SHARE_BLOCK.
- In other environments, the default is the same as MQCNO_HANDLE_SHARE_NONE.

Reconnection options

Reconnection options determine if a connection is reconnectable. Only client connections are reconnectable.

MQCNO_RECONNECT_AS_DEF

The reconnection option is resolved to its default value. If no default is set, the value of this option resolves to DISABLED. The value of the option is passed to the server, and can be queried by PCF and MQSC.

MQCNO_RECONNECT

The application can be reconnected to any queue manager consistent with the value of the **QmgrName** parameter of MQCONNX. Use the MQCNO_RECONNECT option only if there is no affinity between the client application and the queue manager with which it initially established a connection. The value of the option is passed to the server, and can be queried by PCF and MQSC.

MQCNO_RECONNECT_DISABLED

The application cannot be reconnected. The value of the option is not passed to the server.

MQCNO_RECONNECT_Q_MGR

The application can be reconnected only to the queue manager with which it originally connected. Use this value if a client can be reconnected, but there is an affinity between the client application and the queue manager with which it originally established a connection. Choose this value if you want a client to automatically reconnect to the standby instance of a highly available queue manager. The value of the option is passed to the server, and can be queried by PCF and MQSC.

Use the options MQCNO_RECONNECT, MQCNO_RECONNECT_DISABLED and MQCNO_RECONNECT_Q_MGR only for client connections. If the options are used for a binding connection, MQCONNX fails with completion code `MQCC_FAILED` and reason code `MQRC_OPTIONS_ERROR`. Automatic client reconnect is not supported by IBM MQ classes for Java

Conversation-sharing options

The following options apply only to TCP/IP client connections. For SNA, SPX and NetBios channels, these values are ignored and the channel runs as in previous versions of the product

MQCNO_NO_CONV_SHARING

This option does not permit conversation sharing.

You might use MQCNO_NO_CONV_SHARING in situations where conversations are heavily loaded and, therefore, where contention is a possibility on the server-connection end of the channel instance on which the sharing conversations exist. MQCNO_NO_CNV_SHARING behaves like sharecnv(1) when connected to a channel that supports conversation sharing, and sharecnv(0) when connected to a channel that does not support conversation sharing.

MQCNO_ALL_CONVS_SHARE

This option permits conversation sharing; the application does not place any limit on the number of connections on the channel instance. This option is the default value.

If the application indicates that the channel instance can share, but the *SharingConversations* (SHARECNV) definition on the server-connection end of the channel is set to one, no sharing occurs and no warning is given to the application.

Similarly, if the application indicates that sharing is permitted but the server-connection *SharingConversations* definition is set to zero, no warning is given, and the application exhibits the same behavior as a client in versions of the product earlier than Version 7.0; the application setting relating to sharing conversations is ignored.

MQCNO_NO_CONV_SHARING and MQCNO_ALL_CONVS_SHARE are mutually exclusive. If both options are specified on a particular connection, the connection is rejected with a reason code of MQRC_OPTIONS_ERROR.

Channel definition options

The following options control the use of the channel definition structure passed in the MQCNO:

MQCNO_CD_FOR_OUTPUT_ONLY

This option permits channel definition structure in the MQCNO to be used only to return the channel name used on a successful MQCONNX call.

If a valid channel definition structure is not provided, the call fails with the reason code MQRC_CD_ERROR.

If the application is not running as a client, the option is ignored.

The returned channel name can be used on a subsequent MQCONNX call using the MQCNO_USE_CD_SELECTION option to reconnect using the same channel definition. This can be useful when there are multiple applicable channel definitions in the client channel table.

MQCNO_USE_CD_SELECTION

This option permits MQCONNX call to connect using the channel name contained in the channel definition structure passed in the MQCNO.

If the MQSERVER environment variable is set, the channel definition defined by it is used. If MQSERVER is not set, the client channel table is used.

If a channel definition with matching channel name and queue manager name is not found, the call fails with reason code MQRC_Q_MGR_NAME_ERROR.

If a valid channel definition structure is not provided, the call fails with the reason code MQRC_CD_ERROR.

If the application is not running as a client, the option is ignored.

Default option

If you require none of the options described above, you can use the following option:

MQCNO_NONE

No options are specified.

Use MQCNO_NONE to aid program documentation. It is not intended that this option is used with any other MQCNO_* option, but because its value is zero, such use cannot be detected.

SecurityParmsOffset (MQLONG):

SecurityParmsOffset is the offset in bytes of the MQCSP structure from the start of the MQCNO structure. The offset can be positive or negative. This field is an input field, with an initial value of 0.

This field is ignored if *Version* is less than MQCNO_VERSION_5.

The MQCSP structure is defined in "MQCSP - Security parameters" on page 2292.

SecurityParmsPtr (PMQCSP):

SecurityParmsPtr is the address of the MQCSP structure, used to specify a user ID and password for authentication by the authorization service. This field is an input field, and its initial value is a null pointer or null bytes.

This field is ignored if *Version* is less than MQCNO_VERSION_5.

The MQCSP structure is defined in "MQCSP - Security parameters" on page 2292.

SSLConfigOffset (MQLONG):

SSLConfigOffset is the offset in bytes of an MQSCO structure from the start of the MQCNO structure. The offset can be positive or negative. This field is an input field, with an initial value of 0.

Use *SSLConfigOffset* only when the application issuing the MQCONNX call is running as an IBM MQ MQI client. For information about how to use this field, see the description of the *SSLConfigPtr* field.

This field is ignored if *Version* is less than MQCNO_VERSION_4.

SSLConfigPtr (PMQSCO):

SSLConfigPtr is an input field. Its initial value is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise.

Use *SSLConfigPtr* and *SSLConfigOffset* only when the application issuing the MQCONNX call is running as an IBM MQ MQI client and the channel protocol is TCP/IP. If the application is not running as an IBM MQ client, or the channel protocol is not TCP/IP, *SSLConfigPtr* and *SSLConfigOffset* are ignored.

By specifying *SSLConfigPtr* or *SSLConfigOffset*, plus either *ClientConnPtr* or *ClientConnOffset*, the application can control the use of TLS for the client connection. When the TLS information is specified in this way, the environment variables MQSSLKEYR and MQSSLCRYP are ignored; any TLS-related information in the client channel definition table (CCDT) is also ignored.

The TLS information can be specified only on:

- The first MQCONNX call of the client process, or

- A subsequent MQCONNX call when all previous TLS connections to the queue manager have been concluded using MQDISC.

These are the only states in which the process-wide TLS environment can be initialized. If an MQCONNX call is issued specifying TLS information when the TLS environment already exists, the TLS information on the call is ignored and the connection is made using the existing TLS environment; the call returns completion code MQCC_WARNING and reason code MQRC_SSL_ALREADY_INITIALIZED in this case.

You can provide the MQSCO structure in the same way as the MQCD structure, either by specifying an address in *SSLConfigPtr*, or by specifying an offset in *SSLConfigOffset* ; see the description of *ClientConnPtr* for details of how to do this. However, you can use no more than one of *SSLConfigPtr* and *SSLConfigOffset* ; the call fails with reason code MQRC_SSL_CONFIG_ERROR. if both are nonzero.

Once the MQCONNX call has completed, the MQSCO structure is not referenced again.

This field is ignored if *Version* is less than MQCNO_VERSION_4.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

StrucId (MQCHAR4):

StrucId is always an input field. Its initial value is MQCNO_STRUC_ID.

The value must be:

MQCNO_STRUC_ID
Identifier for connect-options structure.

For the C programming language, the constant MQCNO_STRUC_ID_ARRAY is also defined; this constant has the same value as MQCNO_STRUC_ID, but is an array of characters instead of a string.

Version (MQLONG):

Version is always an input field. Its initial value is MQCNO_VERSION_1.

The value must be one of the following:

MQCNO_VERSION_1
Version-1 connect-options structure.

MQCNO_VERSION_2
Version-2 connect-options structure.

MQCNO_VERSION_3
Version-3 connect-options structure.

MQCNO_VERSION_4
Version-4 connect-options structure.

MQCNO_VERSION_5
Version-5 connect-options structure.

This version of the MQCNO structure extends MQCNO_VERSION_3 on z/OS, and MQCNO_VERSION_4 on all other platforms.

V 9.0.0 **MQCNO_VERSION_6**
Version-6 connect-options structure.

Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQCNO_CURRENT_VERSION
Current version of connect-options structure.

Initial values and language declarations for MQCNO:

Table 186. Initial values of fields in MQCNO for MQCNO

Field name	Name of constant	Value of constant
StrucId	MQCNO_STRUC_ID	'CNO¬'
Version	MQCNO_VERSION_1	1
Options	MQCNO_NONE	0
ClientConnOffset	None	0
ClientConnPtr	None	Null pointer or null bytes
ConnTag	MQCT_NONE	Nulls
SSLConfigPtr	None	Null pointer or null bytes
SSLConfigOffset	None	0
ConnectionId	None	Null pointer or null bytes
SecurityParmsOffset	None	Null pointer or null bytes
SecurityParmsPtr	None	Null pointer or null bytes
Notes: 1. The symbol ¬ represents a single blank character. 2. In the C programming language, the macro variable MQCNO_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure: `MQCNO MyCNO = {MQCNO_DEFAULT};`		

C declaration for MQCNO:

```
typedef struct tagMQCNO MQCNO;
struct tagMQCNO {
  MQCHAR4    StrucId;           /* Structure identifier */
  MQLONG     Version;           /* Structure version number */
  MQLONG     Options;           /* Options that control the action of
                                   MQCONNX */
  MQLONG     ClientConnOffset;  /* Offset of MQCD structure for client
                                   connection */
  MQPTR      ClientConnPtr;     /* Address of MQCD structure for client
                                   connection */
  MQBYTE128  ConnTag;           /* Queue managerconnection tag */
  PMQSCO     SSLConfigPtr;      /* Address of MQSCO structure for client
                                   connection */
  MQLONG     SSLConfigOffset;   /* Offset of MQSCO structure for client
                                   connection */
  MQBYTE24   ConnectionId;      /* Unique connection identifier */
  MQLONG     SecurityParmsOffset /* Security fields */
  PMQCSP     SecurityParmsPtr   /* Security parameters */
};
```

COBOL declaration for MQCNO:

```
**   MQCNO structure
  10 MQCNO.
**    Structure identifier
   15 MQCNO-STRUCID          PIC X(4).
**    Structure version number
   15 MQCNO-VERSION          PIC S9(9) BINARY.
**    Options that control the action of MQCONNX
   15 MQCNO-OPTIONS          PIC S9(9) BINARY.
**    Offset of MQCD structure for client connection
   15 MQCNO-CLIENTCONNOFFSET PIC S9(9) BINARY.
**    Address of MQCD structure for client connection
   15 MQCNO-CLIENTCONNPTR    POINTER.
**    Queue manager connection tag
   15 MQCNO-CONNTAG          PIC X(128).
**    Address of MQSCO structure for client connection
   15 MQCNO-SSLCONFIGPTR     POINTER.
**    Offset of MQSCO structure for client connection
   15 MQCNO-SSLCONFIGOFFSET  PIC S9(9) BINARY.
**    Unique connection identifier
   15 MQCNO-CONNECTIONID     PIC X(24).
**    Offset of MQCSP structure for security parameters
   15 MQCNO-SECURITYPARMSOFFSET PIC S9(9) BINARY.
**    Address of MQCSP structure for security parameters
   15 MQCNO-SECURITYPARMSPTR POINTER.
```

PL/I declaration for MQCNO:

```
dcl
 1 MQCNO based,
  3 StrucId          char(4),       /* Structure identifier */
  3 Version          fixed bin(31), /* Structure version number */
  3 Options          fixed bin(31), /* Options that control the action
                                       of MQCONNX */
  3 ClientConnOffset fixed bin(31), /* Offset of MQCD structure for
                                       client connection */
  3 ClientConnPtr    pointer,       /* Address of MQCD structure for
                                       client connection */
  3 ConnTag          char(128),     /* Queue managerconnection tag */
  3 SSLConfigPtr     pointer,       /* Address of MQSCO structure for
                                       client connection */
  3 SSLConfigOffset  fixed bin(31), /* Offset of MQSCO structure for
                                       client connection */
  3 ConnectionId     char(24),      /* Unique connection identifier
  3 SecurityParmsOffset fixed bin(31); /* Offset of MQCSP structure for
                                       security parameters */
  3 SecurityParmsPtr pointer,       /* Address of MQCSP structure for
                                       security parameters */
```

High Level Assembler declaration for MQCNO:

```
MQCNO                   DSECT
MQCNO_STRUCID           DS   CL4    Structure identifier
MQCNO_VERSION           DS   F      Structure version number
MQCNO_OPTIONS           DS   F      Options that control the action of
*                                   MQCONNX
MQCNO_CLIENTCONNOFFSET  DS   F      Offset of MQCD structure for client
*                                   connection
MQCNO_CLIENTCONNPTR     DS   F      Address of MQCD structure for client
*                                   connection
MQCNO_CONNTAG           DS   XL128  Queue manager connection tag
*
MQCNO_CONNECTIONID      DS   XL24   Unique connection identifier
*
MQCNO_SSLCONFIGOFFSET   DS   F      Offset of MQCSP structure for security
*                                   parameters
MQCNO_SSLCONFIGPTR      DS   F      Address of MQCSP structure for security
*                                   parameters
MQCNO_LENGTH            EQU  *-MQCNO
                        ORG  MQCNO
MQCNO_AREA              DS   CL(MQCNO_LENGTH)
```

Visual Basic declaration for MQCNO:

```
Type MQCNO
  StrucId          As String*4  'Structure identifier'
  Version          As Long      'Structure version number'
  Options          As Long      'Options that control the action of'
                                'MQCONNX'
  ClientConnOffset As Long      'Offset of MQCD structure for client'
                                'connection'
  ClientConnPtr    As MQPTR     'Address of MQCD structure for client'
                                'connection'
  ConnTag          As MQBYTE128 'Queue manager connection tag'
  SSLConfigPtr     As MQPTR     'Address of MQSCO structure for client'
                                'connection'
  SSLConfigOffset  As Long      'Offset of MQSCO structure for client'
                                'connection'
  ConnectionId     As MQBYTE24  'Unique connection identifier'
  SecurityParmsOffset As Long   'Offset of MQCSP structure for security'
                                'parameters'
  SecurityParmsPtr As MQPTR     'Address of MQCSP structure for security'
                                'parameters'
  End Type
```

MQCSP - Security parameters:

The following table summarizes the fields in the structure.

Warning: In some cases, the password in an MQCSP structure for a client application will be sent across a network in plain text. To ensure that client application passwords are protected appropriately, see IBM MQCSP password protection.

Table 187. Fields in MQCSP

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
AuthenticationType	Type of authentication	AuthenticationType
Reserved1	Required for pointer alignment on IBM i	Reserved1
CSPUserIdPtr	Address of user ID	CSPUserIdPtr
CSPUserIdOffset	Offset of user ID	CSPUserIdOffset
CSPUserIdLength	Length of user ID	CSPUserIdLength
Reserved2	Required for pointer alignment on IBM i	Reserved2
CSPPasswordPtr	Address of password	CSPPasswordPtr
CSPPasswordOffset	Offset of password	CSPPasswordOffset
CSPPasswordLength	Length of password	CSPPasswordLength

Overview for MQCSP:

Availability: All IBM MQ products.

Purpose: The MQCSP structure enables the authorization service to authenticate a user ID and password. You specify the MQCSP connection security parameters structure on an MQCONNX call.

Warning: In some cases, the password in an MQCSP structure for a client application will be sent across a network in plain text. To ensure that client application passwords are protected appropriately, see IBM MQCSP password protection.

Character set and encoding: Data in MQCSP must be in the character set and encoding of the local queue manager; these are given by the **CodedCharSetId** queue manager attribute and MQENC_NATIVE, respectively.

Fields for MQCSP:

The MQCSP structure contains the following fields; the fields are described in **alphabetical order**:

AuthenticationType (MQLONG):

AuthenticationType is an input field. Its initial value is MQCSP_AUTH_NONE.

This is the type of authentication to perform. Valid values are:

MQCSP_AUTH_NONE
Do not use user ID and password fields.

MQCSP_AUTH_USER_ID_AND_PWD
Authenticate user ID and password fields.

The default value is MQCSP_AUTH_NONE. With the default setting, no password protection is done.

If you require authentication, you have to set **MQCSP.AuthenticationType** to MQCSP_AUTH_USER_ID_AND_PWD.

See MQCSP password protection for more information.

CSPPasswordLength (MQLONG):

This field is the length of the password to be used in authentication.

The maximum length of the password is MQ_CSP_PASSWORD_LENGTH. If the length of the password is greater than the maximum length permitted, the authentication request fails with MQRC_NOT_AUTHORIZED.

The value of MQ_CSP_PASSWORD_LENGTH is 256.

This field is an input field. The initial value of this field is 0.

CSPPasswordOffset (MQLONG):

This is the offset in bytes of the password to be used in authentication. The offset can be positive or negative.

This is an input field. The initial value of this field is 0.

CSPPasswordPtr (MQPTR):

This is the address in bytes of the password to be used in authentication.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQCNO_VERSION_5.

This field can contain an empty password which is rejected by the operating system or LDAP password checking, depending on setup, but is not rejected by IBM MQ before passing it the authentication method.

CSPUserIdLength (MQLONG):

This field is the length of the user ID to be used in authentication.

The maximum length of the user ID is dependent on the platform, see User IDs. If the length of the user ID is greater than the maximum length permitted, the authentication request fails with MQRC_NOT_AUTHORIZED.

This field is an input field. The initial value of this field is 0.

CSPUserIdOffset (MQLONG):

This is the offset in bytes of the user ID to be used in authentication. The offset can be positive or negative.

This is an input field. The initial value of this field is 0.

CSPUserIdPtr (MQPTR):

This is the address in bytes of the user ID to be used in authentication.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQCNO_VERSION_5.

This field can contain an operating system user ID when an **AUTHTYPE** of *IDPWOS* is named in the CONNAUTH field of the queue manager.

On Windows this can be a fully qualified domain user ID.

This field can contain an LDAP User ID when an **AUTHTYPE** of *IDPWLDAP* is named in the CONNAUTH field of the queue manager.

Reserved1 (MQBYTE4):

A reserved field, required for pointer alignment on IBM i.

This is an input field. The initial value of this field is all null.

Reserved2 (MQBYTE8):

A reserved field, required for pointer alignment on IBM i.

This is an input field. The initial value of this field is all null.

StrucId (MQCHAR4):

Structure identifier.

The value must be:

MQCSP_STRUC_ID
Identifier for the security parameters structure.

For the C programming language, the constant MQCSP_STRUC_ID_ARRAY is also defined; this has the same value as MQCSP_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQCSPSTRUC_ID.

Version (MQLONG):

Structure version number.

The value must be:

MQCSP_VERSION_1
Version-1 security parameters structure.

The following constant specifies the version number of the current version:

MQCSP_CURRENT_VERSION
Current version of security parameters structure.

This is always an input field. The initial value of this field is MQCSP_VERSION_1.

Initial values and language declarations for MQCSP:

Table 188. Initial values of fields in MQCSP for MQCSP

Field name	Name of constant	Value of constant
StrucId	MQCSP_STRUC_ID	'CSP¬'
Version	MQCSP_CURRENT_VERSION	1
AuthenticationType	None	MQCSP_AUTH_NONE
Reserved1	None	Null string or blanks
CSPUserIdPtr	None	Null pointer or null bytes
CSPUserIdOffset	None	0
CSPUserIdLength	None	0
Reserved2	None	Null string or blanks
CSPPasswordPtr	None	Null pointer or null bytes
CSPPasswordOffset	None	0
CSPPasswordLength	None	0

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQCSP_DEFAULT contains the values that are listed in the table. It can be used in the following way to provide initial values for the fields in the structure:

```
MQCSP MyCSP = {MQCSP_DEFAULT};
```

C declaration for MQCSP:

```
typedef struct tagMQCSP MQCSP;
struct tagMQCSP {
  MQCHAR4    StrucId;            /* Structure identifier */
  MQLONG     Version;            /* Structure version number */
  MQLONG     AuthenticationType; /* Type of authentication */
  MQBYTE4    Reserved1;          /* Required for IBM i pointer
                                    alignment */
  MQPTR      CSPUserIdPtr;       /* Address of user ID */
  MQLONG     CSPUserIdOffset;    /* Offset of user ID */
  MQLONG     CSPUserIdLength;    /* Length of user ID */
  MQBYTE8    Reserved2;          /* Required for IBM i pointer
                                    alignment */
                                 alignment */
  MQPTR      CSPPasswordPtr;     /* Address of password */
  MQLONG     CSPPasswordOffset;  /* Offset of password */
  MQLONG     CSPPasswordLength;  /* Length of password */
  };
```

COBOL declaration for MQCSP:

```
**   MQCSP structure
  10 MQCSP.
**    Structure identifier
   15 MQCSP-STRUCID           PIC X(4).
**    Structure version number
   15 MQCSP-VERSION           PIC S9(9) BINARY.
**    Type of authentication
   15 MQCSP-AUTHENTICATIONTYPE PIC S9(9) BINARY.
**    Required for IBM i pointer alignment
   15 MQCSP-RESERVED1         PIC X(4).
**    Address of user ID
   15 MQCSP-CSPUSERIDPTR      POINTER.
**    Offset of user ID
   15 MQCSP-CSPUSERIDOFFSET   PIC S9(9) BINARY.
**    Length of user ID
   15 MQCSP-CSPUSERIDLENGTH   PIC S9(9) BINARY.
**    Required for IBM i pointer alignment
   15 MQCSP-RESERVED2         PIC X(4).
**    Address of password
   15 MQCSP-CSPPASSWORDPTR    POINTER.
**    Offset of password
   15 MQCSP-CSPPASSWORDOFFSET PIC S9(9) BINARY.
**    Length of password
   15 MQCSP-CSPPASSWORDLENGTH PIC S9(9) BINARY.
```

PL/I declaration for MQCSP:

```
dcl
 1 MQCSP based,
  3 StrucId           char(4),       /* Structure identifier */
  3 Version           fixed bin(31), /* Structure version number */
  3 AuthenticationType fixed bin(31),  /* Type of authentication */
  3 Reserved1         char(4),     /* Required for IBM i pointer
                                      alignment */
  3 CSPUserIdPtr      pointer,      /* Address of user ID */
  3 CSPUserIdOffset   fixed bin(31), /* Offset of user ID */
  3 CSPUserIdLength   fixed bin(31), /* Length of user ID */
  3 Reserved2         char(8),     /* Required for IBM i pointer
                                      alignment */
  3 CSPPasswordPtr    pointer,      /* Address of password */
  3 CSPPasswordOffset fixed bin(31), /* Offset of user ID */
  3 CSPPasswordLength fixed bin(31); /* Length of user ID */
```

Visual Basic declaration for MQCSP:

```
Type MQCSP
  StrucId           As String*4   'Structure identifier'
  Version           As Long       'Structure version number'
  AuthenticationType As Long      'Type of authentication'
  Reserved1         As MQBYTE4    'Required for IBM i pointer'
                                  'alignment'
  CSPUserIdPtr      As MQPTR      'Address of user ID'
  CSPUserIdOffset   As Long       'Offset of user ID'
  CSPUserIdLength   As Long       'Length of user ID'
  Reserved2         As MQBYTE8    'Required for IBM i pointer'
                                  'alignment'
  CSPPasswordPtr    As MQPTR      'Address of password'
  CSPPasswordOffset As Long       'Offset of password'
  CSPPasswordLength As Long       'Length of password'
End Type
```

MQCTLO - Control callback options structure:

The following table summarizes the fields in the structure. Structure specifying the control callback function.

Table 189. Fields in MQCTLO

Field	Description	Topic
StrucID	Structure identifier	StrucID
Version	Structure version number	Version
Options	Options	Options
Reserved	Reserved field	Options
ConnectionArea	Field for callback function to use	ConnectionArea

Overview for MQCTLO:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, z/OS, and IBM MQ MQI clients connected to these systems. Overview of the MQCTLO structure.

Purpose: The MQCTLO structure is used to specify options relating to a control callbacks function.

The structure is an input and output parameter on the MQCTL call.

Version: The current version of MQCTLO is MQCTLO_VERSION_1.

Character set and encoding: Data in MQCTLO must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Fields for MQCTLO:

Alphabetic list of fields for the MQCTLO structure.

The MQCTLO structure contains the following fields; the fields are described in alphabetical order:

ConnectionArea (MQPTR):

Control options structure - ConnectionArea field

This is a field that is available for the callback function to use.

The queue manager makes no decisions based on the contents of this field and it is passed unchanged to the ConnectionArea field in the MQCBC structure, which is an input parameter to the callback.

This field is ignored for all operations other than MQOP_START and MQOP_START_WAIT.

This is an input and output field to the callback function. The initial value of this field is a null pointer or null bytes.

Options (MQLONG):

Control options structure - Options field

Options that control the action of MQCTL.

MQCTLO_FAIL_IF_QUIESCING
Force the MQCTL call to fail if the queue manager or connection is in the quiescing state.

Specify MQGMO_FAIL_IF_QUIESCING, in the MQGMO options passed on the MQCB call, to cause notification to message consumers when they are quiescing.

MQCTLO_THREAD_AFFINITY
This option informs the system that the application requires that all message consumers, for the same connection, are called on the same thread. This thread will be used for all invocations of the consumers until the connection is stopped.

Default option: If you do not need any of the options described, use the following option:

MQCTLO_NONE
Use this value to indicate that no other options have been specified; all options assume their default values. MQCTLO_NONE is defined to aid program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

This is an input field. The initial value of the *Options* field is MQCTLO_NONE.

Reserved (MQLONG):

This is a reserved field. The value must be zero.

StrucId (MQCHAR4):

Control options structure - StrucId field

This is the structure identifier; the value must be:

MQCTLO_STRUC_ID
Identifier for Control Options structure.

For the C programming language, the constant MQCTLO_STRUC_ID_ARRAY is also defined; this has the same value as MQCTLO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQCTLO_STRUC_ID.

Version (MQLONG):

Control options structure - Version field

This is the structure version number; the value must be:

MQCTLO_VERSION_1
Version-1 Control options structure.

The following constant specifies the version number of the current version:

MQCTLO_CURRENT_VERSION
Current version of Control options structure.

This is always an input field. The initial value of this field is MQCTLO_VERSION_1.

Initial values and language declarations for MQCTLO:

Control options structure - Initial values

Table 190. Initial values of fields in MQCTLO

Field name	Name of constant	Value of constant
StrucId	MQCTLO_STRUC_ID	'CTLO'
Version	MQCTLO_VERSION_1	1
Options	MQCTLO_NONE	Nulls
Reserved	Reserved field	
ConnectionArea	None	Null pointer or null bytes

Notes:

1. In the C programming language, the macro variable MQCTLO_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure:

```
MQCTLO MyCTLO = {MQCTLO_DEFAULT};
```

C declaration for MQCTLO:

Control Options structure - C language declaration

```
typedef struct tagMQCTLO MQCTLO;
 struct tagMQCTLO {
  MQCHAR4   StrucId;        /* Structure identifier */
  MQLONG    Version;        /* Structure version number */
  MQLONG    Options;        /* Options that control the action of MQCTL */
  MQLONG    Reserved;       /* Reserved field */

MQPTR     ConnectionArea; /* Connection work area passed to the function */
 };
```

COBOL declaration for MQCTLO:

```
** MQCTLO structure
 10  MQCTLO.
** Structure Identifier
 15  MQCTLO-STRUCID                      PIC X(4).
** Structure Version
 15  MQCTLO-VERSION                      PIC S9(9) BINARY.
 ** Options
 15  MQCTLO-OPTIONS                      PIC S9(9) BINARY.
** Reserved
 15  MQCTLO-RESERVED                     PIC S9(9) BINARY.
** ConnectionArea
 15  MQCTLO-CONNECTIONAREA               POINTER
```

PL/I declaration for MQCTLO:

```
dcl
 1 MQCTLO based,
  3 StrucId             char(4),       /* Structure identifier */
  3 Version             fixed bin(31), /* Structure version */
  3 Options             fixed bin(31), /* Options */
  3 Reserved            fixed bin(31),
  3 ConnectionArea      pointer;       /* Connection work area */
```

MQDH - Distribution header:

The following table summarizes the fields in the structure.

Table 191. Fields in MQDH

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
StrucLength	Length of MQDH structure plus following records	StrucLength
Encoding	Numeric encoding of data that follows array of MQPMR records	Encoding
CodedCharSetId	Character set identifier of data that follows array of MQPMR records	CodedCharSetId
Format	Format name of data that follows array of MQPMR records	Format
Flags	General flags	Flags
PutMsgRecFields	Flags indicating which MQPMR fields are present	PutMsgRecFields
RecsPresent	Number of object records present	RecsPresent
ObjectRecOffset	Offset of first object record from start of MQDH	ObjectRecOffset

Table 191. Fields in MQDH (continued)

Field	Description	Topic
PutMsgRecOffset	Offset of first put-message record from start of MQDH	PutMsgRecOffset

Overview for MQDH:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ clients connected to these systems.

Purpose: The MQDH structure describes the additional data that is present in a message when that message is a distribution-list message stored on a transmission queue. A distribution-list message is a message that is sent to multiple destination queues. The additional data consists of the MQDH structure followed by an array of MQOR records and an array of MQPMR records.

This structure is used by specialized applications that put messages directly on transmission queues, or that remove messages from transmission queues (for example: message channel agents).

Applications that want to put messages to distribution lists must not use this structure. Instead, they must use the MQOD structure to define the destinations in the distribution list, and the MQPMO structure to specify message properties or receive information about the messages sent to the individual destinations.

Format name: MQFMT_DIST_HEADER.

Character set and encoding: Data in MQDH must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by MQENC_NATIVE.

Set the character set and encoding of the MQDH into the *CodedCharSetId* and *Encoding* fields in:

- The MQMD (if the MQDH structure is at the start of the message data), or
- The header structure that precedes the MQDH structure (all other cases).

Usage: When an application puts a message to a distribution list, and some or all of the destinations are remote, the queue manager prefixes the application message data with the MQXQH and MQDH structures, and places the message on the relevant transmission queue. The data therefore occurs in the following sequence when the message is on a transmission queue:

- MQXQH structure
- MQDH structure plus arrays of MQOR and MQPMR records
- Application message data

Depending on the destinations, the queue manager can generate more than one such message, and place it on different transmission queues. In this case, the MQDH structures in those messages identify different subsets of the destinations defined by the distribution list opened by the application.

An application that puts a distribution-list message directly on a transmission queue must conform to the sequence described earlier, and must ensure that the MQDH structure is correct. If the MQDH structure is not valid, the queue manager can fail the MQPUT or MQPUT1 call with reason code MQRC_DH_ERROR.

You can store messages on a queue in distribution-list form only if you have defined the queue as being able to support distribution list messages. See the **DistLists** queue attribute described in "Attributes for queues" on page 2833. If an application puts a distribution-list message directly on a queue that does not support distribution lists, the queue manager splits the distribution list message into individual messages,

and places those on the queue instead.

Fields for MQDH:

The MQDH structure contains the following fields; the fields are described in **alphabetical order**:

CodedCharSetId (MQLONG):

This is the character set identifier of the data that follows the arrays of MQOR and MQPMR records; it does not apply to character data in the MQDH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. You can use the following special value:

MQCCSI_INHERIT
Inherit character-set identifier of this structure.

Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the MQGET call does not return the value MQCCSI_INHERIT.

You cannot use MQCCSI_INHERIT if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG):

This is the numeric encoding of the data that follows the arrays of MQOR and MQPMR records; it does not apply to numeric data in the MQDH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is 0.

Flags (MQLONG):

You can specify the following flag:

MQDHF_NEW_MSG_IDS
Generate a new message identifier for each destination in the distribution list. Set this only when there are no put-message records present, or when the records are present but they do not contain the *MsgId* field.

Using this flag defers generation of the message identifiers until the moment when the distribution-list message is finally split into individual messages. This minimizes the amount of control information that must flow with the distribution-list message.

When an application puts a message to a distribution list, the queue manager sets MQDHF_NEW_MSG_IDS in the MQDH that it generates when both of the following statements are true:

- There are no put-message records provided by the application, or the records provided do not contain the *MsgId* field.
- The *MsgId* field in MQMD is MQMI_NONE, or the *Options* field in MQPMO includes MQPMO_NEW_MSG_ID

If no flags are needed, specify the following:

MQDHF_NONE
No flags have been specified. MQDHF_NONE is defined to aid program documentation. It is not intended that this constant be used with any other, but as its value is zero, such use cannot be detected.

The initial value of this field is MQDHF_NONE.

Format (MQCHAR8):

This is the format name of the data that follows the arrays of MQOD and MQPMR records (whichever occurs last).

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *Format* field in MQMD.

The initial value of this field is MQFMT_NONE.

ObjectRecOffset (MQLONG):

This gives the offset in bytes of the first record in the array of MQOR object records containing the names of the destination queues. There are *RecsPresent* records in this array. These records (plus any bytes skipped between the first object record and the previous field) are included in the length given by the *StrucLength* field.

A distribution list must always contain at least one destination, so *ObjectRecOffset* must always be greater than zero.

The initial value of this field is 0.

PutMsgRecFields (MQLONG):

You can specify none or more of the following flags:

MQPMRF_MSG_ID
Message-identifier field is present.

MQPMRF_CORREL_ID
Correlation-identifier field is present.

MQPMRF_GROUP_ID
Group-identifier field is present.

MQPMRF_FEEDBACK
Feedback field is present.

MQPMRF_ACCOUNTING_TOKEN
Accounting-token field is present.

If no MQPMR fields are present, specify the following:

MQPMRF_NONE
No put-message record fields are present. MQPMRF_NONE is defined to aid program documentation. It is not intended that this constant be used with any other, but as its value is zero, such use cannot be detected.

The initial value of this field is MQPMRF_NONE.

PutMsgRecOffset (MQLONG):

This gives the offset in bytes of the first record in the array of MQPMR put message records containing the message properties. If present, there are *RecsPresent* records in this array. These records (plus any bytes skipped between the first put message record and the previous field) are included in the length given by the *StrucLength* field.

Put message records are optional; if no records are provided, *PutMsgRecOffset* is zero, and *PutMsgRecFields* has the value MQPMRF_NONE.

The initial value of this field is 0.

RecsPresent (MQLONG):

This is the number of destinations. A distribution list must always contain at least one destination, so *RecsPresent* must always be greater than zero.

The initial value of this field is 0.

StrucId (MQCHAR4):

The value must be:

MQDH_STRUC_ID
Identifier for distribution header structure.

For the C programming language, the constant MQDH_STRUC_ID_ARRAY is also defined; this has the same value as MQDH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQDH_STRUC_ID.

StrucLength (MQLONG):

This is the number of bytes from the start of the MQDH structure to the start of the message data following the arrays of MQOR and MQPMR records. The data occurs in the following sequence:

- MQDH structure
- Array of MQOR records
- Array of MQPMR records
- Message data

The arrays of MQOR and MQPMR records are addressed by offsets contained within the MQDH structure. If these offsets result in unused bytes between one or more of the MQDH structure, the arrays of records, and the message data, those unused bytes must be included in the value of *StrucLength*, but the content of those bytes is not preserved by the queue manager. It is valid for the array of MQPMR records to precede the array of MQOR records.

The initial value of this field is 0.

Version (MQLONG):

The value must be:

MQDH_VERSION_1
Version number for distribution header structure.

The following constant specifies the version number of the current version:

MQDH_CURRENT_VERSION
Current version of distribution header structure.

The initial value of this field is MQDH_VERSION_1.

Initial values and language declarations for MQDH:

Table 192. Initial values of fields in MQDH for MQDH

Field name	Name of constant	Value of constant
StrucId	MQDH_STRUC_ID	'DH¬¬'
Version	MQDH_VERSION_1	1
StrucLength	None	0
Encoding	None	0
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
Flags	MQDHF_NONE	0
PutMsgRecFields	MQPMRF_NONE	0
RecsPresent	None	0
ObjectRecOffset	None	0
PutMsgRecOffset	None	0
Notes: 1. The symbol ¬ represents a single blank character. 2. In the C programming language, the macro variable MQDH_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure: `MQDH MyDH = {MQDH_DEFAULT};`		

C declaration for MQDH:

```
typedef struct tagMQDH MQDH;
struct tagMQDH {
  MQCHAR4  StrucId;          /* Structure identifier */
  MQLONG   Version;          /* Structure version number */
  MQLONG   StrucLength;      /* Length of MQDH structure plus following
                                MQOR and MQPMR records */
  MQLONG   Encoding;         /* Numeric encoding of data that follows
                                the MQOR and MQPMR records */
  MQLONG   CodedCharSetId;   /* Character set identifier of data that
                                follows the MQOR and MQPMR records */
  MQCHAR8  Format;           /* Format name of data that follows the
                                MQOR and MQPMR records */
  MQLONG   Flags;            /* General flags */
  MQLONG   PutMsgRecFields;  /* Flags indicating which MQPMR fields are
                                present */
  MQLONG   RecsPresent;      /* Number of MQOR records present */
  MQLONG   ObjectRecOffset;  /* Offset of first MQOR record from start
                                of MQDH */
  MQLONG   PutMsgRecOffset;  /* Offset of first MQPMR record from start
                                of MQDH */
};
```

COBOL declaration for MQDH:

```
**   MQDH structure
  10 MQDH.
**    Structure identifier
   15 MQDH-STRUCID         PIC X(4).
**    Structure version number
   15 MQDH-VERSION         PIC S9(9) BINARY.
**    Length of MQDH structure plus following MQOR and MQPMR records
   15 MQDH-STRUCLENGTH     PIC S9(9) BINARY.
**    Numeric encoding of data that follows the MQOR and MQPMR records
   15 MQDH-ENCODING        PIC S9(9) BINARY.
**    Character set identifier of data that follows the MQOR and MQPMR
**    records
   15 MQDH-CODEDCHARSETID  PIC S9(9) BINARY.
**    Format name of data that follows the MQOR and MQPMR records
   15 MQDH-FORMAT          PIC X(8).
**    General flags
   15 MQDH-FLAGS           PIC S9(9) BINARY.
**    Flags indicating which MQPMR fields are present
   15 MQDH-PUTMSGRECFIELDS PIC S9(9) BINARY.
**    Number of MQOR records present
   15 MQDH-RECSPRESENT     PIC S9(9) BINARY.
**    Offset of first MQOR record from start of MQDH
   15 MQDH-OBJECTRECOFFSET PIC S9(9) BINARY.
**    Offset of first MQPMR record from start of MQDH
   15 MQDH-PUTMSGRECOFFSET PIC S9(9) BINARY.
```

PL/I declaration for MQDH:

```
dcl
 1 MQDH based,
  3 StrucId         char(4),       /* Structure identifier */
  3 Version         fixed bin(31), /* Structure version number */
  3 StrucLength     fixed bin(31), /* Length of MQDH structure plus
                                      following MQOR and MQPMR
                                      records */
  3 Encoding        fixed bin(31), /* Numeric encoding of data that
                                      follows the MQOR and MQPMR
                                      records */
  3 CodedCharSetId  fixed bin(31), /* Character set identifier of data
                                      that follows the MQOR and MQPMR
                                      records */
  3 Format          char(8),       /* Format name of data that follows
                                      the MQOR and MQPMR records */
  3 Flags           fixed bin(31), /* General flags */
  3 PutMsgRecFields fixed bin(31), /* Flags indicating which MQPMR
                                      fields are present */
  3 RecsPresent     fixed bin(31), /* Number of MQOR records present */
  3 ObjectRecOffset fixed bin(31), /* Offset of first MQOR record from
                                      start of MQDH */
  3 PutMsgRecOffset fixed bin(31); /* Offset of first MQPMR record from
                                      start of MQDH */
```

Visual Basic declaration for MQDH:

```
Type MQDH
  StrucId         As String*4 'Structure identifier'
  Version         As Long     'Structure version number'
  StrucLength     As Long     'Length of MQDH structure plus following'
                              'MQOR and MQPMR records'
  Encoding        As Long     'Numeric encoding of data that follows'
                              'the MQOR and MQPMR records'
  CodedCharSetId  As Long     'Character set identifier of data that'
                              'follows the MQOR and MQPMR records'
  Format          As String*8 'Format name of data that follows the'
                              'MQOR and MQPMR records'
  Flags           As Long     'General flags'
  PutMsgRecFields As Long     'Flags indicating which MQPMR fields are'
                              'present'
  RecsPresent     As Long     'Number of MQOR records present'
  ObjectRecOffset As Long     'Offset of first MQOR record from start'
                              'of MQDH'
  PutMsgRecOffset As Long     'Offset of first MQPMR record from start'
                              'of MQDH'
End Type
```

MQDLH - Dead-letter header:

The following table summarizes the fields in the structure.

Table 193. Fields in MQDLH

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Reason	Reason message arrived on dead-letter queue	Reason
DestQName	Name of original destination queue	DestQName
DestQMgrName	Name of original destination queue manager	DestQMgrName
Encoding	Numeric encoding of data that follows MQDLH	Encoding
CodedCharSetId	Character set identifier of data that follows MQDLH	CodedCharSetId
Format	Format name of data that follows MQDLH	Format
PutApplType	Type of application that put message on dead-letter queue	PutApplType
PutApplName	Name of application that put message on dead-letter queue	PutApplName
PutDate	Date when message was put on dead-letter queue	PutDate
PutTime	Time when message was put on dead-letter queue	PutTime

Overview for MQDLH:

Availability: All IBM MQ platforms.

Purpose: The MQDLH structure describes the information that prefixes the application message data of messages on the dead-letter (undelivered-message) queue. A message can arrive on the dead-letter queue either because the queue manager or message channel agent has redirected it to the queue, or because an application has put the message directly on the queue.

Format name: MQFMT_DEAD_LETTER_HEADER.

Character set and encoding: The fields in the MQDLH structure are in the character set and encoding given by the *CodedCharSetId* and *Encoding* fields. These are specified in the header structure that precedes the MQDLH, or in the MQMD structure if the MQDLH is at the start of the application message data.

The character set must be one that has single-byte characters for the characters that are valid in queue names.

If you are using the WMQ classes for Java/JMS, and the code page defined in the MQMD is not supported by the Java virtual machine, then the MQDLH is written in the UTF-8 character set.

Usage: Applications that put messages directly on the dead-letter queue must prefix the message data with an MQDLH structure, and initialize the fields with appropriate values. However, the queue manager does not require that an MQDLH structure be present, or that valid values have been specified for the fields.

If a message is too long to put on the dead-letter queue, the application must do one of the following:

- Truncate the message data to fit on the dead-letter queue.

- Record the message on auxiliary storage and place an exception report message on the dead-letter queue indicating this.
- Discard the message and return an error to its originator. If the message is (or might be) a critical message, do this only if it is known that the originator still has a copy of the message; for example, a message received by a message channel agent from a communication channel.

Which of the preceding actions is appropriate (if any) depends on the design of the application.

The queue manager performs special processing when a message that is a segment is put with an MQDLH structure at the front; see the description of the MQMDE structure for further details.

Putting messages on the dead-letter queue: When a message is put on the dead-letter queue, the MQMD structure used for the MQPUT or MQPUT1 call must be identical to the MQMD associated with the message (usually the MQMD returned by the MQGET call), with the exception of the following:

- Set the *CodedCharSetId* and *Encoding* fields to whatever character set and encoding are used for fields in the MQDLH structure.
- Set the *Format* field to MQFMT_DEAD_LETTER_HEADER to indicate that the data begins with a MQDLH structure.
- Set the context fields (*AccountingToken*, *ApplIdentityData*, *ApplOriginData*, *PutApplName*, *PutApplType*, *PutDate*, *PutTime*, *UserIdentifier*) by using a context option appropriate to the circumstances:
  - An application putting on the dead-letter queue a message that is not related to any preceding message must use the MQPMO_DEFAULT_CONTEXT option; this causes the queue manager to set all of the context fields in the message descriptor to their default values.
  - A server application putting on the dead-letter queue a messagethat it has just received must use the MQPMO_PASS_ALL_CONTEXT option to preserve the original context information.
  - A server application putting on the dead-letter queue a *reply* to a message that it has just received must use the MQPMO_PASS_IDENTITY_CONTEXT option; this preserves the identity information but sets the origin information to be that of the server application.
  - A message channel agent putting on the dead-letter queue a messagethat it received from its communication channel must use the MQPMO_SET_ALL_CONTEXT option to preserve the original context information.

In the MQDLH structure itself, set the fields as follows:

- Set the *CodedCharSetId*, *Encoding*, and *Format* fields to the values that describe the data that follows the MQDLH structure, usually the values from the original message descriptor.
- Set the context fields *PutApplType*, *PutApplName*, *PutDate*, and *PutTime* to values appropriate to the application that is putting the message on the dead-letter queue; these values are not related to the original message.
- Set other fields as appropriate.

Ensure that all fields have valid values, and that character fields are padded with blanks to the defined length of the field; do not end the character data prematurely by using a null character, because the queue manager does not convert the null and subsequent characters to blanks in the MQDLH structure.

Getting messages from the dead-letter queue: Applications that get messages from the dead-letter queue must verify that the messages begin with an MQDLH structure. The application can determine whether an MQDLH structure is present by examining the *Format* field in the message descriptor MQMD; if the field has the value MQFMT_DEAD_LETTER_HEADER, the message data begins with an MQDLH structure. Be aware also that messages that applications get from the dead-letter queue might be truncated if they were originally too long for the queue.

Fields for MQDLH:

The MQDLH structure contains the following fields; the fields are described in **alphabetical order**:

CodedCharSetId (MQLONG):

CodedCharSetId is the character set identifier of the data that flows through the MQDLH structure (usually the data from the original message); it does not apply to character data in the MQDLH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

MQCCSI_INHERIT
Character data in the data following this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

You cannot use MQCCSI_INHERIT if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Linux, Windows, plus IBM MQ MQI clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

DestQMgrName (MQCHAR48):

DestQMgrName is the name of the queue manager that was the original destination for the message.

The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

DestQName (MQCHAR48):

DestQName is the name of the message queue that was the original destination for the message.

The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

Encoding (MQLONG):

Encoding is the numeric encoding of the data that follows the MQDLH structure (usually the data from the original message); it does not apply to numeric data in the MQDLH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is `0`.

Format (MQCHAR8):

Format is the format name of the data that follows the MQDLH structure (usually the data from the original message).

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those rules for coding the *Format* field in MQMD.

The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

PutApplName (MQCHAR28):

PutApplName is the name of the application that put the message on the dead-letter (undelivered-message) queue.

The format of the name depends on the *PutApplType* field. The format can vary release to release. See the description of the *PutApplName* field in "MQMD - Message descriptor" on page 2387.

If the queue manager redirects the message to the dead-letter queue, *PutApplName* contains the first 28 characters of the queue manager name, padded with blanks if necessary.

The length of this field is given by MQ_PUT_APPL_NAME_LENGTH. The initial value of this field is the null string in C, and 28 blank characters in other programming languages.

PutApplType (MQLONG):

PutApplType is the type of application that put the message on the dead-letter (undelivered-message) queue.

This field has the same meaning as the *PutApplType* field in the message descriptor MQMD (see "MQMD - Message descriptor" on page 2387 for details).

If the queue manager redirects the message to the dead-letter queue, *PutApplType* has the value MQAT_QMGR.

The initial value of this field is `0`.

PutDate (MQCHAR8):

PutDate is the date when the message was put on the dead-letter (undelivered-message) queue.

The format used for the date when this field is generated by the queue manager is:

- YYYYMMDD

where the characters represent:

YYYY year (four numeric digits)

MM month of year (01 through 12)

DD day of month (01 through 31)

Greenwich Mean Time (GMT) is used for the *PutDate* and *PutTime* fields, subject to the system clock being set accurately to GMT.

The length of this field is given by MQ_PUT_DATE_LENGTH. The initial value of this field is the null string in C, and eight blank characters in other programming languages.

PutTime (MQCHAR8):

PutTime is the time when the message was put on the dead-letter (undelivered-message) queue.

The format used for the time when this field is generated by the queue manager is:

- HHMMSSTH

where the characters represent:

HH hours (00 through 23)

MM minutes (00 through 59)

SS seconds (00 through 59; see note)

T tenths of a second (0 through 9)

H hundredths of a second (0 through 9)

Note: If the system clock is synchronized to an very accurate time standard, it is possible on rare occasions for 60 or 61 to be returned for the seconds in *PutTime*. This happens when leap seconds are inserted into the global time standard.

Greenwich Mean Time (GMT) is used for the *PutDate* and *PutTime* fields, subject to the system clock being set accurately to GMT.

The length of this field is given by MQ_PUT_TIME_LENGTH. The initial value of this field is the null string in C, and eight blank characters in other programming languages.

Reason (MQLONG):

The Reason field identifies the reason why the message was placed on the dead-letter queue instead of on the original destination queue.

This identifies the reason why the message was placed on the dead-letter queue instead of on the original destination queue. It should be one of the MQFB_* or MQRC_* values (for example, MQRC_Q_FULL). See the description of the *Feedback* field in "MQMD - Message descriptor" on page 2387 for details of the common MQFB_* values that can occur.

If the value is in the range MQFB_IMS_FIRST through MQFB_IMS_LAST, the actual IMS error code can be determined by subtracting MQFB_IMS_ERROR from the value of the *Reason* field.

Some MQFB_* values occur only in this field. They relate to repository messages, trigger messages, or transmission-queue messages that have been transferred to the dead-letter queue. These are:

MQFB_APPL_CANNOT_BE_STARTED (X'00000109')
An application processing a trigger message cannot start the application named in the *ApplId* field of the trigger message (see "MQTM - Trigger message" on page 2588).

On z/OS, the CKTI CICS transaction is an example of an application that processes trigger messages.

MQFB_APPL_TYPE_ERROR (X'0000010B')
An application processing a trigger message cannot start the application because the *ApplType* field of the trigger message is not valid (see "MQTM - Trigger message" on page 2588).

On z/OS, the CKTI CICS transaction is an example of an application that processes trigger messages.

MQFB_BIND_OPEN_CLUSRCVR_DEL (X'00000119')
The message was on the SYSTEM.CLUSTER.TRANSMIT.QUEUE intended for a cluster queue

that was opened with the MQOO_BIND_ON_OPEN option, but the remote cluster-receiver channel to be used to transmit the message to the destination queue was deleted before the message could be sent. Because MQOO_BIND_ON_OPEN was specified, only the channel selected when the queue was opened can be used to transmit the message. As this channel is no longer available, the message is placed on the dead-letter queue.

MQFB_NOT_A_REPOSITORY_MSG (X'00000118')
The message is not a repository message.

MQFB_STOPPED_BY_CHAD_EXIT (X'00000115')
The message was stopped by channel auto-definition exit.

MQFB_STOPPED_BY_MSG_EXIT (X'0000010D')
The message was stopped by channel message exit.

MQFB_TM_ERROR (X'0000010A')
The *Format* field in MQMD specifies MQFMT_TRIGGER, but the message does not begin with a valid MQTM structure. For example, the *StrucId* mnemonic eye-catcher might not be valid, the *Version* might not be recognized, or the length of the trigger message might be insufficient to contain the MQTM structure.

On z/OS, the CKTI CICS transaction is an example of an application that processes trigger messages and can generate this feedback code.

MQFB_XMIT_Q_MSG_ERROR (X'0000010F')
A message channel agent has found that a message on the transmission queue is not in the correct format. The message channel agent puts the message on the dead-letter queue using this feedback code.

One common cause is that a message has been put directly to the transmission queue, so the message does not have the expected XQH header. Messages should be put to a transmission queue through a remote queue, unless the application builds the MQXQH header.

The initial value of this field is MQRC_NONE.

StrucId (MQCHAR4):

StrucId is the structure identifier.

The value must be:

MQDLH_STRUC_ID
Identifier for dead-letter header structure.

For the C programming language, the constant MQDLH_STRUC_ID_ARRAY is also defined; this has the same value as MQDLH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQDLH_STRUC_ID.

Version (MQLONG):

Version is the structure version number.

The value must be:

MQDLH_VERSION_1
Version number for dead-letter header structure.

The following constant specifies the version number of the current version:

MQDLH_CURRENT_VERSION
Current version of dead-letter header structure.

The initial value of this field is MQDLH_VERSION_1.

Initial values and language declarations for MQDLH:

Table 194. Initial values of fields in MQDLH for MQDLH

Field name	Name of constant	Value of constant
StrucId	MQDLH_STRUC_ID	'DLH¬'
Version	MQDLH_VERSION_1	1
Reason	MQRC_NONE	0
DestQName	None	Null string or blanks
DestQMgrName	None	Null string or blanks
Encoding	None	0
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
PutApplType	None	0
PutApplName	None	Null string or blanks
PutDate	None	Null string or blanks
PutTime	None	Null string or blanks

Notes:

1. The symbol ¬ represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQDLH_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure:

```
MQDLH MyDLH = {MQDLH_DEFAULT};
```

C declaration for MQDLH:

```
typedef struct tagMQDLH MQDLH;
struct tagMQDLH {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    Reason;          /* Reason message arrived on dead-letter
                                (undelivered-message) queue */
  MQCHAR48  DestQName;       /* Name of original destination queue */
  MQCHAR48  DestQMgrName;    /* Name of original destination queue
                                manager */
  MQLONG    Encoding;        /* Numeric encoding of data that follows
                                MQDLH */
  MQLONG    CodedCharSetId;  /* Character set identifier of data that
                                follows MQDLH */
  MQCHAR8   Format;          /* Format name of data that follows
                                MQDLH */
  MQLONG    PutApplType;     /* Type of application that put message on
                                dead-letter (undelivered-message)
                                queue */
  MQCHAR28  PutApplName;     /* Name of application that put message on
                                dead-letter (undelivered-message)
                                queue */
  MQCHAR8   PutDate;         /* Date when message was put on dead-letter
                                (undelivered-message) queue */
  MQCHAR8   PutTime;         /* Time when message was put on the
                                dead-letter (undelivered-message)
                                queue */
};
```

COBOL declaration for MQDLH:

```
**   MQDLH structure
  10 MQDLH.
**    Structure identifier
   15 MQDLH-STRUCID        PIC X(4).
**    Structure version number
   15 MQDLH-VERSION        PIC S9(9) BINARY.
**    Reason message arrived on dead-letter (undelivered-message) queue
   15 MQDLH-REASON         PIC S9(9) BINARY.
**    Name of original destination queue
   15 MQDLH-DESTQNAME      PIC X(48).
**    Name of original destination queue manager
   15 MQDLH-DESTQMGRNAME   PIC X(48).
**    Numeric encoding of data that follows MQDLH
   15 MQDLH-ENCODING       PIC S9(9) BINARY.
**    Character set identifier of data that follows MQDLH
   15 MQDLH-CODEDCHARSETID PIC S9(9) BINARY.
**    Format name of data that follows MQDLH
   15 MQDLH-FORMAT         PIC X(8).
**    Type of application that put message on dead-letter
**    (undelivered-message) queue
   15 MQDLH-PUTAPPLTYPE    PIC S9(9) BINARY.
**    Name of application that put message on dead-letter
**    (undelivered-message) queue
   15 MQDLH-PUTAPPLNAME    PIC X(28).
**    Date when message was put on dead-letter (undelivered-message)
**    queue
   15 MQDLH-PUTDATE        PIC X(8).
**    Time when message was put on the dead-letter (undelivered-message)
**    queue
   15 MQDLH-PUTTIME        PIC X(8).
```

PL/I declaration for MQDLH:

```
dcl
 1 MQDLH based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 Reason         fixed bin(31), /* Reason message arrived on
                                     dead-letter (undelivered-message)
                                     queue */
  3 DestQName      char(48),      /* Name of original destination
                                     queue */
  3 DestQMgrName   char(48),      /* Name of original destination queue
                                     manager */
  3 Encoding       fixed bin(31), /* Numeric encoding of data that
                                     follows MQDLH */
  3 CodedCharSetId fixed bin(31), /* Character set identifier of data
                                     that follows MQDLH */
  3 Format         char(8),       /* Format name of data that follows
                                     MQDLH */
  3 PutApplType    fixed bin(31), /* Type of application that put
                                     message on dead-letter
                                     (undelivered-message) queue */
  3 PutApplName    char(28),      /* Name of application that put
                                     message on dead-letter
                                     (undelivered-message) queue */
  3 PutDate        char(8),       /* Date when message was put on
                                     dead-letter (undelivered-message)
                                     queue */
  3 PutTime        char(8);       /* Time when message was put on the
                                     dead-letter (undelivered-message)
                                     queue */
```

High Level Assembler declaration for MQDLH:

```
MQDLH                 DSECT
MQDLH_STRUCID         DS   CL4   Structure identifier
MQDLH_VERSION         DS   F     Structure version number
MQDLH_REASON          DS   F     Reason message arrived on dead-letter
*                                (undelivered-message) queue
MQDLH_DESTQNAME       DS   CL48  Name of original destination queue
MQDLH_DESTQMGRNAME    DS   CL48  Name of original destination queue
*                                manager
MQDLH_ENCODING        DS   F     Numeric encoding of data that follows
*                                MQDLH
MQDLH_CODEDCHARSETID  DS   F     Character set identifier of data that
*                                follows MQDLH
MQDLH_FORMAT          DS   CL8   Format name of data that follows MQDLH
MQDLH_PUTAPPLTYPE     DS   F     Type of application that put message on
*                                dead-letter (undelivered-message) queue
MQDLH_PUTAPPLNAME     DS   CL28  Name of application that put message on
*                                dead-letter (undelivered-message) queue
MQDLH_PUTDATE         DS   CL8   Date when message was put on
*                                dead-letter (undelivered-message) queue
MQDLH_PUTTIME         DS   CL8   Time when message was put on the
*                                dead-letter (undelivered-message) queue
*
MQDLH_LENGTH          EQU  *-MQDLH
                      ORG  MQDLH
MQDLH_AREA            DS   CL(MQDLH_LENGTH)
```

Visual Basic declaration for MQDLH:

```
Type MQDLH
  StrucId        As String*4  'Structure identifier'
  Version        As Long      'Structure version number'
  Reason         As Long      'Reason message arrived on dead-letter'
                              '(undelivered-message) queue'
  DestQName      As String*48 'Name of original destination queue'
  DestQMgrName   As String*48 'Name of original destination queue'
                              'manager'
  Encoding       As Long      'Numeric encoding of data that follows'
                              'MQDLH'
  CodedCharSetId As Long      'Character set identifier of data that'
                              'follows MQDLH'
  Format         As String*8  'Format name of data that follows MQDLH'
  PutApplType    As Long      'Type of application that put message on'
                              'dead-letter (undelivered-message) queue'
  PutApplName    As String*28 'Name of application that put message on'
                              'dead-letter (undelivered-message) queue'
  PutDate        As String*8  'Date when message was put on dead-letter'
                              '(undelivered-message) queue'
  PutTime        As String*8  'Time when message was put on the'
                              'dead-letter (undelivered-message) queue'
End Type
```

MQDMHO - Delete message handle options:

The following table summarizes the fields in the structure.

Table 195. Fields in MQDMHO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options	Options

Overview for MQDMHO:

Availability: All IBM MQ systems and IBM MQ clients.

Purpose: The **MQDMHO** structure allows applications to specify options that control how message handles are deleted. The structure is an input parameter on the **MQDLTMH** call.

Character set and encoding: Data in **MQDMHO** must be in the character set of the application and encoding of the application (**MQENC_NATIVE**).

Fields for MQDMHO:

The MQDMHO structure contains the following fields; the fields are described in **alphabetical order**:

Options (MQLONG):

The value must be:

MQDMHO_NONE

No options specified.

This is always an input field. The initial value of this field is **MQDMHO_NONE**.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQDMHO_STRUC_ID

Identifier for delete message handle options structure.

For the C programming language, the constant **MQDMHO_STRUC_ID_ARRAY** is also defined; this has the same value as **MQDMHO_STRUC_ID**, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is **MQDMHO_STRUC_ID**.

Version (MQLONG):

This is the structure version number; the value must be:

MQDMHO_VERSION_1
Version-1 delete message handle options structure.

The following constant specifies the version number of the current version:

MQDMHO_CURRENT_VERSION
Current version of delete message handle options structure.

This is always an input field. The initial value of this field is **MQDMHO_VERSION_1**.

Initial values and language declarations for MQDMHO:

Table 196. Initial values of fields in MQDMHO

Field name	Name of constant	Value of constant
StrucId	MQDMHO_STRUC_ID	'DMHO'
Version	MQDMHO_VERSION_1	1
Options	MQDMHO_NONE	0

Notes:

1. In the C programming language, the macro variable MQDMHO_DEFAULT contains the values that are listed in the table. It can be used in the following way to provide initial values for the fields in the structure:

```
MQDMHO MyDMHO = {MQDMHO_DEFAULT};
```

C declaration for MQDMHO:

```
typedef struct tagMQDMHO;
struct tagMQDMHO {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    Options;         /* Options that control the action of MQDLTMH */
};
```

COBOL declaration for MQDMHO:

```
**   MQDMHO structure
  10 MQDMHO.
**    Structure identifier
   15 MQDMHO-STRUCID       PIC X(4).
**    Structure version number
   15 MQDMHO-VERSION       PIC S9(9) BINARY.
**    Options that control the action of MQDLTMH
   15 MQDMHO-OPTIONS       PIC S9(9) BINARY.
```

PL/I declaration for MQDMHO:

```
dcl
 1 MQDMHO based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 Options        fixed bin(31), /* Options that control the action of MQDLTMH */
```

High Level Assembler declaration for MQDMHO:

```
MQDMHO                DSECT
MQDMHO_STRUCID        DS   CL4   Structure identifier
MQDMHO_VERSION        DS   F     Structure version number
MQDMHO_OPTIONS        DS   F     Options that control the action of
*                                MQDLTMH
MQDMHO_LENGTH         EQU  *-MQDMHO
MQDMHO_AREA           DS   CL(MQDMHO_LENGTH)
```

MQDMPO - Delete message property options:

The following table summarizes the fields in the structure. MQDMPO structure - delete message property options

Table 197. Fields in MQDMPO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options controlling the action of MQDMPO	Options

Overview for MQDMPO:

Availability: All IBM MQ systems and IBM MQ clients.

Purpose: The MQDMPO structure allows applications to specify options that control how properties of messages are deleted. The structure is an input parameter on the MQDLTMP call.

Character set and encoding: Data in MQDMPO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields for MQDMPO:

Delete message property options structure - fields

The MQDMPO structure contains the following fields; the fields are described in **alphabetical order**:

Options (MQLONG):

Delete message property options structure - Options field

Location options: The following options relate to the relative location of the property compared to the property cursor.

MQDMPO_DEL_FIRST
Deletes the first property that matches the specified name.

MQDMPO_DEL_PROP_UNDER_CURSOR
Deletes the property pointed to by the property cursor; that is the property that was last inquired by using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused. It is also reset when the message handle is specified in the *MsgHandle* field of the MQGMO structure on an MQGET call, or MQPMO structure on an MQPUT call.

If this option is used when the property cursor has not yet been established, the call fails with completion code MQCC_FAILED and reason MQRC_PROPERTY_NOT_AVAILABLE. If the property pointed to by the property cursor has already been deleted, the call also fails with completion code MQCC_FAILED and reason MQRC_PROPERTY_NOT_AVAILABLE.

If neither of thees options is required, the following option can be used:

MQDMPO_NONE
No options specified.

This field is always an input field. The initial value of this field is MQDMPO_DEL_FIRST.

StrucId (MQCHAR4):

Delete message property options structure - StrucId field

This is the structure identifier. The value must be:

MQDMPO_STRUC_ID
Identifier for delete message property options structure.

For the C programming language, the constant MQDMPO_STRUC_ID_ARRAY is also defined; this has the same value as MQDMPO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQDMPO_STRUC_ID.

Version (MQLONG):

Delete message property options structure - Version field

This is the structure version number. The value must be:

MQDMPO_VERSION_1
Version number for delete message property options structure.

The following constant specifies the version number of the current version:

MQDMPO_CURRENT_VERSION
Current version of delete message property options structure.

This is always an input field. The initial value of this field is MQDMPO_VERSION_1.

Initial values and language declarations for MQDMPO:

Delete message property options structure - Initial values

Table 198. Initial values of fields in MQDPMO

Field name	Name of constant	Value of constant
StrucId	MQDMPO_STRUC_ID	'DMPO'
Version	MQDMPO_VERSION_1	1
Options	Options that control the action of MQDLTMP	MQDMPO_NONE

Notes:

1. In the C programming language, the macro variable MQDMPO_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure:

```
MQDMPO MyDMPO = {MQDMPO_DEFAULT};
```

C declaration for MQDMPO:

Delete message property options structure - C language declaration

```
typedef struct tagMQDMPO MQDMPO;
struct tagMQDMPO {
  MQCHAR4  StrucId;        /* Structure identifier */
  MQLONG   Version;        /* Structure version number */
  MQLONG   Options;        /* Options that control the action of
                              MQDLTMP */
};
```

COBOL declaration for MQDMPO:

Delete message property options structure - COBOL language declaration

```
**   MQDMPO structure
  10 MQDMPO.
**    Structure identifier
   15 MQDMPO-STRUCID               PIC X(4).
**    Structure version number
   15 MQDMPO-VERSION               PIC S9(9) BINARY.
**    Options that control the action of MQDLTMP
   15 MQDMPO-OPTIONS               PIC S9(9) BINARY.
```

PL/I declaration for MQDMPO:

Delete message property options structure - PL/I language declaration

```
Dcl
 1 MQDMPO based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 Options        fixed bin(31), /* Options that control the action
                                     of MQDLTMP */
```

High Level Assembler declaration for MQDMPO:

Delete message property options structure - Assembler language declaration

```
MQDMPO                  DSECT
MQDMPO_STRUCID          DS   CL4  Structure identifier
MQDMPO_VERSION          DS   F    Structure version number
MQDMPO_OPTIONS          DS   F    Options that control the
*                                 action of MQDLTMP
MQDMPO_LENGTH           EQU  *-MQDMPO
MQDMPO_AREA             DS   CL(MQDMPO_LENGTH)
```

MQEPH - Embedded PCF header:

The following table summarizes the fields in the structure.

Table 199. Fields in MQEPH

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
StrucLength	Length of MQEPH structure plus the MQCFH and parameter structures that follow it	StrucLength
Encoding	Numeric encoding of data that follows last PCF parameter structure	Encoding
CodedCharSetId	Character set identifier of data that follows last PCF parameter structure	CodedCharSetId
Format	Format name of data that follows last PCF parameter structure	Format
Flags	Flags	Flags
PCFHeader	Programmable command format (PCF) header	PCFHeader

Overview for MQEPH:

Availability: All IBM MQ platforms.

Purpose: The MQEPH structure describes the additional data that is present in a message when that message is a programmable command format (PCF) message. The *PCFHeader* field defines the PCF parameters that follow this structure and this allows you to follow the PCF message data with other headers.

Format name: MQFMT_EMBEDDED_PCF

Character set and encoding: Data in MQEPH must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by MQENC_NATIVE.

Set the character set and encoding of the MQEPH into the *CodedCharSetId* and *Encoding* fields in:

- The MQMD (if the MQEPH structure is at the start of the message data), or
- The header structure that precedes the MQEPH structure (all other cases).

Usage: You cannot use MQEPH structures to send commands to the command server or any other queue manager PCF-accepting server.

Similarly, the command server or any other queue manager PCF-accepting server do not generate responses or events containing MQEPH structures.

Fields for MQEPH:

The MQEPH structure contains the following fields; the fields are described in **alphabetical order**:

CodedCharSetId (MQLONG):

This is the character set identifier of the data that follows the MQEPH structure and the associated PCF parameters; it does not apply to character data in the MQEPH structure itself.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG):

This is the numeric encoding of the data that follows the MQEPH structure and the associated PCF parameters; it does not apply to character data in the MQEPH structure itself.

The initial value of this field is 0.

Flags (MQLONG):

The following values are available:

MQEPH_NONE
No flags have been specified. MQEPH_NONE is defined to aid program documentation. It is not intended that this constant be used with any other, but as its value is zero, such use cannot be detected.

MQEPH_CCSID_EMBEDDED
The character set of the parameters containing character data is specified individually within the CodedCharSetId field in each structure. The character set of the StrucId and Format fields is defined by the CodedCharSetId field in the header structure that precedes the MQEPH structure, or by the CodedCharSetId field in the MQMD if the MQEPH is at the start of the message.

The initial value of this field is MQEPH_NONE.

Format (MQCHAR8):

This is the format name of the data that follows the MQEPH structure and the associated PCF parameters.

The initial value of this field is MQFMT_NONE.

PCFHeader (MQCFH):
This is the programmable command format (PCF) header, defining the PCF parameters that follow the MQEPH structure. This enables you to follow the PCF message data with other headers.

The PCF header is initially defined with the following values:

Table 200. Initial values of fields in MQCFH

Field name	Name of constant	Value of constant
Type	MQCFT_NONE	0
StrucLength	MQCFH_STRUC_LENGTH	36
Version	MQCFH_VERSION_3	3
StrucLength	None	0
Command	MQCMD_NONE	0

Table 200. Initial values of fields in MQCFH (continued)

Field name	Name of constant	Value of constant
MsgSeqNumber	None	1
Control	MQCFC_LAST	1
CompCode	MQCC_OK	0
Reason	MQRC_NONE	0
ParameterCount	None	0

The application must change the Type from MQCFT_NONE to a valid structure type for the use it is making of the embedded PCF header.

StrucId (MQCHAR4):

The value must be:

MQEPH_STRUC_ID
Identifier for distribution header structure.

For the C programming language, the constant MQEPH_STRUC_ID_ARRAY is also defined; this has the same value as MQDH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQEPH_STRUC_ID.

StrucLength (MQLONG):

This is the amount of data preceding the next header structure. It includes:
- The length of the MQEPH header
- The length of all PCF parameters following the header
- Any blank padding following those parameters

StrucLength must be a multiple of 4.

The fixed length part of the structure is defined by MQEPH_STRUC_LENGTH_FIXED.

The initial value of this field is 68.

Version (MQLONG):

The value must be:

MQEPH_VERSION_1
Version number for embedded PCF header structure.

The following constant specifies the version number of the current version:

MQCFH_VERSION_3
Current version of embedded PCF header structure.

The initial value of this field is MQEPH_VERSION_1.

Initial values and language declarations for MQEPH:

Table 201. Initial values of fields in MQEPH for MQEPH

Field name	Name of constant	Value of constant
StrucId	MQEPH_STRUC_ID	'EPH¬'
Version	MQEPH_VERSION_1	1
StrucLength	MQEPH_STRUC_LENGTH_FIXED	68
Encoding	None	0
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
Flags	MQEPH_NONE	0
PCFHeader	Names and values as defined in Table 200 on page 2325	0

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQEPH_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure:

```
MQEPH MyEPH = {MQEPH_DEFAULT};
```

C declaration:

```
typedef struct tagMQEPH MQEPH;
struct tagMQDH {
  MQCHAR4  StrucId;          /* Structure identifier */
  MQLONG   Version;          /* Structure version number */
  MQLONG   StrucLength;      /* Total length of MQEPH including the MQCFH
                                and parameter structures that follow it */
  MQLONG   Encoding;         /* Numeric encoding of data that follows last
                                PCF parameter structure */
  MQLONG   CodedCharSetId;   /* Character set identifier of data that
                                follows last PCF parameter structure */
  MQCHAR8  Format;           /* Format name of data that follows last PCF
                                parameter structure */
  MQLONG   Flags;            /* Flags */
  MQCFH    PCFHeader;        /* Programmable command format header */
 };
```

COBOL declaration:

```
**   MQEPH structure
  10 MQEPH.
**    Structure identifier
   15 MQEPH-STRUCID         PIC X(4).
**    Structure version number
   15 MQEPH-VERSION         PIC S9(9) BINARY.
**    Total length of MQEPH structure including the MQCFH
**    and parameter structures that follow it
   15 MQEPH-STRUCLENGTH     PIC S9(9) BINARY.
**    Numeric encoding of data that follows last
**    PCF structure
   15 MQEPH-ENCODING        PIC S9(9) BINARY.
**    Character set identifier of data that
**    follows last PCF parameter structure
   15 MQEPH-CODEDCHARSETID  PIC S9(9) BINARY.
**    Format name of data that follows last PCF
**    parameter structure
   15 MQEPH-FORMAT          PIC X(8).
**    Flags
   15 MQEPH-FLAGS           PIC S9(9) BINARY.
**    Programmable command format header
   15 MQEPH-PCFHEADER.
**      Structure type
     20 MQEPH-PCFHEADER-TYPE           PIC S9(9) BINARY.
**      Structure length
     20 MQEPH-PCFHEADER-STRUCLENGTH    PIC S9(9) BINARY.
**      Structure version number
     20 MQEPH-PCFHEADER-VERSION        PIC S9(9) BINARY.
**      Command identifier
     20 MQEPH-PCFHEADER-COMMAND        PIC S9(9) BINARY.
**      Message sequence number
     20 MQEPH-PCFHEADER-MSGSEQNUMBER   PIC S9(9) BINARY.
**      Control options
     20 MQEPH-PCFHEADER-CONTROL        PIC S9(9) BINARY.
**      Completion code
     20 MQEPH-PCFHEADER-COMPCODE       PIC S9(9) BINARY.
**      Reason code qualifying completion code
     20 MQEPH-PCFHEADER-REASON         PIC S9(9) BINARY.
**      Count of parameter structures
     20 MQEPH-PCFHEADER-PARAMETERCOUNT PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQEPH based,
  3 StrucId         char(4),       /* Structure identifier */
  3 Version         fixed bin(31), /* Structure version number */
  3 StrucLength     fixed bin(31), /* Total Length of MQEPH including the
                                      MQCFH and parameter structures that
                                      follow it
  3 Encoding        fixed bin(31), /* Numeric encoding of data that follows
                                      last PCF parameter structure
  3 CodedCharSetId  fixed bin(31), /* Character set identifier of data that
                                      follows last PCF parameter structure
  3 Format          char(8),       /* Format name of data that follows last
                                      PCF parameter structure */
  3 Flags           fixed bin(31), /* Flags */
  3 PCFHeader,                     /* Programmable command format header
   5 Type           fixed bin(31), /* Structure type */
   5 StrucLength    fixed bin(31), /* Structure length */
   5 Version        fixed bin(31), /* Structure version number */
   5 Command        fixed bin(31), /* Command identifier */
   5 MsgseqNumber   fixed bin(31), /* Message sequence number */
   5 Control        fixed bin(31), /* Control options */
```

```
   5 CompCode       fixed bin(31), /* Completion code */
   5 Reason         fixed bin(31), /* Reason code qualifying completion code */
   5 ParameterCount fixed bin(31); /* Count of parameter structures */
```

High Level Assembler declaration for MQEPH:

```
MQEPH                           DSECT
MQEPH_STRUCID                   DS   CL4   Structure identifier
MQEPH_VERSION                   DS   F     Structure version number
MQEPH_STRUCLENGTH               DS   F     Total length of MQEPH including the
*                                          MQCFH and parameter structures that
                                           follow it
MQEPH_ENCODING                  DS   F     Numeric encoding of data that follows
*                                          last PCF parameter structure
MQEPH_CODEDCHARSETID            DS   F     Character set identifier of data that
*                                          follows last PCF parameter structure
MQEPH_FORMAT                    DS   CL8   Format name of data that follows last
*                                          PCF parameter structure
MQEPH_FLAGS                     DS   F     Flags
MQEPH_PCFHEADER                 DS   0F    Force fullword alignment
MQEPH_PCFHEADER_TYPE            DS   F     Structure type
MQEPH_PCFHEADER_STRUCLENGTH     DS   F     Structure length
MQEPH_PCFHEADER_VERSION         DS   F     Structure version number
MQEPH_PCFHEADER_COMMAND         DS   F     Command identifier
MQEPH_PCFHEADER_MSGSEQNUMBER    DS   F     Structure length
MQEPH_PCFHEADER_CONTROL         DS   F     Control options
MQEPH_PCFHEADER_COMPCODE        DS   F     Completion code
MQEPH_PCFHEADER_REASON          DS   F     Reason code qualifying completion code
MQEPH_PCFHEADER_PARAMETER COUNT DS   F     Count of parameter structures
MQEPH_PCFHEADER_LENGTH          EQU  *-MQEPH_PCFHEADER
                                ORG  MQEPH_PCFHEADER
MQEPH_PCFHEADER_AREA            DS   CL(MQEPH_PCFHEADER_LENGTH)
*
MQEPH_LENGTH                    EQU  *-MQEPH
                                ORG  MQEPH
MQEPH_AREA                      DS   CL(MQEPH_LENGTH)
```

Visual Basic declaration for MQEPH:

```
Type MQEPH
  StrucId         As String*4 'Structure identifier'
  Version         As Long     'Structure version number'
  StrucLength     As Long     'Total length of MQEPH structure including the MQCFH'
                              'and parameter structures that follow it'
  Encoding        As Long     'Numeric encoding of data that follows last'
                              'PCF parameter structure'
  CodedCharSetId  As Long     'Character set identifier of data that'
                              'follows last PCF parameter structure'
  Format          As String*8 'Format name of data that follows last PCF'
                              'parameter structure'
  Flags           As Long     'Flags'
  PCFHeader       As MQCFH    'Programmable command format header'
 End Type

Global MQEPH_DEFAULT As MQEPH
```

MQGMO - Get-message options:

The following table summarizes the fields in the structure.

Table 202. Fields in MQGMO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options that control the action of MQGET	MQGMO - Options field
WaitInterval	Wait interval	WaitInterval
Signal1	Signal	Signal1
Signal2	Signal identifier	Signal2
ResolvedQName	Resolved name of destination queue	ResolvedQName
Note: The remaining fields are ignored if *Version* is less than MQGMO_VERSION_2.		
MatchOptions	Options controlling selection criteria used for MQGET	MatchOptions
GroupStatus	Flag indicating whether message retrieved is in a group	GroupStatus
SegmentStatus	Flag indicating whether message retrieved is a segment of a logical message	SegmentStatus
Segmentation	Flag indicating whether further segmentation is allowed for the message retrieved	Segmentation
Reserved1	Reserved	Reserved1
Note: The remaining fields are ignored if *Version* is less than MQGMO_VERSION_3.		
MsgToken	Message token	MsgToken
ReturnedLength	Length of message data returned (bytes)	ReturnedLength
Note: The remaining fields are ignored if *Version* is less than MQGMO_VERSION_4.		
Reserved2	Reserved	Reserved2
MsgHandle	The handle to a message that is to be populated with the properties of the message being retrieved from the queue.	MsgHandle

Overview for MQGMO:

Availability: All IBM MQ platforms.

Purpose: The MQGMO structure allows the application to control how messages are removed from queues. The structure is an input/output parameter on the MQGET call.

Version: The current version of MQGMO is MQGMO_VERSION_4. Certain fields are available only in certain versions of MQGMO. If you need to port applications between several environments, you must ensure that the version of MQGMO is consistent across all environments. Fields that exist only in particular versions of the structure are identified as such in "MQGMO - Get-message options" and in the field descriptions.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQGMO that is supported by the environment, but with the initial value of the *Version* field set to MQGMO_VERSION_1. To use fields that are not present in the version-1 structure, set the *Version* field to the version number of the version required.

Character set and encoding: Data in MQGMO must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Fields for MQGMO:

The MQGMO structure contains the following fields; the fields are described in **alphabetical order**:

GroupStatus (MQCHAR):

This flag indicates whether the message retrieved is in a group.

It has one of the following values:

MQGS_NOT_IN_GROUP
Message is not in a group.

MQGS_MSG_IN_GROUP
Message is in a group, but is not the last in the group.

MQGS_LAST_MSG_IN_GROUP
Message is the last in the group.

This is also the value returned if the group consists of only one message.

This is an output field. The initial value of this field is MQGS_NOT_IN_GROUP. This field is ignored if *Version* is less than MQGMO_VERSION_2.

MatchOptions (MQLONG):

These options allow the application to choose which fields in the **MsgDesc** parameter to use to select the message returned by the MQGET call. The application sets the required options in this field, and then sets the corresponding fields in the **MsgDesc** parameter to the values required for those fields. Only messages that have those values in the MQMD for the message are candidates for retrieval using that **MsgDesc** parameter on the MQGET call. Fields for which the corresponding match option is not specified are ignored when selecting the message to be returned. If you specify no selection criteria on the MQGET call (that is, *any* message is acceptable), set *MatchOptions* to MQMO_NONE.

- On z/OS, the selection criteria that can be used might be restricted by the type of index used for the queue. See the **IndexType** queue attribute for further details.

If you specify MQGMO_LOGICAL_ORDER, only certain messages are eligible for return by the next MQGET call:

- If there is no current group or logical message, only messages that have *MsgSeqNumber* equal to 1 and *Offset* equal to 0 are eligible for return. In this situation, you can use one or more of the following match options to select which of the eligible messages is returned:
  - MQMO_MATCH_MSG_ID
  - MQMO_MATCH_CORREL_ID
  - MQMO_MATCH_GROUP_ID
- If there is a current group or logical message, only the next message in the group or next segment in the logical message is eligible for return, and this cannot be altered by specifying MQMO_* options.

In both of the preceding cases, you can specify match options that do not apply, but the value of the relevant field in the **MsgDesc** parameter must match the value of the corresponding field in the message to be returned; the call fails with reason code MQRC_MATCH_OPTIONS_ERROR is this condition is not satisfied.

MatchOptions is ignored if you specify either MQGMO_MSG_UNDER_CURSOR or MQGMO_BROWSE_MSG_UNDER_CURSOR.

Getting messages based on message property is not done using match options; for more information, see "SelectionString (MQCHARV)" on page 2463.

You can specify one or more of the following match options:

MQMO_MATCH_MSG_ID
The message to be retrieved must have a message identifier that matches the value of the *MsgId* field in the **MsgDesc** parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the correlation identifier).

If you omit this option, the *MsgId* field in the **MsgDesc** parameter is ignored, and any message identifier will match.

Note: The message identifier MQMI_NONE is a special value that matches any message identifier in the MQMD for the message. Therefore, specifying MQMO_MATCH_MSG_ID with MQMI_NONE is the same as not specifying MQMO_MATCH_MSG_ID.

MQMO_MATCH_CORREL_ID
The message to be retrieved must have a correlation identifier that matches the value of the *CorrelId* field in the **MsgDesc** parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the message identifier).

If you omit this option, the *CorrelId* field in the **MsgDesc** parameter is ignored, and any correlation identifier will match.

Note: The correlation identifier MQCI_NONE is a special value that matches *any* correlation identifier in the MQMD for the message. Therefore, specifying MQMO_MATCH_CORREL_ID with MQCI_NONE is the same as not specifying MQMO_MATCH_CORREL_ID.

MQMO_MATCH_GROUP_ID
The message to be retrieved must have a group identifier that matches the value of the *GroupId* field in the **MsgDesc** parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the correlation identifier).

If you omit this option, the *GroupId* field in the **MsgDesc** parameter is ignored, and any group identifier will match.

Note: The group identifier MQGI_NONE is a special value that matches *any* group identifier in the MQMD for the message. Therefore, specifying MQMO_MATCH_GROUP_ID with MQGI_NONE is the same as not specifying MQMO_MATCH_GROUP_ID.

MQMO_MATCH_MSG_SEQ_NUMBER
The message to be retrieved must have a message sequence number that matches the value of the *MsgSeqNumber* field in the **MsgDesc** parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the group identifier).

If you omit this option, the *MsgSeqNumber* field in the **MsgDesc** parameter is ignored, and any message sequence number will match.

MQMO_MATCH_OFFSET
The message to be retrieved must have an offset that matches the value of the *Offset* field in the **MsgDesc** parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the message sequence number).

If you omit this option is not specified, the *Offset* field in the **MsgDesc** parameter is ignored, and any offset will match.

- This option is not supported on z/OS.

MQMO_MATCH_MSG_TOKEN
The message to be retrieved must have a message token that matches the value of the *MsgToken* field in the MQGMO structure specified on the MQGET call.

You can specify this option for all local queues. If you specify it for a queue that has an *IndexType* of MQIT_MSG_TOKEN (a WLM-managed queue), you can specify no other match options with MQMO_MATCH_MSG_TOKEN.

You cannot specify MQMO_MATCH_MSG_TOKEN with MQGMO_WAIT or MQGMO_SET_SIGNAL. If the application wants to wait for a message to arrive on a queue that has an *IndexType* of MQIT_MSG_TOKEN, specify MQMO_NONE.

If you omit this option, the *MsgToken* field in MQGMO is ignored, and any message token will match.

If you specify none of the options described, you can use the following option:

MQMO_NONE
Use no matches in selecting the message to be returned; all messages on the queue are eligible for retrieval (but subject to control by the MQGMO_ALL_MSGS_AVAILABLE, MQGMO_ALL_SEGMENTS_AVAILABLE, and MQGMO_COMPLETE_MSG options).

MQMO_NONE aids program documentation. It is not intended that this option be used with any other MQMO_* option, but as its value is zero, such use cannot be detected.

This is an input field. The initial value of this field is MQMO_MATCH_MSG_ID with MQMO_MATCH_CORREL_ID. This field is ignored if *Version* is less than MQGMO_VERSION_2.

Note: The initial value of the *MatchOptions* field is defined for compatibility with earlier MQSeries queue managers. However, when reading a series of messages from a queue without using selection criteria, this initial value requires the application to reset the *MsgId* and *CorrelId* fields to MQMI_NONE and MQCI_NONE before each MQGET call. Avoid the need to reset *MsgId* and *CorrelId* by setting *Version* to MQGMO_VERSION_2, and *MatchOptions* to MQMO_NONE.

Related information:

Message selectors in JMS

MsgHandle (MQHMSG):

If the MQGMO_PROPERTIES_AS_Q_DEF option is specified and the `PropertyControl` queue attribute is not set to MQPROP_FORCE_MQRFH2 then this is the handle to a message which will be populated with the properties of the message being retrieved from the queue. The handle is created by an MQCRTMH call. Any properties already associated with the handle will be cleared before retrieving a message.

The following value can also be specified:
MQHM_NONE
No message handle supplied.

No message descriptor is required on the MQGET call if a valid message handle is supplied and used on output to contain the message properties, the message descriptor associated with the message handle is used for input fields.

If a message descriptor is specified on the MQGET call, it always takes precedence over the message descriptor associated with a message handle.

If MQGMO_PROPERTIES_FORCE_MQRFH2 is specified, or the MQGMO_PROPERTIES_AS_Q_DEF is specified and the `PropertyControl` queue attribute is MQPROP_FORCE_MQRFH2 then the call fails with reason code MQRC_MD_ERROR when no message descriptor parameter is specified.

On return from the MQGET call, the properties and message descriptor associated with this message handle are updated to reflect the state of the message retrieved (as well as the message descriptor if one was supplied on the MQGET call). The properties of the message can then be inquired using the MQINQMP call.

Except for message descriptor extensions, when present, a property that can be inquired with the MQINQMP call is not contained in the message data; if the message on the queue contained properties in the message data these are removed from the message data before the data is returned to the application.

If no message handle is provided or Version is less than MQGMO_VERSION_4 then you must supply a valid message descriptor on the MQGET call. Any message properties (except those contained in the message descriptor) are returned in the message data subject to the value of the property options in the MQGMO structure and the `PropertyControl` queue attribute.

This is an always an input field. The initial value of this field is MQHM_NONE. This field is ignored if `Version` is less than MQGMO_VERSION_4.

MsgToken (MQBYTE16):

MsgToken field - MQGMO structure. This field is used by the queue manager to uniquely identify a message.

This is a byte string that is generated by the queue manager to identify a message uniquely on a queue. The message token is generated when the message is first placed on the queue manager, and remains with the message until the message is permanently removed from the queue manager, unless the queue manager is restarted.

When the message is removed from the queue, the *MsgToken* that identified that instance of the message is no longer valid, and is never reused. If the queue manager is restarted, the *MsgToken* that identified a message on the queue before restart might not be valid after restart. However, the *MsgToken* is never reused to identify a different message instance. The *MsgToken* is generated by the queue manager and is not visible to any external application.

When a message is returned by a call to MQGET where a Version 3 or higher MQGMO is supplied, the *MsgToken* identifying the message on the queue is returned in the MQGMO by the queue manager. There is one exception to this: when the message is being removed from the queue outside syncpoint, the queue manager might not return a *MsgToken* because it is not useful to identify the returned message on a subsequent MQGET call. Applications should only use *MsgToken* to refer to the message on subsequent MQGET calls.

If a *MsgToken* is supplied and the *MatchOption* MQMO_MATCH_MSG_TOKEN is specified and neither MQGMO_MSG_UNDER_CURSOR nor MQGMO_BROWSE_MSG_UNDER_CURSOR is specified, only the message identified by that *MsgToken* can be returned. The option is valid on all local queues regardless of INDXTYPE, and on z/OS you must use INDXTYPE(MSGTOKEN) only on Workload Manager (WLM) queues.

Any other *MatchOptions* specified are checked, and if they do not match, MQRC_NO_MSG_AVAILABLE is returned. If MQGMO_BROWSE_NEXT is coded with MQMO_MATCH_MSG_TOKEN, the message identified by the *MsgToken* is returned only if it is beyond the browse-cursor for the calling handle.

If MQGMO_MSG_UNDER_CURSOR or MQGMO_BROWSE_MSG_UNDER_CURSOR is specified, MQMO_MATCH_MSG_TOKEN is ignored.

MQMO_MATCH_MSG_TOKEN is not valid with the following get message options:

- MQGMO_WAIT
- MQGMO_SET_SIGNAL

For an MQGET call specifying MQMO_MATCH_MSG_TOKEN, an MQGMO of version 3 or later must be supplied to the call, otherwise MQRC_WRONG_GMO_VERSION is returned.

If the *MsgToken* is not valid at this time, MQCC_FAILED with MQRC_NO_MSG_AVAILABLE is returned, unless there is another error.

Options (MQLONG) for MQGMO:

MQGMO options control the action of MQGET. You can specify zero or more of the options. If you need more than one optional value:

- Add the values (do not add the same constant more than once), or
- Combine the values using the bitwise OR operation (if the programming language supports bit operations).

Combinations of options that are not valid are noted; all other combinations are valid.

Wait options: The following options relate to waiting for messages to arrive on the queue:

MQGMO_WAIT

The application waits until a suitable message arrives. The maximum time that the application waits is specified in *WaitInterval*.

Important: There is no wait, or delay, if a suitable message is available immediately.

If MQGET requests are inhibited, or MQGET requests become inhibited while waiting, the wait is canceled. The call completes with MQCC_FAILED and reason code MQRC_GET_INHIBITED, regardless of whether there are suitable messages on the queue.

You can use MQGMO_WAIT with the MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT options.

If several applications are waiting on the same shared queue, the following rules select which application is activated when a suitable message arrives:

Table 203. Rules for activating MQGET calls on a shared queue.

Number of MQGET calls waiting to be activated		**Result**
With a BROWSE option	**Without a BROWSE option (An MQGET call specifying the MQGMO_LOCK option is treated as a nonbrowse call.)**	
None	One or more	One MQGET call without a BROWSE option is activated.
One or more	None	All MQGET calls with a BROWSE option are activated.
One or more	One or more	One MQGET call without a BROWSE option is activated. The number of MQGET calls with a BROWSE option that are activated is unpredictable.

If more than one MQGET call without a BROWSE option is waiting on the same queue, only one is activated. The queue manager attempts to give priority to waiting calls in the following order:

1. Specific get-wait requests that can be satisfied only by certain messages, for example, ones with a specific *MsgId* or *CorrelId* (or both).
2. General get-wait requests that can be satisfied by any message.

Note:

- Within the first category, no additional priority is given to more specific get-wait requests. For example, requests that specify both *MsgId* and *CorrelId*.

- Within either category, it cannot be predicted which application is selected. In particular, the application waiting longest is not necessarily the one selected.
- Path length, and priority-scheduling considerations of the operating system, can mean that a waiting application of lower operating system priority than expected retrieves the message.
- It can also happen that an application that is not waiting retrieves the message in preference to one that is.

On z/OS, the following points apply:

- If you want the application to proceed with other work while waiting for the message to arrive, consider using the signal option (MQGMO_SET_SIGNAL) instead. However the signal option is environment-specific; applications that you to port between different environments must not use it.
- If there is more than one MQGET call waiting for the same message, with a mixture of wait and signal options, each waiting call is considered equally. It is an error to specify MQGMO_SET_SIGNAL with MQGMO_WAIT. It is also an error to specify this option with a queue handle for which a signal is outstanding.
- If you specify MQGMO_WAIT or MQGMO_SET_SIGNAL for a queue that has an *IndexType* of MQIT_MSG_TOKEN, no selection criteria are permitted. This means that:
  - If you are using a version-1 MQGMO, set the *MsgId* and *CorrelId* fields in the MQMD specified on the MQGET call to MQMI_NONE and MQCI_NONE.
  - If you are using a version-2 or later MQGMO, set the *MatchOptions* field to MQMO_NONE.
- For an MQGET call on a shared queue and the call is a browse request, or a destructive get of a group message, and neither *MsgId* nor *CorrelId* are to be matched, your signal ECB is posted MQEC_MSG_ARRIVED after 200 milliseconds.

  This occurs, even though a suitable message might not have arrived on the queue, until the wait interval has expired, when the queue is posted with MQEC_WAIT_INTERVAL_EXPIRED. When MQEC_MSG_ARRIVED is posted, you must reissue a second MQGET call to retrieve the message, if one is available.

  This technique is used to ensure that you are informed in a timely manner of a message arrival, but can appear as an unexpected processing overhead when compared with a similar call sequence on a nonshared queue.

MQGMO_WAIT is ignored if specified with MQGMO_BROWSE_MSG_UNDER_CURSOR or MQGMO_MSG_UNDER_CURSOR ; no error is raised.

MQGMO_NO_WAIT
The application does not wait if no suitable message is available. MQGMO_NO_WAIT is the opposite of the MQGMO_WAIT. MQGMO_NO_WAIT is defined to aid program documentation. It is the default if neither is specified.

MQGMO_SET_SIGNAL
Use this option with the *Signal1* and *Signal2* fields. It allows applications to proceed with other work while waiting for a message to arrive. It also allows (if suitable operating system facilities are available) applications to wait for messages arriving on more than one queue.

Note: The MQGMO_SET_SIGNAL option is environment-specific; do not use it for applications that you want to port.

In two circumstances, the call completes in the same way as if this option had not been specified:

1. If a currently available message satisfies the criteria specified in the message descriptor.
2. If a parameter error or other synchronous error is detected.

If no message satisfying the criteria specified in the message descriptor is currently available, control returns to the application without waiting for a message to arrive. The **CompCode** and **Reason** parameters are set to MQCC_WARNING and MQRC_SIGNAL_REQUEST_ACCEPTED. Other output

fields in the message descriptor and the output parameters of the MQGET call are not set. When a suitable message arrives later, the signal is delivered by posting the ECB.

The caller must then reissue the MQGET call to retrieve the message. The application can wait for this signal, using functions provided by the operating system.

If the operating system provides a multiple wait mechanism, you can use it to wait for a message arriving on any one of several queues.

If a nonzero *WaitInterval* is specified, the signal is delivered after the wait interval expires. The queue manager can also cancel the wait, in which case the signal is delivered.

More than one MQGET call can set a signal for the same message. The order in which applications are activated is the same as described for MQGMO_WAIT.

If more than one MQGET call is waiting for the same message, each waiting call is considered equally. The calls can include a mixture of wait and signal options.

Under certain conditions the MQGET call can retrieve a message, and a signal resulting from the arrival of the same message can be delivered. When a signal is delivered, an application must be prepared for no message to be available.

A queue handle can have no more than one signal request outstanding.

This option is not valid with any of the following options:

- MQGMO_UNLOCK
- MQGMO_WAIT

For an MQGET call on a shared queue and the call is a browse request, or a destructive get of a group message, and neither *MsgId* or *CorrelId* are to be matched, the user's signal ECB is posted MQEC_MSG_ARRIVED after 200 milliseconds.

This occurs, even though a suitable message might not have arrived on the queue, until the wait interval has expired, when the queue is posted with MQEC_WAIT_INTERVAL_EXPIRED. When MQEC_MSG_ARRIVED is posted, you must reissue a second MQGET call to retrieve the message, if one is available.

This technique is used to ensure that you are informed in a timely manner of a message arrival, but can appear as an unexpected processing overhead when compared with a similar call sequence on a nonshared queue.

This is not an efficient method of message retrieval when messages are added infrequently. To avoid this overhead for the browse case, specify *MsgId* (if non-indexed or indexed by *MsgId*) or *CorrelId* (if indexed by *CorrelId*) matching on the MQGET call.

This option is supported on z/OS only.

MQGMO_FAIL_IF_QUIESCING
Force the MQGET call to fail if the queue manager is in the quiescing state.

On z/OS, this option also forces the MQGET call to fail if the connection (for a CICS or IMS application) is in the quiescing state.

If this option is specified with MQGMO_WAIT or MQGMO_SET_SIGNAL, and the wait or signal is outstanding at the time the queue manager enters the quiescing state:

- The wait is canceled and the call returns completion code MQCC_FAILED with reason code MQRC_Q_MGR_QUIESCING or MQRC_CONNECTION_QUIESCING.
- The signal is canceled with an environment-specific signal completion code.

  On z/OS, the signal completes with event completion code MQEC_Q_MGR_QUIESCING or MQEC_CONNECTION_QUIESCING.

If MQGMO_FAIL_IF_QUIESCING is not specified and the queue manager or connection enters the quiescing state, the wait or signal is not canceled.

Sync point options: The following options relate to the participation of the MQGET call within a unit of work:

MQGMO_SYNCPOINT
The request is to operate within the normal unit-of-work protocols. The message is marked as being unavailable to other applications, but it is deleted from the queue only when the unit of work is committed. The message is made available again if the unit of work is backed out.

You can leave MQGMO_SYNCPOINT and MQGMO_NO_SYNCPOINT unset. In which case, the inclusion of the get request in unit-of-work protocols is determined by the environment running the queue manager. It is not determined by the environment running the application. On z/OS, the get request is within a unit of work. In all other environments, the get request is not within a unit of work.

Because of these differences, an application that you want to port must not allow this option to default; specify MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT explicitly.

This option is not valid with any of the following options:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT
- MQGMO_LOCK
- MQGMO_NO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

MQGMO_SYNCPOINT_IF_PERSISTENT
The request is to operate within the normal unit-of-work protocols, but only if the message retrieved is persistent. A persistent message has the value MQPER_PERSISTENT in the *Persistence* field in MQMD.

- If the message is persistent, the queue manager processes the call as though the application had specified MQGMO_SYNCPOINT.
- If the message is not persistent, the queue manager processes the call as though the application had specified MQGMO_NO_SYNCPOINT.

This option is not valid with any of the following options:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT
- MQGMO_COMPLETE_MSG
- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_NO_SYNCPOINT
- MQGMO_SYNCPOINT
- MQGMO_UNLOCK

This option is supported in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, and Linux, plus IBM MQ MQI clients connected to these systems.

MQGMO_NO_SYNCPOINT
The request is to operate outside the normal unit-of-work protocols. If you get a message without a browse option, it is deleted from the queue immediately. The message cannot be made available again by backing out the unit of work.

This option is assumed if you specify MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT.

You can leave MQGMO_SYNCPOINT and MQGMO_NO_SYNCPOINT unset. In which case, the inclusion of the get request in unit-of-work protocols is determined by the environment running the queue

manager. It is not determined by the environment running the application. On z/OS, the get request is within a unit of work. In all other environments, the get request is not within a unit of work.

Because of these differences, an application that you want to port must not allow this option to default; specify either MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT explicitly.

This option is not valid with any of the following options:

- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT

MQGMO_MARK_SKIP_BACKOUT

Back out a unit of work without reinstating on the queue the message that was marked with this option.

This option is supported only on z/OS.
If this option is specified, MQGMO_SYNCPOINT must also be specified. MQGMO_MARK_SKIP_BACKOUT is not valid with any of the following options:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT
- MQGMO_LOCK
- MQGMO_NO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

Note: On IMS and CICS, you might have to issue an extran IBM MQ call after backing out a unit of work containing a message marked with MQGMO_MARK_SKIP_BACKOUT. You must issue an IBM MQ call before you commit the new unit of work containing the marked message. The call can be any IBM MQ call you like.

1. On IMS, if you have not applied IMS APAR PN60855 and you are running an IMS MPP or BMP application.
2. On CICS, if you are running any application.

In both cases, issue any IBM MQ call before committing the new unit of work containing the backed out message.

Note: Within a unit of work, there can be only one get request marked as skipping backout, as well as none or several unmarked get requests.
If an application backs out of a unit of work, a message that was retrieved using MQGMO_MARK_SKIP_BACKOUT is not restored to its previous state. Other resource updates are backed out. The message is treated as if it had been retrieved in a new unit of work started by the backout request. The message is retrieved without the MQGMO_MARK_SKIP_BACKOUT option. MQGMO_MARK_SKIP_BACKOUT is useful if, after some resources have been changed, it becomes apparent that the unit of work cannot complete successfully. If you omit this option, backing out the unit of work reinstates the message on the queue. The same sequence of events occurs again, when the message is next retrieved.
However, if you specify MQGMO_MARK_SKIP_BACKOUT on the original MQGET call, backing out the unit of work backs out the updates to the other resources. The message is treated as if it had been retrieved under a new unit of work. The application can perform appropriate error handling. It can send a report message to the sender of the original message, or place the original message on the dead-letter queue. It can then commit the new unit of work. Committing the new unit of work removes the message permanently from the original queue.
MQGMO_MARK_SKIP_BACKOUT marks a single physical message. If the message belongs to a message

group, the other messages in the group are not marked. Similarly, if the marked message is a segment of a logical message, the other segments in the logical message are not marked.
Any message in a group can be marked, but if messages are retrieved using MQGMO_LOGICAL_ORDER, it is advantageous to mark the first message in the group. If the unit of work is backed out, the first (marked) message is moved to the new unit of work. The second and later messages in the group are reinstated on the queue. The messages left on the queue cannot be retrieved by another application using MQGMO_LOGICAL_ORDER. The first message in the group is no longer on the queue. However, the application that backed the unit of work out can retrieve the second and later messages into the new unit of work using the MQGMO_LOGICAL_ORDER option. The first message has been retrieved already.
Occasionally you might need to back out the new unit of work. For example, because the dead-letter queue is full and the message must not be discarded. Backing out the new unit of work reinstates the message on the original queue, which prevents the message being lost. However, in this situation processing cannot continue. After backing out the new unit of work, the application must inform the operator or administrator that there is an unrecoverable error, and then finish.
MQGMO_MARK_SKIP_BACKOUT only works if the unit of work containing the get request is interrupted by the application backing it out. If the unit of work containing the get request is backed out because the transaction or system fails, MQGMO_MARK_SKIP_BACKOUT is ignored. Any message retrieved using this option is reinstated on the queue in the same way as messages retrieved without this option.

Browse options: The following options relate to browsing messages on the queue:

MQGMO_BROWSE_FIRST
When a queue is opened with the MQOO_BROWSE option, a browse cursor is established, positioned logically before the first message on the queue. You can then use MQGET calls specifying the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT, or MQGMO_BROWSE_MSG_UNDER_CURSOR option to retrieve messages from the queue nondestructively. The browse cursor marks the position, within the messages on the queue, from which the next MQGET call with MQGMO_BROWSE_NEXT searches for a suitable message.

MQGMO_BROWSE_FIRST is not valid with any of the following options:

- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT
- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_MSG_UNDER_CURSOR
- MQGMO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

It is also an error if the queue was not opened for browse.

An MQGET call with MQGMO_BROWSE_FIRST ignores the previous position of the browse cursor. The first message on the queue that satisfies the conditions specified in the message descriptor is retrieved. The message remains on the queue, and the browse cursor is positioned on this message.

After this call, the browse cursor is positioned on the message that has been returned. The message might be removed from the queue before the next MQGET call with MQGMO_BROWSE_NEXT is issued. In this case, the browse cursor remains at the position in the queue that the message occupied, even though that position is now empty.

Use the MQGMO_MSG_UNDER_CURSOR option with a non-browse MQGET call, to remove the message from the queue.

The browse cursor is not moved by a non-browse MQGET call, even if using the same *Hobj* handle. Nor is it moved by a browse MQGET call that returns a completion code of MQCC_FAILED, or a reason code of MQRC_TRUNCATED_MSG_FAILED.

Specify the MQGMO_LOCK option with this option, to lock the message that is browsed.

You can specify MQGMO_BROWSE_FIRST with any valid combination of the MQGMO_* and MQMO_* options that control the processing of messages in groups and segments of logical messages.

If you specify MQGMO_LOGICAL_ORDER, the messages are browsed in logical order. If you omit that option, the messages are browsed in physical order. If you specify MQGMO_BROWSE_FIRST, you can switch between logical order and physical order. Subsequent MQGET calls using MQGMO_BROWSE_NEXT browse the queue in the same order as the most recent call that specified MQGMO_BROWSE_FIRST for the queue handle.

The queue manager retains two sets of group and segment information for MQGET calls. The group and segment information for browse calls are retained separately from the information for calls that remove messages from the queue. If you specify MQGMO_BROWSE_FIRST, the queue manager ignores the group and segment information for browsing. It scans the queue as though there were no current group and no current logical message. If the MQGET call is successful, completion code MQCC_OK or MQCC_WARNING, the group and segment information for browsing is set to that of the message returned. If the call fails, the group and segment information remain the same as they were before the call.

MQGMO_BROWSE_NEXT

Advance the browse cursor to the next message on the queue that satisfies the selection criteria specified on the MQGET call. The message is returned to the application, but remains on the queue.

MQGMO_BROWSE_NEXT is not valid with any of the following options:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_MSG_UNDER_CURSOR
- MQGMO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

It is also an error if the queue was not opened for browse.
MQGMO_BROWSE_NEXT behaves the same way as MQGMO_BROWSE_FIRST, if it is the first call to browse a queue, after the queue has been opened for browse.
The message under cursor might be removed from the queue before the next MQGET call with MQGMO_BROWSE_NEXT is issued. The browse cursor logically remains at the position in the queue that the message occupied, even though that position is now empty.
Messages are stored on the queue in one of two ways:

- FIFO within priority (MQMDS_PRIORITY), or
- FIFO *regardless* of priority (MQMDS_FIFO)

The **MsgDeliverySequence** queue attribute indicates which method applies (see "Attributes for queues" on page 2833 for details).
A queue might have a *MsgDeliverySequence* of MQMDS_PRIORITY. A message arrives on the queue that is of a higher priority than the one currently pointed to by the browse cursor. In which case, the higher priority message is not found during the current sweep of the queue using MQGMO_BROWSE_NEXT. It can be found only after the browse cursor has been reset with MQGMO_BROWSE_FIRST, or by reopening the queue.
The MQGMO_MSG_UNDER_CURSOR option can be used with a non-browse MQGET call if required, to remove the message from the queue.
The browse cursor is not moved by non-browse MQGET calls using the same *Hobj* handle.

Specify the MQGMO_LOCK option with this option to lock the message that is browsed.
You can specify MQGMO_BROWSE_NEXT with any valid combination of the MQGMO_* and MQMO_* options that control the processing of messages in groups and segments of logical messages.
If you specify MQGMO_LOGICAL_ORDER, the messages are browsed in logical order. If you omit that option, the messages are browsed in physical order. If you specify MQGMO_BROWSE_FIRST, you can switch between logical order and physical order. Subsequent MQGET calls using MQGMO_BROWSE_NEXT browse the queue in the same order as the most recent call that specified MQGMO_BROWSE_FIRST for the queue handle. The call fails with reason code MQRC_INCONSISTENT_BROWSE if this condition is not satisfied.

Note: Take special care when using an MQGET call to browse beyond the end of a message group if MQGMO_LOGICAL_ORDER is not specified. For example, suppose that the last message in the group precedes the first message in the group on the queue. Using MQGMO_BROWSE_NEXT to browse beyond the end of the group, specifying MQMO_MATCH_MSG_SEQ_NUMBER with *MsgSeqNumber* set to 1 returns the first message in the group already browsed. This result can happen immediately, or a number of MQGET calls later if there are intervening groups. The same consideration applies for a logical message not in a group.
The group and segment information for browse calls are retained separately from the information for calls that remove messages from the queue.

MQGMO_BROWSE_MSG_UNDER_CURSOR

Retrieve the message pointed to by the browse cursor nondestructively, regardless of the MQMO_* options specified in the *MatchOptions* field in MQGMO.

MQGMO_BROWSE_MSG_UNDER_CURSOR is not valid with any of the following options:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_NEXT
- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_MSG_UNDER_CURSOR
- MQGMO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

It is also an error if the queue was not opened for browse.
The message pointed to by the browse cursor is the one that was last retrieved using either the MQGMO_BROWSE_FIRST or the MQGMO_BROWSE_NEXT option. The call fails if neither of these calls has been issued for this queue since it was opened. The call also fails if the message that was under the browse cursor has since been retrieved destructively.
The position of the browse cursor is not changed by this call.
The MQGMO_MSG_UNDER_CURSOR option can be used with a non-browse MQGET call, to remove the message from the queue.
The browse cursor is not moved by a non-browse MQGET call, even if using the same *Hobj* handle. Nor is it moved by a browse MQGET call that returns a completion code of MQCC_FAILED, or a reason code of MQRC_TRUNCATED_MSG_FAILED.
If MQGMO_BROWSE_MSG_UNDER_CURSOR is specified with MQGMO_LOCK:

- If there is already a message locked, it must be the one under the cursor, so that is returned without unlocking and locking again. The message remains locked.
- If there is no locked message and there is a message under the browse cursor, it is locked and returned to the application. If there is no message under the browse cursor, the call fails.

If MQGMO_BROWSE_MSG_UNDER_CURSOR is specified without MQGMO_LOCK:

- If there is already a message locked, it must be the one under the cursor. The message is returned to the application and then unlocked. Because the message is now unlocked, there is

no guarantee that it can be browsed again, or retrieved destructively by the same application. It might have been retrieved destructively by another application getting messages from the queue.

- If there is no locked message and there is a message under the browse cursor, it is returned to the application. If there is no message under the browse cursor the call fails.

If MQGMO_COMPLETE_MSG is specified with MQGMO_BROWSE_MSG_UNDER_CURSOR, the browse cursor must identify a message whose *Offset* field in MQMD is zero. If this condition is not satisfied, the call fails with reason code MQRC_INVALID_MSG_UNDER_CURSOR.
The group and segment information for browse calls are retained separately from the information for calls that remove messages from the queue.

MQGMO_MSG_UNDER_CURSOR
Retrieve the message pointed to by the browse cursor, regardless of the MQMO_* options specified in the *MatchOptions* field in MQGMO. The message is removed from the queue.

The message pointed to by the browse cursor is the one that was last retrieved using either the MQGMO_BROWSE_FIRST or the MQGMO_BROWSE_NEXT option.

If MQGMO_COMPLETE_MSG is specified with MQGMO_MSG_UNDER_CURSOR, the browse cursor must identify a message whose *Offset* field in MQMD is zero. If this condition is not satisfied, the call fails with reason code MQRC_INVALID_MSG_UNDER_CURSOR.

This option is not valid with any of the following options:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT
- MQGMO_UNLOCK

It is also an error if the queue was not opened both for browse and for input. If the browse cursor is not currently pointing to a retrievable message, an error is returned by the MQGET call.

MQGMO_MARK_BROWSE_HANDLE
The message that is returned by a successful MQGET, or identified by the returned *MsgToken*, is marked. The mark is specific to the object handle used in the call.

The message is not removed from the queue.

MQGMO_MARK_BROWSE_HANDLE is valid only if one of the following options is also specified:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT

MQGMO_MARK_BROWSE_HANDLE is not valid with any of the following options:

- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE
- MQGMO_COMPLETE_MSG
- MQGMO_LOCK
- MQGMO_LOGICAL_ORDER
- MQGMO_UNLOCK

The message remains in this state until one of the following events occurs:

- The object handle concerned is closed, either normally or otherwise.
- The message is unmarked for this handle by a call to MQGET with the option MQGMO_UNMARK_BROWSE_HANDLE.
- The message is returned from a call to destructive MQGET, which completes with MQCC_OK or MQCC_WARNING. The message state remains changed even if the MQGET is later rolled-back.

- The message expires.

MQGMO_MARK_BROWSE_CO_OP

The message that is returned by a successful MQGET, or identified by the returned *MsgToken*, is marked for all handles in the cooperating set.

The cooperative level mark is in addition to any handle level mark that might have been set.

The message is not removed from the queue.

MQGMO_MARK_BROWSE_CO_OP is valid only if the object handle used was returned by a call to MQOPEN that specified MQOO_CO_OP. You must also specify one of the following MQGMO options:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT

This option is not valid with any of the following options:

- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE
- MQGMO_COMPLETE_MSG
- MQGMO_LOCK
- MQGMO_LOGICAL_ORDER
- MQGMO_UNLOCK

If the message is already marked, and the option MQGMO_UNMARKED_BROWSE_MSG is not specified, the call fails with MQCC_FAILED and reason code MQRC_MSG_MARKED_BROWSE_CO_OP.

The message remains in this state until one of the following events occurs:

- All object handles in the cooperating set are closed.
- The message is unmarked for cooperating browsers by a call to MQGET with the option MQGMO_UNMARK_BROWSE_CO_OP.
- The message is automatically unmarked by the queue manager.
- The message is returned from a call to a non-browse MQGET. The message state remains changed even if the MQGET is later rolled-back.
- The message expires.

MQGMO_UNMARKED_BROWSE_MSG

A call to MQGET that specifies MQGMO_UNMARKED_BROWSE_MSG returns a message that is considered to be unmarked for its handle. It does not return a message if the message was marked for its handle. It also does not return the message if the queue was opened by a call to MQOPEN, with the option MQOO_CO_OP, and the message has been marked by a member of the cooperating set.

This option is not valid with any of the following options:

- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE
- MQGMO_COMPLETE_MSG
- MQGMO_LOCK
- MQGMO_LOGICAL_ORDER
- MQGMO_UNLOCK

MQGMO_UNMARK_BROWSE_CO_OP

After a call to MQGET that specifies this option, the message is no longer considered by any open handles in the set of cooperating handles to be marked for the cooperating set. The message is still considered to be marked at handle level if it was marked at handle level before this call.

Using MQGMO_UNMARK_BROWSE_CO_OP is valid only with a handle returned by a successful call to MQOPEN with the option MQOO_CO_OP. The MQGET succeeds even if the message is not considered to be marked by the cooperating set of handles.

MQGMO_UNMARK_BROWSE_CO_OP is not valid on a non-browse MQGET call, or with any of the following options:

- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE
- MQGMO_COMPLETE_MSG
- MQGMO_LOCK
- MQGMO_LOGICAL_ORDER
- MQGMO_MARK_BROWSE_CO_OP
- MQGMO_UNLOCK
- MQGMO_UNMARKED_BROWSE_MSG

MQGMO_UNMARK_BROWSE_HANDLE
After a call to MQGET that specifies this option, the message located is no longer considered to be marked by this handle.

The call succeeds even if the message is not marked for this handle.

This option is not valid on a non-browse MQGET call, or with any of the following options:

- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE
- MQGMO_COMPLETE_MSG
- MQGMO_LOCK
- MQGMO_LOGICAL_ORDER
- MQGMO_MARK_BROWSE_CO_OP
- MQGMO_UNLOCK
- MQGMO_UNMARKED_BROWSE_MSG

Lock options: The following options relate to locking messages on the queue:

MQGMO_LOCK
Lock the message that is browsed, so that the message becomes invisible to any other handle open for the queue. The option can be specified only if one of the following options is also specified:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_NEXT
- MQGMO_BROWSE_MSG_UNDER_CURSOR

Only one message can be locked for each queue handle. The message can be a logical message or a physical message:

- If you specify MQGMO_COMPLETE_MSG, all the message segments that make up the logical message are locked to the queue handle. The messages must all be present on the queue and available for retrieval.
- If you omit MQGMO_COMPLETE_MSG, only a single physical message is locked to the queue handle. If this message happens to be a segment of a logical message, the locked segment prevents other applications using MQGMO_COMPLETE_MSG to retrieve or browse the logical message.

The locked message is always the one under the browse cursor. The message can be removed from the queue by a later MQGET call that specifies the MQGMO_MSG_UNDER_CURSOR option. Other MQGET calls using the queue handle can also remove the message (for example, a call that specifies the message identifier of the locked message).

If the call returns completion code MQCC_FAILED, or MQCC_WARNING with reason code MQRC_TRUNCATED_MSG_FAILED, no message is locked.

If the application does not remove the message from the queue, the lock is released by one of the following actions:

- Issuing another MQGET call for this handle, specifying either MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT. The lock is released if the call completes with MQCC_OK or MQCC_WARNING. The message remains locked if the call completes with MQCC_FAILED. However, the following exceptions apply:
  - The message is not unlocked if MQCC_WARNING is returned with MQRC_TRUNCATED_MSG_FAILED.
  - The message is unlocked if MQCC_FAILED is returned with MQRC_NO_MSG_AVAILABLE.

  If you also specify MQGMO_LOCK, the message returned is locked. If you omit MQGMO_LOCK, there is no locked message after the call.

  If you specify MQGMO_WAIT, and no message is immediately available, the original message is unlocked before the start of the wait.
- Issuing another MQGET call for this handle, with MQGMO_BROWSE_MSG_UNDER_CURSOR, without MQGMO_LOCK. The lock is released if the call completes with MQCC_OK or MQCC_WARNING. The message remains locked if the call completes with MQCC_FAILED. However, the following exception applies:
  - The message is not unlocked if MQCC_WARNING is returned with MQRC_TRUNCATED_MSG_FAILED.
- Issuing another MQGET call for this handle with MQGMO_UNLOCK.
- Issuing an MQCLOSE call using the handle. The MQCLOSE might be implicit, caused by the application ending.

No special MQOPEN option is required to specify MQGMO_LOCK, other than MQOO_BROWSE, which is needed to specify an accompanying browse option.

MQGMO_LOCK is not valid with any of the following options:

- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

NSS Client MQGMO_LOCK is not possible when you are using an IBM MQ client for HP Integrity NonStop Server to a z/OS queue manager when coordinated by TMF.

MQGMO_UNLOCK

The message to be unlocked must have been previously locked by an MQGET call with the MQGMO_LOCK option. If there is no message locked for this handle, the call completes with MQCC_WARNING and MQRC_NO_MSG_LOCKED.

The **MsgDesc**, **BufferLength**, **Buffer**, and **DataLength** parameters are not checked or altered if you specify MQGMO_UNLOCK. No message is returned in *Buffer*.

No special open option is required to specify MQGMO_UNLOCK (although MQOO_BROWSE is needed to issue the lock request in the first place).

This option is not valid with any options except the following:

- MQGMO_NO_WAIT
- MQGMO_NO_SYNCPOINT

Both of these options are assumed whether specified or not.

Message-data options: The following options relate to the processing of the message data when the message is read from the queue:

MQGMO_ACCEPT_TRUNCATED_MSG

If the message buffer is too small to hold the complete message, allow the MQGET call to fill the buffer. MQGET fills the buffer with as much of the message it can. It issues a warning completion code, and completes its processing. This means that:

- When browsing messages, the browse cursor is advanced to the returned message.
- When removing messages, the returned message is removed from the queue.
- Reason code MQRC_TRUNCATED_MSG_ACCEPTED is returned if no other error occurs.

Without this option, the buffer is still filled with as much of the message as it can hold. A warning completion code is issued, but processing is not completed. This means that:

- When browsing messages, the browse cursor is not advanced.
- When removing messages, the message is not removed from the queue.
- Reason code MQRC_TRUNCATED_MSG_FAILED is returned if no other error occurs.

MQGMO_CONVERT

This option converts the application data in the message to conform to the *CodedCharSetId* and *Encoding* values specified in the **MsgDesc** parameter on the MQGET call. The data is converted before it is copied to the **Buffer** parameter.

The *Format* field specified when the message was put is assumed by the conversion process to identify the nature of the data in the message. The message data is converted by the queue manager for built-in formats, and by a user-written exit for other formats. See "Data conversion" on page 2909 for details of the data-conversion exit.

- If conversion is successful, the *CodedCharSetId* and *Encoding* fields specified in the **MsgDesc** parameter are unchanged on return from the MQGET call.
- If only conversion fails the message data is returned unconverted The *CodedCharSetId* and *Encoding* fields in *MsgDesc* are set to the values for the unconverted message. The completion code is MQCC_WARNING in this case.

In either case, these fields describe the character-set identifier and encoding of the message data that is returned in the **Buffer** parameter.

See the *Format* field described in "MQMD - Message descriptor" on page 2387 for a list of format names for which the queue manager performs the conversion.

Group and segment options: The following options relate to the processing of messages in groups and segments of logical messages. Before the option descriptions, here are some definitions of important terms:

Physical message

A physical message is the smallest unit of information that can be placed on or removed from a queue. It often corresponds to the information specified or retrieved on a single MQPUT, MQPUT1, or MQGET call. Every physical message has its own message descriptor, MQMD. Typically, physical messages are distinguished by differing values for the message identifier, the *MsgId* field in MQMD. The queue manager does not enforce different values.

Logical message

A logical message is a single unit of application information. In the absence of system constraints, a logical message is the same as a physical message. If logical messages are large, system constraints might make it advisable or necessary to split a logical message into two or more physical messages, called segments.

A logical message that has been segmented consists of two or more physical messages that have the same nonnull group identifier, *GroupId* field in MQMD. They have the same message sequence number, *MsgSeqNumber* field in MQMD. The segments are distinguished by differing values for the segment offset, *Offset* field in MQMD. The segment offset is the offset of the data in the physical message from the start of the data in the logical message. Because each segment is a physical message, the segments in a logical message typically have different message identifiers.

A logical message that has not been segmented, but for which segmentation has been permitted by the sending application, also has a nonnull group identifier. In this case there is only one physical message with that group identifier if the logical message does not belong to a message group. Logical messages, for which segmentation has been inhibited by the sending application, have a null group identifier, MQGI_NONE, unless the logical message belongs to a message group.

Message group

A message group is a set of one or more logical messages that have the same nonnull group identifier. The logical messages in the group are distinguished by different values for the message sequence number. The sequence number is an integer in the range 1 through n, where n is the number of logical messages in the group. If one or more of the logical messages is segmented, there are more than n physical messages in the group.

MQGMO_LOGICAL_ORDER

MQGMO_LOGICAL_ORDER controls the order in which messages are returned by successive MQGET calls for the queue handle. The option must be specified on each call.

If MQGMO_LOGICAL_ORDER is specified for successive MQGET calls for the same queue handle, messages in groups are returned in the order of their message sequence numbers. Segments of logical messages are returned in the order given by their segment offsets. This order might be different from the order in which those messages and segments occur on the queue.

Note: Specifying MQGMO_LOGICAL_ORDER has no adverse consequences on messages that do not belong to groups and that are not segments. In effect, such messages are treated as though each belonged to a message group consisting of only one message. It is safe to specify MQGMO_LOGICAL_ORDER when retrieving messages from queues that contain a mixture of messages in groups, message segments, and unsegmented messages not in groups.

To return the messages in the required order, the queue manager retains the group and segment information between successive MQGET calls. The group and segment information identifies the current message group and current logical message for the queue handle. It also identifies the current position within the group and logical message, and whether the messages are being retrieved within a unit of work. Because the queue manager retains this information, the application does not need to set the group and segment information before each MQGET call. Specifically, it means that the application does not need to set the *GroupId*, *MsgSeqNumber*, and *Offset* fields in MQMD. However, the application must set the MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT option correctly on each call.

When the queue is opened, there is no current message group and no current logical message. A message group becomes the current message group when a message that has the MQMF_MSG_IN_GROUP flag is returned by the MQGET call. With MQGMO_LOGICAL_ORDER specified on successive calls, that group remains the current group until a message is returned that has:

- MQMF_LAST_MSG_IN_GROUP without MQMF_SEGMENT (that is, the last logical message in the group is not segmented), or
- MQMF_LAST_MSG_IN_GROUP with MQMF_LAST_SEGMENT (that is, the message returned is the last segment of the last logical message in the group).

When such a message is returned, the message group is terminated, and on successful completion of the MQGET call there is no longer a current group. In a similar way, a logical message becomes the current logical message when a message that has the MQMF_SEGMENT flag is returned by the MQGET call. The logical message is terminated when the message that has the MQMF_LAST_SEGMENT flag is returned.

If no selection criteria are specified, successive MQGET calls return, in the correct order, the messages for the first message group on the queue. They then return the messages for the second message group, and so on, until there are no more messages available. It is possible to select the particular message groups returned by specifying one or more of the following options in the *MatchOptions* field:

- MQMO_MATCH_MSG_ID

- MQMO_MATCH_CORREL_ID
- MQMO_MATCH_GROUP_ID

However, these options are effective only when there is no current message group or logical message. See the *MatchOptions* field described in "MQGMO - Get-message options" on page 2330 for further details.

Table 204 shows the values of the *MsgId*, *CorrelId*, *GroupId*, *MsgSeqNumber*, and *Offset* fields that the queue manager looks for when attempting to find a message to return on the MQGET call. The rules apply both to removing messages from the queue, and browsing messages on the queue. In the table, Either means Yes or No:

LOG ORD
Indicates whether the MQGMO_LOGICAL_ORDER option is specified on the call.

Cur grp
Indicates whether a current message group exists before the call.

Cur log msg
Indicates whether a current logical message exists before the call.

Other columns
Show the values that the queue manager looks for. Previous denotes the value returned for the field in the previous message for the queue handle.

Table 204. MQGET options relating to messages in groups and segments of logical messages

Options you specify	Group and log-msg status before call		Values the queue manager looks for				
LOG ORD	**Cur grp**	**Cur log msg**	*MsgId*	*CorrelId*	*GroupId*	*MsgSeqNumber*	*Offset*
Yes	No	No	Controlled by *MatchOptions*	Controlled by *MatchOptions*	Controlled by *MatchOptions*	1	0
Yes	No	Yes	Any message identifier	Any correlation identifier	Previous group identifier	1	Previous offset + previous segment length
Yes	Yes	No	Any message identifier	Any correlation identifier	Previous group identifier	Previous sequence number + 1	0
Yes	Yes	Yes	Any message identifier	Any correlation identifier	Previous group identifier	Previous sequence number	Previous offset + previous segment length
No	Either	Either	Controlled by *MatchOptions*	Controlled by *MatchOptions*	Controlled by *MatchOptions*	Controlled by *MatchOptions*	Controlled by *MatchOptions*

If multiple message groups are present on the queue and eligible for return, the groups are returned in the order determined by the position on the queue of the first segment of the first logical message in each group. That is, the physical messages that have message sequence numbers of 1, and offsets of 0, determine the order in which eligible groups are returned.

The MQGMO_LOGICAL_ORDER option affects units of work as follows:

- If the first logical message or segment in a group is retrieved within a unit of work, all the other logical messages and segments in the group must be retrieved within a unit of work, if the same queue handle is used. However, they need not be retrieved within the same unit of work. This allows a message group consisting of many physical messages to be split across two or more consecutive units of work for the queue handle.

- If the first logical message or segment in a group is not retrieved within a unit of work, and the same queue handle is used, none of the other logical messages and segments in the group can be retrieved within a unit of work.

If these conditions are not satisfied, the MQGET call fails with reason code MQRC_INCONSISTENT_UOW.

When MQGMO_LOGICAL_ORDER is specified, the MQGMO supplied on the MQGET call must not be less than MQGMO_VERSION_2, and the MQMD must not be less than MQMD_VERSION_2. If this condition is not satisfied, the call fails with reason code MQRC_WRONG_GMO_VERSION or MQRC_WRONG_MD_VERSION, as appropriate.

If MQGMO_LOGICAL_ORDER is not specified for successive MQGET calls for the queue handle, messages are returned without regard for whether they belong to message groups, or whether they are segments of logical messages. This means that messages or segments from a particular group or logical message might be returned out of order, or intermingled with messages or segments from other groups or logical messages, or with messages that are not in groups and are not segments. In this situation, the particular messages that are returned by successive MQGET calls is controlled by the MQMO_* options specified on those calls (see the *MatchOptions* field described in "MQGMO - Get-message options" on page 2330 for details of these options).

This is the technique that can be used to restart a message group or logical message in the middle, after a system failure has occurred. When the system restarts, the application can set the *GroupId*, *MsgSeqNumber*, *Offset*, and *MatchOptions* fields to the appropriate values, and then issue the MQGET call with MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT set, but *without* specifying MQGMO_LOGICAL_ORDER. If this call is successful, the queue manager retains the group and segment information, and subsequent MQGET calls using that queue handle can specify MQGMO_LOGICAL_ORDER as normal.

The group and segment information that the queue manager retains for the MQGET call is separate from the group and segment information that it retains for the MQPUT call. In addition, the queue manager retains separate information for:

- MQGET calls that remove messages from the queue.
- MQGET calls that browse messages on the queue.

For any given queue handle, the application can mix MQGET calls that specify MQGMO_LOGICAL_ORDER with MQGET calls that do not. However, note the following points:

- If you omit MQGMO_LOGICAL_ORDER, each successful MQGET call causes the queue manager to set the saved group and segment information to the values corresponding to the message returned; this replaces the existing group and segment information retained by the queue manager for the queue handle. Only the information appropriate to the action of the call (browse or remove) is modified.
- If you omit MQGMO_LOGICAL_ORDER, the call does not fail if there is a current message group or logical message; the call might succeed with an MQCC_WARNING completion code. Table 205 on page 2351 shows the various cases that can arise. In these cases, if the completion code is not MQCC_OK, the reason code is one of the following (as appropriate):
  - MQRC_INCOMPLETE_GROUP
  - MQRC_INCOMPLETE_MSG
  - MQRC_INCONSISTENT_UOW

  Note: The queue manager does not check the group and segment information when browsing a queue, or when closing a queue that was opened for browse but not input; in those cases the completion code is always MQCC_OK (assuming no other errors).

Table 205. Outcome when MQGET or MQCLOSE call is not consistent with group and segment information

Current call is	Previous call was MQGET with MQGMO_LOGICAL_ORDER	Previous call was MQGET without MQGMO_LOGICAL_ORDER
MQGET with MQGMO_LOGICAL_ORDER	MQCC_FAILED	MQCC_FAILED
MQGET without MQGMO_LOGICAL_ORDER	MQCC_WARNING	MQCC_OK
MQCLOSE with an unterminated group or logical message	MQCC_WARNING	MQCC_OK

Applications that want to retrieve messages and segments in logical order are recommended to specify MQGMO_LOGICAL_ORDER, as this is the simplest option to use. This option relieves the application of the need to manage the group and segment information, because the queue manager manages that information. However, specialized applications might need more control than that provided by the MQGMO_LOGICAL_ORDER option, and this can be achieved by not specifying that option. The application must then ensure that the *MsgId*, *CorrelId*, *GroupId*, *MsgSeqNumber*, and *Offset* fields in MQMD, and the MQMO_* options in *MatchOptions* in MQGMO, are set correctly, before each MQGET call.

For example, an application that wants to *forward* physical messages that it receives, without regard for whether those messages are in groups or segments of logical messages, must not specify MQGMO_LOGICAL_ORDER. In a complex network with multiple paths between sending and receiving queue managers, the physical messages might arrive out of order. By specifying neither MQGMO_LOGICAL_ORDER, nor the corresponding MQPMO_LOGICAL_ORDER on the MQPUT call, the forwarding application can retrieve and forward each physical message as soon as it arrives, without having to wait for the next one in logical order to arrive.

You can specify MQGMO_LOGICAL_ORDER with any of the other MQGMO_* options, and with various of the MQMO_* options in appropriate circumstances (see preceding section).

- On z/OS, this option is supported for private and shared queues, but the queue must have an index type of MQIT_GROUP_ID. For shared queues, the CFSTRUCT object that the queue maps to must be at CFLEVEL(3) or higher.
- On AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ MQI clients connected to these systems, this option is supported for all local queues.

MQGMO_COMPLETE_MSG

Only a complete logical message can be returned by the MQGET call. If the logical message is segmented, the queue manager reassembles the segments and returns the complete logical message to the application; the fact that the logical message was segmented is not apparent to the application retrieving it.

Note: This is the only option that causes the queue manager to reassemble message segments. If not specified, segments are returned individually to the application if they are present on the queue (and they satisfy the other selection criteria specified on the MQGET call). Applications that do not want to receive individual segments must always specify MQGMO_COMPLETE_MSG.

To use this option, the application must provide a buffer that is big enough to accommodate the complete message, or specify the MQGMO_ACCEPT_TRUNCATED_MSG option.

If the queue contains segmented messages with some of the segments missing (perhaps because they have been delayed in the network and have not yet arrived), specifying MQGMO_COMPLETE_MSG prevents the retrieval of segments belonging to incomplete logical messages. However, those message segments still contribute to the value of the **CurrentQDepth** queue attribute; this means that there might be no retrievable logical messages, even though *CurrentQDepth* is greater than zero.

For *persistent* messages, the queue manager can reassemble the segments only within a unit of work:

- If the MQGET call is operating within a user-defined unit of work, that unit of work is used. If the call fails during the reassembly process, the queue manager reinstates on the queue any segments that were removed during reassembly. However, the failure does not prevent the unit of work being committed successfully.
- If the call is operating outside a user-defined unit of work, and there is no user-defined unit of work in existence, the queue manager creates a unit of work for the duration of the call. If the call is successful, the queue manager commits the unit of work automatically (the application does not need to do this). If the call fails, the queue manager backs out the unit of work.
- If the call is operating outside a user-defined unit of work, but a user-defined unit of work exists, the queue manager cannot reassemble. If the message does not require reassembly, the call can still succeed. But if the message requires reassembly, the call fails with reason code MQRC_UOW_NOT_AVAILABLE.

For *nonpersistent* messages, the queue manager does not require a unit of work to be available to perform reassembly.

Each physical message that is a segment has its own message descriptor. For the segments constituting a single logical message, most of the fields in the message descriptor are the same for all segments in the logical message; typically it is only the *MsgId*, *Offset*, and *MsgFlags* fields that differ between segments in the logical message. However, if a segment is placed on a dead-letter queue at an intermediate queue manager, the DLQ handler retrieves the message specifying the MQGMO_CONVERT option, and this can result in the character set or encoding of the segment being changed. If the DLQ handler successfully sends the segment on its way, the segment might have a character set or encoding that differs from the other segments in the logical message when the segment arrives at the destination queue manager.

A logical message consisting of segments in which the *CodedCharSetId* and *Encoding* fields differ cannot be reassembled by the queue manager into a single logical message. Instead, the queue manager reassembles and returns the first few consecutive segments at the start of the logical message that have the same character-set identifiers and encodings, and the MQGET call completes with completion code MQCC_WARNING and reason code MQRC_INCONSISTENT_CCSIDS or MQRC_INCONSISTENT_ENCODINGS, as appropriate. This happens regardless of whether MQGMO_CONVERT is specified. To retrieve the remaining segments, the application must reissue the MQGET call without the MQGMO_COMPLETE_MSG option, retrieving the segments one by one. MQGMO_LOGICAL_ORDER can be used to retrieve the remaining segments in order.

An application that puts segments can also set other fields in the message descriptor to values that differ between segments. However, there is no advantage in doing this if the receiving application uses MQGMO_COMPLETE_MSG to retrieve the logical message. When the queue manager reassembles a logical message, it returns in the message descriptor the values from the message descriptor for the *first* segment; the only exception is the *MsgFlags* field, which the queue manager sets to indicate that the reassembled message is the only segment.

If MQGMO_COMPLETE_MSG is specified for a report message, the queue manager performs special processing. The queue manager checks the queue to see if all the report messages of that report type relating to the different segments in the logical message are present on the queue. If they are, they can be retrieved as a single message by specifying MQGMO_COMPLETE_MSG. For this to be possible, either the report messages must be generated by a queue manager or MCA which supports segmentation, or the originating application must request at least 100 bytes of message data (that is, the appropriate MQRO_*_WITH_DATA or MQRO_*_WITH_FULL_DATA options must be specified). If less than the full amount of application data is present for a segment, the missing bytes are replaced by nulls in the report message returned.

If MQGMO_COMPLETE_MSG is specified with MQGMO_MSG_UNDER_CURSOR or MQGMO_BROWSE_MSG_UNDER_CURSOR, the browse cursor must be positioned on a message whose *Offset* field in MQMD has a value of 0. If this condition is not satisfied, the call fails with reason code MQRC_INVALID_MSG_UNDER_CURSOR.

MQGMO_COMPLETE_MSG implies MQGMO_ALL_SEGMENTS_AVAILABLE, which need not therefore be specified.

MQGMO_COMPLETE_MSG can be specified with any of the other MQGMO_* options apart from MQGMO_SYNCPOINT_IF_PERSISTENT, and with any of the MQMO_* options apart from MQMO_MATCH_OFFSET.

- On z/OS, this option is supported for private and shared queues, but the queue must have an index type of MQIT_GROUP_ID. For shared queues, the CFSTRUCT object that the queue map to must be at CFLEVEL(3) or higher.
- On AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ MQI clients connected to these systems, this option is supported for all local queues.

MQGMO_ALL_MSGS_AVAILABLE

Messages in a group become available for retrieval only when *all* messages in the group are available. If the queue contains message groups with some of the messages missing (perhaps because they have been delayed in the network and have not yet arrived), specifying MQGMO_ALL_MSGS_AVAILABLE prevents retrieval of messages belonging to incomplete groups. However, those messages still contribute to the value of the **CurrentQDepth** queue attribute; this means that there may be no retrievable message groups, even though *CurrentQDepth* is greater than zero. If there are no other messages that are retrievable, reason code MQRC_NO_MSG_AVAILABLE is returned after the specified wait interval (if any) has expired.

The processing of MQGMO_ALL_MSGS_AVAILABLE depends on whether MQGMO_LOGICAL_ORDER is also specified:

- If both options are specified, MQGMO_ALL_MSGS_AVAILABLE has an effect only when there is no current group or logical message. If there is a current group or logical message, MQGMO_ALL_MSGS_AVAILABLE is ignored. This means that MQGMO_ALL_MSGS_AVAILABLE can remain on when processing messages in logical order.
- If MQGMO_ALL_MSGS_AVAILABLE is specified without MQGMO_LOGICAL_ORDER, MQGMO_ALL_MSGS_AVAILABLE *always* has an effect. This means that the option must be turned off after the first message in the group has been removed from the queue, in order to be able to remove the remaining messages in the group.

Successful completion of an MQGET call specifying MQGMO_ALL_MSGS_AVAILABLE means that at the time that the MQGET call was issued, all the messages in the group were on the queue. However, be aware that other applications can still remove messages from the group (the group is not locked to the application that retrieves the first message in the group).

If you omit this option, messages belonging to groups can be retrieved even when the group is incomplete.

MQGMO_ALL_MSGS_AVAILABLE implies MQGMO_ALL_SEGMENTS_AVAILABLE, which need not therefore be specified.

MQGMO_ALL_MSGS_AVAILABLE can be specified with any of the other MQGMO_* options, and with any of the MQMO_* options.

- On z/OS, this option is supported for private and shared queues, but the queue must have an index type of MQIT_GROUP_ID. For shared queues, the CFSTRUCT object that the queue map to must be at CFLEVEL(3) or higher.
- On AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ MQI clients connected to these systems, this option is supported for all local queues.

MQGMO_ALL_SEGMENTS_AVAILABLE

Segments in a logical message become available for retrieval only when *all* segments in the logical message are available. If the queue contains segmented messages with some of the segments missing (perhaps because they have been delayed in the network and have not yet arrived), specifying MQGMO_ALL_SEGMENTS_AVAILABLE prevents retrieval of segments belonging to incomplete logical messages. However, those segments still contribute to the value of the **CurrentQDepth**

queue attribute; this means that there might be no retrievable logical messages, even though *CurrentQDepth* is greater than zero. If there are no other messages that are retrievable, reason code MQRC_NO_MSG_AVAILABLE is returned after the specified wait interval (if any) has expired.

The processing of MQGMO_ALL_SEGMENTS_AVAILABLE depends on whether MQGMO_LOGICAL_ORDER is also specified:

- If both options are specified, MQGMO_ALL_SEGMENTS_AVAILABLE has an effect only when there is no current logical message. If there is a current logical message, MQGMO_ALL_SEGMENTS_AVAILABLE is ignored. This means that MQGMO_ALL_SEGMENTS_AVAILABLE can remain on when processing messages in logical order.
- If MQGMO_ALL_SEGMENTS_AVAILABLE is specified without MQGMO_LOGICAL_ORDER, MQGMO_ALL_SEGMENTS_AVAILABLE *always* has an effect. This means that the option must be turned off after the first segment in the logical message has been removed from the queue, in order to be able to remove the remaining segments in the logical message.

If this option is not specified, message segments can be retrieved even when the logical message is incomplete.

While both MQGMO_COMPLETE_MSG and MQGMO_ALL_SEGMENTS_AVAILABLE require all segments to be available before any of them can be retrieved, the former returns the complete message, whereas the latter allows the segments to be retrieved one by one.

If MQGMO_ALL_SEGMENTS_AVAILABLE is specified for a report message, the queue manager checks the queue to see if there is at least one report message for each of the segments that make up the complete logical message. If there is, the MQGMO_ALL_SEGMENTS_AVAILABLE condition is satisfied. However, the queue manager does not check the *type* of the report messages present, and so there might be a mixture of report types in the report messages relating to the segments of the logical message. As a result, the success of MQGMO_ALL_SEGMENTS_AVAILABLE does not imply that MQGMO_COMPLETE_MSG will succeed. If there is a mixture of report types present for the segments of a particular logical message, those report messages must be retrieved one by one.

You can specify MQGMO_ALL_SEGMENTS_AVAILABLE with any of the other MQGMO_* options, and with any of the MQMO_* options.

- On z/OS, this option is supported for private and shared queues, but the queue must have an index type of MQIT_GROUP_ID. For shared queues, the CFSTRUCT object that the queue map to must be at CFLEVEL(3) or higher.
- On AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ MQI clients connected to these systems, this option is supported for all local queues.

Property options: The following options relate to the properties of the message:

MQGMO_PROPERTIES_AS_Q_DEF

Properties of the message, except those contained in the message descriptor (or extension) should be represented as defined by the **PropertyControl** queue attribute. If a *MsgHandle* is provided this option is ignored and the properties of the message are available via the *MsgHandle*, unless the value of the **PropertyControl** queue attribute is MQPROP_FORCE_MQRFH2.

This is the default action if no property options are specified.

MQGMO_PROPERTIES_IN_HANDLE

Properties of the message should be made available via the *MsgHandle*. If no message handle is provided the call fails with reason MQRC_HMSG_ERROR.

Note: If the message is later read by an application that does not create a message handle, the queue manager places any message properties into an MQRFH2 structure. You might find that the presence of an unexpected MQRFH2 header disrupts the behavior of an existing application.

MQGMO_NO_PROPERTIES

No properties of the message, except those contained in the message descriptor (or extension) will be retrieved. If a *MsgHandle* is provided it will be ignored.

MQGMO_PROPERTIES_FORCE_MQRFH2

Properties of the message, except those contained in the message descriptor (or extension) should be represented using MQRFH2 headers. This provides compatibility with earlier version for applications which are expecting to retrieve properties but are unable to be changed to use message handles. If a *MsgHandle* is provided it is ignored.

MQGMO_PROPERTIES_COMPATIBILITY

If the message contains a property with a prefix of **"mcd."**, **"jms."**, **"usr."**, or **"mqext."**, all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except those contained in the message descriptor (or extension), are discarded and are no longer accessible to the application.

Default option: If none of the options described is required, the following option can be used:

MQGMO_NONE

Use this value to indicate that no other options have been specified; all options assume their default values. MQGMO_NONE aids program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

The initial value of the *Options* field is MQGMO_NO_WAIT plus MQGMO_PROPERTIES_AS_Q_DEF.

Reserved1 (MQCHAR):

This is a reserved field. The initial value of this field is a blank character. This field is ignored if *Version* is less than MQGMO_VERSION_2.

Reserved2 (MQLONG):

This is a reserved field. The initial value of this field is a blank character. This field is ignored if *Version* is less than **MQGMO_VERSION_4**.

ResolvedQName (MQCHAR48):

This is an output field that the queue manager sets to the local name of the queue from which the message was retrieved, as defined to the local queue manager. This is different from the name used to open the queue if:

- An alias queue was opened (in which case, the name of the local queue to which the alias resolved is returned), or
- A model queue was opened (in which case, the name of the dynamic local queue is returned).

The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ReturnedLength (MQLONG):

This is an output field that the queue manager sets to the length in bytes of the message data returned by the MQGET call in the **Buffer** parameter. If the queue manager does not support this capability, *ReturnedLength* is set to the value MQRL_UNDEFINED.

When messages are converted between encodings or character sets, the message data can sometimes change size. On return from the MQGET call:

- If *ReturnedLength* is not MQRL_UNDEFINED, the number of bytes of message data returned is given by *ReturnedLength*.
- If *ReturnedLength* has the value MQRL_UNDEFINED, the number of bytes of message data returned is usually given by the smaller of *BufferLength* and *DataLength*, but can be *less than* this if the MQGET call completes with reason code MQRC_TRUNCATED_MSG_ACCEPTED. If this happens, the insignificant bytes in the **Buffer** parameter are set to nulls.

The following special value is defined:

MQRL_UNDEFINED
Length of returned data not defined.

On z/OS, the value returned for the *ReturnedLength* field is always MQRL_UNDEFINED.

The initial value of this field is MQRL_UNDEFINED. This field is ignored if *Version* is less than MQGMO_VERSION_3.

Segmentation (MQCHAR):

This is a flag that indicates whether further segmentation is allowed for the message retrieved. It has one of the following values:

MQSEG_INHIBITED
Segmentation not allowed.

MQSEG_ALLOWED
Segmentation allowed.

On z/OS, the queue manager always sets this field to MQSEG_INHIBITED.

This is an output field. The initial value of this field is MQSEG_INHIBITED. This field is ignored if *Version* is less than MQGMO_VERSION_2.

SegmentStatus (MQCHAR):

This is a flag that indicates whether the message retrieved is a segment of a logical message. It has one of the following values:

MQSS_NOT_A_SEGMENT
Message is not a segment.

MQSS_SEGMENT
Message is a segment, but is not the last segment of the logical message.

MQSS_LAST_SEGMENT
Message is the last segment of the logical message.

This is also the value returned if the logical message consists of only one segment.

On z/OS, the queue manager always sets this field to MQSS_NOT_A_SEGMENT.

This is an output field. The initial value of this field is MQSS_NOT_A_SEGMENT. This field is ignored if *Version* is less than MQGMO_VERSION_2.

Signal1 (MQLONG):

This is an input field that is used only in conjunction with the MQGMO_SET_SIGNAL option; it identifies a signal that is to be delivered when a message is available.

Note: The data type and usage of this field are determined by the environment; for this reason, applications that you want to port between different environments must not use signals.

- On z/OS, this field must contain the address of an Event Control Block (ECB). The ECB must be cleared by the application before the MQGET call is issued. The storage containing the ECB must not be freed until the queue is closed. The ECB is posted by the queue manager with one of the signal completion codes described. These completion codes are set in bits 2 through 31 of the ECB, the area defined in the z/OS mapping macro IHAECB as being for a user completion code.
- In all other environments, this is a reserved field; its value is not significant.

The signal completion codes are:

MQEC_MSG_ARRIVED
: A suitable message has arrived on the queue. This message has not been reserved for the caller; a second MQGET request must be issued, but another application might retrieve the message before the second request is made.

MQEC_WAIT_INTERVAL_EXPIRED
: The specified *WaitInterval* has expired without a suitable message arriving.

MQEC_WAIT_CANCELED
: The wait was canceled for an indeterminate reason (such as the queue manager terminating or the queue being disabled). Reissue the request if you want further diagnosis.

MQEC_Q_MGR_QUIESCING
: The wait was canceled because the queue manager has entered the quiescing state (MQGMO_FAIL_IF_QUIESCING was specified on the MQGET call).

MQEC_CONNECTION_QUIESCING
: The wait was canceled because the connection has entered the quiescing state (MQGMO_FAIL_IF_QUIESCING was specified on the MQGET call).

The initial value of this field is determined by the environment:

- On z/OS, the initial value is the null pointer.
- In all other environments, the initial value is `0`.

Signal2 (MQLONG):

This is an input field that is used only in conjunction with the MQGMO_SET_SIGNAL option. It is a reserved field; its value is not significant.

The initial value of this field is `0`.

StrucId (MQCHAR4):

This is the structure identifier. The value must be:

MQGMO_STRUC_ID
Identifier for get-message options structure.

For the C programming language, the constant MQGMO_STRUC_ID_ARRAY is also defined; this has the same value as MQGMO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQGMO_STRUC_ID.

Version (MQLONG):

Version is the structure version number.

The value must be one of the following:

MQGMO_VERSION_1
Version-1 get-message options structure.

This version is supported in all environments.

MQGMO_VERSION_2
Version-2 get-message options structure.

This version is supported in all environments.

MQGMO_VERSION_3
Version-3 get-message options structure.

This version is supported in all environments.

MQGMO_VERSION_4
Version-4 get-message options structure.

This version is supported in all environments.

Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQGMO_CURRENT_VERSION
Current version of get-message options structure.

This is always an input field. The initial value of this field is MQGMO_VERSION_1.

WaitInterval (MQLONG):

This is the approximate time, expressed in milliseconds, that the MQGET call waits for a suitable message to arrive (that is, a message satisfying the selection criteria specified in the **MsgDesc** parameter of the MQGET call.

Important: There is no wait, or delay, if a suitable message is available immediately.

See the *MsgId* field described in "MQMD - Message descriptor" on page 2387 for more details). If no suitable message has arrived after this time has elapsed, the call completes with MQCC_FAILED and reason code MQRC_NO_MSG_AVAILABLE.

On z/OS, the period of time that the MQGET call actually waits is affected by system loading and work-scheduling considerations, and can vary between the value specified for *WaitInterval* and approximately 250 milliseconds greater than *WaitInterval*.

WaitInterval is used in conjunction with the MQGMO_WAIT or MQGMO_SET_SIGNAL option. It is ignored if neither of these is specified. If one of these is specified, *WaitInterval* must be greater than or equal to zero, or the following special value:

MQWI_UNLIMITED
Unlimited wait interval.

The initial value of this field is 0.

Initial values and language declarations for MQGMO:

Table 206. Initial values of fields in MQGMO for MQGMO

Field name	Name of constant	Value of constant
StrucId	MQGMO_STRUC_ID	'GMO¬'
Version	MQGMO_VERSION_1	1
Options	MQGMO_NO_WAIT	0
WaitInterval	None	0
Signal1	None	Null pointer on z/OS ; 0 otherwise
Signal2	None	0
ResolvedQName	None	Null string or blanks
MatchOptions	MQMO_MATCH_MSG_ID + MQMO_MATCH_CORREL_ID	3
GroupStatus	MQGS_NOT_IN_GROUP	'⌂'
SegmentStatus	MQSS_NOT_A_SEGMENT	'⌂'
Segmentation	MQSEG_INHIBITED	'⌂'
Reserved1	None	'⌂'
MsgToken	MQMTOK_NONE	Nulls
ReturnedLength	MQRL_UNDEFINED	-1
Reserved2	None	'⌂'
MsgHandle	MQHM_NONE	0

Table 206. Initial values of fields in MQGMO for MQGMO (continued)

Field name	Name of constant	Value of constant
Notes: 1. The symbol ¬ represents a single blank character. 2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages. 3. In the C programming language, the macro variable MQGMO_DEFAULT contains the values that are listed in the table. It can be used in the following way to provide initial values for the fields in the structure: `MQGMO MyGMO = {MQGMO_DEFAULT};`		

C declaration for MQGMO:

```
typedef struct tagMQGMO MQGMO;
struct tagMQGMO {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    Options;         /* Options that control the action of */
                             MQGET */
  MQLONG    WaitInterval;    /* Wait interval */
  MQLONG    Signal1;         /* Signal */
  MQLONG    Signal2;         /* Signal identifier */
  MQCHAR48  ResolvedQName;   /* Resolved name of destination queue */
  /* Ver:1 */
  MQLONG    MatchOptions;    /* Options controlling selection */
                             criteria used for MQGET */
  MQCHAR    GroupStatus;     /* Flag indicating whether message */
                             retrieved is in a group */
  MQCHAR    SegmentStatus;   /* Flag indicating whether message */
                             retrieved is a segment of a logical */
                             message */
  MQCHAR    Segmentation;    /* Flag indicating whether further */
                             segmentation is allowed for the */
                             message retrieved */
  MQCHAR    Reserved1;       /* Reserved */
  /* Ver:2 */
  MQBYTE16  MsgToken;        /* Message token */
  MQLONG    ReturnedLength;  /* Length of message data returned */
                             (bytes) */
  /* Ver:3 */
  MQLONG    Reserved2;       /* Reserved */
  MQHMSG    MsgHandle;       /* Message handle */
  /* Ver:4 */
};
```

- On z/OS, the *Signal1* field is declared as PMQLONG.

COBOL declaration for MQGMO:

```
**   MQGMO structure
  10 MQGMO.
**    Structure identifier
   15 MQGMO-STRUCID        PIC X(4).
**    Structure version number
   15 MQGMO-VERSION        PIC S9(9) BINARY.
**    Options that control the action of MQGET
   15 MQGMO-OPTIONS        PIC S9(9) BINARY.
**    Wait interval
   15 MQGMO-WAITINTERVAL   PIC S9(9) BINARY.
**    Signal
   15 MQGMO-SIGNAL1        PIC S9(9) BINARY.
**    Signal identifier
   15 MQGMO-SIGNAL2        PIC S9(9) BINARY.
**    Resolved name of destination queue
   15 MQGMO-RESOLVEDQNAME  PIC X(48).
**    Options controlling selection criteria used for MQGET
   15 MQGMO-MATCHOPTIONS   PIC S9(9) BINARY.
**    Flag indicating whether message retrieved is in a group
   15 MQGMO-GROUPSTATUS    PIC X.
**    Flag indicating whether message retrieved is a segment of a
**    logical message
   15 MQGMO-SEGMENTSTATUS  PIC X.
**    Flag indicating whether further segmentation is allowed for the
**    message retrieved
   15 MQGMO-SEGMENTATION   PIC X.
**    Reserved
   15 MQGMO-RESERVED1      PIC X.
**    Message token
   15 MQGMO-MSGTOKEN       PIC X(16).
**    Length of message data returned (bytes)
   15 MQGMO-RETURNEDLENGTH PIC S9(9) BINARY.
**    Reserved
   15 MQGMO-RESERVED2      PIC S9(9) BINARY.
**    Message handle
   15 MQGMO-MSGHANDLE      PIC S9(18) BINARY.
```

- On z/OS, the *Signal1* field is declared as POINTER.

PL/I declaration for MQGMO:

```
dcl
 1 MQGMO based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 Options        fixed bin(31), /* Options that control the action of
                                     MQGET */
  3 WaitInterval   fixed bin(31), /* Wait interval */
  3 Signal1        fixed bin(31), /* Signal */
  3 Signal2        fixed bin(31), /* Signal identifier */
  3 ResolvedQName  char(48),      /* Resolved name of destination
                                     queue */
  3 MatchOptions   fixed bin(31), /* Options controlling selection
                                     criteria used for MQGET */
  3 GroupStatus    char(1),       /* Flag indicating whether message
                                     retrieved is in a group */
  3 SegmentStatus  char(1),       /* Flag indicating whether message
                                     retrieved is a segment of a logical
                                     message */
  3 Segmentation   char(1),       /* Flag indicating whether further
                                     segmentation is allowed for the
                                     message retrieved */
  3 Reserved1      char(1),       /* Reserved */
  3 MsgToken       char(16),      /* Message token */
  3 ReturnedLength fixed bin(31); /* Length of message data returned
```

```
                                  (bytes) */
  3 Reserved2      fixed bin(31); /* Reserved */
  3 MsgHandle      fixed bin(63); /* Message handle */
```

- On z/OS, the *Signal1* field is declared as pointer.

High Level Assembler declaration for MQGMO:

```
MQGMO                 DSECT
MQGMO_STRUCID         DS   CL4   Structure identifier
MQGMO_VERSION         DS   F     Structure version number
MQGMO_OPTIONS         DS   F     Options that control the action of
*                                MQGET
MQGMO_WAITINTERVAL    DS   F     Wait interval
MQGMO_SIGNAL1         DS   F     Signal
MQGMO_SIGNAL2         DS   F     Signal identifier
MQGMO_RESOLVEDQNAME   DS   CL48  Resolved name of destination queue
MQGMO_MATCHOPTIONS    DS   F     Options controlling selection criteria
*                                used for MQGET
MQGMO_GROUPSTATUS     DS   CL1   Flag indicating whether message
*                                retrieved is in a group
MQGMO_SEGMENTSTATUS   DS   CL1   Flag indicating whether message
*                                retrieved is a segment of a logical
*                                message
MQGMO_SEGMENTATION    DS   CL1   Flag indicating whether further
*                                segmentation is allowed for the message
*                                retrieved
MQGMO_RESERVED1       DS   CL1   Reserved
MQGMO_MSGTOKEN        DS   XL16  Message token
MQGMO_RETURNEDLENGTH  DS   F     Length of message data returned (bytes)
MQGMO_RESERVED2       DS   F     Reserved
MQGMO_MSGHANDLE       DS   D     Message handle
MQGMO_LENGTH          EQU  *-MQGMO
                      ORG  MQGMO
MQGMO_AREA            DS   CL(MQGMO_LENGTH)
```

Visual Basic declaration for MQGMO:

```
Type MQGMO
  StrucId        As String*4  'Structure identifier'
  Version        As Long      'Structure version number'
  Options        As Long      'Options that control the action of MQGET'
  WaitInterval   As Long      'Wait interval'
  Signal1        As Long      'Signal'
  Signal2        As Long      'Signal identifier'
  ResolvedQName  As String*48 'Resolved name of destination queue'
  MatchOptions   As Long      'Options controlling selection criteria'
                              'used for MQGET'
  GroupStatus    As String*1  'Flag indicating whether message'
                              'retrieved is in a group'
  SegmentStatus  As String*1  'Flag indicating whether message'
                              'retrieved is a segment of a logical'
                              'message'
  Segmentation   As String*1  'Flag indicating whether further'
                              'segmentation is allowed for the message'
                              'retrieved'
  Reserved1      As String*1  'Reserved'
  MsgToken       As MQBYTE16  'Message token'
  ReturnedLength As Long      'Length of message data returned (bytes)'
End Type
```

PROPCTL *channel options:*

Use **PROPCTL** channel attribute to control which message properties are included in a message that is sent from a Version 9.0 queue manager to a partner queue manager from an earlier version of IBM MQ.

Table 207. Channel message property attribute settings

PROPCTL	**Description**
ALL	Use this option if applications connected to the partner queue manager from an earlier version are able to process any properties placed in a message by a Version 9.0 application. All properties are sent to the partner queue manager, in addition to any name-value pairs placed in the MQRFH2. You must consider two application design issues: 1. An application connected to the partner queue manager must be able to process messages containing MQRFH2 headers generated on a Version 9.0 queue manager. 2. The application connected to the partner queue manager must process new message properties that are flagged with MQPD_SUPPORT_REQUIRED correctly. With the ALL channel option set, JMS applications can interoperate between IBM MQ Version 9.0 and an earlier version using the channel. New Version 9.0 applications using message properties can interoperate with applications from an earlier version, depending on how the earlier version application handles MQRFH2 headers.
COMPAT	Use this option to send message properties to applications connected to an earlier version partner queue manager in some cases, but not all. Message properties are only sent if two conditions are met: 1. No property must be marked as requiring message property processing. 2. At least one of the message properties must be in a "reserved" folder; see Note. With the COMPAT channel option set, JMS applications can interoperate between IBM MQ Version 9.0 and an earlier version using the channel. The channel is not available to every application using message properties, only to those applications that use the reserved folders. The rules concerning whether the message or the property is sent are: 1. If the message has properties, but none of the properties are associated with a "reserved" folder, then no message properties are sent. 2. If any message property has been created in a "reserved" property folder, all message properties associated with the message are sent. However: a. If any of the message properties are marked as support being required, MQPD_SUPPORT_REQUIRED or MQPD_SUPPORT_REQUIRED_IF_LOCAL, the whole message is rejected. It is returned, discarded, or sent to the dead letter queue according to the value of its report options. b. If no message properties are marked as support being required, an individual property might not be sent. If any of the message property descriptor fields are set to non-default values the individual property is not sent. The message is still sent. An example of a non-default property descriptor field value is MQPD_USER_CONTEXT. **Note:** The "reserved" folders names start with mcd., jms., usr., or mqext.. These folders are created for applications that use the JMS interface. In Version 9.0 any name-value pairs that are placed in these folders are treated as message properties. Message properties are sent in an MQRFH2 header, in addition to any name-value pairs placed in an MQRFH2 header. Any name-value pairs placed in an MQRFH2 header are sent as long as the message is not rejected.

Table 207. Channel message property attribute settings (continued)

PROPCTL	**Description**
NONE	Use this option to prevent any message properties being sent to applications connected to an earlier version partner queue manager. An MQRFH2 that contains name-value pairs and message properties is still sent, but only with the name-value pairs. With the NONE channel option set, a JMS message is sent as a JMSTextMessage or a JMSBytesMessage without any JMS message properties. If it is possible for an earlier version application to ignore all properties set in a Version 9.0 application, it can interoperate with it.

PROPCTL *queue options:*

Use the **PROPCTL** queue attribute to control how message properties are returned to an application that calls MQGET without setting any MQGMO message property options.

Table 208. Queue message property attribute settings

PROPCTL	**Description**
ALL	Use this option so that different applications reading a message from the same queue can process the message in different ways. • An application, migrated unchanged from an earlier version, can continue to read the MQRFH2 directly. Properties are directly accessible in the MQRFH2 header. You must modify the application to handle any new properties, and new property attributes. It is possible that the application might be affected by changes in the layout and number of MQRFH2 headers. Some folder attributes might be removed, or that IBM MQ reports an error in the layout of the MQRFH2 header that it ignored in an earlier version. • A new or changed application can use the message property MQI to query message properties, and read name-value pairs in MQRFH2 header directly. All the properties in the message are returned to the application. • If the application calls MQCRTMH to create a message handle, it must query the message properties using MQINQMP. Name-value pairs that are not message properties remain in the MQRFH2, which is stripped of any message properties. • If the application does not create a message handle, all the message properties and name-value pairs remain in the MQRFH2. ALL only has this affect if the receiving application has not set a MQGMO_PROPERTIES option, or has set it to MQGMO_PROPERTIES_AS_Q_DEF.

Table 208. Queue message property attribute settings (continued)

PROPCTL	**Description**
COMPAT	COMPAT is the default option. If GMO_PROPERTIES_* is not set, as in an unmodified application from an earlier version, COMPAT is assumed. By defaulting to the COMPAT option, an earlier version application that did not explicitly create an MQRFH2, works without change on Version 9.0. Use this option if you have written an earlier version application MQI application to read JMS messages. • The JMS properties, which are stored in an MQRFH2 header, are returned to the application in an MQRFH2 header in folders with names starting with mcd., jms., usr., or mqext. . • If the message has JMS folders, and if a Version 9.0 application adds new property folders to the message, these properties are also returned in the MQRFH2. Consequently, you must modify the application to handle any new properties, and new property attributes. It is possible that an unmodified application might be affected by changes in the layout and number of MQRFH2 headers. It might find some folder attributes are removed, or that IBM MQ finds errors in the layout of the MQRFH2 header that it ignored in an earlier version. **Note:** In this scenario, the behavior of the application is the same whether it is connected to an earlier version or Version 9.0 queue manager. If the channel **PROPCTL** attribute is set to COMPAT or ALL any new message properties are sent in the message to the earlier version partner queue manager. • If the message is not a JMS message, but contains other properties, those properties are not returned to the application in an MQRFH2 header. (The existence of specific property folders created by the IBM MQ classes for JMS indicates a JMS message. The property folders are mcd., jms., usr., or mqext.) • The option also enables earlier version applications that explicitly create an MQRFH2 to work correctly, in many cases. For example, An MQI program that creates an MQRFH2 containing JMS message properties continues to work correctly. If a message is created without JMS message properties, but with some other MQRFH2 folders, the folders are returned to the application. Only if the folders are message property folders are those specific folders are removed from the MQRFH2. Message property folders are identified by having the new folder attribute content='properties', or are folders with names listed in Defined property folder name or Ungrouped property folder name. • If the application calls MQCRTMH to create a message handle, it must query the message properties using MQINQMP. Message properties are removed from the MQRFH2 headers. Name-value pairs that are not message properties remain in the MQRFH2. • If the application calls MQCRTMH to create a message handle, it can query all message properties, regardless of whether the message has JMS folders. • If the application does not create a message handle, all the message properties and name-value pairs remain in the MQRFH2. If a message contains new user property folders, you can infer that the message was created by a new or changed Version 9.0 application. If the receiving application is to process these new properties directly in an MQRFH2, you must modify the application to use the ALL option. With the default COMPAT option set, an unmodified application continues to process the rest of the MQRFH2, without the Version 9.0 properties. The intent of the PROPCTL interface is to support old applications reading MQRFH2 folders, and new and changed applications using the message property interface. Aim for new applications to use the message property interface for all user message properties, and to avoid reading and writing MQRFH2 headers directly. COMPAT only has this affect if the receiving application has not set a MQGMO_PROPERTIES option, or has set it to MQGMO_PROPERTIES_AS_Q_DEF.

Table 208. Queue message property attribute settings (continued)

PROPCTL	**Description**
FORCE	The FORCE option places all messages properties into MQRFH2 headers. All message properties and name-value pairs in the MQRFH2 headers remain in the message. Message properties are not removed from the MQRFH2, and made available through a message handle. The effect of choosing the FORCE option is to enable a newly migrated application to read message properties from MQRFH2 headers. Suppose you have modified an application to process Version 9.0 message properties, but have also retained its ability to work directly with MQRFH2 headers, as before. You can decide when to switch the application over to using message properties by initially setting the PROPCTL queue attribute to FORCE. Set the **PROPCTL** queue attribute to another value when you are ready to start using message properties. If the new function in the application does not behave as you expected, set the **PROPCTL** option back to FORCE. FORCE only has this affect if the receiving application has not set a MQGMO_PROPERTIES option, or has set it to MQGMO_PROPERTIES_AS_Q_DEF.
NONE	Use this option so that an existing application can process a message, ignoring all message properties, and a new or changed application can query message properties. • If the application calls MQCRTMH to create a message handle, it must query the message properties using MQINQMP. Name-value pairs that are not message properties remain in the MQRFH2, which is stripped of any message properties. • If the application does not create a message handle, all the message properties are removed from the MQRFH2. Name-value pairs in the MQRFH2 headers remain in the message. NONE only has this affect if the receiving application has not set a MQGMO_PROPERTIES option, or has set it to MQGMO_PROPERTIES_AS_Q_DEF.
V6COMPAT	Use this option to receive an MQRFH2 in the same format as it was sent. If the sending application, or the queue manager, creates additional message properties, these are returned in the message handle. This option has to be set on both the sending and receiving queues, and any intervening transmission queues. It overrides any other PROPCTL options set on queue definitions in the queue name resolution path. Use this option only in exceptional circumstances. For example, if you are migrating applications from an earlier version to Version 9.0 the option is valuable because it preserves the behavior of the earlier version. The option is likely to have an impact on message throughput. It is also more difficult to administer; you need to ensure the option is set on the sender, receiver, and intervening transmission queues. V6COMPAT only has this affect if the receiving application has not set a MQGMO_PROPERTIES option, or has set it to MQGMO_PROPERTIES_AS_Q_DEF.

Related information:
PROPCTL

MQGMO message property option settings:

Use MQGMO message property options to control how message properties are returned to an application.

Table 209. MQGMO message property option settings

MQGMO Option	**Description**
MQGMO_PROPERTIES_AS_Q_DEF	Applications prior to Version 7.0, and Version 9.0 applications that read from the same queue, and do not set GMO_PROPERTIES_*, receive the message properties differently. Applications prior to Version 7.0, and Version 9.0 applications that do not create a message handle, are controlled by the queue **PROPCTL** attribute. A Version 9.0 application can choose to receive message properties in the MQRFH2, or create a message handle and query the message properties. If the application creates a message handle, properties are removed from the MQRFH2. • An unmodified application prior to Version 7.0 does not set GMO_PROPERTIES_*. Any message properties it receives are in the MQRFH2 headers. • A new or changed Version 9.0 application that does not set GMO_PROPERTIES_* or sets it to MQGMO_PROPERTIES_AS_Q_DEF can choose to query message properties. It must set MQCRTMH to create a message handle and query message properties using the MQINQMP MQI call. • If a new or changed application does not create a message handle, it behaves like an application prior to Version 7.0. It must read any message properties it receives directly from the MQRFH2 headers. • If the queue attribute **PROPCTL** is set to FORCE, no properties are returned in the message handle. All properties are returned in MQRFH2 headers. • If the queue attribute **PROPCTL** is set to NONE, or COMPAT, a Version 9.0 application that creates a message handle, receives all message properties. • If the queue attribute **PROPCTL** is set to V6COMPAT, and also set to V6COMPAT on all the queues the message was placed on between the sender and the receiver, properties set by MQSETMP are returned in the message handle, and properties and name-value pairs created in an MQRFH2 are returned in the MQRFH2. The format of an MQRFH2 sent in Version 9.0 is the same as an application prior to Version 7.0, when sent by the same application.
MQGMO_PROPERTIES_IN_HANDLE	Force an application to use message properties. Use this option to detect if a modified application fails to create message handle. The application might be trying to read message properties directly from an MQRFH2, rather than call MQINQMP.
MQGMO_NO_PROPERTIES	An application prior to Version 7.0 and a Version 9.0 application behave the same way, even if the Version 9.0 application creates a message handle. • All properties are removed. An unchanged application, prior to Version 7.0, connected to a Version 9.0 queue manager might behave differently to when it was connected to a partner queue manager from a version prior to Version 7.0. Queue manager generated properties, such as JMS properties, are removed. • Properties are removed even if a message handle is created. Name-value pairs in other MQRFH2 folders are available in the message data.

Table 209. MQGMO message property option settings (continued)

MQGMO Option	Description
MQGMO_PROPERTIES_FORCE_MQRFH2	Applications prior to Version 7.0 and Version 9.0 applications behave the same way. Properties are returned in the MQRFH2 headers, even if a message handle is created. • MQINQMP returns no message properties, even if a message handle is created. MQRC_PROPERTY_NOT_AVAILABLE is returned if a property is inquired upon.
MQGMO_PROPERTIES_COMPATIBILITY	An application prior to Version 7.0 connected to a Version 9.0 queue manager behaves the same as when it is connected to a queue manager from a version prior to Version 7.0. If the message is from a JMS client, the JMS properties are returned in the MQRFH2 headers. New or modified Version 9.0 applications, that create a message handle, behave differently. • All properties in any message property folders are returned if the message contains a mcd., jms., usr., or mqext. folder. • If the message contains property folders, but not a mcd., jms., usr., or mqext. folder, no message properties are returned in an MQRFH2. • If a new or modified Version 9.0 application creates a message handle, query message properties using the MQINQMP MQI call. All message properties are removed from the MQRFH2. • If a new or modified Version 9.0 application creates a message handle, all properties in the message can be queried. Even if the message does not contain a mcd., jms., usr., or mqext. folder, all message properties are queriable.

Related information:
PROPCTL
2471 (09A7) (RC2471): MQRC_PROPERTY_NOT_AVAILABLE

MQIIH - IMS information header:

The following table summarizes the fields in the structure.

Table 210. Fields in MQIIH

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
StrucLength	Length of MQIIH structure	StrucLength
Encoding	Reserved	Encoding
CodedCharSetId	Reserved	CodedCharSetId
Format	MQ format name of data that follows MQIIH	Format
Flags	Flags	Flags
LTermOverride	Logical terminal override	LTermOverride
MFSMapName	Message format services map name	MFSMapName
ReplyToFormat	MQ format name of reply message	ReplyToFormat
Authenticator	RACF™ password or passticket	Authenticator
TranInstanceId	Transaction instance identifier	TranInstanceId
TranState	Transaction state	TranState

Table 210. Fields in MQIIH (continued)

Field	Description	Topic
CommitMode	Commit mode	CommitMode
SecurityScope	Security scope	SecurityScope
Reserved	Reserved	Reserved

Overview for MQIIH:

The MQIIH structure describes the header information for a message sent to IMS across the IMS bridge. For any IBM MQ supported platform you can create and transmit a message that includes the MQIIH structure, but only an IBM MQ for z/OS queue manager can use the IMS bridge. Therefore, for the message to get to IMS from a non-z/OS queue manager, your queue manager network must include at least one z/OS queue manager through which the message can be routed.

Availability: All IBM MQ systems and IBM MQ clients.

Purpose: The MQIIH structure describes the information that must be present at the start of a message sent to the IMS bridge through IBM MQ for z/OS.

Format name: MQFMT_IMS.

Character set and encoding: Special conditions apply to the character set and encoding used for the MQIIH structure and application message data:

- Applications that connect to the queue manager that owns the IMS bridge queue must provide an MQIIH structure that is in the character set and encoding of the queue manager. This is because data conversion of the MQIIH structure is not performed in this case.
- Applications that connect to other queue managers can provide an MQIIH structure that is in any of the supported character sets and encodings; the receiving message channel agent connected to the queue manager that owns the IMS bridge queue converts the MQIIH.
- The application message data following the MQIIH structure must be in the same character set and encoding as the MQIIH structure. Do not use the *CodedCharSetId* and *Encoding* fields in the MQIIH structure to specify the character set and encoding of the application message data.

  You must provide a data-conversion exit to convert the application message data if the data is not one of the built-in formats supported by the queue manager.

Fields for MQIIH:

The MQIIH structure contains the following fields; the fields are described in **alphabetical order**:

Authenticator (MQCHAR8):

This is the RACF password or PassTicket. It is optional; if specified, it is used with the user ID in the MQMD security context to build a UTOKEN that is sent to IMS to provide a security context. If it is not specified, the user ID is used without verification. This depends on the setting of the RACF switches, which might require an authenticator to be present.

This is ignored if the first byte is blank or null. The following special value can be used:

MQIAUT_NONE
No authentication.

For the C programming language, the constant MQIAUT_NONE_ARRAY is also defined; this has the same value as MQIAUT_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_AUTHENTICATOR_LENGTH. The initial value of this field is MQIAUT_NONE.

CodedCharSetId (MQLONG):

This is a reserved field; its value is not significant. The initial value of this field is 0.

The Character Set Id for supported structures which follow a MQIIH structure is the same as that of the MQIIH structure itself and taken from any preceding MQ header.

CommitMode (MQCHAR):

This is the IMS commit mode. See the *OTMA Reference* for more information about IMS commit modes. The value must be one of the following:

MQICM_COMMIT_THEN_SEND
Commit then send.

This mode implies double queuing of output, but shorter region occupancy times. Fast-path and conversational transactions cannot run with this mode.

MQICM_SEND_THEN_COMMIT
Send then commit.

Any IMS transaction initiated as a result of a commit mode of MQICM_SEND_THEN_COMMIT runs in RESPONSE mode regardless of how the transaction is defined in the IMS system definition (MSGTYPE parameter in the TRANSACT macro). This also applies to transactions initiated by means of a transaction switch.

The initial value of this field is MQICM_COMMIT_THEN_SEND.

Encoding (MQLONG):

This is a reserved field; its value is not significant. The initial value of this field is 0.

The Encoding for supported structures which follow a MQIIH structure is the same as that of the MQIIH structure itself and taken from any preceding MQ header.

Flags (MQLONG):

The flags value must be:

MQIIH_NONE
No flags.

MQIIH_PASS_EXPIRATION
The reply message contains:

- The same expiry report options as the request message
- The remaining expiry time from the request message with no adjustment made for the bridge's processing time

If this value is not set, the expiry time is set to *unlimited*.

MQIIH_REPLY_FORMAT_NONE
Sets the MQIIH.Format field of the reply to MQFMT_NONE.

MQIIH_IGNORE_PURG
Sets the TMAMIPRG indicator in the OTMA prefix, which requests that OTMA ignores PURG calls on the TP PCB for CM0 transactions.

MQIIH_CM0_REQUEST_RESPONSE
For Commit Mode 0 (CM0) transactions this flag sets the TMAMHRSP indicator in the OTMA prefix. Setting this indicator requests that OTMA/IMS generate a DFS2082 RESPONSE MODE TRANSACTION TERMINATED WITHOUT REPLY message when the original IMS application program does not reply to the IOPCB nor message switch to another transaction.

The initial value of this field is MQIIH_NONE.

Format (MQCHAR8):

This specifies the MQ format name of the data that follows the MQIIH structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

LTermOverride (MQCHAR8):

The logical terminal override, placed in the IO PCB field. It is optional; if it is not specified, the TPIPE name is used. It is ignored if the first byte is blank, or null.

The length of this field is given by MQ_LTERM_OVERRIDE_LENGTH. The initial value of this field is 8 blank characters.

MFSMapName (MQCHAR8):

The message format services map name, placed in the IO PCB field. It is optional. On input it represents the MID, on output it represents the MOD. It is ignored if the first byte is blank or null.

The length of this field is given by MQ_MFS_MAP_NAME_LENGTH. The initial value of this field is 8 blank characters.

ReplyToFormat (MQCHAR8):

This is the MQ format name of the reply message that is sent in response to the current message. The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

To convert the data in the reply message using MQGMO_CONVERT, specify either MQIIH.replyToFormat=MQFMT_STRING or MQIIH.replyToFormat=MQFMT_IMS_VAR_STRING. For an explanation of the use of these fields, see "Format (MQCHAR8)" on page 2404.

If the default value (MQIIH.replyToFormat=MQFMT_NONE) is used on the request message and the reply message is retrieved using MQGMO_CONVERT then no data conversion is performed.

Reserved (MQCHAR):

This is a reserved field; it must be blank.

SecurityScope (MQCHAR):

This indicates the IMS security processing required. The following values are defined:

MQISS_CHECK
Check security scope: an ACEE is built in the control region, but not in the dependent region.

MQISS_FULL
Full security scope: a cached ACEE is built in the control region and a non-cached ACEE is built in the dependent region. If you use MQISS_FULL, ensure that the user ID for which the ACEE is built has access to the resources used in the dependent region.

If neither MQISS_CHECK nor MQISS_FULL is specified for this field, MQISS_CHECK is assumed.

The initial value of this field is MQISS_CHECK.

StrucId (MQCHAR4):

This is the structure identifier. The value must be:

MQIIH_STRUC_ID
Identifier for the IMS information header structure.

For the C programming language, the constant MQIIH_STRUC_ID_ARRAY is also defined; this has the same value as MQIIH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQIIH_STRUC_ID.

StrucLength (MQLONG):

This is the length of MQIIH structure. The value must be:

MQIIH_LENGTH_1
Length of the IMS information header structure.

The initial value of this field is MQIIH_LENGTH_1.

TranInstanceId (MQBYTE16):

This is the transaction instance identifier. This field is used by output messages from IMS, so is ignored on first input. If you set *TranState* to MQITS_IN_CONVERSATION, this must be provided in the next input, and all subsequent inputs, to enable IMS to correlate the messages to the correct conversation. You can use the following special value:

MQITII_NONE
No transaction instance identifier.

For the C programming language, the constant MQITII_NONE_ARRAY is also defined; this has the same value as MQITII_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_TRAN_INSTANCE_ID_LENGTH. The initial value of this field is MQITII_NONE.

TranState (MQCHAR):

This indicates the IMS conversation state. This is ignored on first input because no conversation exists. On subsequent inputs it indicates whether a conversation is active or not. On output it is set by IMS. The value must be one of the following:

MQITS_IN_CONVERSATION
In conversation.

MQITS_NOT_IN_CONVERSATION
Not in conversation.

MQITS_ARCHITECTED
Return transaction state data in architected form.

This value is used only with the IMS `/DISPLAY TRAN` command. It returns the transaction state data in the IMS architected form instead of character form. z/OS For more information, see Writing IMS transaction programs through IBM MQ.

The initial value of this field is MQITS_NOT_IN_CONVERSATION.

Version (MQLONG):

This is the structure version number. The value must be:

MQIIH_VERSION_1
Version number for IMS information header structure.

The following constant specifies the version number of the current version:

MQIIH_CURRENT_VERSION
Current version of IMS information header structure.

The initial value of this field is MQIIH_VERSION_1.

Initial values and language declarations for MQIIH:

Table 211. Initial values of fields in MQIIH for MQIIH

Field name	Name of constant	Value of constant
StrucId	MQIIH_STRUC_ID	'IIH?'
Version	MQIIH_VERSION_1	1
StrucLength	MQIIH_LENGTH_1	84
Encoding	None	0
CodedCharSetId	None	0
Format	MQFMT_NONE	Blanks
Flags	MQIIH_NONE	0
LTermOverride	None	Blanks
MFSMapName	None	Blanks
ReplyToFormat	MQFMT_NONE	Blanks
Authenticator	MQIAUT_NONE	Blanks
TranInstanceId	MQITII_NONE	Nulls
TranState	MQITS_NOT_IN_CONVERSATION	'?'
CommitMode	MQICM_COMMIT_THEN_SEND	'0'
SecurityScope	MQISS_CHECK	'C'
Reserved	None	'?'

Notes:

1. The symbol ? represents a single blank character.
2. In the C programming language, the macro variable MQIIH_DEFAULT contains the values that are listed in the table. It can be used in the following way to provide initial values for the fields in the structure:

```
MQIIH MyIIH = {MQIIH_DEFAULT};
```

C declaration for MQIIH:

```
typedef struct tagMQIIH MQIIH;
struct tagMQIIH {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    StrucLength;     /* Length of MQIIH structure */
  MQLONG    Encoding;        /* Reserved */
  MQLONG    CodedCharSetId;  /* Reserved */
  MQCHAR8   Format;          /* MQ format name of data that follows
                                MQIIH */
  MQLONG    Flags;           /* Flags */
  MQCHAR8   LTermOverride;   /* Logical terminal override */
  MQCHAR8   MFSMapName;      /* Message format services map name */
  MQCHAR8   ReplyToFormat;   /* MQ format name of reply message */
  MQCHAR8   Authenticator;   /* RACF password or passticket */
  MQBYTE16  TranInstanceId;  /* Transaction instance identifier */
  MQCHAR    TranState;       /* Transaction state */
  MQCHAR    CommitMode;      /* Commit mode */
  MQCHAR    SecurityScope;   /* Security scope */
  MQCHAR    Reserved;        /* Reserved */
};
```

COBOL declaration for MQIIH:

```
**   MQIIH structure
  10 MQIIH.
**    Structure identifier
   15 MQIIH-STRUCID        PIC X(4).
**    Structure version number
   15 MQIIH-VERSION        PIC S9(9) BINARY.
**    Length of MQIIH structure
   15 MQIIH-STRUCLENGTH    PIC S9(9) BINARY.
**    Reserved
   15 MQIIH-ENCODING       PIC S9(9) BINARY.
**    Reserved
   15 MQIIH-CODEDCHARSETID PIC S9(9) BINARY.
**    MQ format name of data that follows MQIIH
   15 MQIIH-FORMAT         PIC X(8).
**    Flags
   15 MQIIH-FLAGS          PIC S9(9) BINARY.
**    Logical terminal override
   15 MQIIH-LTERMOVERRIDE  PIC X(8).
**    Message format services map name
   15 MQIIH-MFSMAPNAME     PIC X(8).
**    MQ format name of reply message
   15 MQIIH-REPLYTOFORMAT  PIC X(8).
**    RACF password or passticket
   15 MQIIH-AUTHENTICATOR  PIC X(8).
**    Transaction instance identifier
   15 MQIIH-TRANINSTANCEID PIC X(16).
**    Transaction state
   15 MQIIH-TRANSTATE      PIC X.
**    Commit mode
   15 MQIIH-COMMITMODE     PIC X.
**    Security scope
   15 MQIIH-SECURITYSCOPE  PIC X.
**    Reserved
   15 MQIIH-RESERVED       PIC X.
```

PL/I declaration for MQIIH:

```
dcl
 1 MQIIH based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 StrucLength    fixed bin(31), /* Length of MQIIH structure */
  3 Encoding       fixed bin(31), /* Reserved */
  3 CodedCharSetId fixed bin(31), /* Reserved */
  3 Format         char(8),       /* MQ format name of data that follows
                                     MQIIH */
  3 Flags          fixed bin(31), /* Flags */
  3 LTermOverride  char(8),       /* Logical terminal override */
  3 MFSMapName     char(8),       /* Message format services map name */
  3 ReplyToFormat  char(8),       /* MQ format name of reply message */
  3 Authenticator  char(8),       /* RACF password or passticket */
  3 TranInstanceId char(16),      /* Transaction instance identifier */
  3 TranState      char(1),       /* Transaction state */
  3 CommitMode     char(1),       /* Commit mode */
  3 SecurityScope  char(1),       /* Security scope */
  3 Reserved       char(1);       /* Reserved */
```

High Level Assembler declaration for MQIIH:

```
MQIIH                 DSECT
MQIIH_STRUCID         DS   CL4   Structure identifier
MQIIH_VERSION         DS   F     Structure version number
MQIIH_STRUCLENGTH     DS   F     Length of MQIIH structure
MQIIH_ENCODING        DS   F     Reserved
MQIIH_CODEDCHARSETID  DS   F     Reserved
MQIIH_FORMAT          DS   CL8   MQ format name of data that follows
*                                MQIIH
MQIIH_FLAGS           DS   F     Flags
MQIIH_LTERMOVERRIDE   DS   CL8   Logical terminal override
MQIIH_MFSMAPNAME      DS   CL8   Message format services map name
MQIIH_REPLYTOFORMAT   DS   CL8   MQ format name of reply message
MQIIH_AUTHENTICATOR   DS   CL8   RACF password or passticket
MQIIH_TRANINSTANCEID  DS   XL16  Transaction instance identifier
MQIIH_TRANSTATE       DS   CL1   Transaction state
MQIIH_COMMITMODE      DS   CL1   Commit mode
MQIIH_SECURITYSCOPE   DS   CL1   Security scope
MQIIH_RESERVED        DS   CL1   Reserved
*
MQIIH_LENGTH          EQU  *-MQIIH
                      ORG  MQIIH
MQIIH_AREA            DS   CL(MQIIH_LENGTH)
```

Visual Basic declaration for MQIIH:

```
Type MQIIH
  StrucId        As String*4 'Structure identifier'
  Version        As Long     'Structure version number'
  StrucLength    As Long     'Length of MQIIH structure'
  Encoding       As Long     'Reserved'
  CodedCharSetId As Long     'Reserved'
  Format         As String*8 'MQ format name of data that follows MQIIH'
  Flags          As Long     'Flags'
  LTermOverride  As String*8 'Logical terminal override'
  MFSMapName     As String*8 'Message format services map name'
  ReplyToFormat  As String*8 'MQ format name of reply message'
  Authenticator  As String*8 'RACF password or passticket'
  TranInstanceId As MQBYTE16 'Transaction instance identifier'
  TranState      As String*1 'Transaction state'
  CommitMode     As String*1 'Commit mode'
  SecurityScope  As String*1 'Security scope'
  Reserved       As String*1 'Reserved'
End Type
```

MQIMPO - Inquire message property options:

The following table summarizes the fields in the structure. MQIMPO structure - inquire message property options

Table 212. Fields in MQIMPO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options controlling the action of MQINQMP	Options
RequestedEncoding	Encoding into which the inquired property is to be converted	RequestedEncoding
RequestedCCSID	Character set of the inquired property	RequestedCCSID
ReturnedEncoding	Encoding of the returned value	ReturnedEncoding
ReturnedCCSID	Character set of returned value	ReturnedCCSID
Reserved1	Reserved field	ReturnedCCSID
ReturnedName	Name of the inquired property	ReturnedName
TypeString	String representation of the data type of the property	TypeString

Overview for MQIMPO:

The inquire message properties options structure.

Availability: All IBM MQ systems and IBM MQ clients.

Purpose: The MQIMPO structure allows applications to specify options that control how properties of messages are inquired. The structure is an input parameter on the MQINQMP call.

Character set and encoding: Data in MQIMPO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields for MQIMPO:

Inquire message property options structure - fields

The MQIMPO structure contains the following fields; the fields are described in **alphabetical order**:

Options (MQLONG):

Inquire message property options structure - Options field

The following options control the action of MQINQMP. You can specify one or more of these options. To specify more than one option, either add the values together (do not add the same constant more than once), or combine the values using the bitwise OR operation (if the programming language supports bit operations).

Combinations of options that are not valid are noted; all other combinations are valid.

Value data options: The following options relate to the processing of the value data when the property is retrieved from the message.

MQIMPO_CONVERT_VALUE
This option requests that the value of the property be converted to conform to the *RequestedCCSID* and *RequestedEncoding* values specified before the MQINQMP call returns the property value in the *Value* area.

- If conversion is successful, the *ReturnedCCSID* and *ReturnedEncoding* fields are set to the same as *RequestedCCSID* and *RequestedEncoding* on return from the MQINQMP call.
- If conversion fails, but the MQINQMP call otherwise completes without error, the property value is returned unconverted.

  If the property is a string, the *ReturnedCCSID* and *ReturnedEncoding* fields are set to the character set and encoding of the unconverted string. The completion code is MQCC_WARNING in this case, with reason code MQRC_PROP_VALUE_NOT_CONVERTED. The property cursor is advanced to the returned property.

If the property value expands during conversion, and exceeds the size of the **Value** parameter, the value is returned unconverted, with completion code MQCC_FAILED; the reason code is set to MQRC_PROPERTY_VALUE_TOO_BIG.

The **DataLength** parameter of the MQINQMP call returns the length that the property value would have converted to, in order to allow the application to determine the size of the buffer required to accommodate the converted property value. The property cursor is unchanged.

This option also requests that:

- If the property name contains a wildcard, and
- The *ReturnedName* field is initialized with an address or offset for the returned name,

then the returned name is converted to conform to the *RequestedCCSID* and *RequestedEncoding* values.

- If conversion is successful, the *VSCCSID* field of *ReturnedName* and the encoding of the returned name are set to the input value of *RequestedCCSID* and *RequestedEncoding*.
- If conversion fails, but the MQINQMP call otherwise completes without error or warning, the returned name is unconverted. The completion code is MQCC_WARNING in this case, with reason code MQRC_PROP_NAME_NOT_CONVERTED.

  The property cursor is advanced to the returned property. MQRC_PROP_VALUE_NOT_CONVERTED is returned if both the value and the name are not converted.

If the returned name expands during conversion, and exceeds the size of the *VSBufsize* field of the *RequestedName*, the returned string is left unconverted, with completion code MQCC_FAILED and the reason code is set to MQRC_PROPERTY_NAME_TOO_BIG.

The *VSLength* field of the MQCHARV structure returns the length that the property value would have converted to, in order to allow the application to determine the size of the buffer required to accommodate the converted property value. The property cursor is unchanged.

MQIMPO_CONVERT_TYPE

This option requests that the value of the property be converted from its current data type, into the data type specified on the **Type** parameter of the MQINQMP call.

- If conversion is successful, the **Type** parameter is unchanged on return of the MQINQMP call.
- If conversion fails, but the MQINQMP call otherwise completes without error, the call fails with reason MQRC_PROP_CONV_NOT_SUPPORTED. The property cursor is unchanged.

If the conversion of the data type causes the value to expand during conversion, and the converted value exceeds the size of the **Value** parameter, the value is returned unconverted, with completion code MQCC_FAILED and the reason code is set to MQRC_PROPERTY_VALUE_TOO_BIG.

The **DataLength** parameter of the MQINQMP call returns the length that the property value would have converted to, in order to allow the application to determine the size of the buffer required to accommodate the converted property value. The property cursor is unchanged.

If the value of the **Type** parameter of the MQINQMP call is not valid, the call fails with reason MQRC_PROPERTY_TYPE_ERROR.

If the requested data type conversion is not supported, the call fails with reason MQRC_PROP_CONV_NOT_SUPPORTED. The following data type conversions are supported:

Property data type	Supported target data types
MQTYPE_BOOLEAN	MQTYPE_STRING, MQTYPE_INT8, MQTYPE_INT16, MQTYPE_INT32, MQTYPE_INT64
MQTYPE_BYTE_STRING	MQTYPE_STRING
MQTYPE_INT8	MQTYPE_STRING, MQTYPE_INT16, MQTYPE_INT32, MQTYPE_INT64
MQTYPE_INT16	MQTYPE_STRING, MQTYPE_INT32, MQTYPE_INT64
MQTYPE_INT32	MQTYPE_STRING, MQTYPE_INT64
MQTYPE_INT64	MQTYPE_STRING
MQTYPE_FLOAT32	MQTYPE_STRING, MQTYPE_FLOAT64
MQTYPE_FLOAT64	MQTYPE_STRING
MQTYPE_STRING	MQTYPE_BOOLEAN, MQTYPE_INT8, MQTYPE_INT16, MQTYPE_INT32, MQTYPE_INT64, MQTYPE_FLOAT32, MQTYPE_FLOAT64
MQTYPE_NULL	None

The general rules governing the supported conversions are as follows:

- Numeric property values can be converted from one data type to another, provided that no data is lost during the conversion.

  For example, the value of a property with data type MQTYPE_INT32 can be converted into a value with data type MQTYPE_INT64, but cannot be converted into a value with data type MQTYPE_INT16.
- A property value of any data type can be converted into a string.
- A string property value can be converted to any other data type provided the string is formatted correctly for the conversion. If an application attempts to convert a string property value that is not formatted correctly, IBM MQ returns reason code MQRC_PROP_NUMBER_FORMAT_ERROR.
- If an application attempts a conversion that is not supported, IBM MQ returns reason code MQRC_PROP_CONV_NOT_SUPPORTED.

The specific rules for converting a property value from one data type to another are as follows:

- When converting an MQTYPE_BOOLEAN property value to a string, the value TRUE is converted to the string "TRUE", and the value false is converted to the string "FALSE".
- When converting an MQTYPE_BOOLEAN property value to a numeric data type, the value TRUE is converted to one, and the value FALSE is converted to zero.
- When converting a string property value to an MQTYPE_BOOLEAN value, the string "TRUE" , or "1" , is converted to TRUE, and the string "FALSE", or "0", is converted to FALSE.

  Note that the terms "TRUE" and "FALSE" are not case sensitive.

  Any other string cannot be converted; IBM MQ returns reason code MQRC_PROP_NUMBER_FORMAT_ERROR.
- When converting a string property value to a value with data type MQTYPE_INT8, MQTYPE_INT16, MQTYPE_INT32 or MQTYPE_INT64, the string must have the following format:

  ```
  [blanks][sign]digits
  ```

  The meanings of the components of the string are as follows:

  blanks Optional leading blank characters

  sign An optional plus sign (+) or minus sign (-) character.

  digits A contiguous sequence of digit characters (0-9). At least one digit character must be present.

  After the sequence of digit characters, the string can contain other characters that are not digit characters, but the conversion stops as soon as the first of these characters is reached. The string is assumed to represent a decimal integer.

  IBM MQ returns reason code MQRC_PROP_NUMBER_FORMAT_ERROR if the string is not formatted correctly.
- When converting a string property value to a value with data type MQTYPE_FLOAT32 or MQTYPE_FLOAT64, the string must have the following format:

  ```
  [blanks][sign]digits[.digits][e_char[e_sign]e_digits]
  ```

  The meanings of the components of the string are as follows:

  blanks Optional leading blank characters

  sign An optional plus sign (+) or minus sign (-) character.

  digits A contiguous sequence of digit characters (0-9). At least one digit character must be present.

  e_char An exponent character, which is either "E" or "e".

  e_sign An optional plus sign (+) or minus sign (-) character for the exponent.

  e_digits
  A contiguous sequence of digit characters (0-9) for the exponent. At least one digit character must be present if the string contains an exponent character.

  After the sequence of digit characters, or the optional characters representing an exponent, the string can contain other characters that are not digit characters, but the conversion stops as soon as the first of these characters is reached. The string is assumed to represent a decimal floating point number with an exponent that is a power of 10.

  IBM MQ returns reason code MQRC_PROP_NUMBER_FORMAT_ERROR if the string is not formatted correctly.
- When converting a numeric property value to a string, the value is converted to the string representation of the value as a decimal number, not the string containing the ASCII character for that value. For example, the integer 65 is converted to the string "65", not the string "A".
- When converting a byte string property value to a string, each byte is converted to the two hexadecimal characters that represent the byte. For example, the byte array {0xF1, 0x12, 0x00, 0xFF} is converted to the string "F11200FF".

MQIMPO_QUERY_LENGTH
Query the type and length of the property value. The length is returned in the **DataLength** parameter of the MQINQMP call. The property value is not returned.

If a **ReturnedName** buffer is specified, the *VSLength* field of the MQCHARV structure is filled in with the length of the property name. The property name is not returned.

Iteration options: The following options relate to iterating over properties, using a name with a wildcard character

MQIMPO_INQ_FIRST
Inquire on the first property that matches the specified name. After this call, a cursor is established on the property that is returned.

This is the default value.

The MQIMPO_INQ_PROP_UNDER_CURSOR option can subsequently be used with an MQINQMP call, if required, to inquire on the same property again.

Note that there is only one property cursor; therefore, if the property name, specified in the MQINQMP call, changes the cursor is reset.

This option is not valid with either of the following options:

MQIMPO_INQ_NEXT

MQIMPO_INQ_PROP_UNDER_CURSOR

MQIMPO_INQ_NEXT
Inquires on the next property that matches the specified name, continuing the search from the property cursor. The cursor is advanced to the property that is returned.

If this is the first MQINQMP call for the specified name, then the first property that matches the specified name is returned.

The MQIMPO_INQ_PROP_UNDER_CURSOR option can subsequently be used with an MQINQMP call if required, to inquire on the same property again.

If the property under the cursor has been deleted, MQINQMP returns the next matching property following the one that has been deleted.

If a property is added that matches the wildcard, while an iteration is in progress, the property might or might not be returned during the completion of the iteration. The property is returned once the iteration restarts using MQIMPO_INQ_FIRST.

A property matching the wildcard that was deleted, while the iteration was in progress, is not returned subsequent to its deletion.

This option is not valid with either of the following options:

MQIMPO_INQ_FIRST

MQIMPO_INQ_PROP_UNDER_CURSOR

MQIMPO_INQ_PROP_UNDER_CURSOR
Retrieve the value of the property pointed to by the property cursor. The property pointed to by the property cursor is the one that was last inquired, using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused, when the message handle is specified in the *MsgHandle* field of the MQGMO on an MQGET call, or when the message handle is specified in *OriginalMsgHandle* or *NewMsgHandle* fields of the MQPMO structure on an MQPUT call.

If this option is used when the property cursor has not yet been established, or if the property pointed to by the property cursor has been deleted, the call fails with completion code MQCC_FAILED and reason MQRC_PROPERTY_NOT_AVAILABLE.

This option is not valid with either of the following options:

MQIMPO_INQ_FIRST

MQIMPO_INQ_NEXT

If none of the options previously described is required, the following option can be used:

MQIMPO_NONE

Use this value to indicate that no other options have been specified; all options assume their default values.

MQIMPO_NONE aids program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

This is always an input field. The initial value of this field is MQIMPO_INQ_FIRST.

RequestedCCSID (MQLONG):

Inquire message property options structure - RequestedCCSID field

The character set that the inquired property value is to be converted into if the value is a character string. This is also the character set into which the *ReturnedName* is to be converted when MQIMPO_CONVERT_VALUE or MQIMPO_CONVERT_TYPE is specified.

The initial value of this field is MQCCSI_APPL.

RequestedEncoding (MQLONG):

Inquire message property options structure - RequestedEncoding field

This is the encoding into which the inquired property value is to be converted when MQIMPO_CONVERT_VALUE or MQIMPO_CONVERT_TYPE is specified.

The initial value of this field is MQENC_NATIVE.

Reserved1 (MQCHAR):

This is a reserved field. The initial value of this field is a blank character (4 byte field).

ReturnedCCSID (MQLONG):

Inquire message property options structure - ReturnedCCSID field

On output, this is the character set of the value returned if the **Type** parameter of the MQINQMP call is MQTYPE_STRING.

If the MQIMPO_CONVERT_VALUE option is specified and conversion was successful, the *ReturnedCCSID* field, on return, is the same value as the value passed in.

The initial value of this field is zero.

ReturnedEncoding (MQLONG):

Inquire message property options structure - ReturnedEncoding field

On output, this is the encoding of the value returned.

If the MQIMPO_CONVERT_VALUE option is specified and conversion was successful, the *ReturnedEncoding* field, on return, is the same value as the value passed in.

The initial value of this field is MQENC_NATIVE.

ReturnedName (MQCHARV):

Inquire message property options structure - ReturnedName field

The actual name of the inquired property.

On input a string buffer can be passed in using the *VSPtr* or *VSOffset* field of the MQCHARV structure. The length of the string buffer is specified using the *VSBufsize* field of the MQCHARV structure.

On return from the MQINQMP call, the string buffer is completed with the name of the property that was inquired, provided the string buffer was long enough to fully contain the name. The *VSLength* field of the MQCHARV structure is filled in with the length of the property name. The *VSCCSID* field of the MQCHARV structure is filled in to indicate the character set of the returned name, whether or not conversion of the name failed.

This is an input/output field. The initial value of this field is MQCHARV_DEFAULT.

StrucId (MQCHAR4):

Inquire message property options structure - StrucId field

This is the structure identifier. The value must be:

MQIMPO_STRUC_ID
Identifier for inquire message property options structure.

For the C programming language, the constant MQIMPO_STRUC_ID_ARRAY is also defined; this has the same value as MQIMPO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQIMPO_STRUC_ID.

TypeString (MQCHAR8):

Inquire message property options structure - TypeString field

A string representation of the data type of the property.

If the property was specified in an MQRFH2 header and the MQRFH2 `dt` attribute is not recognized, this field can be used to determine the data type of the property. *TypeString* is returned in coded character set 1208 (UTF-8), and is the first eight bytes of the value of the `dt` attribute of the property that failed to be recognized

This is always an output field. The initial value of this field is the null string in the C programming language, and 8 blank characters in other programming languages.

Version (MQLONG):

Inquire message property options structure - Version field

This is the structure version number. The value must be:

MQIMPO_VERSION_1
Version number for inquire message property options structure.

The following constant specifies the version number of the current version:

MQIMPO_CURRENT_VERSION
Current version of inquire message property options structure.

This is always an input field. The initial value of this field is MQIMPO_VERSION_1.

Initial values and language declarations for MQIMPO:

Inquire message property options structure - Initial values

Table 213. Initial values of fields in MQIPMO

Field name	Name of constant	Value of constant
StrucId	MQIMPO_STRUC_ID	'IMPO'
Version	MQIMPO_VERSION_1	1
Options	MQIMPO_INQ_FIRST	
RequestedEncoding	MQENC_NATIVE	
RequestedCCSID	MQCCSI_APPL	
ReturnedEncoding	MQENC_NATIVE	
ReturnedCCSID	0	
Reserved1	0	
ReturnedName	MQCHARV_DEFAULT	
TypeString	Null string or blanks	

Notes:

1. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
2. In the C programming language, the macro variable MQIMPO_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure:

```
MQIMPO MyIMPO = {MQIMPO_DEFAULT};
```

C declaration for MQIMPO:

Inquire message property options structure - C language declaration

```
typedef struct tagMQIMPO MQIMPO;
struct tagMQIMPO {
  MQCHAR4  StrucId;            /* Structure identifier */
  MQLONG   Version;            /* Structure version number */
  MQLONG   Options;            /* Options that control the action of
                                  MQINQMP */
  MQLONG   RequestedEncoding;  /* Requested encoding of Value */
  MQLONG   RequestedCCSID;     /* Requested character set identifier
                                  of Value */
  MQLONG   ReturnedEncoding;   /* Returned encoding of Value */
  MQLONG   ReturnedCCSID;      /* Returned character set identifier
                                  of Value */
  MQCHAR   Reserved1           /* Reserved field */
  MQCHARV  ReturnedName;       /* Returned property name */
  MQCHAR8  TypeString;         /* Property data type as a string */
};
```

COBOL declaration for MQIMPO:

Inquire message property options structure - COBOL language declaration

```
**   MQIMPO structure
  10 MQIMPO.
**    Structure identifier
   15 MQIMPO-STRUCID                PIC X(4).
**    Structure version number
   15 MQIMPO-VERSION                PIC S9(9) BINARY.
**    Options that control the action of MQINQMP
   15 MQIMPO-OPTIONS                PIC S9(9) BINARY.
**    Requested encoding of VALUE
   15 MQIMPO-REQUESTEDENCODING      PIC S9(9) BINARY.
**    Requested character set identifier of VALUE
   15 MQIMPO-REQUESTEDCCSID         PIC S9(9) BINARY.
**    Returned encoding of VALUE
   15 MQIMPO-RETURNEDENCODING       PIC S9(9) BINARY.
**    Returned character set identifier of VALUE
   15 MQIMPO-RETURNEDCCSID          PIC S9(9) BINARY.
**    Reserved field
   15 MQIMPO-RESERVED1
**    Returned property name
   15 MQIMPO-RETURNEDNAME.
**     Address of variable length string
    20 MQIMPO-RETURNEDNAME-VSPTR    POINTER.
**     Offset of variable length string
    20 MQIMPO-RETURNEDNAME-VSOFFSET PIC S9(9) BINARY.
**     CCSID of variable length string
    20 MQIMPO-RETURNEDNAME-VSCCSID  PIC S9(9) BINARY.
**    Property data type as string
   15 MQIMPO-TYPESTRING             PIC S9(9) BINARY.
```

PL/I declaration for MQIMPO:

Inquire message property options structure - PL/I language declaration

```
dcl
 1 MQIMPO based,
  3 StrucId            char(4),       /* Structure identifier */
  3 Version            fixed bin(31), /* Structure version number */
  3 Options            fixed bin(31), /* Options that control the
                                         action of MQINQMP */
  3 RequestedEncoding  fixed bin(31), /* Requested encoding of
                                         Value */
  3 RequestedCCSID     fixed bin(31), /* Requested character set
                                         identifier of Value */
  3 ReturnedEncoding   fixed bin(31), /* Returned encoding of
                                         Value */
  3 ReturnedCCSID      fixed bin(31), /* Returned character set
                                         identifier of Value */
  3 Reserved1          fixed bin(31), /* Reserved field */
  3 ReturnedName,                     /* Returned property name */
   5 ReturnedName_VSPtr      pointer,       /* Address of returned
                                               name */
   5 5 ReturnedName_VSOffset fixed bin(31), /* Offset of returned
                                               name */
    5 5 ReturnedName_VSCCSID  fixed bin(31), /* CCSID of returned
                                               name */
  3 TypeString         char(8);       /* Property data type as
                                         string */
```

High Level Assembler declaration for MQIMPO:

Inquire message property options structure - Assembler language declaration

```
MQIMPO                      DSECT
MQIMPO_STRUCID              DS   CL4  Structure identifier
MQIMPO_VERSION              DS   F    Structure version number
MQIMPO_OPTIONS              DS   F    Options that control the
*                                     action of MQINQMP
MQIMPO_REQUESTEDENCODING    DS   F    Requested encoding of VALUE
MQIMPO_REQUESTEDCCSID       DS   F    Requested character set
*                                     identifier of VALUE
MQIMPO_RETURNEDENCODING     DS   F    Returned encoding of VALUE
MQIMPO_RETURNEDCCSID        DS   F    Returned character set
*                                     identifier of VALUE
MQIMPO_RESERVED1            DS   F    Reserved field
MQIMPO_RETURNEDNAME         DS   0F   Force fullword alignment
MQIMPO_RETURNEDNAME_VSPTR   DS   F    Address of returned name
MQIMPO_RETURNEDNAME_VSOFFSET DS  F    Offset of returned name
MQIMPO_RETURNEDNAME_VSLENGTH DS  F    Length of returned name
MQIMPO_RETURNEDNAME_VSCCSID DS   F    CCSID of returned name
MQIMPO_RETURNEDNAME_LENGTH  EQU  *-MQIMPO_RETURNEDNAME
                            ORG  MQIMPO_RETURNEDNAME
MQIMPO_RETURNEDNAME_AREA    DS   CL(MQIMPO_RETURNEDNAME_LENGTH)
*
MQIMPO_TYPESTRING           DS   CL8  Property data type as string
MQIMPO_LENGTH               EQU  *-MQIMPO
MQIMPO_AREA                 DS   CL(MQIMPO_LENGTH)
```

MQMD - Message descriptor:

The following table summarizes the fields in the structure.

Table 214. Fields in MQMD

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Report	Options for report messages	Report
MsgType	Message type	MsgType
Expiry	Message lifetime	MQMD - Expiry field
Feedback	Feedback or reason code	MQMD - Feedback field
Encoding	Numeric encoding of message data	Encoding
CodedCharSetId	Character set identifier of message data	CodedCharSetId
Format	Format name of message data	Format
Priority	Message priority	Priority
Persistence	Message persistence	Persistence
MsgId	Message identifier	MQMD - MsgId field
CorrelId	Correlation identifier	CorrelId
BackoutCount	Backout counter	BackoutCount
ReplyToQ	Name of reply queue	ReplyToQ
ReplyToQMgr	Name of reply queue manager	ReplyToQMgr
UserIdentifier	User identifier	UserIdentifier
AccountingToken	Accounting token	AccountingToken
ApplIdentityData	Application data relating to identity	ApplIdentityData
PutApplType	Type of application that put the message	PutApplType
PutApplName	Name of application that put the message	PutApplName
PutDate	Date when message was put	PutDate
PutTime	Time when message was put	PutTime
ApplOriginData	Application data relating to origin	ApplOriginData
Note: The remaining fields are ignored if *Version* is less than MQMD_VERSION_2.		
GroupId	Group identifier	GroupId
MsgSeqNumber	Sequence number of logical message within group	MsgSeqNumber
Offset	Offset of data in physical message from start of logical message	Offset
MsgFlags	Message flags	MQMD - MsgFlags field
OriginalLength	Length of original message	OriginalLength

Overview for MQMD:

Availability: All IBM MQ systems, plus IBM MQ MQI clients connected to these systems.

Purpose: The MQMD structure contains the control information that accompanies the application data when a message travels between the sending and receiving applications. The structure is an input/output parameter on the MQGET, MQPUT, and MQPUT1 calls.

Version: The current version of MQMD is MQMD_VERSION_2. Applications that are intended to be portable between several environments must ensure that the required version of MQMD is supported in all the environments concerned. Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions that follow.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQMD that is supported by the environment, but with the initial value of the *Version* field set to MQMD_VERSION_1. To use fields that are not present in the version-1 structure, the application must set the *Version* field to the version number of the version required.

A declaration for the version-1 structure is available with the name MQMD1.

Character set and encoding: Data in MQMD must be in the character set and encoding of the local queue manager; these are given by the **CodedCharSetId** queue manager attribute and MQENC_NATIVE. However, if the application is running as an IBM MQ MQI client, the structure must be in the character set and encoding of the client.

If the sending and receiving queue managers use different character sets or encodings, the data in MQMD is converted automatically. It is not necessary for the application to convert the MQMD.

Using different versions of MQMD: A version-2 MQMD is equivalent to using a version-1 MQMD and prefixing the message data with an MQMDE structure. However, if all the fields in the MQMDE structure have their default values, the MQMDE can be omitted. A version-1 MQMD plus MQMDE are used as described:

- On the MQPUT and MQPUT1 calls, if the application provides a version-1 MQMD, the application can optionally prefix the message data with an MQMDE, setting the *Format* field in MQMD to MQFMT_MD_EXTENSION to indicate that an MQMDE is present. If the application does not provide an MQMDE, the queue manager assumes default values for the fields in the MQMDE.

  Note: Several of the fields that exist in the version-2 MQMD but not the version-1 MQMD are input/output fields on the MQPUT and MQPUT1 calls. However, the queue manager does not return any values in the equivalent fields in the MQMDE on output from the MQPUT and MQPUT1 calls; if the application requires those output values, it must use a version-2 MQMD.
- On the MQGET call, if the application provides a version-1 MQMD, the queue manager prefixes the message returned with an MQMDE, but only if one or more of the fields in the MQMDE has a non-default value. The *Format* field in MQMD will have the value MQFMT_MD_EXTENSION to indicate that an MQMDE is present.

The default values that the queue manager uses for the fields in the MQMDE are the same as the initial values of those fields, shown in Table 219 on page 2447.

When a message is on a transmission queue, some of the fields in MQMD are set to particular values; see "MQXQH - Transmission-queue header" on page 2612 for details.

Message context: Certain fields in MQMD contain the message context. There are two types of message context: *identity context* and *origin context*. Typically:

- Identity context relates to the application that *originally* put the message

- Origin context relates to the application that *most recently* put the message.

These two applications can be the same application, but they can also be different applications (for example, when a message is forwarded from one application to another).

Although identity and origin context typically have the meanings described, the content of both types of context fields in MQMD depends on the MQPMO_*_CONTEXT options that are specified when the message is put. As a result, identity context does not necessarily relate to the application that originally put the message, and origin context does not necessarily relate to the application that most-recently put the message; it depends on the design of the application suite.

The message channel agent (MCA) never alters message context. MCAs that receive messages from remote queue managers use the context option MQPMO_SET_ALL_CONTEXT on the MQPUT or MQPUT1 call. This allows the receiving MCA to preserve exactly the message context that traveled with the message from the sending MCA. However, the result is that the origin context does not relate to either of the MCAs that sent and received the message. The origin context refers to an earlier application that put the message. If all the intermediate applications have passed the message context, the origin context refers to the originating application itself.

In the descriptions, the context fields are described as though they are used as described previously. For more information about message context, see Message context.

Fields for MQMD:

The MQMD structure contains the following fields; the fields are described in **alphabetical order**:

Table 215. Fields in MQMD

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Report	Options for report messages	Report
MsgType	Message type	MsgType
Expiry	Message lifetime	MQMD - Expiry field
Feedback	Feedback or reason code	MQMD - Feedback field
Encoding	Numeric encoding of message data	Encoding
CodedCharSetId	Character set identifier of message data	CodedCharSetId
Format	Format name of message data	Format
Priority	Message priority	Priority
Persistence	Message persistence	Persistence
MsgId	Message identifier	MQMD - MsgId field
CorrelId	Correlation identifier	CorrelId
BackoutCount	Backout counter	BackoutCount
ReplyToQ	Name of reply queue	ReplyToQ
ReplyToQMgr	Name of reply queue manager	ReplyToQMgr
UserIdentifier	User identifier	UserIdentifier
AccountingToken	Accounting token	AccountingToken
ApplIdentityData	Application data relating to identity	ApplIdentityData
PutApplType	Type of application that put the message	PutApplType
PutApplName	Name of application that put the message	PutApplName

Table 215. Fields in MQMD (continued)

Field	Description	Topic
PutDate	Date when message was put	PutDate
PutTime	Time when message was put	PutTime
ApplOriginData	Application data relating to origin	ApplOriginData
Note: The remaining fields are ignored if *Version* is less than MQMD_VERSION_2.		
GroupId	Group identifier	GroupId
MsgSeqNumber	Sequence number of logical message within group	MsgSeqNumber
Offset	Offset of data in physical message from start of logical message	Offset
MsgFlags	Message flags	MQMD - MsgFlags field
OriginalLength	Length of original message	OriginalLength

AccountingToken (MQBYTE32):

This is the accounting token, part of the **identity context** of the message. For more information about message context, see "Overview for MQMD" on page 2388 ; also see Message context.

AccountingToken allows an application to charge appropriately for work done as a result of the message. The queue manager treats this information as a string of bits and does not check its content.

The queue manager generates this information as follows:

- The first byte of the field is set to the length of the accounting information present in the bytes that follow; this length is in the range zero through 30, and is stored in the first byte as a binary integer.
- The second and subsequent bytes (as specified by the length field) are set to the accounting information appropriate to the environment.
  - On z/OS the accounting information is set to:
    - For z/OS batch, the accounting information from the JES JOB card or from a JES ACCT statement in the EXEC card (comma separators are changed to X'FF'). This information is truncated, if necessary, to 31 bytes.
    - For TSO, the user's account number.
    - For CICS, the LU 6.2 unit of work identifier (UEPUOWDS) (26 bytes).
    - For IMS, the 8-character PSB name concatenated with the 16-character IMS recovery token.
  - On IBM i, the accounting information is set to the accounting code for the job.
  - On UNIX, the accounting information is set to the numeric user identifier, in ASCII characters.
  - On Windows, the accounting information is set to a Windows security identifier (SID) in a compressed format. The SID uniquely identifies the user identifier stored in the *UserIdentifier* field. When the SID is stored in the *AccountingToken* field, the 6-byte Identifier Authority (located in the third and subsequent bytes of the SID) is omitted. For example, if the Windows SID is 28 bytes long, 22 bytes of SID information are stored in the *AccountingToken* field.
- The last byte (byte 32) of the accounting field is set to the accounting token type (in this case MQACTT_NT_SECURITY_ID, x '0b'):

  MQACTT_CICS_LUOW_ID
  CICS LUOW identifier.

  MQACTT_NT_SECURITY_ID
  Windows security identifier.

  MQACTT_OS400_ACCOUNT_TOKEN
  IBM i accounting token.

MQACTT_UNIX_NUMERIC_ID
UNIX numeric identifier.

MQACTT_USER
User-defined accounting token.

MQACTT_UNKNOWN
Unknown accounting-token type.

The accounting-token type is set to an explicit value only in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ MQI clients connected to these systems. In other environments, the accounting-token type is set to the value MQACTT_UNKNOWN. In these environments use the *PutApplType* field to deduce the type of accounting token received.

- All other bytes are set to binary zero.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT is specified in the **PutMsgOpts** parameter. If neither MQPMO_SET_IDENTITY_CONTEXT nor MQPMO_SET_ALL_CONTEXT is specified, this field is ignored on input and is an output-only field. For more information about message context, see Message context.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *AccountingToken* that was transmitted with the message if it was put to a queue. This will be the value of *AccountingToken* that is kept with the message if it is retained (see description of MQPMO_RETAIN in "MQPMO options (MQLONG)" on page 2484 for more details about retained publications) but is not used as the *AccountingToken* when the message is sent as a publication to subscribers since they provide a value to override *AccountingToken* in all publications sent to them. If the message has no context, the field is entirely binary zero.

This is an output field for the MQGET call.

This field is not subject to any translation based on the character set of the queue manager; the field is treated as a string of bits, and not as a string of characters.

The queue manager does nothing with the information in this field. The application must interpret the information if it wants to use the information for accounting purposes.

You can use the following special value for the *AccountingToken* field:

MQACT_NONE
No accounting token is specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQACT_NONE_ARRAY is also defined; this has the same value as MQACT_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_ACCOUNTING_TOKEN_LENGTH. The initial value of this field is MQACT_NONE.

ApplIdentityData (MQCHAR32):

This is part of the **identity context** of the message. For more information about message context, see "Overview for MQMD" on page 2388 and Message context.

ApplIdentityData is information that is defined by the application suite, and can be used to provide additional information about the message or its originator. The queue manager treats this information as character data, but does not define the format of it. When the queue manager generates this information, it is entirely blank.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT is specified in the **PutMsgOpts** parameter. If a null character is present, the null and any following characters are converted to blanks by the queue manager. If neither MQPMO_SET_IDENTITY_CONTEXT nor MQPMO_SET_ALL_CONTEXT is specified, this field is ignored on input and is an output-only field. For more information about message context, see Message context.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *ApplIdentityData* that was transmitted with the message if it was put to a queue. This will be the value of *ApplIdentityData* that is kept with the message if it is retained (see description of MQPMO_RETAIN for more details about retained publications) but is not used as the *ApplIdentityData* when the message is sent as a publication to subscribers because they provide a value to override *ApplIdentityData* in all publications sent to them. If the message has no context, the field is entirely blank.

This is an output field for the MQGET call. The length of this field is given by MQ_APPL_IDENTITY_DATA_LENGTH. The initial value of this field is the null string in C, and 32 blank characters in other programming languages.

ApplOriginData (MQCHAR4):

This is part of the **origin context** of the message. For more information about message context, see "Overview for MQMD" on page 2388 and Message context.

ApplOriginData is information that is defined by the application suite that can be used to provide additional information about the origin of the message. For example, it could be set by applications running with suitable user authority to indicate whether the identity data is trusted.

The queue manager treats this information as character data, but does not define the format of it. When the queue manager generates this information, it is entirely blank.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the **PutMsgOpts** parameter. Any information following a null character within the field is discarded. The queue manager converts the null character and any following characters to blanks. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

This is an output field for the MQGET call. The length of this field is given by MQ_APPL_ORIGIN_DATA_LENGTH. The initial value of this field is the null string in C, and 4 blank characters in other programming languages.

BackoutCount (MQLONG):

This is a count of the number of times that the message has been previously returned by the MQGET call as part of a unit of work, and subsequently backed out. It helps the application to detect processing errors that are based on message content. The count excludes MQGET calls that specify any of the MQGMO_BROWSE_* options.

The accuracy of this count is affected by the **HardenGetBackout** queue attribute; see "Attributes for queues" on page 2833.

On z/OS, a value of 255 means that the message has been backed out 255 or more times; the value returned is never greater than 255.

This is an output field for the MQGET call. It is ignored for the MQPUT and MQPUT1 calls. The initial value of this field is `0`.

CodedCharSetId (MQLONG):

This field specifies the character set identifier of character data within the message body.

Note: Character data in MQMD and the other MQ data structures that are parameters on calls must be in the character set of the queue manager. This is defined by the queue manager's **CodedCharSetId** attribute; see "Attributes for the queue manager" on page 2792 for details of this attribute.

If this field is set to MQCCSI_Q_MGR when calling MQGET with MQGMO_CONVERT in the options, the behavior is different between client and server applications. For server applications, the code page used for character conversion is the *CodedCharSetId* of the queue manager; for client applications, the code page used for character conversion is the current locale code page.

For client applications, MQCCSI_Q_MGR is filled in, based on the locale of the client rather than the one on the queue manager. The exception to that rule is when you put a message to an IMS bridge queue; what is returned, in the *CodedCharSetId* field of MQMD, is the CCSID of the queue manager.

You must not use the following special value:

MQCCSI_APPL
This results in an incorrect value in the *CodedCharSetId* field of the MQMD and causes a return code of MQRC_SOURCE_CCSID_ERROR (or MQRC_FORMAT_ERROR for z/OS) when the message is received using the MQGET call with the MQGMO_CONVERT option.

You can use the following special values:

MQCCSI_Q_MGR
Character data in the message is in the queue manager's character set.

On the MQPUT and MQPUT1 calls, the queue manager changes this value in the MQMD that is sent with the message to the true character set identifier of the queue manager. As a result, the value MQCCSI_Q_MGR is never returned by the MQGET call.

MQCCSI_DEFAULT
The *CodedCharSetId* of the data in the *String* field is defined by the *CodedCharSetId* field in the header structure that precedes the MQCFH structure, or by the *CodedCharSetId* field in the MQMD if the MQCFH is at the start of the message.

MQCCSI_INHERIT
Character data in the message is in the same character set as this structure; this is the queue manager's character set. (For MQMD only, MQCCSI_INHERIT has the same meaning as MQCCSI_Q_MGR).

The queue manager changes this value in the MQMD that is sent with the message to the actual character set identifier of MQMD. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

Do not use MQCCSI_INHERIT if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

MQCCSI_EMBEDDED

Character data in the message is in a character set with the identifier that is contained within the message data itself. There can be any number of character set identifiers embedded within the message data, applying to different parts of the data. This value must be used for PCF messages (with a format of MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF) that contain data in a mixture of character sets. Each MQCFST, MQCFSL, and MQCFSF structure contained within the PCF message must have an explicit character set identifier specified and not MQCCSI_DEFAULT.

If a message of format MQFMT_EMBEDDED_PCF is to contain data in a mixture of character sets, do not use MQCCSI_EMBEDDED. Instead set MQEPH_CCSID_EMBEDDED in the Flags field in the MQEPH structure. This is equivalent to setting MQCCSI_EMBEDDED in the preceding structure. Each MQCFST, MQCFSL, and MQCFSF structure contained within the PCF message must then have an explicit character set identifier specified and not MQCCSI_DEFAULT. For more information on the MQEPH structure, see "MQEPH - Embedded PCF header" on page 2324.

Specify this value only on the MQPUT and MQPUT1 calls. If it is specified on the MQGET call, it prevents conversion of the message.

On the MQPUT and MQPUT1 calls, the queue manager changes the values MQCCSI_Q_MGR and MQCCSI_INHERIT in the MQMD that is sent with the message as described above, but does not change the MQMD specified on the MQPUT or MQPUT1 call. No other check is carried out on the value specified.

Applications that retrieve messages must compare this field against the value the application is expecting; if the values differ, the application might need to convert character data in the message.

V 9.0.0 On z/OS, the *Encoding* field of the MQMD is used to specify the integer encoding of character data in the message body, when the *CodedCharSetId* field of the MQMD indicates that the representation of the character set is dependent on the encoding used for binary integers. On Multiplatforms, the byte order of character data is assumed to be the same as the native integer encoding for the platform where the queue manager is running. This only affects certain multibyte character sets (for example UTF-16 character sets).

If you specify the MQGMO_CONVERT option on the MQGET call, this field is an input/output field. The value specified by the application is the coded character set identifier to which to convert the message data if necessary. If conversion is successful or unnecessary, the value is unchanged (except that the value MQCCSI_Q_MGR or MQCCSI_INHERIT is converted to the actual value). If conversion is unsuccessful, the value after the MQGET call represents the coded character set identifier of the unconverted message that is returned to the application.

Otherwise, this is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQCCSI_Q_MGR.

CorrelId (MQBYTE24):

The CorrelId field is property in the message header that may be used to identify a specific message or group of messages.

This is a byte string that the application can use to relate one message to another, or to relate the message to other work that the application is performing. The correlation identifier is a permanent property of the message, and persists across restarts of the queue manager. Because the correlation identifier is a byte string and not a character string, the correlation identifier is not converted between character sets when the message flows from one queue manager to another.

For the MQPUT and MQPUT1 calls, the application can specify any value. The queue manager transmits this value with the message and delivers it to the application that issues the get request for the message.

If the application specifies MQPMO_NEW_CORREL_ID, the queue manager generates a unique correlation identifier which is sent with the message, and also returned to the sending application on output from the MQPUT or MQPUT1 call.

A correlation identifier generated by the queue manager consists of a 3-byte product identifier (AMQ or CSQ in either ASCII or EBCDIC), followed by one reserved byte and a product-specific implementation of a unique string. In IBM MQ this product-specific implementation string contains the first 12 characters of the queue manager name, and a value derived from the system clock. All queue managers that can intercommunicate must therefore have names that differ in the first 12 characters to ensure that message identifiers are unique. The ability to generate a unique string also depends on the system clock not being changed backward. To eliminate the possibility of a message identifier generated by the queue manager duplicating one generated by the application, the application must avoid generating identifiers with initial characters in the range A through I in ASCII or EBCDIC (X'41' through X'49' and X'C1' through X'C9'). However, the application is not prevented from generating identifiers with initial characters in these ranges.

This generated correlation identifier is kept with the message if it is retained, and is used as the correlation identifier when the message is sent as a publication to subscribers who specify MQCI_NONE in the SubCorrelId field in the MQSD passed on the MQSUB call. See MQPMO options for more details about retained publications.

When the queue manager or a message channel agent generates a report message, it sets the *CorrelId* field in the way specified by the *Report* field of the original message, either MQRO_COPY_MSG_ID_TO_CORREL_ID or MQRO_PASS_CORREL_ID. Applications that generate report messages must also do this.

For the MQGET call, *CorrelId* is one of the five fields that can be used to select a particular message to be retrieved from the queue. See the description of the *MsgId* field for details of how to specify values for this field.

Specifying MQCI_NONE as the correlation identifier has the same effect as not specifying MQMO_MATCH_CORREL_ID, that is, *any* correlation identifier will match.

If the MQGMO_MSG_UNDER_CURSOR option is specified in the **GetMsgOpts** parameter on the MQGET call, this field is ignored.

On return from an MQGET call, the *CorrelId* field is set to the correlation identifier of the message returned (if any).

The following special values can be used:

MQCI_NONE
No correlation identifier is specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQCI_NONE_ARRAY is also defined; this has the same value as MQCI_NONE, but is an array of characters instead of a string.

MQCI_NEW_SESSION
Message is the start of a new session.

This value is recognized by the CICS bridge as indicating the start of a new session, that is, the start of a new sequence of messages.

For the C programming language, the constant MQCI_NEW_SESSION_ARRAY is also defined; this has the same value as MQCI_NEW_SESSION, but is an array of characters instead of a string.

For the MQGET call, this is an input/output field. For the MQPUT and MQPUT1 calls, this is an input field if MQPMO_NEW_CORREL_ID is not specified, and an output field if MQPMO_NEW_CORREL_ID is specified. The length of this field is given by MQ_CORREL_ID_LENGTH. The initial value of this field is MQCI_NONE.

Note:

You cannot pass the correlation identifier of a publication in a hierarchy. The field is used by the queue manager.

Encoding (MQLONG):

This specifies the numeric encoding of numeric data in the message; it does not apply to numeric data in the MQMD structure itself. The numeric encoding defines the representation used for binary integers, packed-decimal integers, and floating-point numbers.

V 9.0.0 On z/OS, the binary integer portion of the *Encoding* field is also used to specify the integer encoding of character data in the message body when the corresponding character set identifier indicates that the representation of the character set is dependent on the encoding used for binary integers. This only affects certain multibyte character sets (for example UTF-16 character sets).

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that the field is valid. The following special value is defined:

MQENC_NATIVE
The encoding is the default for the programming language and machine on which the application is running.

Note: The value of this constant depends on the programming language and environment. For this reason, applications must be compiled using the header, macro, COPY, or INCLUDE files appropriate to the environment in which the application will run.

Applications that put messages usually specify MQENC_NATIVE. Applications that retrieve messages must compare this field against the value MQENC_NATIVE; if the values differ, the application might need to convert numeric data in the message. Use the MQGMO_CONVERT option to request the queue manager to convert the message as part of the processing of the MQGET call. See “Machine encodings” on page 2902 for details of how the *Encoding* field is constructed.

If you specify the MQGMO_CONVERT option on the MQGET call, this field is an input/output field. The value specified by the application is the encoding to which to convert the message data if necessary.

If conversion is successful or unnecessary, the value is unchanged. If conversion is unsuccessful, the value after the MQGET call represents the encoding of the unconverted message that is returned to the application.

In other cases, this is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQENC_NATIVE.

Expiry (MQLONG):

This is a period of time expressed in tenths of a second, set by the application that puts the message. The message becomes eligible to be discarded if it has not been removed from the destination queue before this period of time elapses.

The value is decremented to reflect the time that the message spends on the destination queue, and also on any intermediate transmission queues if the put is to a remote queue. It can also be decremented by message channel agents to reflect transmission times, if these are significant. Likewise, an application forwarding this message to another queue might decrement the value if necessary, if it has retained the message for a significant time. However, the expiration time is treated as approximate, and the value need not be decremented to reflect small time intervals.

When the message is retrieved by an application using the MQGET call, the *Expiry* field represents the expiry time that still remains.

After a message's expiry time has elapsed, it becomes eligible to be discarded by the queue manager. The message is discarded when a browse or nonbrowse MQGET call occurs that would have returned the message had it not already expired. For example, a nonbrowse MQGET call with the *MatchOptions* field in MQGMO set to MQMO_NONE reading from a FIFO ordered queue discards all the expired messages up to the first unexpired message. With a priority ordered queue, the same call will discard expired messages of higher priority and messages of an equal priority that arrived on the queue before the first unexpired message.

A message that has expired is never returned to an application (either by a browse or a non-browse MQGET call), so the value in the *Expiry* field of the message descriptor after a successful MQGET call is either greater than zero, or the special value MQEI_UNLIMITED.

If a message is put on a remote queue, the message might expire (and be discarded) while it is on an intermediate transmission queue, before the message reaches the destination queue.

A report is generated when an expired message is discarded, if the message specified one of the MQRO_EXPIRATION_* report options. If none of these options is specified, no such report is generated; the message is assumed to be no longer relevant after this time period (perhaps because a later message has superseded it).

For a message put within syncpoint, the expiry interval begins at the time the message is put, not the time the syncpoint is committed. It is possible that the expiry interval can pass before the syncpoint is committed. In this case the message will be discarded at some time after the commit operation and the message will not be returned to an application in response to an MQGET operation.

Any other program that discards messages based on expiry time must also send an appropriate report message if one was requested.

Notes:

1. If a message is put with an *Expiry* time of zero or a number greater than 999 999 999, the MQPUT or MQPUT1 call fails with reason code MQRC_EXPIRY_ERROR; no report message is generated in this case.

2. Because a message with an expiry time that has elapsed might not be discarded until later, there might be messages on a queue that have passed their expiry time, and that are not therefore eligible for retrieval. These messages nevertheless count toward the number of messages on the queue for all purposes, including depth triggering.
3. An expiration report is generated, if requested, when the message is discarded, not when it becomes eligible for discarding.
4. Discarding an expired message, and generating an expiration report if requested, are never part of the application's unit of work, even if the message was scheduled for discarding as a result of an MQGET call operating within a unit of work.
5. If a nearly-expired message is retrieved by an MQGET call within a unit of work, and the unit of work is subsequently backed out, the message might become eligible to be discarded before it can be retrieved again.
6. If a nearly-expired message is locked by an MQGET call with MQGMO_LOCK, the message might become eligible to be discarded before it can be retrieved by an MQGET call with MQGMO_MSG_UNDER_CURSOR; reason code MQRC_NO_MSG_UNDER_CURSOR is returned on this subsequent MQGET call if that happens.
7. When a request message with an expiry time greater than zero is retrieved, the application can take one of the following actions when it sends the reply message:
   - Copy the remaining expiry time from the request message to the reply message.
   - Set the expiry time in the reply message to an explicit value greater than zero.
   - Set the expiry time in the reply message to MQEI_UNLIMITED.

   The action to take depends on the design of the application. However, the default action for putting messages to a dead-letter (undelivered-message) queue must be to preserve the remaining expiry time of the message, and to continue to decrement it.
8. Trigger messages are always generated with MQEI_UNLIMITED.
9. A message (normally on a transmission queue) that has a *Format* name of MQFMT_XMIT_Q_HEADER has a second message descriptor within the MQXQH. It therefore has two *Expiry* fields associated with it. The following additional points should be noted in this case:
   - When an application puts a message on a remote queue, the queue manager places the message initially on a local transmission queue, and prefixes the application message data with an MQXQH structure. The queue manager sets the values of the two *Expiry* fields to be the same as that specified by the application.

     If an application puts a message directly on a local transmission queue, the message data must already begin with an MQXQH structure, and the format name must be MQFMT_XMIT_Q_HEADER. In this case, the application need not set the values of these two *Expiry* fields to be the same. (The queue manager checks that the *Expiry* field within the MQXQH contains a valid value, and that the message data is long enough to include it). For an application that can write directly to the transmission queue, the application has to create a transmission queue header with the embedded message descriptor. However, if the expiry value in the message descriptor written to the transmission queue is inconsistent with the value in the embedded message descriptor, an expiry error rejection occurs.
   - When a message with a *Format* name of MQFMT_XMIT_Q_HEADER is retrieved from a queue (whether this is a normal or a transmission queue), the queue manager decrements *both* these *Expiry* fields with the time spent waiting on the queue. No error is raised if the message data is not long enough to include the *Expiry* field in the MQXQH.
   - The queue manager uses the *Expiry* field in the separate message descriptor (that is, not the one in the message descriptor embedded within the MQXQH structure) to test whether the message is eligible for discarding.
   - If the initial values of the two *Expiry* fields are different, the *Expiry* time in the separate message descriptor when the message is retrieved might be greater than zero (so the message is not eligible for discarding), while the time according to the *Expiry* field in the MQXQH has elapsed. In this case the *Expiry* field in the MQXQH is set to zero.

10. The expiry time on a reply message returned from the IMS bridge is unlimited unless MQIIH_PASS_EXPIRATION is set in the Flags field of the MQIIH. See Flags for more information.

The following special value is recognized:

MQEI_UNLIMITED
The message has an unlimited expiration time.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQEI_UNLIMITED.

Expired messages on z/OS:

On IBM MQ for z/OS, messages that have expired are discarded by the next appropriate MQGET call.

However, if no such call occurs, the expired message is not discarded, and, for some queues, a large number of expired messages can accumulate. To remedy this, set the queue manager to scan queues periodically and discard expired messages on one or more queues in one of the following ways:

Periodic scan
You can specify a period using the EXPRYINT (expiry interval) queue manager attribute. Each time the expiry interval is reached, the queue manager looks for candidate queues that are worth scanning to discard expired messages.

The queue manager maintains information about the expired messages on each queue, and knows whether a scan for expired messages is worthwhile. So, only a selection of queues is scanned at any time.

Shared queues are scanned by only one queue manager in a queue-sharing group. Generally, it is the first queue manager to restart, or the first to have EXPRYINT set. If this queue manager terminates, another queue manager in the queue-sharing group takes over the queue scanning. Set the expiry interval value for all queue managers within a queue-sharing group to the same value.

Note that expiry processing takes place for every queue when a queue manager restarts, regardless of the EXPRYINT setting.

Explicit request
Issue the REFRESH QMGR TYPE(EXPIRY) command, specifying the queue or queues that you want scanned.

Enforcing lower expiration times:

Administrators can limit the expiry time of any message put to a queue or topic by using the *CAPEXPRY* attribute specified in the **CUSTOM** attribute on the queue or topic.

z/OS You can either enter the command at a console, or to the queue manager configuration, by adding to a file processed in the CSQINP2 concatenation during queue manager startup. Note that the effect of the command does not persist over a queue manager restart.

An expiry time specified in the **Expiry** field of the MQMD, by an application,which is greater than the *CAPEXPRY* value specified in the **CUSTOM** attribute on the queue or topic will be replaced by that *CAPEXPRY* value. An expiry time specified by an application, which is lower than the *CAPEXPRY* value will be used.

If more than one object is used on the resolution path, for example, when a message is put to an alias or remote queue, then the lowest of all the *CAPEXPRY* values is used as the upper limit for message expiry.

Changes to the *CAPEXPRY* values take effect immediately. The expiry value is evaluated for each put to a queue or topic and so is sensitive to the object resolution, which may differ between each put operation.

However, note that existing messages in the queue, prior to a change in **CAPEXPRY**, are not affected by the change (that is, their expiry time remains intact). Only new messages that are put into the queue after the change in **CAPEXPRY** have the new expiry time.

For example, in a cluster where a put is performed to a queue opened with *MQOO_BIND_NOT_FIXED*, messages can be assigned different expiry values on each put, depending on the *CAPEXPRY* value set for the transmission queue, used by the channel, that sends the message to the selected target queue manager.

Note, that a put to a queue or topic by a JMS application specifying a delivery delay fails with *MQRC_EXPIRY_ERROR*, if the delivery delay is beyond the resolved expiry time for the target queue or topic. A *CAPEXPRY* attribute set on a queue resolved for a JMS Destination can cause this error.

Feedback (MQLONG):

The Feedback field is used with a message of type MQMT_REPORT to indicate the nature of the report, and is only meaningful with that type of message.

The field can contain one of the MQFB_* values, or one of the MQRC_* values. Feedback codes are grouped as follows:

MQFB_NONE
No feedback provided.

MQFB_SYSTEM_FIRST
Lowest value for system-generated feedback.

MQFB_SYSTEM_LAST
Highest value for system-generated feedback.

The range of system-generated feedback codes MQFB_SYSTEM_FIRST through MQFB_SYSTEM_LAST includes the general feedback codes listed in this topic (MQFB_*), and also the reason codes (MQRC_*) that can occur when the message cannot be put on the destination queue.

MQFB_APPL_FIRST
Lowest value for application-generated feedback.

MQFB_APPL_LAST
Highest value for application-generated feedback.

Applications that generate report messages must not use feedback codes in the system range (other than MQFB_QUIT), unless they want to simulate report messages generated by the queue manager or message channel agent.

On the MQPUT or MQPUT1 calls, the value specified must either be MQFB_NONE, or be within the system range or application range. This is checked whatever the value of *MsgType*.

General feedback codes:

MQFB_COA
Confirmation of arrival on the destination queue (see MQRO_COA).

MQFB_COD
Confirmation of delivery to the receiving application (see MQRO_COD).

MQFB_EXPIRATION
Message was discarded because it had not been removed from the destination queue before its expiry time had elapsed.

MQFB_PAN
Positive action notification (see MQRO_PAN).

MQFB_NAN
Negative action notification (see MQRO_NAN).

MQFB_QUIT
End application.

This can be used by a workload scheduling program to control the number of instances of an application program that are running. Sending an MQMT_REPORT message with this feedback code to an instance of the application program indicates to that instance that it should stop processing. However, adherence to this convention is a matter for the application; it is not enforced by the queue manager.

Channel feedback codes:

MQFB_CHANNEL_COMPLETED
A channel ended normally.

MQFB_CHANNEL_FAIL
A channel ended abnormally and goes into STOPPED state.

MQFB_CHANNEL_FAIL_RETRY
A channel ended abnormally and goes into RETRY state.

IMS-bridge feedback codes

These codes are used when an unexpected IMS-OTMA sense code is received. The sense code or, when the sense code is 0x1A the reason code associated with that sense code, is indicated in the *Feedback*.

1. For *Feedback* codes in range MQFB_IMS_FIRST (300) through MQFB_IMS_LAST (399), a sense code other than 0x1A was received. The *sense code* is given by the expression (*Feedback* - MQFB_IMS_FIRST+1)
2. For *Feedback* codes in range MQFB_IMS_NACK_1A_REASON_FIRST (600) through MQFB_IMS_NACK_1A_REASON_LAST (855), a sense code of 0x1A was received. The *reason code* associated with the sense code is given by the expression (*Feedback* - MQFB_IMS_NACK_1A_REASON_FIRST)

The meaning of the IMS-OTMA sense codes and corresponding reason codes are described in *Open Transaction Manager Access Guide and Reference*.

The following feedback codes can be generated by the IMS bridge:

MQFB_DATA_LENGTH_ZERO
A segment length was zero in the application data of the message.

MQFB_DATA_LENGTH_NEGATIVE
A segment length was negative in the application data of the message.

MQFB_DATA_LENGTH_TOO_BIG
A segment length was too large in the application data of the message.

MQFB_BUFFER_OVERFLOW
The value of one of the length fields would cause the data to overflow the message buffer.

MQFB_LENGTH_OFF_BY_ONE
The value of one of the length fields was 1 byte too short.

MQFB_IIH_ERROR
The *Format* field in MQMD specifies MQFMT_IMS, but the message does not begin with a valid MQIIH structure.

MQFB_NOT_AUTHORIZED_FOR_IMS
The user ID contained in the message descriptor MQMD, or the password contained in the *Authenticator* field in the MQIIH structure, failed the validation performed by the IMS bridge. As a result the message was not passed to IMS.

MQFB_IMS_ERROR
An unexpected error was returned by IMS. Consult the IBM MQ error log on the system on which the IMS bridge resides for more information about the error.

MQFB_IMS_FIRST
When the IMS-OTMA sense code is not 0x1A, IMS-generated feedback codes are in the range MQFB_IMS_FIRST (300) through MQFB_IMS_LAST (399). The IMS-OTMA sense code itself is *Feedback* minus MQFB_IMS_ERROR.

MQFB_IMS_LAST
Highest value for IMS-generated feedback when the sense code is not 0x1A.

MQFB_IMS_NACK_1A_REASON_FIRST
When the sense code is 0x1A, IMS-generated feedback codes are in the range MQFB_IMS_NACK_1A_REASON_FIRST (600) through MQFB_IMS_NACK_1A_REASON_LAST (855).

MQFB_IMS_NACK_1A_REASON_LAST
Highest value for IMS-generated feedback when the sense code is 0x1A

CICS-bridge feedback codes: The following feedback codes can be generated by the CICS bridge:

MQFB_CICS_APPL_ABENDED
The application program specified in the message abnormally ended. This feedback code occurs only in the *Reason* field of the MQDLH structure.

MQFB_CICS_APPL_NOT_STARTED
The EXEC CICS LINK for the application program specified in the message failed. This feedback code occurs only in the *Reason* field of the MQDLH structure.

MQFB_CICS_BRIDGE_FAILURE
CICS bridge terminated abnormally without completing normal error processing.

MQFB_CICS_CCSID_ERROR
Character set identifier not valid.

MQFB_CICS_CIH_ERROR
CICS information header structure missing or not valid.

MQFB_CICS_COMMAREA_ERROR
Length of CICS COMMAREA not valid.

MQFB_CICS_CORREL_ID_ERROR
Correlation identifier not valid.

MQFB_CICS_DLQ_ERROR
The CICS bridge task was unable to copy a reply to this request to the dead-letter queue. The request was backed out.

MQFB_CICS_ENCODING_ERROR
Encoding not valid.

MQFB_CICS_INTERNAL_ERROR
CICS bridge encountered an unexpected error.

This feedback code occurs only in the *Reason* field of the MQDLH structure.

MQFB_CICS_NOT_AUTHORIZED
User identifier not authorized or password not valid.

This feedback code occurs only in the *Reason* field of the MQDLH structure.

MQFB_CICS_UOW_BACKED_OUT
The unit of work was backed out, for one of the following reasons:

- A failure was detected while processing another request within the same unit of work.
- A CICS abend occurred while the unit of work was in progress.

MQFB_CICS_UOW_ERROR
Unit-of-work control field *UOWControl* not valid.

Trace-route message feedback codes:

MQFB_ACTIVITY
Used with the MQFMT_EMBEDDED_PCF format to allow the option of user data following activity reports.

MQFB_MAX_ACTIVITIES
Returned when the trace-route message is discarded because the number of activities the message has been involved in exceeds the maximum activities limit.

MQFB_NOT_FORWARDED
Returned when the trace-route message is discarded because it is about to be sent to a remote queue manager that does not support trace-route messages.

MQFB_NOT_DELIVERED
Returned when the trace-route message is discarded because it is about to be put on a local queue.

MQFB_UNSUPPORTED_FORWARDING
Returned when the trace-route message is discarded because a value in the forwarding parameter is unrecognized, and is in the rejected bit mask.

MQFB_UNSUPPORTED_DELIVERY
Returned when the trace-route message is discarded because a value in the delivery parameter is unrecognized, and is in the rejected bit mask.

IBM MQ reason codes: For exception report messages, *Feedback* contains an IBM MQ reason code. Among possible reason codes are:

MQRC_PUT_INHIBITED
(2051, X'803') Put calls inhibited for the queue.

MQRC_Q_FULL
(2053, X'805') Queue already contains maximum number of messages.

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_Q_SPACE_NOT_AVAILABLE
(2056, X'808') No space available on disk for queue.

MQRC_PERSISTENT_NOT_ALLOWED
(2048, X'800') Queue does not support persistent messages.

MQRC_MSG_TOO_BIG_FOR_Q_MGR
(2031, X'7EF') Message length greater than maximum for queue manager.

MQRC_MSG_TOO_BIG_FOR_Q
(2030, X'7EE') Message length greater than maximum for queue.

For a full list of reason codes, see:
- For IBM MQ for z/OS, see API reason codes.
- For all other platforms, see API completion and reason codes.

.

This is an output field for the MQGET call, and an input field for MQPUT and MQPUT1 calls. The initial value of this field is MQFB_NONE.

Format (MQCHAR8):

This is a name that the sender of the message uses to indicate to the receiver the nature of the data in the message. Any characters that are in the character set of the queue manager can be specified for the name, but you must restrict the name to the following:

- Uppercase A through Z
- Numeric digits 0 through 9

If other characters are used, it might not be possible to translate the name between the character sets of the sending and receiving queue managers.

Pad the name with blanks to the length of the field, or use a null character to terminate the name before the end of the field; the null and any subsequent characters are treated as blanks. Do not specify a name with leading or embedded blanks. For the MQGET call, the queue manager returns the name padded with blanks to the length of the field.

The queue manager does not check that the name complies with the recommendations described above.

Names beginning MQ in upper, lower, and mixed case have meanings that are defined by the queue manager; do not use names beginning with these letters for your own formats. The queue manager built-in formats are:

MQFMT_NONE
The nature of the data is undefined: the data cannot be converted when the message is retrieved from a queue using the MQGMO_CONVERT option.

If you specify MQGMO_CONVERT on the MQGET call, and the character set or encoding of data in the message differs from that specified in the **MsgDesc** parameter, the message is returned with the following completion and reason codes (assuming no other errors):

- Completion code MQCC_WARNING and reason code MQRC_FORMAT_ERROR if the MQFMT_NONE data is at the beginning of the message.
- Completion code MQCC_OK and reason code MQRC_NONE if the MQFMT_NONE data is at the end of the message (that is, preceded by one or more MQ header structures). The MQ header structures are converted to the requested character set and encoding in this case.

For the C programming language, the constant MQFMT_NONE_ARRAY is also defined; this has the same value as MQFMT_NONE, but is an array of characters instead of a string.

MQFMT_ADMIN
The message is a command-server request or reply message in programmable command format (PCF). Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call. See Using Programmable Command Formats for more information about using programmable command format messages.

For the C programming language, the constant MQFMT_ADMIN_ARRAY is also defined; this has the same value as MQFMT_ADMIN, but is an array of characters instead of a string.

MQFMT_CICS
The message data begins with the CICS information header MQCIH, followed by the application data. The format name of the application data is given by the *Format* field in the MQCIH structure.

On z/OS, specify the MQGMO_CONVERT option on the MQGET call to convert messages that have format MQFMT_CICS.

For the C programming language, the constant MQFMT_CICS_ARRAY is also defined; this has the same value as MQFMT_CICS, but is an array of characters instead of a string.

MQFMT_COMMAND_1
The message is an MQSC command-server reply message containing the object count, completion code, and reason code. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_COMMAND_1_ARRAY is also defined; this has the same value as MQFMT_COMMAND_1, but is an array of characters instead of a string.

MQFMT_COMMAND_2
The message is an MQSC command-server reply message containing information about the objects requested. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_COMMAND_2_ARRAY is also defined; this has the same value as MQFMT_COMMAND_2, but is an array of characters instead of a string.

MQFMT_DEAD_LETTER_HEADER
The message data begins with the dead-letter header MQDLH. The data from the original message immediately follows the MQDLH structure. The format name of the original message data is given by the *Format* field in the MQDLH structure; see "MQDLH - Dead-letter header" on page 2309 for details of this structure. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

COA and COD reports are not generated for messages that have a *Format* of MQFMT_DEAD_LETTER_HEADER.

For the C programming language, the constant MQFMT_DEAD_LETTER_HEADER_ARRAY is also defined; this has the same value as MQFMT_DEAD_LETTER_HEADER, but is an array of characters instead of a string.

MQFMT_DIST_HEADER
The message data begins with the distribution-list header MQDH; this includes the arrays of MQOR and MQPMR records. The distribution-list header can be followed by additional data. The format of the additional data (if any) is given by the *Format* field in the MQDH structure; see "MQDH - Distribution header" on page 2301 for details of this structure. Messages with format MQFMT_DIST_HEADER can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

This format is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ MQI clients connected to these systems.

For the C programming language, the constant MQFMT_DIST_HEADER_ARRAY is also defined; this has the same value as MQFMT_DIST_HEADER, but is an array of characters instead of a string.

MQFMT_EMBEDDED_PCF
Format for a trace-route message, provided that the PCF command value is set to MQCMD_TRACE_ROUTE. Using this format allows user data to be sent along with the trace-route message, provided that their applications can cope with preceding PCF parameters.

The PCF header **must** be the first header, or the message will not be treated as a trace-route message. This means that the message cannot be in a group, and that trace-route messages cannot be segmented. If a trace-route message is sent in a group the message is rejected with reason code MQRC_MSG_NOT_ALLOWED_IN_GROUP.

Note that MQFMT_ADMIN can also be used for the format of a trace-route message, but in this case no user data can be sent along with the trace-route message.

MQFMT_EVENT
The message is an MQ event message that reports an event that occurred. Event messages have the same structure as programmable commands; see PCF command messages for more information about this structure, and Event monitoring for information about events.

Version-1 event messages can be converted in all environments if the MQGMO_CONVERT option is specified on the MQGET call. Version-2 event messages can be converted only on z/OS.

For the C programming language, the constant MQFMT_EVENT_ARRAY is also defined; this has the same value as MQFMT_EVENT, but is an array of characters instead of a string.

MQFMT_IMS
The message data begins with the IMS information header MQIIH, which is followed by the application data. The format name of the application data is given by the *Format* field in the MQIIH structure.

For details of how MQIIH structure is handled when using MQGET with MQGMO_CONVERT, see "Format (MQCHAR8)" on page 2371 and "ReplyToFormat (MQCHAR8)" on page 2372.

For the C programming language, the constant MQFMT_IMS_ARRAY is also defined; this has the same value as MQFMT_IMS, but is an array of characters instead of a string.

MQFMT_IMS_VAR_STRING
The message is an IMS variable string, which is a string of the form `llzzccc`, where:

ll is a 2-byte length field specifying the total length of the IMS variable string item. This length is equal to the length of `ll` (2 bytes), plus the length of `zz` (2 bytes), plus the length of the character string itself. `ll` is a 2-byte binary integer in the encoding specified by the *Encoding* field.

zz is a 2-byte field containing flags that are significant to IMS. `zz` is a byte string consisting of two MQBYTE fields, and is transmitted without change from sender to receiver (that is, `zz` is not subject to any conversion).

ccc is a variable-length character string containing `ll-4` characters. `ccc` is in the character set specified by the *CodedCharSetId* field.

On z/OS, the message data can consist of a sequence of IMS variable strings butted together, with each string being of the form `llzzccc`. There must be no bytes skipped between successive IMS variable strings. This means that if the first string has an odd length, the second string will be misaligned, that is, it will not begin on a boundary that is a multiple of two. Take care when constructing such strings on machines that require alignment of elementary data types.

Use the MQGMO_CONVERT option on the MQGET call to convert messages that have format MQFMT_IMS_VAR_STRING.

For the C programming language, the constant MQFMT_IMS_VAR_STRING_ARRAY is also defined; this has the same value as MQFMT_IMS_VAR_STRING, but is an array of characters instead of a string.

MQFMT_MD_EXTENSION
The message data begins with the message-descriptor extension MQMDE, and is optionally followed by other data (usually the application message data). The format name, character set, and encoding of the data that follow the MQMDE are given by the *Format*, *CodedCharSetId*, and

Encoding fields in the MQMDE. See "MQMDE - Message descriptor extension" on page 2442 for details of this structure. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_MD_EXTENSION_ARRAY is also defined; this has the same value as MQFMT_MD_EXTENSION, but is an array of characters instead of a string.

MQFMT_PCF
The message is a user-defined message that conforms to the structure of a programmable command format (PCF) message. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call. See Using Programmable Command Formats for more information about using programmable command format messages.

For the C programming language, the constant MQFMT_PCF_ARRAY is also defined; this has the same value as MQFMT_PCF, but is an array of characters instead of a string.

MQFMT_REF_MSG_HEADER
The message data begins with the reference message header MQRMH, and is optionally followed by other data. The format name, character set, and encoding of the data is given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQRMH. See "MQRMH - Reference message header" on page 2529 for details of this structure. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

This format is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ MQI clients connected to these systems.

For the C programming language, the constant MQFMT_REF_MSG_HEADER_ARRAY is also defined; this has the same value as MQFMT_REF_MSG_HEADER, but is an array of characters instead of a string.

MQFMT_RF_HEADER
The message data begins with the rules and formatting header MQRFH, and is optionally followed by other data. The format name, character set, and encoding of the data (if any) are given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQRFH. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_RF_HEADER_ARRAY is also defined; this has the same value as MQFMT_RF_HEADER, but is an array of characters instead of a string.

MQFMT_RF_HEADER_2
The message data begins with the version-2 rules and formatting header MQRFH2, and is optionally followed by other data. The format name, character set, and encoding of the optional data (if any) are given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQRFH2. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_RF_HEADER_2_ARRAY is also defined; this has the same value as MQFMT_RF_HEADER_2, but is an array of characters instead of a string.

MQFMT_STRING
The application message data can be either an SBCS string (single-byte character set), or a DBCS string (double-byte character set). Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_STRING_ARRAY is also defined; this has the same value as MQFMT_STRING, but is an array of characters instead of a string.

MQFMT_TRIGGER
The message is a trigger message, described by the MQTM structure; see "MQTM - Trigger

message" on page 2588 for details of this structure. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_TRIGGER_ARRAY is also defined; this has the same value as MQFMT_TRIGGER, but is an array of characters instead of a string.

MQFMT_WORK_INFO_HEADER
The message data begins with the work information header MQWIH, which is followed by the application data. The format name of the application data is given by the *Format* field in the MQWIH structure.

On z/OS, specify the MQGMO_CONVERT option on the MQGET call to convert the *user data* in messages that have format MQFMT_WORK_INFO_HEADER. However, the MQWIH structure itself is always returned in the queue manager's character set and encoding (that is, the MQWIH structure is converted whether or not the MQGMO_CONVERT option is specified).

For the C programming language, the constant MQFMT_WORK_INFO_HEADER_ARRAY is also defined; this has the same value as MQFMT_WORK_INFO_HEADER, but is an array of characters instead of a string.

MQFMT_XMIT_Q_HEADER
The message data begins with the transmission queue header MQXQH. The data from the original message immediately follows the MQXQH structure. The format name of the original message data is given by the *Format* field in the MQMD structure, which is part of the transmission queue header MQXQH. See "MQXQH - Transmission-queue header" on page 2612 for details of this structure.

COA and COD reports are not generated for messages that have a *Format* of MQFMT_XMIT_Q_HEADER.

For the C programming language, the constant MQFMT_XMIT_Q_HEADER_ARRAY is also defined; this has the same value as MQFMT_XMIT_Q_HEADER, but is an array of characters instead of a string.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

GroupId (MQBYTE24):

This is a byte string that is used to identify the particular message group or logical message to which the physical message belongs. *GroupId* is also used if segmentation is allowed for the message. In all these cases, *GroupId* has a non-null value, and one or more of the following flags is set in the *MsgFlags* field:

- MQMF_MSG_IN_GROUP
- MQMF_LAST_MSG_IN_GROUP
- MQMF_SEGMENT
- MQMF_LAST_SEGMENT
- MQMF_SEGMENTATION_ALLOWED

If none of these flags is set, *GroupId* has the special null value MQGI_NONE.

The application does not need to set this field on the MQPUT or MQGET call if:

- On the MQPUT call, MQPMO_LOGICAL_ORDER is specified.
- On the MQGET call, MQMO_MATCH_GROUP_ID is not specified.

These are the recommended ways of using these calls for messages that are not report messages. However, if the application requires more control, or the call is MQPUT1, the application must ensure that *GroupId* is set to an appropriate value.

Message groups and segments can be processed correctly only if the group identifier is unique. For this reason, *applications must not generate their own group identifiers* ; instead, applications must do one of the following:

- If MQPMO_LOGICAL_ORDER is specified, the queue manager automatically generates a unique group identifier for the first message in the group or segment of the logical message, and uses that group identifier for the remaining messages in the group or segments of the logical message, so the application does not need to take any special action. This is the recommended procedure.
- If MQPMO_LOGICAL_ORDER is not specified, the application must request the queue manager to generate the group identifier, by setting *GroupId* to MQGI_NONE on the first MQPUT or MQPUT1 call for a message in the group or segment of the logical message. The group identifier returned by the queue manager on output from that call must then be used for the remaining messages in the group or segments of the logical message. If a message group contains segmented messages, the same group identifier must be used for all segments and messages in the group.

  When MQPMO_LOGICAL_ORDER is not specified, messages in groups and segments of logical messages can be put in any order (for example, in reverse order), but the group identifier must be allocated by the *first* MQPUT or MQPUT1 call that is issued for any of those messages.

On input to the MQPUT and MQPUT1 calls, the queue manager uses the value described in Physical order on a queue. On output from the MQPUT and MQPUT1 calls, the queue manager sets this field to the value that was sent with the message if the object opened is a single queue and not a distribution list, but leaves it unchanged if the object opened is a distribution list. In the latter case, if the application needs to know the group identifiers generated, the application must provide MQPMR records containing the *GroupId* field.

On input to the MQGET call, the queue manager uses the value described in Table 204 on page 2349. On output from the MQGET call, the queue manager sets this field to the value for the message retrieved.

The following special value is defined:

MQGI_NONE
No group identifier specified.

The value is binary zero for the length of the field. This is the value that is used for messages that are not in groups, not segments of logical messages, and for which segmentation is not allowed.

For the C programming language, the constant MQGI_NONE_ARRAY is also defined; this has the same value as MQGI_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_GROUP_ID_LENGTH. The initial value of this field is MQGI_NONE. This field is ignored if *Version* is less than MQMD_VERSION_2.

MsgFlags (MQLONG):

MsgFlags are flags that specify attributes of the message, or control its processing.

MsgFlags are divided into the following categories:

- Segmentation flags
- Status flags

Segmentation flags: When a message is too big for a queue, an attempt to put the message on the queue typically fails. Segmentation is a technique whereby the queue manager or application splits the message into smaller pieces called segments, and places each segment on the queue as a separate physical message. The application that retrieves the message can either retrieve the segments one by one, or request the queue manager to reassemble the segments into a single message that is returned by the MQGET call. The latter is achieved by specifying the MQGMO_COMPLETE_MSG option on the MQGET call, and supplying a buffer that is big enough to accommodate the complete message. (See "MQGMO - Get-message options" on page 2330 for details of the MQGMO_COMPLETE_MSG option.) A message can be segmented at the sending queue manager, at an intermediate queue manager, or at the destination queue manager.

You can specify one of the following to control the segmentation of a message:

MQMF_SEGMENTATION_INHIBITED
This option prevents the message being broken into segments by the queue manager. If specified for a message that is already a segment, this option prevents the segment being broken into smaller segments.

The value of this flag is binary zero. This is the default.

MQMF_SEGMENTATION_ALLOWED
This option allows the message to be broken into segments by the queue manager. If specified for a message that is already a segment, this option allows the segment to be broken into smaller segments. MQMF_SEGMENTATION_ALLOWED can be set without either MQMF_SEGMENT or MQMF_LAST_SEGMENT being set.

- On z/OS, the queue manager does not support the segmentation of messages. If a message is too big for the queue, the MQPUT or MQPUT1 call fails with reason code MQRC_MSG_TOO_BIG_FOR_Q. However, the MQMF_SEGMENTATION_ALLOWED option can still be specified, and allows the message to be segmented at a remote queue manager.

When the queue manager segments a message, the queue manager turns on the MQMF_SEGMENT flag in the copy of the MQMD that is sent with each segment, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call. For the last segment in the logical message, the queue manager also turns on the MQMF_LAST_SEGMENT flag in the MQMD that is sent with the segment.

Note: Take care when putting messages with MQMF_SEGMENTATION_ALLOWED but without MQPMO_LOGICAL_ORDER. If the message is:

- Not a segment, and
- Not in a group, and
- Not being forwarded,

the application must reset the *GroupId* field to MQGI_NONE before *each* MQPUT or MQPUT1 call, so that the queue manager can generate a unique group identifier for each message. If this is not done, unrelated messages can have the same group identifier, which might lead to incorrect processing subsequently. See the descriptions of the *GroupId* field and the MQPMO_LOGICAL_ORDER option for more information about when to reset the *GroupId* field.

The queue manager splits messages into segments as necessary so that the segments (plus any required header data) fit on the queue. However, there is a lower limit for the size of a segment generated by the queue manager, and only the last segment created from a message can be smaller than this limit (the lower limit for the size of an application-generated segment is one byte). Segments generated by the queue manager might be of unequal length. The queue manager processes the message as follows:

- User-defined formats are split on boundaries that are multiples of 16 bytes; the queue manager does not generate segments that are smaller than 16 bytes (other than the last segment).
- Built-in formats other than MQFMT_STRING are split at points appropriate to the nature of the data present. However, the queue manager never splits a message in the middle of an IBM MQ header structure. This means that a segment containing a single MQ header structure cannot be split further by the queue manager, and as a result the minimum possible segment size for that message is greater than 16 bytes.

  The second or later segment generated by the queue manager begins with one of the following:
  - An MQ header structure
  - The start of the application message data
  - Part of the way through the application message data
- MQFMT_STRING is split without regard for the nature of the data present (SBCS, DBCS, or mixed SBCS/DBCS). When the string is DBCS or mixed SBCS/DBCS, this might result in segments that cannot be converted from one character set to another. The queue manager never splits MQFMT_STRING messages into segments that are smaller than 16 bytes (other than the last segment).
- The queue manager sets the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQMD of each segment to describe correctly the data present at the *start* of the segment; the format name is either the name of a built-in format, or the name of a user-defined format.
- The *Report* field in the MQMD of segments with *Offset* greater than zero is modified. For each report type, if the report option is MQRO_*_WITH_DATA, but the segment cannot contain any of the first 100 bytes of user data (that is, the data following any IBM MQ header structures that may be present), the report option is changed to MQRO_*.

The queue manager follows the above rules, but otherwise splits messages unpredictably; do not make assumptions about where a message is split.

For *persistent* messages, the queue manager can perform segmentation only within a unit of work:

- If the MQPUT or MQPUT1 call is operating within a user-defined unit of work, that unit of work is used. If the call fails during the segmentation process, the queue manager removes any segments that were placed on the queue as a result of the failing call. However, the failure does not prevent the unit of work being committed successfully.
- If the call is operating outside a user-defined unit of work, and there is no user-defined unit of work in existence, the queue manager creates a unit of work just for the duration of the call. If the call is successful, the queue manager commits the unit of work automatically. If the call fails, the queue manager backs out the unit of work.
- If the call is operating outside a user-defined unit of work, but a user-defined unit of work exists, the queue manager cannot perform segmentation. If the message does not require segmentation, the call can still succeed. But if the message requires segmentation, the call fails with reason code MQRC_UOW_NOT_AVAILABLE.

For *nonpersistent* messages, the queue manager does not require a unit of work to be available in order to perform segmentation.

Take special care when converting data in messages that might be segmented:

- If the receiving application converts data on the MQGET call, and specifies the MQGMO_COMPLETE_MSG option, the data-conversion exit is passed the complete message for the exit to convert, and the fact that the message was segmented is apparent to the exit.

- If the receiving application retrieves one segment at a time, the data-conversion exit is invoked to convert one segment at a time. The exit must therefore convert the data in a segment independently of the data in any of the other segments.

  If the nature of the data in the message is such that arbitrary segmentation of the data on 16-byte boundaries might result in segments that cannot be converted by the exit, or the format is MQFMT_STRING and the character set is DBCS or mixed SBCS/DBCS, the sending application must create and put the segments, specifying MQMF_SEGMENTATION_INHIBITED to suppress further segmentation. In this way, the sending application can ensure that each segment contains sufficient information to allow the data-conversion exit to convert the segment successfully.
- If sender conversion is specified for a sending message channel agent (MCA), the MCA converts only messages that are not segments of logical messages; the MCA never attempts to convert messages that are segments.

This flag is an input flag on the MQPUT and MQPUT1 calls, and an output flag on the MQGET call. On the latter call, the queue manager also echoes the value of the flag to the *Segmentation* field in MQGMO.

The initial value of this flag is MQMF_SEGMENTATION_INHIBITED.

Status flags: These are flags that indicate whether the physical message belongs to a message group, is a segment of a logical message, both, or neither. One or more of the following can be specified on the MQPUT or MQPUT1 call, or returned by the MQGET call:

MQMF_MSG_IN_GROUP
Message is a member of a group.

MQMF_LAST_MSG_IN_GROUP
Message is the last logical message in a group.

If this flag is set, the queue manager turns on MQMF_MSG_IN_GROUP in the copy of MQMD that is sent with the message, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call.

It is valid for a group to consist of only one logical message. If this is the case, MQMF_LAST_MSG_IN_GROUP is set, but the *MsgSeqNumber* field has the value one.

MQMF_SEGMENT
Message is a segment of a logical message.

When MQMF_SEGMENT is specified without MQMF_LAST_SEGMENT, the length of the application message data in the segment (*excluding* the lengths of any IBM MQ header structures that might be present) must be at least one. If the length is zero, the MQPUT or MQPUT1 call fails with reason code MQRC_SEGMENT_LENGTH_ZERO.

On z/OS, this option is not supported if the message is being put on a queue that has an index type of MQIT_GROUP_ID.

MQMF_LAST_SEGMENT
Message is the last segment of a logical message.

If this flag is set, the queue manager turns on MQMF_SEGMENT in the copy of MQMD that is sent with the message, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call.

A logical message can consist of only one segment. If so, MQMF_LAST_SEGMENT is set, but the *Offset* field has the value zero.

When MQMF_LAST_SEGMENT is specified, the length of the application message data in the segment (*excluding* the lengths of any header structures that might be present) can be zero.

On z/OS, this option is not supported if the message is being put on a queue that has an index type of MQIT_GROUP_ID.

The application must ensure that these flags are set correctly when putting messages. If MQPMO_LOGICAL_ORDER is specified, or was specified on the preceding MQPUT call for the queue handle, the settings of the flags must be consistent with the group and segment information retained by the queue manager for the queue handle. The following conditions apply to *successive* MQPUT calls for the queue handle when MQPMO_LOGICAL_ORDER is specified:

- If there is no current group or logical message, all these flags (and combinations of them) are valid.
- Once MQMF_MSG_IN_GROUP has been specified, it must remain on until MQMF_LAST_MSG_IN_GROUP is specified. The call fails with reason code MQRC_INCOMPLETE_GROUP if this condition is not satisfied.
- Once MQMF_SEGMENT has been specified, it must remain on until MQMF_LAST_SEGMENT is specified. The call fails with reason code MQRC_INCOMPLETE_MSG if this condition is not satisfied.
- Once MQMF_SEGMENT has been specified without MQMF_MSG_IN_GROUP, MQMF_MSG_IN_GROUP must remain *off* until after MQMF_LAST_SEGMENT has been specified. The call fails with reason code MQRC_INCOMPLETE_MSG if this condition is not satisfied.

Physical order on a queue shows the valid combinations of the flags, and the values used for various fields.

These flags are input flags on the MQPUT and MQPUT1 calls, and output flags on the MQGET call. On the latter call, the queue manager also echoes the values of the flags to the *GroupStatus* and *SegmentStatus* fields in MQGMO.

You cannot use grouped or segmented messages with Publish/Subscribe.

Default flags: The following can be specified to indicate that the message has default attributes:

MQMF_NONE
No message flags (default message attributes).

This inhibits segmentation, and indicates that the message is not in a group and is not a segment of a logical message. MQMF_NONE is defined to aid program documentation. It is not intended that this flag be used with any other, but as its value is zero, such use cannot be detected.

The *MsgFlags* field is partitioned into subfields; for details see "Report options and message flags" on page 2905.

The initial value of this field is MQMF_NONE. This field is ignored if *Version* is less than MQMD_VERSION_2.

MsgId (MQBYTE24):

This is a byte string that is used to distinguish one message from another. Generally, no two messages should have the same message identifier, although this is not disallowed by the queue manager. The message identifier is a permanent property of the message, and persists across restarts of the queue manager. Because the message identifier is a byte string and not a character string, the message identifier is not converted between character sets when the message flows from one queue manager to another.

For the MQPUT and MQPUT1 calls, if MQMI_NONE or MQPMO_NEW_MSG_ID is specified by the application, the queue manager generates a unique message identifier[1] when the message is put, and places it in the message descriptor sent with the message. The queue manager also returns this message identifier in the message descriptor belonging to the sending application. The application can use this value to record information about particular messages, and to respond to queries from other parts of the application.

If the message is being put to a topic, the queue manager generates unique message identifiers as necessary for each message published. If MQPMO_NEW_MSG_ID is specified by the application, the queue manager generates a unique message identifier to return on output. If MQMI_NONE is specified by the application, the value of the *MsgId* field in the MQMD is unchanged on return from the call.

See the description of MQPMO_RETAIN in "MQPMO options (MQLONG)" on page 2484 for more details about retained publications.

If the message is being put to a distribution list, the queue manager generates unique message identifiers as necessary, but the value of the *MsgId* field in MQMD is unchanged on return from the call, even if MQMI_NONE or MQPMO_NEW_MSG_ID was specified. If the application needs to know the message identifiers generated by the queue manager, the application must provide MQPMR records containing the *MsgId* field.

The sending application can also specify a value for the message identifier other than MQMI_NONE; this stops the queue manager generating a unique message identifier. An application that is forwarding a message can use this to propagate the message identifier of the original message.

The queue manager does not use this field except to:

- Generate a unique value if requested, as described above
- Deliver the value to the application that issues the get request for the message
- Copy the value to the *CorrelId* field of any report message that it generates about this message (depending on the *Report* options)

When the queue manager or a message channel agent generates a report message, it sets the *MsgId* field in the way specified by the *Report* field of the original message, either MQRO_NEW_MSG_ID or MQRO_PASS_MSG_ID. Applications that generate report messages must also do this.

For the MQGET call, *MsgId* is one of the five fields that can be used to retrieve a particular message from the queue. Normally the MQGET call returns the next message on the queue, but a particular message can be obtained by specifying one or more of the five selection criteria, in any combination; these fields are:

---

1. A *MsgId* generated by the queue manager consists of a 4-byte product identifier (AMQ¬ or CSQ¬ in either ASCII or EBCDIC, where ¬ represents a blank character), followed by a product-specific implementation of a unique string. In IBM MQ this contains the first 12 characters of the queue manager name, and a value derived from the system clock. All queue managers that can intercommunicate must therefore have names that differ in the first 12 characters, in order to ensure that message identifiers are unique. The ability to generate a unique string also depends on the system clock not being changed backward. To eliminate the possibility of a message identifier generated by the queue manager duplicating one generated by the application, the application must avoid generating identifiers with initial characters in the range A through I in ASCII or EBCDIC (X'41' through X'49' and X'C1' through X'C9'). However, the application is not prevented from generating identifiers with initial characters in these ranges.

- *MsgId*
- *CorrelId*
- *GroupId*
- *MsgSeqNumber*
- *Offset*

The application sets one or more of these field to the values required, and then sets the corresponding MQMO_* match options in the *MatchOptions* field in MQGMO to use those fields as selection criteria. Only messages that have the specified values in those fields are candidates for retrieval. The default for the *MatchOptions* field (if not altered by the application) is to match both the message identifier and the correlation identifier.

On z/OS, the selection criteria that you can use are restricted by the type of index used for the queue. See the **IndexType** queue attribute for further details.

Normally, the message returned is the *first* message on the queue that satisfies the selection criteria. But if MQGMO_BROWSE_NEXT is specified, the message returned is the *next* message that satisfies the selection criteria; the scan for this message starts with the message *following* the current cursor position.

Note: The queue is scanned sequentially for a message that satisfies the selection criteria, so retrieval times are slower than if no selection criteria are specified, especially if many messages have to be scanned before a suitable one is found. The exceptions to this are:

- Multi an MQGET call by *CorrelId* on 64-bit Multiplatforms where the *CorrelId* index eliminates the need to perform a true sequential scan.
- z/OS an MQGET call by *IndexType* on z/OS.

In both these cases, retrieval performance is improved.

See Table 204 on page 2349 for more information about how selection criteria are used in various situations.

Specifying MQMI_NONE as the message identifier has the same effect as not specifying MQMO_MATCH_MSG_ID, that is, *any* message identifier matches.

This field is ignored if the MQGMO_MSG_UNDER_CURSOR option is specified in the **GetMsgOpts** parameter on the MQGET call.

On return from an MQGET call, the *MsgId* field is set to the message identifier of the message returned (if any).

The following special value can be used:

MQMI_NONE
No message identifier is specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQMI_NONE_ARRAY is also defined; this has the same value as MQMI_NONE, but is an array of characters instead of a string.

This is an input/output field for the MQGET, MQPUT, and MQPUT1 calls. The length of this field is given by MQ_MSG_ID_LENGTH. The initial value of this field is MQMI_NONE.

MsgSeqNumber (MQLONG):

This is the sequence number of a logical message within a group.

Sequence numbers start at 1, and increase by 1 for each new logical message in the group, up to a maximum of 999 999 999. A physical message that is not in a group has a sequence number of 1.

The application does not have to set this field on the MQPUT or MQGET call if:

- On the MQPUT call, MQPMO_LOGICAL_ORDER is specified.
- On the MQGET call, MQMO_MATCH_MSG_SEQ_NUMBER is not specified.

These are the recommended ways of using these calls for messages that are not report messages. However, if the application requires more control, or the call is MQPUT1, the application must ensure that *MsgSeqNumber* is set to an appropriate value.

On input to the MQPUT and MQPUT1 calls, the queue manager uses the value described in Physical order on a queue. On output from the MQPUT and MQPUT1 calls, the queue manager sets this field to the value that was sent with the message.

On input to the MQGET call, the queue manager uses the value shown in Table 204 on page 2349. On output from the MQGET call, the queue manager sets this field to the value for the message retrieved.

The initial value of this field is one. This field is ignored if *Version* is less than MQMD_VERSION_2.

MsgType (MQLONG):

This indicates the type of the message. Message types are grouped as follows:

MQMT_SYSTEM_FIRST
Lowest value for system-defined message types.

MQMT_SYSTEM_LAST
Highest value for system-defined message types.

The following values are currently defined within the system range:

MQMT_DATAGRAM
The message is one that does not require a reply.

MQMT_REQUEST
The message is one that requires a reply.

Specify the name of the queue to which to send the reply in the *ReplyToQ* field. The *Report* field indicates how to set the *MsgId* and *CorrelId* of the reply.

MQMT_REPLY
The message is the reply to an earlier request message (MQMT_REQUEST). The message must be sent to the queue indicated by the *ReplyToQ* field of the request message. Use the *Report* field of the request to control how to set the *MsgId* and *CorrelId* of the reply.

Note: The queue manager does not enforce the request-reply relationship; this is an application responsibility.

MQMT_REPORT
The message is reporting on some expected or unexpected occurrence, usually related to some other message (for example, a request message was received that contained data that was not valid). Send the message to the queue indicated by the *ReplyToQ* field of the message descriptor of the original message. Set the *Feedback* field s to indicate the nature of the report. Use the *Report* field of the original message to control how to set the *MsgId* and *CorrelId* of the report message.

Report messages generated by the queue manager or message channel agent are always sent to the *ReplyToQ* queue, with the *Feedback* and *CorrelId* fields set as described above.

Application-defined values can also be used. They must be within the following range:

MQMT_APPL_FIRST
Lowest value for application-defined message types.

MQMT_APPL_LAST
Highest value for application-defined message types.

For the MQPUT and MQPUT1 calls, the *MsgType* value must be within either the system-defined range or the application-defined range; if it is not, the call fails with reason code MQRC_MSG_TYPE_ERROR.

This is an output field for the MQGET call, and an input field for MQPUT and MQPUT1 calls. The initial value of this field is MQMT_DATAGRAM.

Offset (MQLONG):

This is the offset in bytes of the data in the physical message from the start of the logical message of which the data forms part. This data is called a *segment*. The offset is in the range 0 through 999 999 999. A physical message that is not a segment of a logical message has an offset of zero.

The application does not need to set this field on the MQPUT or MQGET call if:

- On the MQPUT call, MQPMO_LOGICAL_ORDER is specified.
- On the MQGET call, MQMO_MATCH_OFFSET is not specified.

These are the recommended ways of using these calls for messages that are not report messages. However, if the application does not comply with these conditions, or the call is MQPUT1, the application must ensure that *Offset* is set to an appropriate value.

On input to the MQPUT and MQPUT1 calls, the queue manager uses the value described in Physical order on a queue. On output from the MQPUT and MQPUT1 calls, the queue manager sets this field to the value that was sent with the message.

For a report message reporting on a segment of a logical message, the *OriginalLength* field (provided it is not MQOL_UNDEFINED) is used to update the offset in the segment information retained by the queue manager.

On input to the MQGET call, the queue manager uses the value shown in Table 204 on page 2349. On output from the MQGET call, the queue manager sets this field to the value for the message retrieved.

The initial value of this field is zero. This field is ignored if *Version* is less than MQMD_VERSION_2.

OriginalLength (MQLONG):

This field is relevant only for report messages that are segments. It specifies the length of the message segment to which the report message relates; it does not specify the length of the logical message of which the segment forms part, or the length of the data in the report message.

Note: When generating a report message for a message that is a segment, the queue manager and message channel agent copy into the MQMD for the report message the *GroupId*, *MsgSeqNumber*, *Offset*, and *MsgFlags*, fields from the original message. As a result, the report message is also a segment. Applications that generate report messages must do the same, and set the *OriginalLength* field correctly.

The following special value is defined:

MQOL_UNDEFINED
Original length of message not defined.

OriginalLength is an input field on the MQPUT and MQPUT1 calls, but the value that the application provides is accepted only in particular circumstances:

- If the message being put is a segment and is also a report message, the queue manager accepts the value specified. The value must be:
  - Greater than zero if the segment is not the last segment
  - Not less than zero if the segment is the last segment
  - Not less than the length of data present in the message

  If these conditions are not satisfied, the call fails with reason code MQRC_ORIGINAL_LENGTH_ERROR.
- If the message being put is a segment but not a report message, the queue manager ignores the field and uses the length of the application message data instead.
- In all other cases, the queue manager ignores the field and uses the value MQOL_UNDEFINED instead.

This is an output field on the MQGET call.

The initial value of this field is MQOL_UNDEFINED. This field is ignored if *Version* is less than MQMD_VERSION_2.

Persistence (MQLONG):

This indicates whether the message survives system failures and restarts of the queue manager. For the MQPUT and MQPUT1 calls, the value must be one of the following:

MQPER_PERSISTENT
The message survives system failures and restarts of the queue manager. Once the message has been put, and the unit of work in which it was put has been committed (if the message is put as part of a unit of work), the message is preserved on auxiliary storage. It remains there until the message is removed from the queue, and the unit of work in which it was got has been committed (if the message is retrieved as part of a unit of work).

When a persistent message is sent to a remote queue, a store-and-forward mechanism holds the message at each queue manager along the route to the destination, until the message is known to have arrived at the next queue manager.

Persistent messages cannot be placed on:

- Temporary dynamic queues
- Shared queues that map to a CFSTRUCT object at CFLEVEL(2) or below, or where the CFSTRUCT object is defined as RECOVER(NO).

Persistent messages can be placed on permanent dynamic queues, and predefined queues.

MQPER_NOT_PERSISTENT
The message does not usually survive system failures or queue manager restarts. This applies even if an intact copy of the message is found on auxiliary storage when the queue manager restarts.

In the case of NPMCLASS (HIGH) queues nonpersistent messages survive a normal queue manager shutdown and restart.

In the case of shared queues, nonpersistent messages survive queue manager restarts in the queue-sharing group, but do not survive failures of the coupling facility used to store messages on the shared queues.

MQPER_PERSISTENCE_AS_Q_DEF

- If the queue is a cluster queue, the persistence of the message is taken from the **DefPersistence** attribute defined at the *destination* queue manager that owns the particular instance of the queue on which the message is placed. Usually, all instances of a cluster queue have the same value for the **DefPersistence** attribute, although this is not mandated.

  The value of *DefPersistence* is copied into the *Persistence* field when the message is placed on the destination queue. If *DefPersistence* is changed subsequently, messages that have already been placed on the queue are not affected.
- If the queue is not a cluster queue, the persistence of the message is taken from the **DefPersistence** attribute defined at the *local* queue manager, even if the destination queue manager is remote.

  If there is more than one definition in the queue-name resolution path, the default persistence is taken from the value of this attribute in the *first* definition in the path. This can be:
  - An alias queue
  - A local queue
  - A local definition of a remote queue
  - A queue manager alias
  - A transmission queue (for example, the *DefXmitQName* queue)

  The value of *DefPersistence* is copied into the *Persistence* field when the message is put. If *DefPersistence* is changed subsequently, messages that have already been put are not affected.

Both persistent and nonpersistent messages can exist on the same queue.

When replying to a message, applications must use the persistence of the request message for the reply message.

For an MQGET call, the value returned is either MQPER_PERSISTENT or MQPER_NOT_PERSISTENT.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQPER_PERSISTENCE_AS_Q_DEF.

Priority (MQLONG):

For the MQPUT and MQPUT1 calls, the value must be greater than or equal to zero; zero is the lowest priority. The following special value can also be used:

MQPRI_PRIORITY_AS_Q_DEF

- If the queue is a cluster queue, the priority for the message is taken from the **DefPriority** attribute as defined at the *destination* queue manager that owns the particular instance of the queue on which the message is placed. Usually, all instances of a cluster queue have the same value for the **DefPriority** attribute, although this is not mandated.

  The value of *DefPriority* is copied into the *Priority* field when the message is placed on the destination queue. If *DefPriority* is changed subsequently, messages that have already been placed on the queue are not affected.
- If the queue is not a cluster queue, the priority for the message is taken from the **DefPriority** attribute as defined at the *local* queue manager, even if the destination queue manager is remote.

  If there is more than one definition in the queue-name resolution path, the default priority is taken from the value of this attribute in the *first* definition in the path. This can be:
  - An alias queue
  - A local queue
  - A local definition of a remote queue
  - A queue manager alias
  - A transmission queue (for example, the *DefXmitQName* queue)

  The value of *DefPriority* is copied into the *Priority* field when the message is put. If *DefPriority* is changed subsequently, messages that have already been put are not affected.

The value returned by the MQGET call is always greater than or equal to zero; the value MQPRI_PRIORITY_AS_Q_DEF is never returned.

If a message is put with a priority greater than the maximum supported by the local queue manager (this maximum is given by the **MaxPriority** queue manager attribute), the message is accepted by the queue manager, but placed on the queue at the queue manager's maximum priority; the MQPUT or MQPUT1 call completes with MQCC_WARNING and reason code MQRC_PRIORITY_EXCEEDS_MAXIMUM. However, the *Priority* field retains the value specified by the application that put the message.

On z/OS, if a message with a MsgSeqNumber of 1 is put to a queue that has a message delivery sequence of MQMDS_PRIORITY and an index type of MQIT_GROUP_ID, the queue might treat the message with a different priority. If the message was placed on the queue with a priority of 0 or 1, it is processed as though it has a priority of 2. This is because the order of messages placed on this type of queue is optimized to enable efficient group completeness tests. For more information on the message delivery sequence MQMDS_PRIORITY and the index type MQIT_GROUP_ID, see MsgDeliverySequence attribute.

When replying to a message, applications must use the priority of the request message for the reply message. In other situations, specifying MQPRI_PRIORITY_AS_Q_DEF allows priority tuning to be carried out without changing the application.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQPRI_PRIORITY_AS_Q_DEF.

PutApplName (MQCHAR28):

This is the name of application that put the message, and is part of the *origin context* of the message. The contents differ between platforms, and might differ between releases.

For more information about message context, see "Overview for MQMD" on page 2388 and Message context.

The format of *PutApplName* depends on the value of *PutApplType* and can change from one release to another. Changes are rare, but do happen if the environment changes.

When the queue manager sets this field (that is, for all options except MQPMO_SET_ALL_CONTEXT), it sets the field to a value that is determined by the environment:

- z/OS On z/OS, the queue manager uses:
  - For z/OS batch, the 8-character job name from the JES JOB card
  - For TSO, the 7-character TSO user identifier
  - For CICS, the 8-character applid, followed by the 4-character tranid
  - For IMS, the 8-character IMS system identifier, followed by the 8-character PSB name
  - For XCF, the 8-character XCF group name, followed by the 16-character XCF member name
  - For a message generated by a queue manager, the first 28 characters of the queue manager name
  - For distributed queuing without CICS, the 8-character jobname of the channel initiator followed by the 8-character name of the module putting to the dead-letter queue followed by an 8-character task identifier.

  The name or names are each padded to the right with blanks, as is any space in the remainder of the field. Where there is more than one name, there is no separator between them.
- Windows On Windows systems, the queue manager uses the following names:
  - For a CICS application, the CICS transaction name
  - For a non-CICS application, the rightmost 28 characters of the fully-qualified name of the executable
- IBM i On IBM i, the queue manager uses the fully-qualified job name.
- UNIX On UNIX, the queue manager uses the following names:
  - For a CICS application, the CICS transaction name
  - For a non-CICS application, MQ asks the operating system for the name of the process. This is returned as the program file name, without full path. Then MQ places this process name in the MQMD.PutApplName field as follows:

    AIX If the name is less than or equal to 28 bytes, then the name is inserted, padded to the right with spaces.

    If the name is greater than 28 bytes, then the leftmost 28 bytes of the name are inserted.

    Linux and Solaris
    If the name is less than or equal to 15 bytes, then the name is inserted, padded to the right with spaces.

    If the name is greater than 15 bytes, then the leftmost 15 bytes of the name are inserted, padded to the right with spaces.

    HP-UX
    If the name is less than or equal to 14 bytes, then the name is inserted, padded to the right with spaces.

    If the name is greater than 14 bytes, then the leftmost 14 bytes of the name are inserted, padded to the right with spaces.

For example, if you run `/opt/mqm/samp/bin/amqsput QNAME QMNAME`, then the PutApplName is `'amqsput                        '`. There are 21 space characters of padding in this MQCHAR28 field. Note that the full path including `/opt/mqm/samp/bin` is not included in the PutApplName.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the **PutMsgOpts** parameter. Any information following a null character within the field is discarded. The null character and any following characters are converted to blanks by the queue manager. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

PutApplType (MQLONG):

This is the type of application that put the message, and is part of the **origin context** of the message. For more information about message context, see "Overview for MQMD" on page 2388 and Message context.

PutApplType can have one of the following standard types. You can also define your own types, but only with values in the range MQAT_USER_FIRST through MQAT_USER_LAST.

MQAT_AIX
AIX application (same value as MQAT_UNIX).

MQAT_AMQP
AMQP protocol application

MQAT_BROKER
Broker.

MQAT_CICS
CICS transaction.

MQAT_CICS_BRIDGE
CICS bridge.

MQAT_CICS_VSE
CICS/VSE transaction.

MQAT_DOS
IBM MQ MQI client application on PC DOS.

MQAT_DQM
Distributed queue manager agent.

MQAT_GUARDIAN
Tandem Guardian application (same value as MQAT_NSK).

MQAT_IMS
IMS application.

MQAT_IMS_BRIDGE
IMS bridge.

MQAT_JAVA
Java.

MQAT_MVS
MVS or TSO application (same value as MQAT_ZOS).

MQAT_NOTES_AGENT
Lotus Notes Agent application.

MQAT_NSK
HP Integrity NonStop Server application.

MQAT_OS390
OS/390 application (same value as MQAT_ZOS).

MQAT_OS400
IBM i application.

MQAT_QMGR
Queue manager.

MQAT_UNIX
UNIX application.

MQAT_VOS
Stratus VOS application.

MQAT_WINDOWS
16-bit Windows application.

MQAT_WINDOWS_NT
32-bit Windows application.

MQAT_WLM
z/OS workload manager application.

MQAT_XCF
XCF.

MQAT_ZOS
z/OS application.

MQAT_DEFAULT
Default application type.

This is the default application type for the platform on which the application is running.

Note: The value of this constant is environment-specific. Because of this, always compile the application using the header, include, or COPY files that are appropriate to the platform on which the application will run.

MQAT_UNKNOWN
Use this value to indicate that the application type is unknown, even though other context information is present.

MQAT_USER_FIRST
Lowest value for user-defined application type.

MQAT_USER_LAST
Highest value for user-defined application type.

The following special value can also occur:

MQAT_NO_CONTEXT
This value is set by the queue manager when a message is put with no context (that is, the MQPMO_NO_CONTEXT context option is specified).

When a message is retrieved, *PutApplType* can be tested for this value to decide whether the message has context (it is recommended that *PutApplType* is never set to MQAT_NO_CONTEXT, by an application using MQPMO_SET_ALL_CONTEXT, if any of the other context fields are nonblank).

When the queue manager generates this information as a result of an application put, the field is set to a value that is determined by the environment. On IBM i, it is set to MQAT_OS400; the queue manager never uses MQAT_CICS on IBM i.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the **PutMsgOpts** parameter. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

This is an output field for the MQGET call. The initial value of this field is MQAT_NO_CONTEXT.

PutDate (MQCHAR8):

This is the date when the message was put, and is part of the **origin context** of the message. For more information about message context, see "Overview for MQMD" on page 2388 and Message context.

The format used for the date when this field is generated by the queue manager is:

- YYYYMMDD

where the characters represent:

YYYY year (four numeric digits)

MM month of year (01 through 12)

DD day of month (01 through 31)

Greenwich Mean Time (GMT) is used for the *PutDate* and *PutTime* fields, subject to the system clock being set accurately to GMT.

If the message was put as part of a unit of work, the date is that when the message was put, and not the date when the unit of work was committed.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the **PutMsgOpts** parameter. The contents of the field are not checked by the queue manager, except that any information following a null character within the field is discarded. The queue manager converts the null character and any following characters to blanks. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

This is an output field for the MQGET call. The length of this field is given by MQ_PUT_DATE_LENGTH. The initial value of this field is the null string in C, and 8 blank characters in other programming languages.

PutTime (MQCHAR8):

This is the time when the message was put, and is part of the **origin context** of the message. For more information about message context, see "Overview for MQMD" on page 2388 and Message context.

The format used for the time when this field is generated by the queue manager is:

- HHMMSSTH

where the characters represent (in order):

HH hours (00 through 23)

MM minutes (00 through 59)

SS seconds (00 through 59; see note)

T tenths of a second (0 through 9)

H hundredths of a second (0 through 9)

Note: If the system clock is synchronized to a very accurate time standard, it is possible on rare occasions for 60 or 61 to be returned for the seconds in *PutTime*. This happens when leap seconds are inserted into the global time standard.

Greenwich Mean Time (GMT) is used for the *PutDate* and *PutTime* fields, subject to the system clock being set accurately to GMT.

If the message was put as part of a unit of work, the time is that when the message was put, and not the time when the unit of work was committed.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the **PutMsgOpts** parameter. The queue manager does not check the contents of the field, except that any information following a null character within the field is discarded. The queue manager converts the null character and any following characters to blanks. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

This is an output field for the MQGET call. The length of this field is given by MQ_PUT_TIME_LENGTH. The initial value of this field is the null string in C, and 8 blank characters in other programming languages.

ReplyToQ (MQCHAR48):

This is the name of the message queue to which the application that issued the get request for the message sends MQMT_REPLY and MQMT_REPORT messages. The name is the local name of a queue that is defined on the queue manager identified by *ReplyToQMgr*. This queue must not be a model queue, although the sending queue manager does not verify this when the message is put.

For the MQPUT and MQPUT1 calls, this field must not be blank if the *MsgType* field has the value MQMT_REQUEST, or if any report messages are requested by the *Report* field. However, the value specified (or substituted) is passed on to the application that issues the get request for the message, whatever the message type.

If the *ReplyToQMgr* field is blank, the local queue manager looks up the *ReplyToQ* name in its own queue definitions. If a local definition of a remote queue exists with this name, the *ReplyToQ* value in the transmitted message is replaced by the value of the **RemoteQName** attribute from the definition of the remote queue, and this value is returned in the message descriptor when the receiving application issues an MQGET call for the message. If a local definition of a remote queue does not exist, *ReplyToQ* is unchanged.

If the name is specified, it can contain trailing blanks; the first null character and characters following it are treated as blanks. Otherwise no check is made that the name satisfies the naming rules for queues; this is also true for the name transmitted, if the *ReplyToQ* is replaced in the transmitted message. The only check made is that a name has been specified, if the circumstances require it.

If a reply-to queue is not required, set the *ReplyToQ* field to blanks, or (in the C programming language) to the null string, or to one or more blanks followed by a null character; do not leave the field uninitialized.

For the MQGET call, the queue manager always returns the name padded with blanks to the length of the field.

If a message that requires a report message cannot be delivered, and the report message also cannot be delivered to the queue specified, both the original message and the report message go to the dead-letter (undelivered-message) queue (see the **DeadLetterQName** attribute described in "Attributes for the queue manager" on page 2792).

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ReplyToQMgr (MQCHAR48):

This is the name of the queue manager to which to send the reply message or report message. *ReplyToQ* is the local name of a queue that is defined on this queue manager.

If the *ReplyToQMgr* field is blank, the local queue manager looks up the *ReplyToQ* name in its queue definitions. If a local definition of a remote queue exists with this name, the *ReplyToQMgr* value in the transmitted message is replaced by the value of the **RemoteQMgrName** attribute from the definition of the remote queue, and this value is returned in the message descriptor when the receiving application issues an MQGET call for the message. If a local definition of a remote queue does not exist, the *ReplyToQMgr* that is transmitted with the message is the name of the local queue manager.

If the name is specified, it can contain trailing blanks; the first null character and characters following it are treated as blanks. Otherwise no check is made that the name satisfies the naming rules for queue managers, or that this name is known to the sending queue manager; this is also true for the name transmitted, if the *ReplyToQMgr* is replaced in the transmitted message.

If a reply-to queue is not required, set the *ReplyToQMgr* field to blanks, or (in the C programming language) to the null string, or to one or more blanks followed by a null character; do not leave the field uninitialized.

For the MQGET call, the queue manager always returns the name padded with blanks to the length of the field.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

Report (MQLONG):

A report message is a message about another message, used to inform an application about expected or unexpected events that relate to the original message. The *Report* field enables the application sending the original message to specify which report messages are required, whether the application message data is to be included in them, and also (for both reports and replies) how the message and correlation identifiers in the report or reply message are to be set. Any or all (or none) of the following types of report message can be requested:

- Exception
- Expiration
- Confirm on arrival (COA)
- Confirm on delivery (COD)
- Positive action notification (PAN)
- Negative action notification (NAN)

You can specify one or more of these options. To specify more than one option, either add the values together (do not add the same constant more than once), or combine the values using the bitwise OR operation (if the programming language supports bit operations).

The application that receives the report message can determine the reason that the report was generated by examining the *Feedback* field in the MQMD; see the *Feedback* field for more details.

The use of report options when putting a message to a topic can cause zero, one, or many report messages to be generated and sent to the application. This is because the publication message may be sent to zero, one, or many subscribing applications.

Exception options: Specify one of the options listed to request an exception report message.

MQRO_EXCEPTION
A message channel agent generates this type of report when a message is sent to another queue manager and the message cannot be delivered to the specified destination queue. For example, the destination queue or an intermediate transmission queue might be full, or the message might be too big for the queue.

Generation of the exception report message depends on the persistence of the original message, and the speed of the message channel (normal or fast) through which the original message travels:

- For all persistent messages, and for nonpersistent messages traveling through normal message channels, the exception report is generated only if the action specified by the sending application for the error condition can be completed successfully. The sending application can specify one of the following actions to control the disposition of the original message when the error condition arises:
  - MQRO_DEAD_LETTER_Q (this places the original message on the dead-letter queue).
  - MQRO_DISCARD_MSG (this discards the original message).

  If the action specified by the sending application cannot be completed successfully, the original message is left on the transmission queue, and no exception report message is generated.
- For nonpersistent messages traveling through fast message channels, the original message is removed from the transmission queue and the exception report generated *even if* the specified action for the error condition cannot be completed successfully. For example, if MQRO_DEAD_LETTER_Q is specified, but the original message cannot be placed on the dead-letter queue because that queue is full, the exception report message is generated and the original message discarded.

  For more information about normal and fast message channels, see Nonpersistent message speed (NPMSPEED).

An exception report is not generated if the application that put the original message can be notified synchronously of the problem by means of the reason code returned by the MQPUT or MQPUT1 call.

Applications can also send exception reports, to indicate that a message cannot be processed (for example, because it is a debit transaction that would cause the account to exceed its credit limit).

Message data from the original message is not included with the report message.

Do not specify more than one of MQRO_EXCEPTION, MQRO_EXCEPTION_WITH_DATA, and MQRO_EXCEPTION_WITH_FULL_DATA.

MQRO_EXCEPTION_WITH_DATA
This is the same as MQRO_EXCEPTION, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of MQRO_EXCEPTION, MQRO_EXCEPTION_WITH_DATA, and MQRO_EXCEPTION_WITH_FULL_DATA.

MQRO_EXCEPTION_WITH_FULL_DATA
Exception reports with full data required.

This is the same as MQRO_EXCEPTION, except that all the application message data from the original message is included in the report message.

Do not specify more than one of MQRO_EXCEPTION, MQRO_EXCEPTION_WITH_DATA, and MQRO_EXCEPTION_WITH_FULL_DATA.

Expiration options: Specify one of the options listed to request an expiration report message.

MQRO_EXPIRATION
This type of report is generated by the queue manager if the message is discarded before delivery to an application because its expiry time has passed (see the *Expiry* field). If this option is not set, no report message is generated if a message is discarded for this reason (even if you specify one of the MQRO_EXCEPTION_* options).

Message data from the original message is not included with the report message.

Do not specify more than one of MQRO_EXPIRATION, MQRO_EXPIRATION_WITH_DATA, and MQRO_EXPIRATION_WITH_FULL_DATA.

MQRO_EXPIRATION_WITH_DATA
This is the same as MQRO_EXPIRATION, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of MQRO_EXPIRATION, MQRO_EXPIRATION_WITH_DATA, and MQRO_EXPIRATION_WITH_FULL_DATA.

MQRO_EXPIRATION_WITH_FULL_DATA
This is the same as MQRO_EXPIRATION, except that all the application message data from the original message is included in the report message.

Do not specify more than one of MQRO_EXPIRATION, MQRO_EXPIRATION_WITH_DATA, and MQRO_EXPIRATION_WITH_FULL_DATA.

Confirm-on-arrival options: Specify one of the options listed to request a confirm-on-arrival report message.

MQRO_COA

This type of report is generated by the queue manager that owns the destination queue when the message is placed on the destination queue. Message data from the original message is not included with the report message.

If the message is put as part of a unit of work, and the destination queue is a local queue, the COA report message generated by the queue manager can be retrieved only if the unit of work is committed.

A COA report is not generated if the *Format* field in the message descriptor is MQFMT_XMIT_Q_HEADER or MQFMT_DEAD_LETTER_HEADER. This prevents a COA report being generated if the message is put on a transmission queue, or is undeliverable and put on a dead-letter queue.

In the case of an IMS bridge queue, the COA report is generated when the message reaches the IMS queue (acknowledgment received from IMS) and not when the message is put in the MQ bridge queue. That means that if IMS is not active, no COA report is generated until IMS is started and a message is queued on the IMS queue.

The user that runs a program that puts a message with MQMD.Report=MQRO_COA must have +passid authority on the reply queue. If the user does not have +passid authority, the COA report message does not reach the reply queue. An attempt is made to put the report message on the dead letter queue.

Do not specify more than one of MQRO_COA, MQRO_COA_WITH_DATA, and MQRO_COA_WITH_FULL_DATA.

MQRO_COA_WITH_DATA
This is the same as MQRO_COA, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of MQRO_COA, MQRO_COA_WITH_DATA, and MQRO_COA_WITH_FULL_DATA.

MQRO_COA_WITH_FULL_DATA
This is the same as MQRO_COA, except that all the application message data from the original message is included in the report message.

Do not specify more than one of MQRO_COA, MQRO_COA_WITH_DATA, and MQRO_COA_WITH_FULL_DATA.

Confirm-on-delivery options: Specify one of the options listed to request a confirm-on-delivery report message.

MQRO_COD
This type of report is generated by the queue manager when an application retrieves the message from the destination queue in a way that deletes the message from the queue. Message data from the original message is not included with the report message.

If the message is retrieved as part of a unit of work, the report message is generated within the same unit of work, so that the report is not available until the unit of work is committed. If the unit of work is backed out, the report is not sent.

A COD report is not always generated if a message is retrieved with the MQGMO_MARK_SKIP_BACKOUT option. If the primary unit of work is backed out but the secondary unit of work is committed, the message is removed from the queue, but a COD report is not generated.

A COD report is not generated if the *Format* field in the message descriptor is MQFMT_DEAD_LETTER_HEADER. This prevents a COD report being generated if the message is undeliverable and put on a dead-letter queue.

MQRO_COD is not valid if the destination queue is an XCF queue.

Do not specify more than one of MQRO_COD, MQRO_COD_WITH_DATA, and MQRO_COD_WITH_FULL_DATA.

MQRO_COD_WITH_DATA
This is the same as MQRO_COD, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

If MQGMO_ACCEPT_TRUNCATED_MSG is specified on the MQGET call for the original message, and the message retrieved is truncated, the amount of application message data placed in the report message depends on the environment:

- On z/OS, it is the minimum of:
  - The length of the original message
  - The length of the buffer used to retrieve the message
  - 100 bytes.
- In other environments, it is the minimum of:
  - The length of the original message
  - 100 bytes.

MQRO_COD_WITH_DATA is not valid if the destination queue is an XCF queue.

Do not specify more than one of MQRO_COD, MQRO_COD_WITH_DATA, and MQRO_COD_WITH_FULL_DATA.

MQRO_COD_WITH_FULL_DATA
This is the same as MQRO_COD, except that all the application message data from the original message is included in the report message.

MQRO_COD_WITH_FULL_DATA is not valid if the destination queue is an XCF queue.

Do not specify more than one of MQRO_COD, MQRO_COD_WITH_DATA, and MQRO_COD_WITH_FULL_DATA.

Action-notification options: Specify one or both of the options listed to request that the receiving application send a positive-action or negative-action report message.

MQRO_PAN
This type of report is generated by the application that retrieves the message and acts upon it. It indicates that the action requested in the message has been performed successfully. The application generating the report determines whether any data is to be included with the report.

Other than conveying this request to the application retrieving the message, the queue manager takes no action based on this option. The retrieving application must generate the report if appropriate.

MQRO_NAN
This type of report is generated by the application that retrieves the message and acts upon it. It indicates that the action requested in the message has not been performed successfully. The application generating the report determines whether any data is to be included with the report. For example, you might want to include some data indicating why the request could not be performed.

Other than conveying this request to the application retrieving the message, the queue manager takes no action based on this option. The retrieving application must generate the report if appropriate.

The application must determine which conditions correspond to a positive action and which correspond to a negative action. However, if the request has been only partially performed, generate a NAN report rather than a PAN report if requested. Every possible condition must correspond to either a positive action, or a negative action, but not both.

Message-identifier options: Specify one of the options listed to control how the *MsgId* of the report message (or of the reply message) is to be set.

MQRO_NEW_MSG_ID
This is the default action, and indicates that if a report or reply is generated as a result of this message, a new *MsgId* is generated for the report or reply message.

MQRO_PASS_MSG_ID
If a report or reply is generated as a result of this message, the *MsgId* of this message is copied to the *MsgId* of the report or reply message.

The *MsgId* of a publication message will be different for each subscriber that receives a copy of the publication and therefore the *MsgId* copied into the report or reply message will be different for each one.

If this option is not specified, MQRO_NEW_MSG_ID is assumed.

Correlation-identifier options: Specify one of the options listed to control how the *CorrelId* of the report message (or of the reply message) is to be set.

MQRO_COPY_MSG_ID_TO_CORREL_ID
This is the default action, and indicates that if a report or reply is generated as a result of this message, the *MsgId* of this message is copied to the *CorrelId* of the report or reply message.

The *MsgId* of a publication message will be different for each subscriber that receives a copy of the publication and therefore the *MsgId* copied into the *CorrelId* of the report or reply message will be different for each one.

MQRO_PASS_CORREL_ID
If a report or reply is generated as a result of this message, the *CorrelId* of this message is copied to the *CorrelId* of the report or reply message.

The *CorrelId* of a publication message will be specific to a subscriber unless it uses the MQSO_SET_CORREL_ID option and sets the SubCorrelId field in the MQSD to MQCI_NONE. Therefore it is possible that the *CorrelId* copied into the *CorrelId* of the report or reply message will be different for each one.

If this option is not specified, MQRO_COPY_MSG_ID_TO_CORREL_ID is assumed.

Servers replying to requests or generating report messages must check whether the MQRO_PASS_MSG_ID or MQRO_PASS_CORREL_ID options were set in the original message. If they were, the servers must take the action described for those options. If neither is set, the servers must take the corresponding default action.

Disposition options: Specify one of the options listed to control the disposition of the original message when it cannot be delivered to the destination queue. The application can set the disposition options independently of requesting exception reports.

MQRO_DEAD_LETTER_Q
This is the default action, and places the message on the dead-letter queue if the message cannot be delivered to the destination queue. This happens in the following situations:

- When the application that put the original message cannot be notified synchronously of the problem by means of the reason code returned by the MQPUT or MQPUT1 call. An exception report message is generated, if one was requested by the sender.
- When the application that put the original message was putting to a topic

MQRO_DISCARD_MSG
This discards the message if it cannot be delivered to the destination queue. This happens in the following situations:

- When the application that put the original message cannot be notified synchronously of the problem by means of the reason code returned by the MQPUT or MQPUT1 call. An exception report message is generated, if one was requested by the sender.
- When the application that put the original message was putting to a topic

If you want to return the original message to the sender, without the original message being placed on the dead-letter queue, the sender must specify MQRO_DISCARD_MSG with MQRO_EXCEPTION_WITH_FULL_DATA.

MQRO_PASS_DISCARD_AND_EXPIRY
If this option is set on a message, and a report or reply is generated because of it, the message descriptor of the report inherits:

- MQRO_DISCARD_MSG if it was set.
- The remaining expiry time of the message (if this is not an expiry report). If this is an expiry report the expiry time is set to 60 seconds.

Activity option

MQRO_ACTIVITY
Using this value allows the route of **any** message to be traced throughout a queue manager network. The report option can be specified on any current user message, instantly allowing you to begin calculating the route of the message through the network.

If the application generating the message cannot enable activity report generation, reporting can be enabled using an API crossing exit supplied by queue manager administrators.

Note:

1. The fewer the queue managers in the network that are able to generate activity reports, the less detailed the route.
2. The activity reports might be difficult to place in the correct order to determine the route taken.
3. The activity reports might not be able to find a route to their requested destination.
4. Messages with this report option set must be accepted by any queue manager, even if they do not understand the option. This allows the report option to be set on any user message, even if they are processed by a non Version 6.0 or later queue manager.
5. If a process, either a queue manager or a user process, performs an activity on a message with this option set it can choose to generate and put an activity report.

Default option: Specify the following if no report options are required:

MQRO_NONE
Use this value to indicate that no other options have been specified. MQRO_NONE is defined to aid program documentation. It is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

General information:

1. All report types required must be specifically requested by the application sending the original message. For example, if a COA report is requested but an exception report is not, a COA report is generated when the message is placed on the destination queue, but no exception report is generated if the destination queue is full when the message arrives there. If no *Report* options are set, no report messages are generated by the queue manager or message channel agent (MCA).

   Some report options can be specified even though the local queue manager does not recognize them; this is useful when the option is to be processed by the *destination* queue manager. See "Report options and message flags" on page 2905 for more details.

   If a report message is requested, the name of the queue to which to send the report must be specified in the *ReplyToQ* field. When a report message is received, the nature of the report can be determined by examining the *Feedback* field in the message descriptor.
2. If the queue manager or MCA that generates a report message cannot put the report message on the reply queue (for example, because the reply queue or transmission queue is full), the report message is placed instead on the dead-letter queue. If that *also* fails, or there is no dead-letter queue, the action taken depends on the type of the report message:
   - If the report message is an exception report, the message that generated the exception report is left on its transmission queue; this ensures that the message is not lost.
   - For all other report types, the report message is discarded and processing continues normally. This is done because either the original message has already been delivered safely (for COA or COD report messages), or is no longer of any interest (for an expiration report message).

   Once a report message has been placed successfully on a queue (either the destination queue or an intermediate transmission queue), the message is no longer subject to special processing; it is treated just like any other message.
3. When the report is generated, the *ReplyToQ* queue is opened and the report message put using the authority of the *UserIdentifier* in the MQMD of the message causing the report, except in the following cases:
   - Exception reports generated by a receiving MCA are put with whatever authority the MCA used when it tried to put the message causing the report.
   - COA reports generated by the queue manager are put with whatever authority was used when the message causing the report was put on the queue manager generating the report. For example, if the message was put by a receiving MCA using the MCA's user identifier, the queue manager puts the COA report using the MCA's user identifier.

Applications generating reports must use the same authority as they use to generate a reply; this is usually the authority of the user identifier in the original message.

If the report has to travel to a remote destination, senders and receivers can decide whether to accept it, in the same way as they do for other messages.

4. If a report message with data is requested:
   - The report message is always generated with the amount of data requested by the sender of the original message. If the report message is too big for the reply queue, the processing described above occurs; the report message is never truncated to fit on the reply queue.
   - If the *Format* of the original message is MQFMT_XMIT_Q_HEADER, the data included in the report does not include the MQXQH. The report data starts with the first byte of the data beyond the MQXQH in the original message. This occurs whether or not the queue is a transmission queue.
5. If a COA, COD, or expiration report message is received at the reply queue, it is guaranteed that the original message arrived, was delivered, or expired, as appropriate. However, if one or more of these report messages is requested and is not received, the reverse cannot be assumed, because one of the following might have occurred:
   a. The report message is held up because a link is down.
   b. The report message is held up because a blocking condition exists at an intermediate transmission queue or at the reply queue (for example, the queue is full or inhibited for puts).
   c. The report message is on a dead-letter queue.
   d. When the queue manager was attempting to generate the report message, it could neither put it on the appropriate queue, nor on the dead-letter queue, so the report message could not be generated.
   e. A failure of the queue manager occurred between the action being reported (arrival, delivery, or expiry), and generation of the corresponding report message. (This does not happen for COD report messages if the application retrieves the original message within a unit of work, as the COD report message is generated within the same unit of work.)

   Exception report messages can be held up in the same way for reasons 1, 2, and 3 above. However, when an MCA cannot generate an exception report message (the report message cannot be put either on the reply queue or the dead-letter queue), the original message remains on the transmission queue at the sender, and the channel is closed. This occurs irrespective of whether the report message was to be generated at the sending or the receiving end of the channel.
6. If the original message is temporarily blocked (resulting in an exception report message being generated and the original message being put on a dead-letter queue), but the blockage clears and an application then reads the original message from the dead-letter queue and puts it again to its destination, the following might occur:
   - Even though an exception report message has been generated, the original message eventually arrives successfully at its destination.
   - More than one exception report message is generated in respect of a single original message, because the original message might encounter another blockage later.

Report messages when putting to a topic:

1. Reports can be generated when putting a message to a topic. This message will be sent to all subscribers to the topic, which could be zero, one, or many. This should be taken into account when choosing to use report options as many report messages could be generated as a result.
2. When putting a message to a topic, there may be many destination queues that are to be given a copy of the message. If some of these destination queues have a problem, such as queue full, then the successful completion of the MQPUT depends on the setting of NPMSGDLV or PMSGDLV (depending on the persistence of the message). If the setting is such that message delivery to the destination queue must be successful (for example, it is a persistent message to a durable subscriber and PMSGDLV is set to ALL or ALLDUR), then success is defined as one of the following criteria being met:
   - Successful put to the subscriber queue

- Use of MQRO_DEAD_LETTER_Q and a successful put to the Dead-letter queue if the subscriber queue cannot take the message
- Use of MQRO_DISCARD_MSG if the subscriber queue cannot take the message.

Report messages for message segments:

1. Report messages can be requested for messages that have segmentation allowed (see the description of the MQMF_SEGMENTATION_ALLOWED flag). If the queue manager finds it necessary to segment the message, a report message can be generated for each of the segments that subsequently encounters the relevant condition. Applications must be prepared to receive multiple report messages for each type of report message requested. Use the *GroupId* field in the report message to correlate the multiple reports with the group identifier of the original message, and the *Feedback* field identify the type of each report message.
2. If MQGMO_LOGICAL_ORDER is used to retrieve report messages for segments, be aware that reports of *different types* might be returned by the successive MQGET calls. For example, if both COA and COD reports are requested for a message that is segmented by the queue manager, the MQGET calls for the report messages might return the COA and COD report messages interleaved in an unpredictable fashion. Avoid this by using the MQGMO_COMPLETE_MSG option (optionally with MQGMO_ACCEPT_TRUNCATED_MSG). MQGMO_COMPLETE_MSG causes the queue manager to reassemble report messages that have the same report type. For example, the first MQGET call might reassemble all the COA messages relating to the original message, and the second MQGET call might reassemble all the COD messages. Which is reassembled first depends on which type of report message occurs first on the queue.
3. Applications that themselves put segments can specify different report options for each segment. However, note the following points:
   - If the segments are retrieved using the MQGMO_COMPLETE_MSG option, only the report options in the *first* segment are honored by the queue manager.
   - If the segments are retrieved one by one, and most of them have one of the MQRO_COD_* options, but at least one segment does not, you cannot use the MQGMO_COMPLETE_MSG option to retrieve the report messages with a single MQGET call, or use the MQGMO_ALL_SEGMENTS_AVAILABLE option to detect when all the report messages have arrived.
4. In an MQ network, the queue managers can have different capabilities. If a report message for a segment is generated by a queue manager or MCA that does not support segmentation, the queue manager or MCA does not by default include the necessary segment information in the report message, and this might make it difficult to identify the original message that caused the report to be generated. Avoid this difficulty by requesting data with the report message, that is, by specifying the appropriate MQRO_*_WITH_DATA or MQRO_*_WITH_FULL_DATA options. However, be aware that if MQRO_*_WITH_DATA is specified, *less than* 100 bytes of application message data might be returned to the application that retrieves the report message, if the report message is generated by a queue manager or MCA that does not support segmentation.

Contents of the message descriptor for a report message: When the queue manager or message channel agent (MCA) generates a report message, it sets the fields in the message descriptor to the following values, and then puts the message in the normal way.

Field in MQMD	Value used
StrucId	MQMD_STRUC_ID
Version	MQMD_VERSION_2
Report	MQRO_NONE
MsgType	MQMT_REPORT
Expiry	MQEI_UNLIMITED
Feedback	As appropriate for the nature of the report (MQFB_COA, MQFB_COD, MQFB_EXPIRATION, or an MQRC_* value)
Encoding	Copied from the original message descriptor
CodedCharSetId	Copied from the original message descriptor
Format	Copied from the original message descriptor
Priority	Copied from the original message descriptor
Persistence	Copied from the original message descriptor
MsgId	As specified by the report options in the original message descriptor
CorrelId	As specified by the report options in the original message descriptor
BackoutCount	0
ReplyToQ	Blanks
ReplyToQMgr	Name of queue manager
UserIdentifier	As set by the MQPMO_PASS_IDENTITY_CONTEXT option
AccountingToken	As set by the MQPMO_PASS_IDENTITY_CONTEXT option
ApplIdentityData	As set by the MQPMO_PASS_IDENTITY_CONTEXT option
PutApplType	MQAT_QMGR, or as appropriate for the message channel agent
PutApplName	First 28 bytes of the queue manager name or message channel agent name. For report messages generated by the IMS bridge, this field contains the XCF group name and XCF member name of the IMS system to which the message relates.
PutDate	Date when report message is sent
PutTime	Time when report message is sent
ApplOriginData	Blanks
GroupId	Copied from the original message descriptor
MsgSeqNumber	Copied from the original message descriptor
Offset	Copied from the original message descriptor
MsgFlags	Copied from the original message descriptor
OriginalLength	Copied from the original message descriptor if not MQOL_UNDEFINED, and set to the length of the original message data otherwise

An application generating a report is recommended to set similar values, except for the following:

- The *ReplyToQMgr* field can be set to blanks (the queue manager changes this to the name of the local queue manager when the message is put).
- Set the context fields using the option that would have been used for a reply, normally MQPMO_PASS_IDENTITY_CONTEXT.

Analyzing the report field: The *Report* field contains subfields; because of this, applications that need to check whether the sender of the message requested a particular report must use one of the techniques described in "Analyzing the report field" on page 2907.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQRO_NONE.

StrucId (MQCHAR4):

This is the structure identifier, and must be:

MQMD_STRUC_ID
Identifier for message descriptor structure.

For the C programming language, the constant MQMD_STRUC_ID_ARRAY is also defined; this has the same value as MQMD_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQMD_STRUC_ID.

UserIdentifier (MQCHAR12):

This is part of the **identity context** of the message. For more information about message context, see "Overview for MQMD" on page 2388 and Message context.

UserIdentifier specifies the user identifier of the application that originated the message. The queue manager treats this information as character data, but does not define the format of it.

After a message has been received, use *UserIdentifier* in the *AlternateUserId* field of the **ObjDesc** parameter of a subsequent MQOPEN or MQPUT1 call to perform the authorization check for the *UserIdentifier* user instead of the application performing the open.

When the queue manager generates this information for an MQPUT or MQPUT1 call:

- On z/OS, the queue manager uses the *AlternateUserId* from the **ObjDesc** parameter of the MQOPEN or MQPUT1 call if the MQOO_ALTERNATE_USER_AUTHORITY or MQPMO_ALTERNATE_USER_AUTHORITY option was specified. If the relevant option was not specified, the queue manager uses a user identifier determined from the environment.
- In other environments, the queue manager always uses a user identifier determined from the environment.

When the user identifier is determined from the environment:

- On z/OS, the queue manager uses:
  - For MVS (batch), the user identifier from the JES JOB card or started task
  - For TSO, the user identifier propagated to the job during job submission
  - For CICS, the user identifier associated with the task
  - For IMS, the user identifier depends on the type of application:
    - For:
      - Nonmessage BMP regions
      - Nonmessage IFP regions
      - Message BMP and message IFP regions that have not issued a successful GU call

      the queue manager uses the user identifier from the region JES JOB card or the TSO user identifier. If these are blank or null, it uses the name of the program specification block (PSB).
    - For:
      - Message BMP and message IFP regions that *have* issued a successful GU call
      - MPP regions

      the queue manager uses one of:
      - The signed-on user identifier associated with the message
      - The logical terminal (LTERM) name
      - The user identifier from the region JES JOB card
      - The TSO user identifier

    - The PSB name
- On IBM i, the queue manager uses the name of the user profile associated with the application job.
- On UNIX, the queue manager uses:
  - The application's logon name
  - The effective user identifier of the process if no logon is available
  - The user identifier associated with the transaction, if the application is a CICS transaction
- On Windows systems, the queue manager uses the first 12 characters of the logged-on user name.

This field is normally an output field generated by the queue manager but for an MQPUT or MQPUT1 call you can make this field an input/output field and specify the UserIdentification field instead of letting the queue manager generate this information. Specify either MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT in the PutMsgOpts parameter and specify a user ID in the UserIdentifier field if you do not want the queue manager to generate the UserIdentifier field for an MQPUT or MQPUT1 call.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT is specified in the **PutMsgOpts** parameter. Any information following a null character within the field is discarded. The queue manager converts the null character and any following characters to blanks. If MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *UserIdentifier* that was transmitted with the message if it was put to a queue. This will be the value of *UserIdentifier* that is kept with the message if it is retained (see description of MQPMO_RETAIN for more details about retained publications) but is not used as the *UserIdentifier* when the message is sent as a publication to subscribers because they provide a value to override *UserIdentifier* in all publications sent to them. If the message has no context, the field is entirely blank.

This is an output field for the MQGET call. The length of this field is given by MQ_USER_ID_LENGTH. The initial value of this field is the null string in C, and 12 blank characters in other programming languages.

Version (MQLONG):

This is the structure version number, and must be one of the following:

MQMD_VERSION_1
Version-1 message descriptor structure.

This version is supported in all environments.

MQMD_VERSION_2
Version-2 message descriptor structure.

This version is supported in all IBM MQ V6.0 and later environments, plus IBM MQ MQI clients connected to these systems.

Note: When a version-2 MQMD is used, the queue manager performs additional checks on any MQ header structures that might be present at the beginning of the application message data; for further details see the usage notes for the MQPUT call.

Fields that exist only in the more-recent version of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQMD_CURRENT_VERSION
Current version of message descriptor structure.

This is always an input field. The initial value of this field is MQMD_VERSION_1.

Initial values and language declarations for MQMD:

Table 216. Initial values of fields in MQMD for MQMD

Field name	Name of constant	Value of constant
StrucId	MQMD_STRUC_ID	'MD'
Version	MQMD_VERSION_1	1
Report	MQRO_NONE	0
MsgType	MQMT_DATAGRAM	8
Expiry	MQEI_UNLIMITED	-1
Feedback	MQFB_NONE	0
Encoding	MQENC_NATIVE	Depends on environment
CodedCharSetId	MQCCSI_Q_MGR	0
Format	MQFMT_NONE	Blanks
Priority	MQPRI_PRIORITY_AS_Q_DEF	-1
Persistence	MQPER_PERSISTENCE_AS_Q_DEF	2
MsgId	MQMI_NONE	Nulls
CorrelId	MQCI_NONE	Nulls
BackoutCount	None	0
ReplyToQ	None	Null string or blanks
ReplyToQMgr	None	Null string or blanks
UserIdentifier	None	Null string or blanks
AccountingToken	MQACT_NONE	Nulls
ApplIdentityData	None	Null string or blanks
PutApplType	MQAT_NO_CONTEXT	0
PutApplName	None	Null string or blanks
PutDate	None	Null string or blanks
PutTime	None	Null string or blanks
ApplOriginData	None	Null string or blanks
GroupId	MQGI_NONE	Nulls
MsgSeqNumber	None	1
Offset	None	0
MsgFlags	MQMF_NONE	0
OriginalLength	MQOL_UNDEFINED	-1

Notes:

1. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
2. In the C programming language, the macro variable MQMD_DEFAULT contains the values that are listed in the table. It can be used in the following way to provide initial values for the fields in the structure:

```
MQMD MyMD = {MQMD_DEFAULT};
```

C declaration for MQMD:

```
typedef struct tagMQMD MQMD;
struct tagMQMD {
  MQCHAR4   StrucId;           /* Structure identifier */
  MQLONG    Version;           /* Structure version number */
  MQLONG    Report;            /* Options for report messages */
  MQLONG    MsgType;           /* Message type */
  MQLONG    Expiry;            /* Message lifetime */
  MQLONG    Feedback;          /* Feedback or reason code */
  MQLONG    Encoding;          /* Numeric encoding of message data */
  MQLONG    CodedCharSetId;    /* Character set identifier of message
                                  data */
  MQCHAR8   Format;            /* Format name of message data */
  MQLONG    Priority;          /* Message priority */
  MQLONG    Persistence;       /* Message persistence */
  MQBYTE24  MsgId;             /* Message identifier */
  MQBYTE24  CorrelId;          /* Correlation identifier */
  MQLONG    BackoutCount;      /* Backout counter */
  MQCHAR48  ReplyToQ;          /* Name of reply queue */
  MQCHAR48  ReplyToQMgr;       /* Name of reply queue manager */
  MQCHAR12  UserIdentifier;    /* User identifier */
  MQBYTE32  AccountingToken;   /* Accounting token */
  MQCHAR32  ApplIdentityData;  /* Application data relating to
                                  identity */
  MQLONG    PutApplType;       /* Type of application that put the
                                  message */
  MQCHAR28  PutApplName;       /* Name of application that put the
                                  message */
  MQCHAR8   PutDate;           /* Date when message was put */
  MQCHAR8   PutTime;           /* Time when message was put */
  MQCHAR4   ApplOriginData;    /* Application data relating to origin */
  MQBYTE24  GroupId;           /* Group identifier */
  MQLONG    MsgSeqNumber;      /* Sequence number of logical message
                                  within group */
  MQLONG    Offset;            /* Offset of data in physical message
                                  from start of logical message */
  MQLONG    MsgFlags;          /* Message flags */
  MQLONG    OriginalLength;    /* Length of original message */
};
```

COBOL declaration for MQMD:

```
**   MQMD structure
  10 MQMD.
**    Structure identifier
   15 MQMD-STRUCID          PIC X(4).
**    Structure version number
   15 MQMD-VERSION          PIC S9(9) BINARY.
**    Options for report messages
   15 MQMD-REPORT           PIC S9(9) BINARY.
**    Message type
   15 MQMD-MSGTYPE          PIC S9(9) BINARY.
**    Message lifetime
   15 MQMD-EXPIRY           PIC S9(9) BINARY.
**    Feedback or reason code
   15 MQMD-FEEDBACK         PIC S9(9) BINARY.
**    Numeric encoding of message data
   15 MQMD-ENCODING         PIC S9(9) BINARY.
**    Character set identifier of message data
   15 MQMD-CODEDCHARSETID   PIC S9(9) BINARY.
**    Format name of message data
   15 MQMD-FORMAT           PIC X(8).
**    Message priority
   15 MQMD-PRIORITY         PIC S9(9) BINARY.
**    Message persistence
   15 MQMD-PERSISTENCE      PIC S9(9) BINARY.
**    Message identifier
```

```
   15 MQMD-MSGID            PIC X(24).
**    Correlation identifier
   15 MQMD-CORRELID         PIC X(24).
**    Backout counter
   15 MQMD-BACKOUTCOUNT     PIC S9(9) BINARY.
**    Name of reply queue
   15 MQMD-REPLYTOQ         PIC X(48).
**    Name of reply queue manager
   15 MQMD-REPLYTOQMGR      PIC X(48).
**    User identifier
   15 MQMD-USERIDENTIFIER   PIC X(12).
**    Accounting token
   15 MQMD-ACCOUNTINGTOKEN  PIC X(32).
**    Application data relating to identity
   15 MQMD-APPLIDENTITYDATA PIC X(32).
**    Type of application that put the message
   15 MQMD-PUTAPPLTYPE      PIC S9(9) BINARY.
**    Name of application that put the message
   15 MQMD-PUTAPPLNAME      PIC X(28).
**    Date when message was put
   15 MQMD-PUTDATE          PIC X(8).
**    Time when message was put
   15 MQMD-PUTTIME          PIC X(8).
**    Application data relating to origin
   15 MQMD-APPLORIGINDATA   PIC X(4).
**    Group identifier
   15 MQMD-GROUPID          PIC X(24).
**    Sequence number of logical message within group
   15 MQMD-MSGSEQNUMBER     PIC S9(9) BINARY.
**    Offset of data in physical message from start of logical message
   15 MQMD-OFFSET           PIC S9(9) BINARY.
**    Message flags
   15 MQMD-MSGFLAGS         PIC S9(9) BINARY.
**    Length of original message
   15 MQMD-ORIGINALLENGTH   PIC S9(9) BINARY.
```

PL/I declaration for MQMD:

```
dcl
 1 MQMD based,
  3 StrucId          char(4),       /* Structure identifier */
  3 Version          fixed bin(31), /* Structure version number */
  3 Report           fixed bin(31), /* Options for report messages */
  3 MsgType          fixed bin(31), /* Message type */
  3 Expiry           fixed bin(31), /* Message lifetime */
  3 Feedback         fixed bin(31), /* Feedback or reason code */
  3 Encoding         fixed bin(31), /* Numeric encoding of message
                                       data */
  3 CodedCharSetId   fixed bin(31), /* Character set identifier of
                                       message data */
  3 Format           char(8),       /* Format name of message data */
  3 Priority         fixed bin(31), /* Message priority */
  3 Persistence      fixed bin(31), /* Message persistence */
  3 MsgId            char(24),      /* Message identifier */
  3 CorrelId         char(24),      /* Correlation identifier */
  3 BackoutCount     fixed bin(31), /* Backout counter */
  3 ReplyToQ         char(48),      /* Name of reply queue */
  3 ReplyToQMgr      char(48),      /* Name of reply queue manager */
  3 UserIdentifier   char(12),      /* User identifier */
  3 AccountingToken  char(32),      /* Accounting token */
  3 ApplIdentityData char(32),      /* Application data relating to
                                       identity */
  3 PutApplType      fixed bin(31), /* Type of application that put the
                                       message */
  3 PutApplName      char(28),      /* Name of application that put the
                                       message */
  3 PutDate          char(8),       /* Date when message was put */
```

```
  3 PutTime          char(8),       /* Time when message was put */
  3 ApplOriginData   char(4),       /* Application data relating to
                                       origin */
  3 GroupId          char(24),      /* Group identifier */
  3 MsgSeqNumber     fixed bin(31), /* Sequence number of logical
                                       message within group */
  3 Offset           fixed bin(31), /* Offset of data in physical
                                       message from start of logical
                                       message */
  3 MsgFlags         fixed bin(31), /* Message flags */
  3 OriginalLength   fixed bin(31); /* Length of original message */
```

High Level Assembler declaration for MQMD:

```
MQMD                  DSECT
MQMD_STRUCID          DS   CL4   Structure identifier
MQMD_VERSION          DS   F     Structure version number
MQMD_REPORT           DS   F     Options for report messages
MQMD_MSGTYPE          DS   F     Message type
MQMD_EXPIRY           DS   F     Message lifetime
MQMD_FEEDBACK         DS   F     Feedback or reason code
MQMD_ENCODING         DS   F     Numeric encoding of message data
MQMD_CODEDCHARSETID   DS   F     Character set identifier of message
*                                data
MQMD_FORMAT           DS   CL8   Format name of message data
MQMD_PRIORITY         DS   F     Message priority
MQMD_PERSISTENCE      DS   F     Message persistence
MQMD_MSGID            DS   XL24  Message identifier
MQMD_CORRELID         DS   XL24  Correlation identifier
MQMD_BACKOUTCOUNT     DS   F     Backout counter
MQMD_REPLYTOQ         DS   CL48  Name of reply queue
MQMD_REPLYTOQMGR      DS   CL48  Name of reply queue manager
MQMD_USERIDENTIFIER   DS   CL12  User identifier
MQMD_ACCOUNTINGTOKEN  DS   XL32  Accounting token
MQMD_APPLIDENTITYDATA DS   CL32  Application data relating to identity
MQMD_PUTAPPLTYPE      DS   F     Type of application that put the
*                                message
MQMD_PUTAPPLNAME      DS   CL28  Name of application that put the
*                                message
MQMD_PUTDATE          DS   CL8   Date when message was put
MQMD_PUTTIME          DS   CL8   Time when message was put
MQMD_APPLORIGINDATA   DS   CL4   Application data relating to origin
MQMD_GROUPID          DS   XL24  Group identifier
MQMD_MSGSEQNUMBER     DS   F     Sequence number of logical message
*                                within group
MQMD_OFFSET           DS   F     Offset of data in physical message
*                                from start of logical message
MQMD_MSGFLAGS         DS   F     Message flags
MQMD_ORIGINALLENGTH   DS   F     Length of original message
*
MQMD_LENGTH           EQU  *-MQMD
                      ORG  MQMD
MQMD_AREA             DS   CL(MQMD_LENGTH)
```

Visual Basic declaration for MQMD:

```
Type MQMD
  StrucId          As String*4  'Structure identifier'
  Version          As Long      'Structure version number'
  Report           As Long      'Options for report messages'
  MsgType          As Long      'Message type'
  Expiry           As Long      'Message lifetime'
  Feedback         As Long      'Feedback or reason code'
  Encoding         As Long      'Numeric encoding of message data'
  CodedCharSetId   As Long      'Character set identifier of message'
                                'data'
  Format           As String*8  'Format name of message data'
  Priority         As Long      'Message priority'
  Persistence      As Long      'Message persistence'
  MsgId            As MQBYTE24  'Message identifier'
  CorrelId         As MQBYTE24  'Correlation identifier'
  BackoutCount     As Long      'Backout counter'
  ReplyToQ         As String*48 'Name of reply queue'
  ReplyToQMgr      As String*48 'Name of reply queue manager'
  UserIdentifier   As String*12 'User identifier'
  AccountingToken  As MQBYTE32  'Accounting token'
  ApplIdentityData As String*32 'Application data relating to identity'
  PutApplType      As Long      'Type of application that put the'
                                'message'
  PutApplName      As String*28 'Name of application that put the'
                                'message'
  PutDate          As String*8  'Date when message was put'
  PutTime          As String*8  'Time when message was put'
  ApplOriginData   As String*4  'Application data relating to origin'
  GroupId          As MQBYTE24  'Group identifier'
  MsgSeqNumber     As Long      'Sequence number of logical message'
                                'within group'
  Offset           As Long      'Offset of data in physical message'
                                'from start of logical message'
  MsgFlags         As Long      'Message flags'
  OriginalLength   As Long      'Length of original message'
End Type
```

MQMDE - Message descriptor extension:

The following table summarizes the fields in the structure.

Table 217. Fields in MQMDE

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
StrucLength	Length of MQMDE structure	StrucLength
Encoding	Numeric encoding of data that follows MQMDE	Encoding
CodedCharSetId	Character set identifier of data that follows MQMDE	CodedCharSetId
Format	Format name of data that follows MQMDE	Format
Flags	General flags	Flags
GroupId	Group identifier	GroupId
MsgSeqNumber	Sequence number of logical message within group	MsgSeqNumber
Offset	Offset of data in physical message from start of logical message	Offset
MsgFlags	Message flags	MsgFlags
OriginalLength	Length of original message	OriginalLength

Overview for MQMDE:

Availability: All IBM MQ systems, plus IBM MQ clients connected to these systems.

Purpose: The MQMDE structure describes the data that sometimes occurs preceding the application message data. The structure contains those MQMD fields that exist in the version-2 MQMD, but not in the version-1 MQMD.

Format name: MQFMT_MD_EXTENSION.

Character set and encoding: Data in MQMDE must be in the character set and encoding of the local queue manager; these are given by the **CodedCharSetId** queue manager attribute and MQENC_NATIVE for the C programming language.

Set the character set and encoding of the MQMDE into the *CodedCharSetId* and *Encoding* fields in:

- The MQMD (if the MQMDE structure is at the start of the message data), or
- The header structure that precedes the MQMDE structure (all other cases).

If the MQMDE is not in the queue manager's character set and encoding, the MQMDE is accepted but not honored, that is, the MQMDE is treated as message data.

Note: On Windows, applications compiled with Micro Focus COBOL use a value of MQENC_NATIVE that is different from the queue manager's encoding. Although numeric fields in the MQMD structure on the MQPUT, MQPUT1, and MQGET calls must be in the Micro Focus COBOL encoding, numeric fields in the MQMDE structure must be in the queue manager's encoding. This latter is given by MQENC_NATIVE for the C programming language, and has the value 546.

Usage: Applications that use a version-2 MQMD will not encounter an MQMDE structure. However, specialized applications, and applications that continue to use a version-1 MQMD, might encounter an MQMDE in some situations. The MQMDE structure can occur in the following circumstances:

- Specified on the MQPUT and MQPUT1 calls
- Returned by the MQGET call
- In messages on transmission queues

MQMDE specified on MQPUT and MQPUT1 calls: On the MQPUT and MQPUT1 calls, if the application provides a version-1 MQMD, the application can optionally prefix the message data with an MQMDE, setting the *Format* field in MQMD to MQFMT_MD_EXTENSION to indicate that an MQMDE is present. If the application does not provide an MQMDE, the queue manager assumes default values for the fields in the MQMDE. The default values that the queue manager uses are the same as the initial values for the structure; see Table 219 on page 2447.

If the application provides a version-2 MQMD *and* prefixes the application message data with an MQMDE, the structures are processed as shown in Table 218 on page 2444.

Table 218. Queue manager action when MQMDE specified on MQPUT or MQPUT1 for MQMDE

MQMD version	Values of version-2 fields	Values of corresponding fields in MQMDE	Action taken by queue manager
1	-	Valid	MQMDE is honored
2	Default	Valid	MQMDE is honored
2	Not default	Valid	MQMDE is treated as message data
1 or 2	Any	Not valid	Call fails with an appropriate reason code
1 or 2	Any	MQMDE is in the wrong character set or encoding, or is an unsupported version	MQMDE is treated as message data
Note: On z/OS, if the application specifies a version-1 MQMD with an MQMDE, the queue manager validates the MQMDE only if the queue has an *IndexType* of MQIT_GROUP_ID.			

There is one special case. If the application uses a version-2 MQMD to put a message that is a segment (that is, the MQMF_SEGMENT or MQMF_LAST_SEGMENT flag is set), and the format name in the MQMD is MQFMT_DEAD_LETTER_HEADER, the queue manager generates an MQMDE structure and inserts it *between* the MQDLH structure and the data that follows it. In the MQMD that the queue manager retains with the message, the version-2 fields are set to their default values.

Several of the fields that exist in the version-2 MQMD but not the version-1 MQMD are input/output fields on MQPUT and MQPUT1. However, the queue manager does not return any values in the equivalent fields in the MQMDE on output from the MQPUT and MQPUT1 calls; if the application requires those output values, it must use a version-2 MQMD.

MQMDE returned by MQGET call: On the MQGET call, if the application provides a version-1 MQMD, the queue manager prefixes the message returned with an MQMDE, but only if one or more of the fields in the MQMDE has a nondefault value. The queue manager sets the *Format* field in MQMD to the value MQFMT_MD_EXTENSION to indicate that an MQMDE is present.

If the application provides an MQMDE at the start of the **Buffer** parameter, the MQMDE is ignored. On return from the MQGET call, it is replaced by the MQMDE for the message (if one is needed), or overwritten by the application message data (if the MQMDE is not needed).

If the MQGET call returns an MQMDE, the data in the MQMDE is usually in the queue manager's character set and encoding. However the MQMDE might be in some other character set and encoding if:

- The MQMDE was treated as data on the MQPUT or MQPUT1 call (see Table 218 for the circumstances that can cause this).
- The message was received from a remote queue manager connected by a TCP connection, and the receiving message channel agent (MCA) was not set up correctly.

Note: On Windows, applications compiled with Micro Focus COBOL use a value of MQENC_NATIVE that is different from the queue manager's encoding (see above).

MQMDE in messages on transmission queues: Messages on transmission queues are prefixed with the MQXQH structure, which contains within it a version-1 MQMD. An MQMDE might also be present, positioned between the MQXQH structure and application message data, but it is usually present only if one or more of the fields in the MQMDE has a nondefault value.

Other MQ header structures can also occur between the MQXQH structure and the application message data. For example, when the dead-letter header MQDLH is present, and the message is not a segment, the order is:

- MQXQH (containing a version-1 MQMD)
- MQMDE
- MQDLH
- application message data

Fields for MQMDE:

The MQMDE structure contains the following fields; the fields are described in **alphabetical order**:

CodedCharSetId (MQLONG):

This specifies the character set identifier of the data that follows the MQMDE structure; it does not apply to character data in the MQMDE structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that this field is valid. The following special value can be used:

MQCCSI_INHERIT
Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

MQCCSI_INHERIT cannot be used if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG):

This specifies the numeric encoding of the data that follows the MQMDE structure; it does not apply to numeric data in the MQMDE structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that the field is valid. See the *Encoding* field described in "MQMD - Message descriptor" on page 2387 for more information about data encodings.

The initial value of this field is MQENC_NATIVE.

Flags (MQLONG):

The following flag can be specified:

MQMDEF_NONE
No flags.

The initial value of this field is MQMDEF_NONE.

Format (MQCHAR8):

This specifies the format name of the data that follows the MQMDE structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that this field is valid. See the *Format* field described in"MQMD - Message descriptor" on page 2387for more information about format names.

The initial value of this field is MQFMT_NONE.

GroupId (MQBYTE24):

See the *GroupId* field described in"MQMD - Message descriptor" on page 2387. The initial value of this field is MQGI_NONE.

MsgFlags (MQLONG):

See the *MsgFlags* field described in "MQMD - Message descriptor" on page 2387. The initial value of this field is MQMF_NONE.

MsgSeqNumber (MQLONG):

See the *MsgSeqNumber* field described in"MQMD - Message descriptor" on page 2387. The initial value of this field is 1.

Offset (MQLONG):

See the *Offset* field described in"MQMD - Message descriptor" on page 2387. The initial value of this field is 0.

OriginalLength (MQLONG):

See the *OriginalLength* field described in"MQMD - Message descriptor" on page 2387. The initial value of this field is MQOL_UNDEFINED.

StrucId (MQCHAR4):

The value must be:

MQMDE_STRUC_ID
Identifier for message descriptor extension structure.

For the C programming language, the constant MQMDE_STRUC_ID_ARRAY is also defined; this has the same value as MQMDE_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQMDE_STRUC_ID.

StrucLength (MQLONG):

This is the length of the MQMDE structure; the following value is defined:

MQMDE_LENGTH_2
Length of version-2 message descriptor extension structure.

The initial value of this field is MQMDE_LENGTH_2.

Version (MQLONG):

This is the structure version number; the value must be:

MQMDE_VERSION_2
Version-2 message descriptor extension structure.

The following constant specifies the version number of the current version:

MQMDE_CURRENT_VERSION
Current version of message descriptor extension structure.

The initial value of this field is MQMDE_VERSION_2.

Initial values and language declarations for MQMDE:

Table 219. Initial values of fields in MQMDE for MQMDE

Field name	Name of constant	Value of constant
StrucId	MQMDE_STRUC_ID	'MDE¬'
Version	MQMDE_VERSION_2	2
StrucLength	MQMDE_LENGTH_2	72
Encoding	MQENC_NATIVE	Depends on environment
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
Flags	MQMDEF_NONE	0
GroupId	MQGI_NONE	Nulls
MsgSeqNumber	None	1
Offset	None	0
MsgFlags	MQMF_NONE	0
OriginalLength	MQOL_UNDEFINED	-1

Table 219. Initial values of fields in MQMDE for MQMDE (continued)

Field name	Name of constant	Value of constant
Notes: 1. The symbol ¬ represents a single blank character. 2. In the C programming language, the macro variable MQMDE_DEFAULT contains the values that are listed in the table. It can be used in the following way to provide initial values for the fields in the structure: `MQMDE MyMDE = {MQMDE_DEFAULT};`		

C declaration for MQMDE:

```
typedef struct tagMQMDE MQMDE;
struct tagMQMDE {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    StrucLength;     /* Length of MQMDE structure */
  MQLONG    Encoding;        /* Numeric encoding of data that follows
                                MQMDE */
  MQLONG    CodedCharSetId;  /* Character-set identifier of data that
                                follows MQMDE */
  MQCHAR8   Format;          /* Format name of data that follows
                                MQMDE */
  MQLONG    Flags;           /* General flags */
  MQBYTE24  GroupId;         /* Group identifier */
  MQLONG    MsgSeqNumber;    /* Sequence number of logical message
                                within group */
  MQLONG    Offset;          /* Offset of data in physical message from
                                start of logical message */
  MQLONG    MsgFlags;        /* Message flags */
  MQLONG    OriginalLength;  /* Length of original message */
};
```

COBOL declaration for MQMDE:

```
**   MQMDE structure
  10 MQMDE.
**    Structure identifier
   15 MQMDE-STRUCID        PIC X(4).
**    Structure version number
   15 MQMDE-VERSION        PIC S9(9) BINARY.
**    Length of MQMDE structure
   15 MQMDE-STRUCLENGTH    PIC S9(9) BINARY.
**    Numeric encoding of data that follows MQMDE
   15 MQMDE-ENCODING       PIC S9(9) BINARY.
**    Character-set identifier of data that follows MQMDE
   15 MQMDE-CODEDCHARSETID PIC S9(9) BINARY.
**    Format name of data that follows MQMDE
   15 MQMDE-FORMAT         PIC X(8).
**    General flags
   15 MQMDE-FLAGS          PIC S9(9) BINARY.
**    Group identifier
   15 MQMDE-GROUPID        PIC X(24).
**    Sequence number of logical message within group
   15 MQMDE-MSGSEQNUMBER   PIC S9(9) BINARY.
**    Offset of data in physical message from start of logical message
   15 MQMDE-OFFSET         PIC S9(9) BINARY.
**    Message flags
   15 MQMDE-MSGFLAGS       PIC S9(9) BINARY.
**    Length of original message
   15 MQMDE-ORIGINALLENGTH PIC S9(9) BINARY.
```

PL/I declaration for MQMDE:

```
dcl
 1 MQMDE based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 StrucLength    fixed bin(31), /* Length of MQMDE structure */
  3 Encoding       fixed bin(31), /* Numeric encoding of data that
                                     follows MQMDE */
  3 CodedCharSetId fixed bin(31), /* Character-set identifier of data
                                     that follows MQMDE */
  3 Format         char(8),       /* Format name of data that follows
                                     MQMDE */
  3 Flags          fixed bin(31), /* General flags */
  3 GroupId        char(24),      /* Group identifier */
  3 MsgSeqNumber   fixed bin(31), /* Sequence number of logical message
                                     within group */
  3 Offset         fixed bin(31), /* Offset of data in physical message
                                     from start of logical message */
  3 MsgFlags       fixed bin(31), /* Message flags */
  3 OriginalLength fixed bin(31); /* Length of original message */
```

High Level Assembler declaration for MQMDE:

```
MQMDE                DSECT
MQMDE_STRUCID        DS   CL4   Structure identifier
MQMDE_VERSION        DS   F     Structure version number
MQMDE_STRUCLENGTH    DS   F     Length of MQMDE structure
MQMDE_ENCODING       DS   F     Numeric encoding of data that follows
*                               MQMDE
MQMDE_CODEDCHARSETID DS   F     Character-set identifier of data that
*                               follows MQMDE
MQMDE_FORMAT         DS   CL8   Format name of data that follows MQMDE
MQMDE_FLAGS          DS   F     General flags
MQMDE_GROUPID        DS   XL24  Group identifier
MQMDE_MSGSEQNUMBER   DS   F     Sequence number of logical message
*                               within group
MQMDE_OFFSET         DS   F     Offset of data in physical message from
*                               start of logical message
MQMDE_MSGFLAGS       DS   F     Message flags
MQMDE_ORIGINALLENGTH DS   F     Length of original message
*
MQMDE_LENGTH         EQU  *-MQMDE
                     ORG  MQMDE
MQMDE_AREA           DS   CL(MQMDE_LENGTH)
```

Visual Basic declaration for MQMDE:

```
Type MQMDE
  StrucId        As String*4 'Structure identifier'
  Version        As Long     'Structure version number'
  StrucLength    As Long     'Length of MQMDE structure'
  Encoding       As Long     'Numeric encoding of data that follows'
                             'MQMDE'
  CodedCharSetId As Long     'Character-set identifier of data that'
                             'follows MQMDE'
  Format         As String*8 'Format name of data that follows MQMDE'
  Flags          As Long     'General flags'
  GroupId        As MQBYTE24 'Group identifier'
  MsgSeqNumber   As Long     'Sequence number of logical message within'
                             'group'
  Offset         As Long     'Offset of data in physical message from'
                             'start of logical message'
  MsgFlags       As Long     'Message flags'
  OriginalLength As Long     'Length of original message'
End Type
```

MQMHBO - Message handle to buffer options:

The following table summarizes the fields in the structure. MQMHBO structure - message handle to buffer options

Table 220. Fields in MQMHBO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options controlling the action of MQMHBUF	Options

Overview for MQMHBO:

Availability: All IBM MQ systems and IBM MQ MQI clients.

Purpose: The MQMHBO structure allows applications to specify options that control how buffers are produced from message handles. The structure is an input parameter on the MQMHBUF call.

Character set and encoding: Data in MQMHBO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields for MQMHBO:

Message handle to buffer options structure - fields

The MQMHBO structure contains the following fields; the fields are described in **alphabetical order**:

Options (MQLONG):

Message handle to buffer options structure - Options field

These options control the action of MQMHBUF.

You must specify the following option:

MQMHBO_PROPERTIES_IN_MQRFH2
When converting properties from a message handle into a buffer, convert them into the MQRFH2 format.

Optionally, you can also specify the following option. To specify more than one option, either add the values together (do not add the same constant more than once), or combine the values using the bitwise OR operation (if the programming language supports bit operations).

MQMHBO_DELETE_PROPERTIES
Properties that are added to the buffer are deleted from the message handle. If the call fails no properties are deleted.

This is always an input field. The initial value of this field is MQMHBO_PROPERTIES_IN_MQRFH2.

StrucId (MQCHAR4):

Message handle to buffer options structure - StrucId field

This is the structure identifier. The value must be:

MQMHBO_STRUC_ID
Identifier for message handle to buffer options structure.

For the C programming language, the constant MQMHBO_STRUC_ID_ARRAY is also defined; this has the same value as MQMHBO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQMHBO_STRUC_ID.

Version (MQLONG):

Message handle to buffer options structure - Version field

This is the structure version number. The value must be:

MQMHBO_VERSION_1
Version number for message handle to buffer options structure.

The following constant specifies the version number of the current version:

MQMHBO_CURRENT_VERSION
Current version of message handle to buffer options structure.

This is always an input field. The initial value of this field is MQMHBO_VERSION_1.

Initial values and language declarations for MQMHBO:

Message handle to buffer structure - Initial values

Table 221. Initial values of fields in MQMHBO

Field name	Name of constant	Value of constant
StrucId	MQMHBO_STRUC_ID	'MHBO'
Version	MQMHBO_VERSION_1	1
Options	MQMHBO_PROPERTIES_IN_MQRFH2	

Notes:

1. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
2. In the C programming language, the macro variable MQMHBO_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure:

```
MQMHBO MyMHBO = {MQMHBO_DEFAULT};
```

C declaration for MQMHBO:

Message handle to buffer options structure - C language declaration

```
typedef struct tagMQMHBO MQMHBO;
struct tagMQMHBO {
  MQCHAR4  StrucId;        /* Structure identifier */
  MQLONG   Version;        /* Structure version number */
  MQLONG   Options;        /* Options that control the action of
                              MQMHBUF */
};
```

COBOL declaration for MQMHBO:

Message handle to buffer options structure - COBOL language declaration

```
**  MQMHBO structure
  10 MQMHBO.
**    Structure identifier
   15 MQMHBO-STRUCID                PIC X(4).
**    Structure version number
   15 MQMHBO-VERSION                PIC S9(9) BINARY.
**    Options that control the action of MQMHBUF
   15 MQMHBO-OPTIONS                PIC S9(9) BINARY.
```

PL/I declaration for MQMHBO:

Message handle to buffer options structure - PL/I language declaration

```
Dcl
 1 MQMHBO based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 Options        fixed bin(31), /* Options that control the action
                                     of MQMHBUF */
```

High Level Assembler declaration for MQMHBO:

Message handle to buffer options structure - Assembler language declaration

```
MQMHBO                   DSECT
MQMHBO_STRUCID           DS   CL4  Structure identifier
MQMHBO_VERSION           DS   F    Structure version number
MQMHBO_OPTIONS           DS   F    Options that control the
*                                   action of MQMHBUF
MQMHBO_LENGTH            EQU  *-MQMHBO
MQMHBO_AREA              DS   CL(MQMHBO_LENGTH)
```

MQOD - Object descriptor:

The following table summarizes the fields in the structure.

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
ObjectType	Object type	ObjectType
ObjectName	Object name	ObjectName
ObjectQMgrName	Object queue manager name	ObjectQMgrName
DynamicQName	Dynamic queue name	DynamicQName
AlternateUserId	Alternate user identifier	AlternateUserId
Note: The remaining fields are ignored if *Version* is less than MQOD_VERSION_2.		
RecsPresent	Number of object records present	RecsPresent
KnownDestCount	Number of local queues opened successfully	KnownDestCount
UnknownDestCount	Number of remote queues opened successfully	UnknownDestCount
InvalidDestCount	Number of queues that failed to open	InvalidDestCount
ObjectRecOffset	Offset of first object record from start of MQOD	ObjectRecOffset
ResponseRecOffset	Offset of first response record from start of MQOD	ResponseRecOffset
ObjectRecPtr	Address of first object record	ObjectRecPtr
ResponseRecPtr	Address of first response record	ResponseRecPtr
Note: The remaining fields are ignored if *Version* is less than MQOD_VERSION_3.		
AlternateSecurityId	Alternate security identifier	AlternateSecurityId
ResolvedQName	Resolved queue name	ResolvedQName
ResolvedQMgrName	Resolved queue manager name	ResolvedQMgrName
Note: The remaining fields are ignored if *Version* is less than MQOD_VERSION_4.		
ObjectString	Long object name	ObjectString
SelectionString	Selection string	SelectionString
ResObjectString	Resolved long object name	ResObjectString
ResolvedType	Resolved object type	ResolvedType

Overview for MQOD:

Availability: All IBM MQ systems, plus IBM MQ MQI clients connected to those systems.

Purpose: The MQOD structure is used to specify an object by name. The following types of object are valid:

- Queue or distribution list
- Namelist
- Process definition
- Queue manager
- Topic

The structure is an input/output parameter on the MQOPEN and MQPUT1 calls.

Version: The current version of MQOD is MQOD_VERSION_4. Applications that you want to port between several environments must ensure that the required version of MQOD is supported in all the environments concerned. Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions that follow.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQOD that is supported by the environment, but with the initial value of the *Version* field set to MQOD_VERSION_1. To use fields that are not present in the version-1 structure, the application must set the *Version* field to the version number of the version required.

To open a distribution list, *Version* must be MQOD_VERSION_2 or greater.

Character set and encoding: Data in MQOD must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Fields for MQOD:

The MQOD structure contains the following fields; the fields are described in **alphabetical order**:

AlternateSecurityId (MQBYTE40):

This is a security identifier that is passed with the *AlternateUserId* to the authorization service to allow appropriate authorization checks to be performed. *AlternateSecurityId* is used only if:

- MQOO_ALTERNATE_USER_AUTHORITY is specified on the MQOPEN call, or
- MQPMO_ALTERNATE_USER_AUTHORITY is specified on the MQPUT1 call,

and the *AlternateUserId* field is not entirely blank up to the first null character or the end of the field.

On Windows, *AlternateSecurityId* can be used to supply the Windows security identifier (SID) that uniquely identifies the *AlternateUserId*. The SID for a user can be obtained from the Windows system by use of the `LookupAccountName()` Windows API call.

On z/OS, this field is ignored.

The *AlternateSecurityId* field has the following structure:

- The first byte is a binary integer containing the length of the significant data that follows; the value excludes the length byte itself. If no security identifier is present, the length is zero.
- The second byte indicates the type of security identifier that is present; the following values are possible:

MQSIDT_NT_SECURITY_ID
Windows security identifier.

MQSIDT_NONE
No security identifier.

- The third and subsequent bytes up to the length defined by the first byte contain the security identifier itself.
- Remaining bytes in the field are set to binary zero.

You can use the following special value:

MQSID_NONE
No security identifier specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQSID_NONE_ARRAY is also defined; this has the same value as MQSID_NONE, but is an array of characters instead of a string.

This is an input field. The length of this field is given by MQ_SECURITY_ID_LENGTH. The initial value of this field is MQSID_NONE. This field is ignored if *Version* is less than MQOD_VERSION_3.

AlternateUserId (MQCHAR12):

If you specify MQOO_ALTERNATE_USER_AUTHORITY for the MQOPEN call, or MQPMO_ALTERNATE_USER_AUTHORITY for the MQPUT1 call, this field contains an alternative user identifier that is used to check the authorization for the open, in place of the user identifier that the application is currently running under. Some checks, however, are still carried out with the current user identifier (for example, context checks).

If MQOO_ALTERNATE_USER_AUTHORITY or MQPMO_ALTERNATE_USER_AUTHORITY is specified and this field is entirely blank up to the first null character or the end of the field, the open can succeed only if no user authorization is needed to open this object with the options specified.

If neither MQOO_ALTERNATE_USER_AUTHORITY nor MQPMO_ALTERNATE_USER_AUTHORITY is specified, this field is ignored.

The following differences exist in the environments indicated:

- On z/OS, only the first 8 characters of *AlternateUserId* are used to check the authorization for the open. However, the current user identifier must be authorized to specify this particular alternative user identifier; all 12 characters of the alternative user identifier are used for this check. The user identifier must contain only characters allowed by the external security manager.

  If *AlternateUserId* is specified for a queue, the value can be used subsequently by the queue manager when messages are put. If the MQPMO_*_CONTEXT options specified on the MQPUT or MQPUT1 call cause the queue manager to generate the identity context information, the queue manager places the *AlternateUserId* into the *UserIdentifier* field in the MQMD of the message, in place of the current user identifier.
- In other environments, *AlternateUserId* is used only for access control checks on the object being opened. If the object is a queue, *AlternateUserId* does not affect the content of the *UserIdentifier* field in the MQMD of messages sent using that queue handle.

This is an input field. The length of this field is given by MQ_USER_ID_LENGTH. The initial value of this field is the null string in C, and 12 blank characters in other programming languages.

DynamicQName (MQCHAR48):

This is the name of a dynamic queue that is to be created by the MQOPEN call. This is of relevance only when *ObjectName* specifies the name of a model queue; in all other cases *DynamicQName* is ignored.

The characters that are valid in the name are the same as those for *ObjectName*, except that an asterisk is also valid. A name that is blank (or one in which only blanks occur before the first null character) is not valid if *ObjectName* is the name of a model queue.

If the last nonblank character in the name is an asterisk (*), the queue manager replaces the asterisk with a string of characters that guarantees that the name generated for the queue is unique at the local queue manager. To allow a sufficient number of characters for this, the asterisk is valid only in positions 1 through 33. There must be no characters other than blanks or a null character following the asterisk.

It is valid for the asterisk to occur in the first character position, in which case the name consists solely of the characters generated by the queue manager.

On z/OS, do not use a name with the asterisk in the first character position, as there can be no security checks made on a queue with a full name that is generated automatically.

This is an input field. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is determined by the environment:

- On z/OS, the value is `'CSQ.*'`.
- On other platforms, the value is `'AMQ.*'`.

The value is a null-terminated string in C, and a blank-padded string in other programming languages.

InvalidDestCount (MQLONG):

This is the number of queues in the distribution list that failed to open successfully. If present, this field is also set when opening a single queue that is not in a distribution list.

Note: If present, this field is set only if the **CompCode** parameter on the MQOPEN or MQPUT1 call is MQCC_OK or MQCC_WARNING; it is not set if the **CompCode** parameter is MQCC_FAILED.

This is an output field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_1.

KnownDestCount (MQLONG):

This is the number of queues in the distribution list that resolve to local queues and that were opened successfully. The count does not include queues that resolve to remote queues (even though a local transmission queue is used initially to store the message). If present, this field is also set when opening a single queue that is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_1.

ObjectName (MQCHAR48):

This is the local name of the object as defined on the queue manager identified by *ObjectQMgrName*. The name can contain the following characters:

- Uppercase alphabetic characters (A through Z)
- Lowercase alphabetic characters (a through z)
- Numeric digits (0 through 9)
- Period (.), forward slash (/), underscore (_), percent (%)

The name must not contain leading or embedded blanks, but can contain trailing blanks. Use a null character to indicate the end of significant data in the name; the null and any characters following it are treated as blanks. The following restrictions apply in the environments indicated:

- On systems that use EBCDIC Katakana, lowercase characters cannot be used.
- On z/OS:
  - Avoid names that begin or end with an underscore; they cannot be processed by the operations and control panels.
  - The percent character has a special meaning to RACF. If RACF is used as the external security manager, names must not contain the percent. If they do, those names are not included in any security checks when RACF generic profiles are used.
- On IBM i, names containing lowercase characters, forward slash, or percent, must be enclosed in quotation marks when specified on commands. These quotation marks must not be specified for names that occur as fields in structures or as parameters on calls.

The full topic name can be built from two different fields: *ObjectName* and *ObjectString*. For details of how these two fields are used, see "Using topic strings" on page 2566.

The following points apply to the types of object indicated:

- If *ObjectName* is the name of a model queue, the queue manager creates a dynamic queue with the attributes of the model queue, and returns in the *ObjectName* field the name of the queue created. A model queue can be specified only on the MQOPEN call; a model queue is not valid on the MQPUT1 call.
- If *ObjectName* is the name of an alias queue with TARGTYPE(TOPIC), a security check is first made on the named alias queue; this is normal when alias queues are used. When the security check completes successfully, the MQOPEN call will continue and will behave like an MQOPEN call on an MQOT_TOPIC; this includes making a security check against the administrative topic object.
- If *ObjectName* and *ObjectQMgrName* identify a shared queue owned by the queue-sharing group to which the local queue manager belongs, there must not also be a queue definition of the same name on the local queue manager. If there is such a definition (a local queue, alias queue, remote queue, or model queue), the call fails with reason code MQRC_OBJECT_NOT_UNIQUE.
- If the object being opened is a distribution list (that is, *RecsPresent* is present and greater than zero), *ObjectName* must be blank or the null string. If this condition is not satisfied, the call fails with reason code MQRC_OBJECT_NAME_ERROR.
- If *ObjectType* is MQOT_Q_MGR, special rules apply; in this case the name must be entirely blank up to the first null character or the end of the field.

This is an input/output field for the MQOPEN call when *ObjectName* is the name of a model queue, and an input-only field in all other cases. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ObjectQMgrName (MQCHAR48):

This is the name of the queue manager on which the *ObjectName* object is defined. The characters that are valid in the name are the same as those for *ObjectName* (see "ObjectName (MQCHAR48)" on page 2457). A name that is entirely blank up to the first null character or the end of the field denotes the queue manager to which the application is connected (the local queue manager).

The following points apply to the types of object indicated:

- If *ObjectType* is MQOT_TOPIC, MQOT_NAMELIST, MQOT_PROCESS, or MQOT_Q_MGR, *ObjectQMgrName* must be blank or the name of the local queue manager.
- If *ObjectName* is the name of a model queue, the queue manager creates a dynamic queue with the attributes of the model queue, and returns in the *ObjectQMgrName* field the name of the queue manager on which the queue is created; this is the name of the local queue manager. A model queue can be specified only on the MQOPEN call; a model queue is not valid on the MQPUT1 call.
- If *ObjectName* is the name of a cluster queue, and *ObjectQMgrName* is blank, the destination of messages sent using the queue handle returned by the MQOPEN call is chosen by the queue manager (or cluster workload exit, if one is installed) as follows:
  - If MQOO_BIND_ON_OPEN is specified, the queue manager selects a particular instance of the cluster queue while processing the MQOPEN call, and all messages put using this queue handle are sent to that instance.
  - If MQOO_BIND_NOT_FIXED is specified, the queue manager can choose a different instance of the destination queue (residing on a different queue manager in the cluster) for each successive MQPUT call that uses this queue handle.

  If the application needs to send a message to a *specific* instance of a cluster queue (that is, a queue instance that resides on a particular queue manager in the cluster), the application must specify the name of that queue manager in the *ObjectQMgrName* field. This forces the local queue manager to send the message to the specified destination queue manager.
- If *ObjectName* is the name of a shared queue that is owned by the queue-sharing group to which the local queue manager belongs, *ObjectQMgrName* can be the name of the queue-sharing group, the name of the local queue manager, or blank; the message is placed on the same queue whichever of these values is specified.

  Queue-sharing groups are supported only on z/OS.
- If *ObjectName* is the name of a shared queue that is owned by a remote queue-sharing group (that is, a queue-sharing group to which the local queue manager does not belong), *ObjectQMgrName* must be the name of the queue-sharing group. You can use the name of a queue manager that belongs to that group, but this can delay the message if that particular queue manager is not available when the message arrives at the queue-sharing group.
- If the object being opened is a distribution list (that is, *RecsPresent* is greater than zero), *ObjectQMgrName* must be blank or the null string. If this condition is not satisfied, the call fails with reason code MQRC_OBJECT_Q_MGR_NAME_ERROR.

This is an input/output field for the MQOPEN call when *ObjectName* is the name of a model queue, and an input-only field in all other cases. The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ObjectRecOffset (MQLONG):

This is the offset in bytes of the first MQOR object record from the start of the MQOD structure. The offset can be positive or negative. *ObjectRecOffset* is used only when a distribution list is being opened. The field is ignored if *RecsPresent* is zero.

When a distribution list is being opened, an array of one or more MQOR object records must be provided in order to specify the names of the destination queues in the distribution list. This can be done in one of two ways:

- By using the offset field *ObjectRecOffset*.

  In this case, the application must declare its own structure containing an MQOD followed by the array of MQOR records (with as many array elements as are needed), and set *ObjectRecOffset* to the offset of the first element in the array from the start of the MQOD. Ensure that this offset is correct and has a value that can be accommodated within an MQLONG (the most restrictive programming language is COBOL, for which the valid range is -999 999 999 through +999 999 999).

  Use *ObjectRecOffset* for programming languages that do not support the pointer data type, or that implement the pointer data type in a way that is not portable to different environments (for example, the COBOL programming language).
- By using the pointer field *ObjectRecPtr*.

  In this case, the application can declare the array of MQOR structures separately from the MQOD structure, and set *ObjectRecPtr* to the address of the array.

  Use *ObjectRecPtr* for programming languages that support the pointer data type in a way that is portable to different environments (for example, the C programming language).

Whatever technique you choose, use one of *ObjectRecOffset* and *ObjectRecPtr* ; the call fails with reason code MQRC_OBJECT_RECORDS_ERROR if both are zero, or both are nonzero.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_2.

ObjectRecPtr (MQPTR):

This is the address of the first MQOR object record. *ObjectRecPtr* is used only when a distribution list is being opened. The field is ignored if *RecsPresent* is zero.

You can use either *ObjectRecPtr* or *ObjectRecOffset* to specify the object records, but not both; for the description of the *ObjectRecOffset* field, see "ObjectRecOffset (MQLONG)." If you do not use *ObjectRecPtr*, set it to the null pointer or null bytes.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQOD_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

ObjectString (MQCHARV):

The ObjectString field specifies the long object name.

This specifies the long object name to be used. This field is only referenced for certain values of *ObjectType*, and is ignored for all other values. See the description of *ObjectType* for details of which values indicate that this field is used.

If *ObjectString* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code MQRC_OBJECT_STRING_ERROR.

This is an input field. The initial values of the fields in this structure are the same as those in the MQCHARV structure.

The full topic name can be built from two different fields: *ObjectName* and *ObjectString*. For details of how these two fields are used, see "Using topic strings" on page 2566.

ObjectType (MQLONG):

The type of object being named in the object descriptor. Possible values are:

MQOT_CLNTCONN_CHANNEL
Client connection channel. The name of the object is found in the *ObjectName* field.

MQOT_Q
Queue. The name of the object is found in the *ObjectName* field.

MQOT_NAMELIST
Namelist. The name of the object is found in the *ObjectName* field

MQOT_PROCESS
Process definition. The name of the object is found in the *ObjectName* field

MQOT_Q_MGR
Queue manager. The name of the object is found in the *ObjectName* field

MQOT_TOPIC
Topic. The full topic name can be built from two different fields: *ObjectName* and *ObjectString*.

For details of how those two fields are used, see "Using topic strings" on page 2566.

This is always an input field. The initial value of this field is MQOT_Q.

RecsPresent (MQLONG):

This is the number of MQOR object records that have been provided by the application. If this number is greater than zero, it indicates that a distribution list is being opened, with *RecsPresent* being the number of destination queues in the list. A distribution list can contain only one destination.

The value of *RecsPresent* must not be less than zero, and if it is greater than zero *ObjectType* must be MQOT_Q; the call fails with reason code MQRC_RECS_PRESENT_ERROR if these conditions are not satisfied.

On z/OS, this field must be zero.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_2.

ResObjectString (MQCHARV):

The ResObjectString field is the long object name after the queue manager resolves the name provided in the *ObjectName* field.

This field is returned only for topics and queue aliases that reference a topic object.

If the long object name is provided in *ObjectString* and nothing is provided in *ObjectName*, then the value returned in this field is the same as provided in *ObjectString*.

If this field is omitted (that is ResObjectString.VSBufSize is zero) then the *ResObjectString* will not be returned, but the length will be returned in ResObjectString.VSLength.

If the buffer length (provided in ResObjectStrng.VSBufSize) is shorter than the full *ResObjectString*, the string will be truncated and will return as many of the rightmost characters as can fit in the provided buffer.

If *ResObjectString* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code MQRC_RES_OBJECT_STRING_ERROR.

ResolvedQMgrName (MQCHAR48):

This is the name of the destination queue manager after the local queue manager resolves the name. The name returned is the name of the queue manager that owns the queue identified by *ResolvedQName*. *ResolvedQMgrName* can be the name of the local queue manager.

If *ResolvedQName* is a shared queue that is owned by the queue-sharing group to which the local queue manager belongs, *ResolvedQMgrName* is the name of the queue-sharing group. If the queue is owned by some other queue-sharing group, *ResolvedQName* can be the name of the queue-sharing group or the name of a queue manager that is a member of the queue-sharing group (the nature of the value returned is determined by the queue definitions that exist at the local queue manager).

A nonblank value is returned only if the object is a single queue opened for browse, input, or output (or any combination). If the object opened is any of the following, *ResolvedQMgrName* is set to blanks:

- Not a queue
- A queue, but not opened for browse, input, or output
- A cluster queue with MQOO_BIND_NOT_FIXED specified (or with MQOO_BIND_AS_Q_DEF in effect when the **DefBind** queue attribute has the value MQBND_BIND_NOT_FIXED)
- A distribution list

This is an output field. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages. This field is ignored if *Version* is less than MQOD_VERSION_3.

ResolvedQName (MQCHAR48):

This is the name of the destination queue after the local queue manager resolves the name. The name returned is the name of a queue that exists on the queue manager identified by *ResolvedQMgrName*.

A nonblank value is returned only if the object is a single queue opened for browse, input, or output (or any combination). If the object opened is any of the following, *ResolvedQName* is set to blanks:

- Not a queue
- A queue, but not opened for browse, input, or output
- A distribution list
- An alias queue that references a topic object (refer to ResObjectString instead).
- An alias queue that resolves to a topic object.

This is an output field. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages. This field is ignored if *Version* is less than MQOD_VERSION_3.

ResolvedType (MQLONG):

The type of the resolved (base) object being opened.

The possible values are:

MQOT_Q
The resolved object is a queue. This value applies when a queue is opened directly or when an alias queue pointing to a queue is opened.

MQOT_TOPIC
The resolved object is a topic. This value applies when a topic is opened directly or when an alias queue pointing to a topic object is opened.

MQOT_NONE
The resolved type is neither a queue nor a topic.

ResponseRecOffset (MQLONG):

This is the offset in bytes of the first MQRR response record from the start of the MQOD structure. The offset can be positive or negative. *ResponseRecOffset* is used only when a distribution list is being opened. The field is ignored if *RecsPresent* is zero.

When a distribution list is being opened, you can provide an array of one or more MQRR response records in order to identify the queues that failed to open (*CompCode* field in MQRR), and the reason for each failure (*Reason* field in MQRR). The data is returned in the array of response records in the same order as the queue names occur in the array of object records. The queue manager sets the response records only when the outcome of the call is mixed (that is, some queues were opened successfully while others failed, or all failed but for different reasons); reason code MQRC_MULTIPLE_REASONS from the call indicates this case. If the same reason code applies to all queues, that reason is returned in the **Reason** parameter of the MQOPEN or MQPUT1 call, and the response records are not set. Response records are optional, but if they are supplied there must be *RecsPresent* of them.

The response records can be provided in the same way as the object records, either by specifying an offset in *ResponseRecOffset*, or by specifying an address in *ResponseRecPtr* ; for details of how to do this, see "ObjectRecOffset (MQLONG)" on page 2459. However, no more than one of *ResponseRecOffset* and *ResponseRecPtr* can be used; the call fails with reason code MQRC_RESPONSE_RECORDS_ERROR if both are nonzero.

For the MQPUT1 call, these response records are used to return information about errors that occur when the message is sent to the queues in the distribution list, as well as errors that occur when the queues are opened. The completion code and reason code from the put operation for a queue replace those from the open operation for that queue only if the completion code from the latter was MQCC_OK or MQCC_WARNING.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_2.

ResponseRecPtr (MQPTR):

This is the address of the first MQRR response record. *ResponseRecPtr* is used only when a distribution list is being opened. The field is ignored if *RecsPresent* is zero.

Use either *ResponseRecPtr* or *ResponseRecOffset* to specify the response records, but not both; for details, see "ResponseRecOffset (MQLONG)" on page 2462. If you do not use *ResponseRecPtr*, set it to the null pointer or null bytes.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQOD_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

SelectionString (MQCHARV):

This is the string used to provide the selection criteria used when retrieving messages off a queue.

SelectionString must not be provided in the following cases:

- If *ObjectType* is not MQOT_Q
- If the queue being opened is not being opened using one of the MQOO_BROWSE, or MQOO_INPUT_* options

If *SelectionString* is provided in these cases, the call fails with reason code MQRC_SELECTOR_INVALID_FOR_TYPE.

If *SelectionString* is specified incorrectly, according to the description of how to use the "MQCHARV - Variable Length String" on page 2250 structure, or if it exceeds the maximum length, the call fails with reason code MQRC_SELECTION_STRING_ERROR. The maximum length of *SelectionString* is MQ_SELECTOR_LENGTH.

SelectionString usage is described in Selectors.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQOD_STRUC_ID
Identifier for object descriptor structure.

For the C programming language, the constant MQOD_STRUC_ID_ARRAY is also defined; this has the same value as MQOD_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQOD_STRUC_ID.

UnknownDestCount (MQLONG):

This is the number of queues in the distribution list that resolve to remote queues and that were opened successfully. If present, this field is also set when opening a single queue that is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_1.

Version (MQLONG):

This is the structure version number; the value must be one of the following:

MQOD_VERSION_1
Version-1 object descriptor structure.

MQOD_VERSION_2
Version-2 object descriptor structure.

MQOD_VERSION_3
Version-3 object descriptor structure.

MQOD_VERSION_4
Version-4 object descriptor structure.

All versions are supported in all IBM MQ V7.0 environments.

Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQOD_CURRENT_VERSION
Current version of object descriptor structure.

This is always an input field. The initial value of this field is MQOD_VERSION_1.

Initial values and language declarations for MQOD:

Field name	Name of constant	Value of constant
StrucId	MQOD_STRUC_ID	'OD¬¬'
Version	MQOD_VERSION_1	1
ObjectType	MQOT_Q	1
ObjectName	None	Null string or blanks
ObjectQMgrName	None	Null string or blanks
DynamicQName	None	'CSQ.*' on z/OS ; 'AMQ.*' otherwise
AlternateUserId	None	Null string or blanks
RecsPresent	None	0
KnownDestCount	None	0
UnknownDestCount	None	0
InvalidDestCount	None	0
ObjectRecOffset	None	0
ResponseRecOffset	None	0
ObjectRecPtr	None	Null pointer or null bytes
ResponseRecPtr	None	Null pointer or null bytes
AlternateSecurityId	MQSID_NONE	Nulls
ResolvedQName	None	Null string or blanks
ResolvedQMgrName	None	Null string or blanks
ObjectString	MQCHARV_DEFAULT	As defined for MQCHARV
SelectionString	MQCHARV_DEFAULT	As defined for MQCHARV
ResObjectString	MQCHARV_DEFAULT	As defined for MQCHARV
ResolvedType	MQOT_NONE	0

Notes:

1. The symbol ¬ represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQOD_DEFAULT contains the values that are listed in the table. It can be used in the following way to provide initial values for the fields in the structure:

```
MQOD MyOD = {MQOD_DEFAULT};
```

C declaration for MQOD:

```
typedef struct tagMQOD MQOD;
struct tagMQOD {
  MQCHAR4    StrucId;              /* Structure identifier */
  MQLONG     Version;              /* Structure version number */
  MQLONG     ObjectType;           /* Object type */
  MQCHAR48   ObjectName;           /* Object name */
  MQCHAR48   ObjectQMgrName;       /* Object queue manager name */
  MQCHAR48   DynamicQName;         /* Dynamic queue name */
  MQCHAR12   AlternateUserId;      /* Alternate user identifier */
  /* Ver:1 */
  MQLONG     RecsPresent;          /* Number of object records present */
  MQLONG     KnownDestCount;       /* Number of local queues opened
                                      successfully */
  MQLONG     UnknownDestCount;     /* Number of remote queues opened
                                      successfully */
  MQLONG     InvalidDestCount;     /* Number of queues that failed to
                                      open */
  MQLONG     ObjectRecOffset;      /* Offset of first object record from
                                      start of MQOD */
  MQLONG     ResponseRecOffset;    /* Offset of first response record
                                      from start of MQOD */
  MQPTR      ObjectRecPtr;         /* Address of first object record */
  MQPTR      ResponseRecPtr;       /* Address of first response record */
  /* Ver:2 */
  MQBYTE40   AlternateSecurityId;  /* Alternate security identifier */
  MQCHAR48   ResolvedQName;        /* Resolved queue name */
  MQCHAR48   ResolvedQMgrName;     /* Resolved queue manager name */
  /* Ver:3 */
  MQCHARV    ObjectString;         /* Object Long name */
  MQCHARV    SelectionString;      /* Message Selector */
  MQCHARV    ResObjectString;      /* Resolved Long object name*/
  MQLONG     ResolvedType          /* Alias queue resolved
                                      oject type */
  /* Ver:4 */
};
```

COBOL declaration for MQOD:

```
** MQOD structure
 10  MQOD.
** Structure identifier
 15  MQOD-STRUCID                     PIC X(4).
** Structure version number
 15  MQOD-VERSION                     PIC S9(9) BINARY.
** Object type
 15  MQOD-OBJECTTYPE                  PIC S9(9) BINARY.
** Object name
 15  MQOD-OBJECTNAME                  PIC X(48).
** Object queue manager name
 15  MQOD-OBJECTQMGRNAME              PIC X(48).
** Dynamic queue name
 15  MQOD-DYNAMICQNAME                PIC X(48).
** Alternate user identifier
 15  MQOD-ALTERNATEUSERID             PIC X(12).
** Number of object records present
 15  MQOD-RECSPRESENT                 PIC S9(9) BINARY.
** Number of local queues opened successfully
 15  MQOD-KNOWNDESTCOUNT              PIC S9(9) BINARY.
** Number of remote queues opened successfully
 15  MQOD-UNKNOWNDESTCOUNT            PIC S9(9) BINARY.
** Number of queues that failed to open
 15  MQOD-INVALIDDESTCOUNT            PIC S9(9) BINARY.
** Offset of first object record from start of MQOD
 15  MQOD-OBJECTRECOFFSET             PIC S9(9) BINARY.
** Offset of first response record from start of MQOD
 15  MQOD-RESPONSERECOFFSET           PIC S9(9) BINARY.
```

```
** Address of first object record
 15  MQOD-OBJECTRECPTR                POINTER.
** Address of first response record
 15  MQOD-RESPONSERECPTR              POINTER.
** Alternate security identifier
 15  MQOD-ALTERNATESECURITYID         PIC X(40).
** Resolved queue name
 15  MQOD-RESOLVEDQNAME               PIC X(48).
** Resolved queue manager name
 15  MQOD-RESOLVEDQMGRNAME            PIC X(48).
** Object Long name
 15 MQOD-OBJECTSTRING.
** Address of variable length string
   20  MQOD-OBJECTSTRING-VSPTR        POINTER.
** Offset of variable length string
   20  MQOD-OBJECTSTRING-VSOFFSET     PIC S9(9) BINARY.
** size of buffer
   20  MQOD-OBJECTSTRING-VSBUFSIZE    PIC S9(9) BINARY.
** Length of variable length string
   20  MQOD-OBJECTSTRING-VSLENGTH     PIC S9(9) BINARY.
** CCSID of variable length string
   20  MQOD-OBJECTSTRING-VSCCSID      PIC S9(9) BINARY.
** Message Selector
 15 MQOD-SELECTIONSTRING.
** Address of variable length string
   20  MQOD-SELECTIONSTRING-VSPTR     POINTER.
** Offset of variable length string
   20  MQOD-SELECTIONSTRING-VSOFFSET  PIC S9(9) BINARY.
** size of buffer
   20  MQOD-SELECTIONSTRING-VSBUFSIZE PIC S9(9) BINARY.
** Length of variable length string
   20  MQOD-SELECTIONSTRING-VSLENGTH  PIC S9(9) BINARY.
** CCSID of variable length string
   20  MQOD-SELECTIONSTRING-VSCCSID   PIC S9(9) BINARY.
** Resolved Long object name
 15 MQOD-RESOBJECTSTRING.
** Address of variable length string
   20  MQOD-RESOBJECTSTRING-VSPTR     POINTER.
** Offset of variable length string
   20  MQOD-RESOBJECTSTRING-VSOFFSET  PIC S9(9) BINARY.
** size of buffer
   20  MQOD-RESOBJECTSTRING-VSBUFSIZE PIC S9(9) BINARY.
** Length of variable length string
   20  MQOD-RESOBJECTSTRING-VSLENGTH  PIC S9(9) BINARY.
** CCSID of variable length string
   20  MQOD-RESOBJECTSTRING-VSCCSID   PIC S9(9) BINARY.
** Alias queue resolved object type
   15  MQOD-RESOLVEDTYPE              PIC S9(9) BINARY.
```

PL/I declaration for MQOD:

```
dcl
 1 MQOD based,
  3 StrucId             char(4),       /* Structure identifier */
  3 Version             fixed bin(31), /* Structure version number */
  3 ObjectType          fixed bin(31), /* Object type */
  3 ObjectName          char(48),      /* Object name */
  3 ObjectQMgrName      char(48),      /* Object queue manager name */
  3 DynamicQName        char(48),      /* Dynamic queue name */
  3 AlternateUserId     char(12),      /* Alternate user identifier */
  3 RecsPresent         fixed bin(31), /* Number of object records
                                          present */
  3 KnownDestCount      fixed bin(31), /* Number of local queues opened
                                          successfully */
  3 UnknownDestCount    fixed bin(31), /* Number of remote queues opened
                                          successfully */
  3 InvalidDestCount    fixed bin(31), /* Number of queues that failed to
                                          open */
  3 ObjectRecOffset     fixed bin(31), /* Offset of first object record
                                          from start of MQOD */
  3 ResponseRecOffset   fixed bin(31), /* Offset of first response record
                                          from start of MQOD */
  3 ObjectRecPtr        pointer,       /* Address of first object record */
  3 ResponseRecPtr      pointer,       /* Address of first response
                                          record */
  3 AlternateSecurityId char(40),      /* Alternate security identifier */
  3 ResolvedQName       char(48),      /* Resolved queue name */
  3 ResolvedQMgrName    char(48),      /* Resolved queue manager name */
  3 ObjectString,                      /* Object Long name */
   5 VSPtr              pointer,       /* Address of variable length string */
   5 VSOffset           fixed bin(31), /* Offset of variable length string */
   5 VSBufSize          fixed bin(31), /* size of buffer */
   5 VSLength           fixed bin(31), /* Length of variable length string */
   5 VSCCSID            fixed bin(31), /* CCSID of variable length string */
  3 SelectionString,                   /* Message Selection */
   5 VSPtr              pointer,       /* Address of variable length string */
   5 VSOffset           fixed bin(31), /* Offset of variable length string */
   5 VSBufSize          fixed bin(31), /* size of buffer */
   5 VSLength           fixed bin(31), /* Length of variable length string */
   5 VSCCSID            fixed bin(31), /* CCSID of variable length string */
  3 ResObjectString,                   /* Resolved Long object name */
   5 VSPtr              pointer,       /* Address of variable length string */
   5 VSOffset           fixed bin(31), /* Offset of variable length string */
   5 VSBufSize          fixed bin(31), /* size of buffer */
   5 VSLength           fixed bin(31), /* Length of variable length string */
   5 VSCCSID            fixed bin(31), /* CCSID of variable length string */
  3 ResolvedType        fixed bin(31); /* Alias queue resolved object type */
```

High Level Assembler declaration for MQOD:

```
MQOD                           DSECT
MQOD_STRUCID                   DS   CL4   Structure identifier
MQOD_VERSION                   DS   F     Structure version number
MQOD_OBJECTTYPE                DS   F     Object type
MQOD_OBJECTNAME                DS   CL48  Object name
MQOD_OBJECTQMGRNAME            DS   CL48  Object queue manager name
MQOD_DYNAMICQNAME              DS   CL48  Dynamic queue name
MQOD_ALTERNATEUSERID           DS   CL12  Alternate user identifier
MQOD_RECSPRESENT               DS   F     Number of object records present
MQOD_KNOWNDESTCOUNT            DS   F     Number of local queues opened
*                                         successfully
MQOD_UNKNOWNDESTCOUNT          DS   F     Number of remote queues opened
*                                         successfully
MQOD_INVALIDDESTCOUNT          DS   F     Number of queues that failed to
*                                         open
MQOD_OBJECTRECOFFSET           DS   F     Offset of first object record from
*                                         start of MQOD
MQOD_RESPONSERECOFFSET         DS   F     Offset of first response record
*                                         from start of MQOD
MQOD_OBJECTRECPTR              DS   F     Address of first object record
MQOD_RESPONSERECPTR            DS   F     Address of first response record
MQOD_ALTERNATESECURITYID       DS   XL40  Alternate security identifier
MQOD_RESOLVEDQNAME             DS   CL48  Resolved queue name
MQOD_RESOLVEDQMGRNAME          DS   CL48  Resolved queue manager name
MQOD_OBJECTSTRING              DS   F     Object Long name
MQOD_OBJECTSTRING_VSPTR        DS   F     Address of variable length string
MQOD_OBJECTSTRING_VSOFFSET     DS   F     Offset of variable length string
MQOD_OBJECTSTRING_VSBUFSIZE    DS   F     size of buffer
MQOD_OBJECTSTRING_VSLENGTH     DS   F     Length of variable length string
MQOD_OBJECTSTRING_VSCCSID      DS   F     CCSID of variable length string
MQOD_OBJECTSTRING_LENGTH       EQU  *- MQOD_OBJECTSTRING
                               ORG  MQOD_OBJECTSTRING
MQOD_OBJECTSTRING_AREA         DS   CL(MQOD_OBJECTSTRING_LENGTH)
*
MQOD_SELECTIONSTRING           DS   F     Message Selector
MQOD_SELECTIONSTRING_VSPTR     DS   F     Address of variable length string
MQOD_SELECTIONSTRING_VSOFFSET  DS   F     Offset of variable length string
MQOD_SELECTIONSTRING_VSBUFSIZE DS   F     size of buffer
MQOD_SELECTIONSTRING_VSLENGTH  DS   F     Length of variable length string
MQOD_SELECTIONSTRING_VSCCSID   DS   F     CCSID of variable length string
MQOD_SELECTIONSTRING_LENGTH    EQU  *- MQOD_SELECTIONSTRING
                               ORG  MQOD_SELECTIONSTRING
MQOD_SELECTIONSTRING_AREA      DS   CL(MQOD_SELECTIONSTRING_LENGTH)
*
MQOD_RESOBJECTSTRING           DS   F     Resolved Long object name
MQOD_RESOBJECTSTRING_VSPTR     DS   F     Address of variable length string
MQOD_RESOBJECTSTRING_VSOFFSET  DS   F     Offset of variable length string
MQOD_RESOBJECTSTRING_VSBUFSIZE DS   F     size of buffer
MQOD_RESOBJECTSTRING_VSLENGTH  DS   F     Length of variable length string
MQOD_RESOBJECTSTRING_VSCCSID   DS   F     CCSID of variable length string
MQOD_RESOBJECTSTRING_LENGTH    EQU  *- MQOD_RESOBJECTSTRING
                               ORG  MQOD_RESOBJECTSTRING
MQOD_RESOBJECTSTRING_AREA      DS   CL(MQOD_RESOBJECTSTRING_LENGTH)
MQOD_RESOLVEDTYPE              DS   F     Alias queue object resolved type
*
MQOD_LENGTH                    EQU  *-MQOD
                               ORG  MQOD
MQOD_AREA                      DS   CL(MQOD_LENGTH)
```

Visual Basic declaration for MQOD:

```
Type MQOD
  StrucId             As String*4  'Structure identifier'
  Version             As Long      'Structure version number'
  ObjectType          As Long      'Object type'
  ObjectName          As String*48 'Object name'
  ObjectQMgrName      As String*48 'Object queue manager name'
  DynamicQName        As String*48 'Dynamic queue name'
  AlternateUserId     As String*12 'Alternate user identifier'
  RecsPresent         As Long      'Number of object records present'
  KnownDestCount      As Long      'Number of local queues opened'
                                   'successfully'
  UnknownDestCount    As Long      'Number of remote queues opened'
                                   'successfully'
  InvalidDestCount    As Long      'Number of queues that failed to'
                                   'open'
  ObjectRecOffset     As Long      'Offset of first object record from'
                                   'start of MQOD'
  ResponseRecOffset   As Long      'Offset of first response record'
                                   'from start of MQOD'
  ObjectRecPtr        As MQPTR     'Address of first object record'
  ResponseRecPtr      As MQPTR     'Address of first response record'
  AlternateSecurityId As MQBYTE40  'Alternate security identifier'
  ResolvedQName       As String*48 'Resolved queue name'
  ResolvedQMgrName    As String*48 'Resolved queue manager name'
End Type
```

MQOR - Object record:

The following table summarizes the fields in the structure.

Table 222. Fields in MQOR

Field	Description	Topic
ObjectName	Object name	ObjectName
ObjectQMgrName	Object queue manager name	ObjectQMgrName

Overview for MQOR:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ MQI clients connected to these systems.

Purpose: Use the MQOR structure to specify the queue name and queue manager name of a single destination queue. MQOR is an input structure for the MQOPEN and MQPUT1 calls.

Character set and encoding: Data in MQOR must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Usage: By providing an array of these structures on the MQOPEN call, you can open a list of queues; this list is called a *distribution list*. Each message put using the queue handle returned by that MQOPEN call is placed on each of the queues in the list, provided that the queue was opened successfully.

Fields for MQOR:

The MQOR structure contains the following fields; the fields are described in **alphabetical order**:

ObjectName (MQCHAR48):

This is the same as the *ObjectName* field in the MQOD structure (see MQOD for details), except that:

- It must be the name of a queue.
- It must not be the name of a model queue.

This is always an input field. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ObjectQMgrName (MQCHAR48):

This is the same as the *ObjectQMgrName* field in the MQOD structure (see MQOD for details).

This is always an input field. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

Initial values and language declarations for MQOR:

Table 223. Initial values of fields in MQOR for MQOR

Field name	Name of constant	Value of constant
ObjectName	None	Null string or blanks
ObjectQMgrName	None	Null string or blanks

Notes:

1. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
2. In the C programming language, the macro variable MQOR_DEFAULT contains the values that are listed in the table. It can be used in the following way to provide initial values for the fields in the structure:

```
MQOR MyOR = {MQOR_DEFAULT};
```

C declaration for MQOR:

```
typedef struct tagMQOR MQOR;
struct tagMQOR {
  MQCHAR48  ObjectName;      /* Object name */
  MQCHAR48  ObjectQMgrName;  /* Object queue manager name */
};
```

COBOL declaration for MQOR:

```
**   MQOR structure
  10 MQOR.
**    Object name
   15 MQOR-OBJECTNAME     PIC X(48).
**    Object queue manager name
   15 MQOR-OBJECTQMGRNAME PIC X(48).
```

PL/I declaration for MQOR:

```
dcl
 1 MQOR based,
  3 ObjectName     char(48), /* Object name */
  3 ObjectQMgrName char(48); /* Object queue manager name */
```

Visual Basic declaration for MQOR:

```
Type MQOR
  ObjectName     As String*48 'Object name'
  ObjectQMgrName As String*48 'Object queue manager name'
End Type
```

MQPD - Property descriptor:

The following table summarizes the fields in the structure.

Table 224. Fields in MQPD

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options	Options
Support	Required support for message property	Support
Context	Message context to which property belongs	Context
CopyOptions	Copy options to which property belongs	CopyOptions

Overview for MQPD:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, z/OS and IBM MQ MQI clients.

Purpose: The **MQPD** is used to define the attributes of a property. The structure is an input/output parameter on the MQSETMP call and an output parameter on the MQINQMP call.

Character set and encoding: Data in **MQPD** must be in the character set of the application and encoding of the application (**MQENC_NATIVE**).

Fields for MQPD:

The MQPD structure contains the following fields; the fields are described in **alphabetical order**:

Context (MQLONG):

This describes what message context the property belongs to.

When a queue manager receives a message containing an IBM MQ-defined property that the queue manager recognizes as being incorrect, the queue manager corrects the value of the *Context* field.

The following option can be specified:

MQPD_USER_CONTEXT
The property is associated with the user context.

No special authorization is required to be able to set a property associated with the user context using the MQSETMP call.

On a IBM WebSphere MQ Version 7.0 queue manager, a property associated with the user context is saved as described for MQOO_SAVE_ALL_CONTEXT. An MQPUT call with MQPMO_PASS_ALL_CONTEXT specified, causes the property to be copied from the saved context into the new message.

If the option previously described is not required, the following option can be used:

MQPD_NO_CONTEXT
The property is not associated with a message context.

An unrecognized value is rejected with a *Reason* code of MQRC_PD_ERROR

This is an input/output field to the MQSETMP call and an output field from the MQINQMP call. The initial value of this field is MQPD_NO_CONTEXT.

CopyOptions (MQLONG):

This describes which type of messages the property should be copied into. This is an output only field for recognized IBM MQ defined properties; IBM MQ sets the appropriate value.

When a queue manager receives a message containing an IBM MQ defined property that the queue manager recognizes as being incorrect, the queue manager corrects the value of the *CopyOptions* field.

You can specify one or more of these options. To specify more than one option, either add the values together (do not add the same constant more than once), or combine the values using the bitwise OR operation (if the programming language supports bit operations).

MQCOPY_FORWARD
This property is copied into a message being forwarded.

MQCOPY_PUBLISH
This property is copied into the message received by a subscriber when a message is being published.

MQCOPY_REPLY
This property is copied into a reply message.

MQCOPY_REPORT
This property is copied into a report message.

MQCOPY_ALL
This property is copied into all types of subsequent messages.

Default option: The following option can be specified to supply the default set of copy options:

MQCOPY_DEFAULT
This property is copied into a message being forwarded, into a report message, or into a message received by a subscriber when a message is being published.

This is equivalent to specifying the combination of options MQCOPY_FORWARD, plus MQCOPY_REPORT, plus MQCOPY_PUBLISH.

If none of the options that are described previously is required, use the following option:

MQCOPY_NONE
Use this value to indicate that no other copy options are specified; programmatically no relationship exists between this property and subsequent messages. This is always returned for message descriptor properties.

This is an input/output field to the MQSETMP call and an output field from the MQINQMP call. The initial value of this field is MQCOPY_DEFAULT.

Options (MQLONG):

The value must be:

MQPD_NONE
No options specified

This is always an input field. The initial value of this field is MQPD_NONE.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQPD_STRUC_ID
Identifier for property descriptor structure.

For the C programming language, the constant **MQPD_STRUC_ID_ARRAY** is also defined; this has the same value as **MQPD_STRUC_ID**, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is **MQPD_STRUC_ID**.

Support (MQLONG):
This field describes what level of support for the message property is required of the queue manager, in order for the message containing this property to be put to a queue. This applies only to IBM MQ-defined properties; support for all other properties is optional.

The field is automatically set to the correct value when the IBM MQ-defined property is known by the queue manager. If the property is not recognized, MQPD_SUPPORT_OPTIONAL is assigned. When a queue manager receives a message containing an IBM MQ-defined property that the queue manager recognizes as being incorrect, the queue manager corrects the value of the *Support* field.

When setting an IBM MQ-defined property using the MQSETMP call on a message handle where the MQCMHO_NO_VALIDATION option was set, *Support* becomes an input field. This allows an application to put an IBM MQ-defined property, with the correct value, where the property is unsupported by the connected queue manager, but where the message is intended to be processed on another queue manager.

The value MQPD_SUPPORT_OPTIONAL is always assigned to properties that are not IBM MQ-defined properties.

If a IBM WebSphere MQ Version 7.0 queue manager, that supports message properties, receives a property that contains an unrecognized *Support* value, the property is treated as if:

- MQPD_SUPPORT_REQUIRED was specified if any of the unrecognized values are contained in the MQPD_REJECT_UNSUP_MASK.
- MQPD_SUPPORT_REQUIRED_IF_LOCAL was specified if any of the unrecognized values are contained in the MQPD_ACCEPT_UNSUP_IF_XMIT_MASK
- MQPD_SUPPORT_OPTIONAL was specified otherwise.

One of the following values is returned by the MQINQMP call, or one of the values can be specified, when using the MQSETMP call on a message handle where the MQCMHO_NO_VALIDATION option is set:

MQPD_SUPPORT_OPTIONAL
The property is accepted by a queue manager even if it is not supported. The property can be discarded in order for the message to flow to a queue manager that does not support message properties. This value is also assigned to properties that are not IBM MQ-defined.

MQPD_SUPPORT_REQUIRED
Support for the property is required. The message is rejected by a queue manager that does not support the IBM MQ-defined property. The MQPUT or MQPUT1 call fails with completion code MQCC_FAILED and reason code MQRC_UNSUPPORTED_PROPERTY.

MQPD_SUPPORT_REQUIRED_IF_LOCAL
The message is rejected by a queue manager that does not support the IBM MQ-defined property if the message is destined for a local queue. The MQPUT or MQPUT1 call fails with completion code MQCC_FAILED and reason code MQRC_UNSUPPORTED_PROPERTY.

The MQPUT or MQPUT1 call succeeds if the message is destined for a remote queue manager.

This is an output field on the MQINQMP call and an input field on the MQSETMP call if the message handle was created with the MQCMHO_NO_VALIDATION option set. The initial value of this field is MQPD_SUPPORT_OPTIONAL.

Version (MQLONG):

This is the structure version number; the value must be:

MQPD_VERSION_1
Version-1 property descriptor structure.

The following constant specifies the version number of the current version:

MQPD_CURRENT_VERSION
Current version of property descriptor structure.

This is always an input field. The initial value of this field is **MQPD_VERSION_1**.

Initial values and language declarations for MQPD:

Table 225. Initial values of fields in MQPD

Field name	Name of constant	Value of constant
StrucId	MQPD_STRUC_ID	'PD'
Version	MQPD_VERSION_1	1
Options	MQPD_NONE	0
Support	MQPD_SUPPORT_OPTIONAL	0
Context	MQPD_NO_CONTEXT	0
CopyOptions	MQCOPY_DEFAULT	0

Notes:

1. In the C programming language, the macro variable MQPD_DEFAULT contains the values that are listed in the table. It can be used in the following way to provide initial values for the fields in the structure:

   ```
   MQPD MyPD = {MQPD_DEFAULT};
   ```

C declaration for MQPD:

```
typedef struct tagMQPD MQPD;
struct tagMQPD {
  MQCHAR4  StrucId;     /* Structure identifier */
  MQLONG   Version;     /* Structure version number */
  MQLONG   Options;     /* Options that control the action of
                           MQSETMP and MQINQMP */
  MQLONG   Support;     /* Property support option */
  MQLONG   Context;     /* Property context */
  MQLONG   CopyOptions; /* Property copy options */
};
```

COBOL declaration for MQPD:

```
**   MQPD structure
  10 MQPD.
**    Structure identifier
   15 MQPD-STRUCID PIC X(4).
**    Structure version number
   15 MQPD-VERSION PIC S9(9) BINARY.
**    Options that control the action of MQSETMP and
**    MQINQMP
   15 MQPD-OPTIONS PIC S9(9) BINARY.
**    Property support option
   15 MQPD-SUPPORT PIC S9(9) BINARY.
**    Property context
   15 MQPD-CONTEXT PIC S9(9) BINARY.
**    Property copy options
   15 MQPD-COPYOPTIONS PIC S9(9) BINARY.
```

PL/I declaration for MQPD:

```
dcl
 1 MQPD based,
  3 StrucId     char(4),       /* Structure identifier */
  3 Version     fixed bin(31), /* Structure version number */
  3 Options     fixed bin(31), /* Options that control the action
                                  of MQSETMP and MQINQMP */
  3 Support     fixed bin(31), /* Property support option */
  3 Context     fixed bin(31), /* Property context */
  3 CopyOptions fixed bin(31); /* Property copy options */
```

High Level Assembler declaration for MQPD:

```
MQPD             DSECT
MQPD_STRUCID     DS   CL4    Structure identifier
MQPD_VERSION     DS   F      Structure version number
MQPD_OPTIONS     DS   F      Options that control the
*                            action of MQSETMP and MQINQMP
MQPD_SUPPORT     DS   F      Property support option
MQPD_CONTEXT     DS   F      Property context
MQPD_COPYOPTIONS DS   F      Property copy options
MQPD_LENGTH      EQU  *-MQPD
MQPD_AREA        DS   CL(MQPD_LENGTH)
```

MQPMO - Put-message options:

The following table summarizes the fields in the structure.

Table 226. MQPMO structure

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options that control the action of MQPUT and MQPUT1	Options
Timeout	Reserved	Timeout
Context	Object handle of input queue	Context
KnownDestCount	Number of messages sent successfully to local queues	KnownDestCount
UnknownDestCount	Number of messages sent successfully to remote queues	UnknownDestCount
InvalidDestCount	Number of messages that could not be sent	InvalidDestCount
ResolvedQName	Resolved name of destination queue	ResolvedQName
ResolvedQMgrName	Resolved name of destination queue manager	ResolvedQMgrName
Note: The remaining fields are ignored if *Version* is less than MQPMO_VERSION_2.		
RecsPresent	Number of put message records or response records present	RecsPresent
PutMsgRecFields	Flags indicating which MQPMR fields are present	PutMsgRecFields
PutMsgRecOffset	Offset of first put-message record from start of MQPMO	PutMsgRecOffset
ResponseRecOffset	Offset of first response record from start of MQPMO	ResponseRecOffset
PutMsgRecPtr	Address of first put message record	PutMsgRecPtr
ResponseRecPtr	Address of first response record	ResponseRecPtr
Note: The remaining fields are ignored if *Version* is less than MQPMO_VERSION_3.		

Table 226. MQPMO structure (continued)

Field	Description	Topic
OriginalMsgHandle	Original message handle	OriginalMsgHandle
NewMsgHandle	New message handle	NewMsgHandle
Action	Type of put being performed and the relationship between the original message specified by the *OriginalMsgHandle* field and the new message specified by the *NewMsgHandle* field	Action
PubLevel	Level of subscription targeted by the publication	PubLevel

Overview for MQPMO:

Availability: All IBM MQ systems, plus IBM MQ clients connected to these systems.

Purpose: The MQPMO structure allows the application to specify options that control how messages are placed on queues, or published to topics. The structure is an input/output parameter on the MQPUT and MQPUT1 calls.

Version: The current version of MQPMO is MQPMO_VERSION_3. Certain fields are available only in certain versions of MQPMO. If you need to port applications between several environments, you must ensure that the version of MQPMO is consistent across all environments. Fields that exist only in particular versions of the structure are identified as such in "MQPMO - Put-message options" on page 2477 and in the field descriptions.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQPMO that is supported by the environment, but with the initial value of the *Version* field set to MQPMO_VERSION_1. To use fields that are not present in the version-1 structure, the application must set the *Version* field to the version number of the version required.

Character set and encoding: Data in MQPMO must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Fields for MQPMO:

The MQPMO structure contains the following fields; the fields are described in **alphabetical order**:

Action (MQLONG):

This specifies the type of put being performed and the relationship between the original message specified by the OriginalMsgHandle field and the new message specified by the NewMsgHandle field. The properties of the message are chosen by the queue manager according to the value of the Action specified.

You can choose to supply the contents of the message descriptor using the MsgDesc parameter on the MQPUT or MQPUT1 calls. Alternatively it is possible not to supply the MsgDesc parameter, or to specify that it is output-only by including MQPMO_MD_FOR_OUTPUT_ONLY in the Options field of the MQPMO structure.

If the MsgDesc parameter is not supplied, or if it is specified to be output-only, then the message descriptor for the new message is populated from the message handle fields of the MQPMO, according to the rules described in this topic.

The context setting and passing activities described in Controlling context information take effect after the message descriptor has been composed.

If an incorrect action value is specified, the call fails with the reason code MQRC_ACTION_ERROR.

Any one of the following actions can be specified:

MQACTP_NEW
A new message is being put, and no relationship to a previous message is being specified by the program. The message descriptor is composed as follows:

- If a MsgDesc is supplied on the MQPUT or MQPUT1 call, and MQPMO_MD_FOR_OUTPUT_ONLY is not in the MQPMO.Options, this is used as the message descriptor unmodified.
- If a MsgDesc is not supplied, or MQPMO_MD_FOR_OUTPUT_ONLY is in the MQPMO.Options then the queue manager generates the message descriptor using a combination of properties from OriginalMsgHandle and NewMsgHandle. Any message descriptor fields explicitly set on the new message handle take precedence over those in the original message handle.

Message data is taken from the MQPUT or MQPUT1 Buffer parameter.

MQACTP_FORWARD
A previously retrieved message is being forwarded. The original message handle specifies the message that was previously retrieved.

The new message handle specifies any modifications to the properties (including any in the message descriptor) in the original message handle.

The message descriptor is composed as follows:

- If a MsgDesc is supplied on the MQPUT or MQPUT1 call, and MQPMO_MD_FOR_OUTPUT_ONLY is not in the MQPMO.Options, this is used as the message descriptor unmodified.
- If a MsgDesc is not supplied, or MQPMO_MD_FOR_OUTPUT_ONLY is in the MQPMO.Options then the queue manager generates the message descriptor using a combination of properties from OriginalMsgHandle and NewMsgHandle. Any message descriptor fields explicitly set on the new message handle take precedence over those in the original message handle.
- If MQPMO_NEW_MSG_ID or MQPMO_NEW_CORREL_ID are specified in the MQPMO.Options, then these are honoured.

The message properties are composed as follows:

- All properties from the original message handle which have MQCOPY_FORWARD in the MQPD.CopyOptions
- All properties from the new message handle. For each property in the new message handle that has the same name as a property in the original message handle, the value is taken from the new message handle. The only exception to this rule is the special case when the property in the new message handle has the same name as a property in the original message handle, but the value of the property is null. In this case the property is removed from the message.

The message data to be forwarded is taken from the MQPUT or MQPUT1 Buffer parameter.

MQACTP_REPLY
A reply is being made to a previously retrieved message. The original message handle specifies the message that was previously retrieved.

The new message handle specifies any modifications to the properties (including any in the message descriptor) in the original message handle.

The message descriptor is composed as follows:

- If a MsgDesc is supplied on the MQPUT or MQPUT1 call, and MQPMO_MD_FOR_OUTPUT_ONLY is not in the MQPMO.Options, this is used as the message descriptor unmodified.
- If a MsgDesc is not supplied, or MQPMO_MD_FOR_OUTPUT_ONLY is in the MQPMO.Options then initial message descriptor fields are chosen as follows:

Table 227. Reply message handle transformation

Field in MQMD	**Value used**
Report	If MQRO_PASS_DISCARD_AND_EXPIRY and MQRO_DISCARD_MSG are set: MQRO_DISCARD_MSG otherwise MQRO_NONE
MsgType	MQMT_REPLY
Expiry	If MQRO_PASS_DISCARD_AND_EXPIRY is set: Copied from the input message otherwise MQEI_UNLIMITED
Feedback	MQFB_NONE
MsgId	If MQPMO_NEW_MSG_ID is set: A new message identifier is generated else if MQRO_PASS_MSG_ID is set: Copied from the input message otherwise MQMI_NONE
CorrelId	If MQPMO_NEW_CORREL_ID is set: A new correlation identifier is generated else if MQRO_COPY_MSG_ID_TO_CORREL_ID is set: Copied from the MsgId field of the input message else if MQRO_PASS_CORREL_ID is set: Copied from the CorrelId field of the input message otherwise MQCI_NONE
BackoutCount	0
ReplyToQ	Blanks
ReplyToQMgr	Blanks
GroupId	MQGI_NONE
MsgSeqNumber	1
Offset	0
MsgFlags	MQMF_NONE
OriginalLength	MQOL_UNDEFINED

- The message descriptor is then modified by the new message handle - any message descriptor fields explicitly set as properties in the new message handle take precedence over the message descriptor fields as described previously.

The message properties are composed as follows:

- All properties from the original message handle which have MQCOPY_REPLY in the MQPD.CopyOptions

- All properties from the new message handle. For each property in the new message handle that has the same name as a property in the original message handle, the value is taken from the new message handle. The only exception to this rule is the special case when the property in the new message handle has the same name as a property in the original message handle, but the value of the property is null. In this case the property is removed from the message.

The message data to be forwarded is taken from the MQPUT/MQPUT1 Buffer parameter.

MQACTP_REPORT

A report is being generated as a result of a previously retrieved message. The original message handle specifies the message causing the report to be generated.

The new message handle specifies any modifications to the properties (including any in the message descriptor) in the original message handle.

The message descriptor is composed as follows:

- If a MsgDesc is supplied on the MQPUT or MQPUT1 call, and MQPMO_MD_FOR_OUTPUT_ONLY is not in the MQPMO.Options, this is used as the message descriptor unmodified.
- If a MsgDesc is not supplied, or MQPMO_MD_FOR_OUTPUT_ONLY is in the MQPMO.Options then initial message descriptor fields are chosen as follows:

Table 228. Report message handle transformation

Field in MQMD	Value used
Report	If MQRO_PASS_DISCARD_AND_EXPIRY and MQRO_DISCARD_MSG are set: MQRO_DISCARD_MSG otherwise MQRO_NONE
MsgType	MQMT_REPORT
Expiry	If MQRO_PASS_DISCARD_AND_EXPIRY is set: Copied from the input message otherwise MQEI_UNLIMITED
MsgId	If MQPMO_NEW_MSG_ID is set: A new message identifier is generated else if MQRO_PASS_MSG_ID is set: Copied from the input message otherwise MQMI_NONE
CorrelId	If MQPMO_NEW_CORREL_ID is set: A new correlation identifier is generated else if MQRO_COPY_MSG_ID_TO_CORREL_ID is set: Copied from the MsgId field of the input message else if MQRO_PASS_CORREL_ID is set: Copied from the CorrelId field of the input message otherwise MQCI_NONE
BackoutCount	0
ReplyToQ	Blanks
ReplyToQMgr	Blanks
OriginalLength	Set to the *BufferLength*

- The message descriptor is then modified by the new message handle - any message descriptor fields explicitly set as properties in the new message handle take precedence over the message descriptor fields as described previously.

The message properties are composed as follows:

- All properties from the original message handle which have MQCOPY_REPORT in the MQPD.CopyOptions
- All properties from the new message handle. For each property in the new message handle that has the same name as a property in the original message handle, the value is taken from the new message handle. The only exception to this rule is the special case when the property in the new message handle has the same name as a property in the original message handle, but the value of the property is null. In this case the property is removed from the message.

The Feedback field in the resultant MQMD represents the report that is to be generated. A Feedback value of MQFB_NONE causes the MQPUT or MQPUT1 call to fail with reason code MQRC_FEEDBACK_ERROR.

To choose the user data of the report message, IBM MQ consults the Report and Feedback fields in the resultant MQMD, and the Buffer and BufferLength parameters of the MQPUT or MQPUT1 call.

- If Feedback is MQFB_COA, MQFB_COD or MQFB_EXPIRATION then the value of Report is inspected.
- If any of the following cases is true, the full message data from Buffer for a length of BufferLength is used.
  - Feedback is MQFB_EXPIRATION and Report contains MQRO_EXPIRATION_WITH_FULL_DATA
  - Feedback is MQFB_COD and Report contains MQRO_COD_WITH_FULL_DATA
  - Feedback is MQFB_COA and Report contains MQRO_COA_WITH_FULL_DATA
- If any of the following cases is true, the first 100 bytes of the message (or BufferLength if this is less than 100) from Buffer are used
  - Feedback is MQFB_EXPIRATION and Report contains MQRO_EXPIRATION_WITH_DATA
  - Feedback is MQFB_COD and Report contains MQRO_COD_WITH_DATA
  - Feedback is MQFB_COA and Report contains MQRO_COA_WITH_DATA
- If Feedback is MQFB_EXPIRATION, MQFB_COD or MQFB_COA, and Report does not contain the *_WITH_FULL_DATA or *_WITH_DATA options relevant to that Feeback value, then no user data is included with the message.
- If Feedback takes a different value from those listed above, then Buffer and BufferLength are used as normal.

The derivation of the user data is shown in the following table:

Context (MQHOBJ):

If MQPMO_PASS_IDENTITY_CONTEXT or MQPMO_PASS_ALL_CONTEXT is specified, this field must contain the input queue handle from which context information to be associated with the message being put is taken.

If neither MQPMO_PASS_IDENTITY_CONTEXT nor MQPMO_PASS_ALL_CONTEXT is specified, this field is ignored.

This is an input field. The initial value of this field is `0`.

InvalidDestCount (MQLONG):

This is the number of messages that could not be sent to queues in the distribution list. The count includes queues that failed to open, as well as queues that were opened successfully but for which the put operation failed. This field is also set when putting a message to a single queue that is not in a distribution list.

Note: This field is set if the **CompCode** parameter on the MQPUT or MQPUT1 call is MQCC_OK or MQCC_WARNING; it might be set if the **CompCode** parameter is MQCC_FAILED, but do not rely on this in application code.

This is an output field. The initial value of this field is 0. This field is not set if *Version* is less than MQPMO_VERSION_1.

This field is undefined on z/OS because distribution lists are not supported.

KnownDestCount (MQLONG):

This is the number of messages that the current MQPUT or MQPUT1 call has sent successfully to queues in the distribution list that are local queues. The count does not include messages sent to queues that resolve to remote queues (even though a local transmission queue is used initially to store the message). This field is also set when putting a message to a single queue that is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is not set if *Version* is less than MQPMO_VERSION_1.

This field is undefined on z/OS because distribution lists are not supported.

NewMsgHandle (MQHMSG):

This is an optional handle to the message being put subject to the value of the Action field. It defines the properties of the message and overrides the values of the *OriginalMsgHandle*, if specified.

On return from the **MQPUT** or **MQPUT1** call, the contents of the handle reflect the message that was actually put.

This is an input field. The initial value of this field is **MQHM_NONE**. This field is ignored if Version is less than **MQPMO_VERSION_3**.

MQPMO options (MQLONG):

The Options field controls the operation of **MQPUT** and **MQPUT1** calls.

Scope option. You can specify any or none of the MQPMO options. To specify more than one option, either add the values together (do not add the same constant more than once), or combine the values using the bitwise OR operation (if the programming language supports bit operations). Combinations that are not valid are noted; any other combinations are valid.

The following option controls the scope of the publications sent:

MQPMO_SCOPE_QMGR

The publication is sent only to subscribers that have subscribed on this queue manager. The publication is not forwarded to any remote publish/subscribe queue managers that have made a subscription to this queue manager, which overrides any behavior that has been set using the PUBSCOPE topic attribute.

Note: If not set, the publication scope is determined by the PUBSCOPE topic attribute.

Publishing options. The following options control the way messages are published to a topic:

MQPMO_SUPPRESS_REPLYTO

Any information specified in the *ReplyToQ* and *ReplyToQMgr* fields of the MQMD of this publication is not passed on to subscribers. If this option is used with a report option that requires a *ReplyToQ*, the call fails with MQRC_MISSING_REPLY_TO_Q.

MQPMO_RETAIN

The publication being sent is to be retained by the queue manager. This retention allows a subscriber to request a copy of this publication after the time it was published, by using the MQSUBRQ call. It also allows a publication to be sent to applications which make their subscription after the time this publication was made (unless they choose not to be sent it by using the option MQSO_NEW_PUBLICATIONS_ONLY). If an application is sent a publication which was retained, it is indicated by the MQIsRetained message property of that publication.

Only one publication can be retained at each node of the topic tree. Therefore, if there already is a retained publication for this topic, published by any other application, it is replaced with this publication. It is therefore better to avoid having more than one publisher retaining messages on the same topic.

When retained publications are requested by a subscriber, the subscription used might contain a wildcard in the topic, in which case a number of retained publications might match (at various nodes in the topic tree) and several publications might be sent to the requesting application. See the description of the "MQSUBRQ - Subscription request" on page 2789 call for more details.

For information about how retained publications interact with subscription levels, see Intercepting publications.

If this option is used and the publication cannot be retained, the message is not published and the call fails with MQRC_PUT_NOT_RETAINED.

MQPMO_NOT_OWN_SUBS

Tells the queue manager that the application does not want to send any of its publications to subscriptions it owns. Subscriptions are considered to be owned by the same application if the connection handles are the same.

MQPMO_WARN_IF_NO_SUBS_MATCHED

If no subscription matches the publication, return a completion code (*CompCode*) of MQCC_WARNING and reason code MQRC_NO_SUBS_MATCHED.

If MQRC_NO_SUBS_MATCHED is returned by the put operation, the publication was not delivered to any subscriptions. However, if the MQPMO_RETAIN option is specified on the put operation, the message is retained and delivered to any subsequently defined matching subscription.

A subscription on the topic matches the publication if any of the following conditions are met:

- The message is delivered to the subscription queue
- The message would have been delivered to the subscription queue but a problem with the queue means that the message cannot be put to the queue, and it was consequently placed on the dead letter queue or discarded.
- A routing exit is defined that suppresses delivery of the message to the subscription

A subscription on the topic does not match the publication if any of the following conditions are met:

- The subscription has a selection string that does not match the publication
- The subscription specified the MQSO_PUBLICATION_ON_REQUEST option
- The publication is not delivered because the MQPMO_NOT_OWN_SUBS option was specified on the put operation and the subscription matches the identity of the publisher

Syncpoint options. The following options relate to the participation of the MQPUT or MQPUT1 call within a unit of work:

MQPMO_SYNCPOINT
The request is to operate within the normal unit-of-work protocols. The message is not visible outside the unit of work until the unit of work is committed. If the unit of work is backed out, the message is deleted.

If MQPMO_SYNCPOINT and MQPMO_NO_SYNCPOINT are not specified, the inclusion of the put request in unit-of-work protocols is determined by the environment running the queue manager and not the environment running the application. On z/OS, the put request is within a unit of work. In all other environments, the put request is not within a unit of work.

Because of these differences, an application that you want to port must not allow this option to default; specify either MQPMO_SYNCPOINT or MQPMO_NO_SYNCPOINT explicitly.

Do not specify MQPMO_SYNCPOINT with MQPMO_NO_SYNCPOINT.

MQPMO_NO_SYNCPOINT
The request is to operate outside the normal unit-of-work protocols. The message is available immediately, and it cannot be deleted by backing out a unit of work.

If MQPMO_NO_SYNCPOINT and MQPMO_SYNCPOINT are not specified, the inclusion of the put request in unit-of-work protocols is determined by the environment running the queue manager and not the environment running the application. On z/OS, the put request is within a unit of work. In all other environments, the put request is not within a unit of work.

Because of these differences, an application that you want to port must not allow this option to default; specify either MQPMO_SYNCPOINT or MQPMO_NO_SYNCPOINT explicitly.

Do not specify MQPMO_NO_SYNCPOINT with MQPMO_SYNCPOINT.

Message-identifier and correlation-identifier options. The following options request the queue manager to generate a new message identifier or correlation identifier:

MQPMO_NEW_MSG_ID
The queue manager replaces the contents of the *MsgId* field in MQMD with a new message identifier. This message identifier is sent with the message, and returned to the application on output from the MQPUT or MQPUT1 call.

The MQPMO_NEW_MSG_ID option can also be specified when the message is being put to a distribution list; see the description of the *MsgId* field in the MQPMR structure for details.

Using this option relieves the application of the need to reset the *MsgId* field to MQMI_NONE before each MQPUT or MQPUT1 call.

MQPMO_NEW_CORREL_ID
The queue manager replaces the contents of the *CorrelId* field in MQMD with a new correlation identifier. This correlation identifier is sent with the message, and returned to the application on output from the MQPUT or MQPUT1 call.

The MQPMO_NEW_CORREL_ID option can also be specified when the message is being put to a distribution list; see the description of the *CorrelId* field in the MQPMR structure for details.

MQPMO_NEW_CORREL_ID is useful in situations where the application requires a unique correlation identifier.

Group and segment options. The following options relate to the processing of messages in groups and segments of logical messages. Read the definitions that follow to help you to understand the option.

Physical message
The is the smallest unit of information that can be placed on or removed from a queue; it often corresponds to the information specified or retrieved on a single MQPUT, MQPUT1, or MQGET call. Every physical message has its own message descriptor (MQMD). Generally, physical messages are distinguished by differing values for the message identifier (*MsgId* field in MQMD), although this is not enforced by the queue manager.

Logical message
A logical message is a single unit of application information for non- z/OS platforms only. In the absence of system constraints, a logical message is the same as a physical message. But where logical messages are extremely large, system constraints might make it advisable or necessary to split a logical message into two or more physical messages, called *segments*.

A logical message that has been segmented consists of two or more physical messages that have the same non-null group identifier (*GroupId* field in MQMD), and the same message sequence number (*MsgSeqNumber* field in MQMD). The segments are distinguished by differing values for the segment offset (*Offset* field in MQMD), which gives the offset of the data in the physical message from the start of the data in the logical message. Because each segment is a physical message, the segments in a logical message usually have differing message identifiers.

A logical message that has not been segmented, but for which segmentation has been permitted by the sending application, also has a non-null group identifier, although in this case there is only one physical message with that group identifier if the logical message does not belong to a message group. Logical messages for which segmentation has been inhibited by the sending application have a null group identifier (MQGI_NONE), unless the logical message belongs to a message group.

Message group
A message group is a set of one or more logical messages that have the same non-null group identifier. The logical messages in the group are distinguished by differing values for the message sequence number, which is an integer in the range 1 through *n*, where *n* is the number of logical messages in the group. If one or more of the logical messages is segmented, there are more than *n* physical messages in the group.

MQPMO_LOGICAL_ORDER
This option tells the queue manager how the application puts messages in groups and segments of logical messages. It can be specified only on the MQPUT call; it is not valid on the MQPUT1 call.

If MQPMO_LOGICAL_ORDER is specified, it indicates that the application uses successive MQPUT calls to:

1. Put the segments in each logical message in the order of increasing segment offset, starting from 0, with no gaps.
2. Put all the segments in one logical message before putting the segments in the next logical message.
3. Put the logical messages in each message group in the order of increasing message sequence number, starting from 1, with no gaps. IBM MQ increments the message sequence number automatically.
4. Put all the logical messages in one message group before putting logical messages in the next message group.

For detailed information about MQPMO_LOGICAL_ORDER, see Logical and physical ordering

Context options. The following options control the processing of message context:

MQPMO_NO_CONTEXT
Both identity and origin context are set to indicate no context. This means that the context fields in MQMD are set to:

- Blanks for character fields
- Nulls for byte fields
- Zeros for numeric fields

MQPMO_DEFAULT_CONTEXT
The message is to have default context information associated with it, for both identity and origin. The queue manager sets the context fields in the message descriptor as follows:

Field in MQMD	Value used
UserIdentifier	Determined from the environment if possible; set to blanks otherwise.
AccountingToken	Determined from the environment if possible; set to MQACT_NONE otherwise.
ApplIdentityData	Set to blanks.
PutApplType	Determined from the environment.
PutApplName	Determined from the environment if possible; set to blanks otherwise.
PutDate	Set to the date when message is put.
PutTime	Set to the time when message is put.
ApplOriginData	Set to blanks.

For more information about message context, see Message context.

These are the default values and actions if no context options are specified.

MQPMO_PASS_IDENTITY_CONTEXT
The message is to have context information associated with it. Identity context is taken from the queue handle specified in the *Context* field. Origin context information is generated by the queue manager in the same way that it is for MQPMO_DEFAULT_CONTEXT (see the preceding table for values). For more information about message context, see Message context.

For the MQPUT call, the queue must have been opened with the MQOO_PASS_IDENTITY_CONTEXT option (or an option that implies it). For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the MQOO_PASS_IDENTITY_CONTEXT option.

MQPMO_PASS_ALL_CONTEXT
The message is to have context information associated with it. Context is taken from the queue handle specified in the *Context* field. For more information about message context, see Controlling context information.

For the MQPUT call, the queue must have been opened with the MQOO_PASS_ALL_CONTEXT option (or an option that implies it). For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the MQOO_PASS_ALL_CONTEXT option.

MQPMO_SET_IDENTITY_CONTEXT
The message is to have context information associated with it. The application specifies the identity context in the MQMD structure. Origin context information is generated by the queue manager in the same way that it is for MQPMO_DEFAULT_CONTEXT (see the preceding table for values). For more information about message context, see Message context.

For the MQPUT call, the queue must have been opened with the MQOO_SET_IDENTITY_CONTEXT option (or an option that implies it). For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the MQOO_SET_IDENTITY_CONTEXT option.

MQPMO_SET_ALL_CONTEXT
The message is to have context information associated with it. The application specifies the identity, origin, and user context in the MQMD structure. For more information about message context, see Message context.

For the MQPUT call, the queue must have been opened with the MQOO_SET_ALL_CONTEXT option. For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the MQOO_SET_ALL_CONTEXT option.

You can specify only one of the MQPMO_*_CONTEXT context options. If you specify none, MQPMO_DEFAULT_CONTEXT is assumed.

Property options. The following option relates to the properties of the message:

MQPMO_MD_FOR_OUTPUT_ONLY
The message descriptor parameter must only be used for output to return the message descriptor of the message that was put. The message descriptor fields associated with the *NewMsgHandle*, *OriginalMsgHandle*, or both fields, of the **MQPMO** structure must be used for input.

If a valid message handle is not provided then the call fails with reason code **MQRC_MD_ERROR**.

Put response options. The following options control the response returned to an MQPUT or MQPUT1 call. You can specify only one of these options. If MQPMO_ASYNC_RESPONSE and MQPMO_SYNC_RESPONSE are not specified, MQPMO_RESPONSE_AS_Q_DEF or MQPMO_RESPONSE_AS_TOPIC_DEF is assumed.

MQPMO_ASYNC_RESPONSE
The MQPMO_ASYNC_RESPONSE option requests that an MQPUT or MQPUT1 operation is completed without the application waiting for the queue manager to complete the call. Using this option can improve messaging performance, particularly for applications using client bindings. An application can periodically check, using the MQSTAT verb, whether an error has occurred during any previous asynchronous calls.

With this option, only the following fields are guaranteed to be completed in the MQMD;

- ApplIdentityData
- PutApplType
- PutApplName
- ApplOriginData

Additionally, if either or both of MQPMO_NEW_MSG_ID or MQPMO_NEW_CORREL_ID are specified as options, the MsgId and CorrelId returned are also completed. (MQPMO_NEW_MSG_ID can be implicitly specified by specifying a blank MsgId field).

Only the preceding specified fields are completed. Other information that would normally be returned in the MQMD or MQPMO structure is undefined.

When requesting asynchronous put response for MQPUT1, the ResolvedQName and ResolvedQMgrName returned in the MQOD structure are undefined.

When requesting asynchronous put response for MQPUT or MQPUT1, a CompCode and Reason of MQCC_OK and MQRC_NONE does not necessarily mean that the message was successfully put to a queue. When developing an MQI application that uses asynchronous put response and requires confirmation that messages have been put to a queue you must check both CompCode and Reason codes from the put operations and also use MQSTAT to query asynchronous error information.

Although the success or failure of each individual MQPUT or MQPUT1 call mightnot be returned immediately, the first error that occurred under an asynchronous call can be determined later through a call to MQSTAT.

If a persistent message under syncpoint fails to be delivered using asynchronous put response, and you attempt to commit the transaction, the commit fails and the transaction is backed out with a completion code of MQCC_FAILED and a reason of MQRC_BACKED_OUT. The application can make a call to MQSTAT to determine the cause of a previous MQPUT or MQPUT1 failure.

MQPMO_SYNC_RESPONSE
Specifying this put response type ensures that the MQPUT or MQPUT1 operation is always issued synchronously. If the put operation is successful, all fields in the MQMD and MQPMO are completed.

This option ensures a synchronous response irrespective of the default put response value defined on the queue or topic object.

MQPMO_RESPONSE_AS_Q_DEF
If this value is specified for an MQPUT call, the put response type used is taken from the DEFPRESP value specified on the queue when it was first opened by the application. If a client application is connected to a queue manager at a level earlier than Version 7.0, it behaves as if MQPMO_SYNC_RESPONSE was specified.

If this option is specified for an MQPUT1 call, the value of the DEFPRESP attribute is not known before the request is sent to the server. By default, if the MQPUT1 call is using MQPMO_SYNCPOINT it behaves as for MQPMO_ASYNC_RESPONSE, and if it is using MQPMO_NO_SYNCPOINT it behaves as for MQPMO_SYNC_RESPONSE. However, you can override this default behavior by setting the Put1DefaultAlwaysSync property in the client configuration file, see CHANNELS stanza of the client configuration file.

MQPMO_RESPONSE_AS_TOPIC_DEF
MQPMO_RESPONSE_AS_TOPIC_DEF is a synonym for MQPMO_RESPONSE_AS_Q_DEF for use with topic objects.

Other options. The following options control authorization checking, what happens when the queue manager is quiescing, and resolving queue and queue manager names:

MQPMO_ALTERNATE_USER_AUTHORITY
MQPMO_ALTERNATE_USER_AUTHORITY indicates that the *AlternateUserId* field in the **ObjDesc** parameter of the MQPUT1 call contains a user identifier that is to be used to validate authority to put messages on the queue. The call can succeed only if *AlternateUserId* is authorized to open the queue with the specified options, regardless of whether the user identifier under which the application is running is authorized to do so. (This does not apply to the context options specified, however, which are always checked against the user identifier under which the application is running.)

This option is valid only with the MQPUT1 call.

MQPMO_FAIL_IF_QUIESCING
This option forces the MQPUT or MQPUT1 call to fail if the queue manager is in the quiescing state.

On z/OS, this option also forces the MQPUT or MQPUT1 call to fail if the connection (for a CICS or IMS application) is in the quiescing state.

The call returns completion code MQCC_FAILED with reason code MQRC_Q_MGR_QUIESCING or MQRC_CONNECTION_QUIESCING.

MQPMO_RESOLVE_LOCAL_Q
Use this option to fill *ResolvedQName* in the MQPMO structure with the name of the local queue to which the message is put, and *ResolvedQMgrName* with the name of the local queue manager that hosts the local queue. For more information about MQPMO_RESOLVE_LOCAL_Q, see topic MQOO_RESOLVE_LOCAL_Q.

If you are authorized to put to a queue, you have the required authority to specify this flag on the MQPUT call; no special authority is needed.

Default option. If you need none of the options described, use the following option:

MQPMO_NONE
Use this value to indicate that no other options have been specified; all options assume their default values. MQPMO_NONE is defined to aid program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

MQPMO_NONE is an input field. The initial value of the *Options* field is MQPMO_NONE.

OriginalMsgHandle (MQHMSG):

This is an optional handle to a message. It might have been previously retrieved from a queue. The use of this handle is subject to the value of the *Action* field; see also NewMsgHandle.

The contents of the original message handle will not be altered by the **MQPUT** or **MQPUT1** call.

This is an input field. The initial value of this field is **MQHM_NONE**. This field is ignored if Version is less than **MQPMO_VERSION_3**.

PubLevel (MQLONG):

The initial value of this field is 9. The level of subscription targeted by this publication. Only those subscriptions with the highest SubLevel less than or equal to this value receive this publication. This value must be in the range zero to 9; zero is the lowest level. However, if a publication has been retained, it is no longer available to subscribers at higher levels because it is republished at PubLevel 1.

For information, see Intercepting publications.

PutMsgRecFields (MQLONG):

This field contains flags that indicate which MQPMR fields are present in the put message records provided by the application. Use *PutMsgRecFields* only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero, or both *PutMsgRecOffset* and *PutMsgRecPtr* are zero.

For fields that are present, the queue manager uses for each destination the values from the fields in the corresponding put message record. For fields that are absent, the queue manager uses the values from the MQMD structure.

Use one or more of the following flags to indicate which fields are present in the put message records:

MQPMRF_MSG_ID
Message-identifier field is present.

MQPMRF_CORREL_ID
Correlation-identifier field is present.

MQPMRF_GROUP_ID
Group-identifier field is present.

MQPMRF_FEEDBACK
Feedback field is present.

MQPMRF_ACCOUNTING_TOKEN
Accounting-token field is present.

If you specify this flag, specify either MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT in the *Options* field; if this condition is not satisfied, the call fails with reason code MQRC_PMO_RECORD_FLAGS_ERROR.

If no MQPMR fields are present, the following can be specified:

MQPMRF_NONE
No put-message record fields are present.

If this value is specified, either *RecsPresent* must be zero, or both *PutMsgRecOffset* and *PutMsgRecPtr* must be zero.

MQPMRF_NONE is defined to aid program documentation. It is not intended that this constant be used with any other, but as its value is zero, such use cannot be detected.

If *PutMsgRecFields* contains flags that are not valid, or put message records are provided but *PutMsgRecFields* has the value MQPMRF_NONE, the call fails with reason code MQRC_PMO_RECORD_FLAGS_ERROR.

This is an input field. The initial value of this field is MQPMRF_NONE. This field is ignored if *Version* is less than MQPMO_VERSION_2.

PutMsgRecOffset (MQLONG):

This is the offset in bytes of the first MQPMR put message record from the start of the MQPMO structure. The offset can be positive or negative. *PutMsgRecOffset* is used only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero.

When the message is being put to a distribution list, an array of one or more MQPMR put message records can be provided in order to specify certain properties of the message for each destination individually; these properties are:

- Message identifier
- Correlation identifier
- Group identifier
- Feedback value
- Accounting token

You do not need to specify all these properties, but whatever subset you choose, specify the fields in the correct order. See the description of the MQPMR structure for further details.

Usually, there must be as many put message records as there are object records specified by MQOD when the distribution list is opened; each put message record supplies the message properties for the queue identified by the corresponding object record. Queues in the distribution list that fail to open must still have put message records allocated for them at the appropriate positions in the array, although the message properties are ignored in this case.

The number of put message records can differ from the number of object records. If there are fewer put message records than object records, the message properties for the destinations that do not have put message records are taken from the corresponding fields in the message descriptor MQMD. If there are more put message records than object records, the excess are not used (although it must still be possible to access them). Put message records are optional, but if they are supplied there must be *RecsPresent* of them.

Provide the put message records in a similar way to the object records in MQOD, either by specifying an offset in *PutMsgRecOffset*, or by specifying an address in *PutMsgRecPtr* ; for details of how to do this, see the *ObjectRecOffset* field described in"MQOD - Object descriptor" on page 2453.

No more than one of *PutMsgRecOffset* and *PutMsgRecPtr* can be used; the call fails with reason code MQRC_PUT_MSG_RECORDS_ERROR if both are nonzero.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQPMO_VERSION_2.

PutMsgRecPtr (MQPTR):

This is the address of the first MQPMR put message record. Use *PutMsgRecPtr* only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero.

You can use either *PutMsgRecPtr* or *PutMsgRecOffset* can be used to specify the put message records, but not both; for details, see "PutMsgRecOffset (MQLONG)" on page 2491. If you do not use *PutMsgRecPtr*, set it to the null pointer or null bytes.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQPMO_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

RecsPresent (MQLONG):

This is the number of MQPMR put message records or MQRR response records that have been provided by the application. This number can be greater than zero only if the message is being put to a distribution list. Put message records and response records are optional; the application need not provide any records, or it can choose to provide records of only one type. However, if the application provides records of both types, it must provide *RecsPresent* records of each type.

The value of *RecsPresent* need not be the same as the number of destinations in the distribution list. If too many records are provided, the excess are not used; if too few records are provided, default values are used for the message properties for those destinations that do not have put message records (see *PutMsgRecOffset*).

If *RecsPresent* is less than zero, or is greater than zero but the message is not being put to a distribution list, the call fails with reason code MQRC_RECS_PRESENT_ERROR.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQPMO_VERSION_2.

ResolvedQMgrName (MQCHAR48):

This is the name of the destination queue manager after name resolution has been performed by the local queue manager. The name returned is the name of the queue manager that owns the queue identified by *ResolvedQName*, and can be the name of the local queue manager.

If *ResolvedQName* is a shared queue that is owned by the queue-sharing group to which the local queue manager belongs, *ResolvedQMgrName* is the name of the queue-sharing group. If the queue is owned by some other queue-sharing group, *ResolvedQName* can be the name of the queue-sharing group or the name of a queue manager that is a member of the queue-sharing group (the nature of the value returned is determined by the queue definitions that exist at the local queue manager).

A nonblank value is returned only if the object is a single queue; if the object is a distribution list or a topic, the value returned is undefined.

This is an output field. The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ResolvedQName (MQCHAR48):

This is the name of the destination queue after name resolution has been performed by the local queue manager. The name returned is the name of a queue that exists on the queue manager identified by *ResolvedQMgrName*.

A nonblank value is returned only if the object is a single queue; if the object is a distribution list or a topic, the value returned is undefined.

This is an output field. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ResponseRecOffset (MQLONG):

This is the offset in bytes of the first MQRR response record from the start of the MQPMO structure. The offset can be positive or negative. *ResponseRecOffset* is used only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero.

When putting the message to a distribution list, you can provide an array of one or more MQRR response records to identify the queues to which the message was not sent successfully (*CompCode* field in MQRR), and the reason for each failure (*Reason* field in MQRR). The message might not have been sent either because the queue failed to open, or because the put operation failed. The queue manager sets the response records only when the outcome of the call is mixed (that is, some messages were sent successfully while others failed, or all failed but for differing reasons); reason code MQRC_MULTIPLE_REASONS from the call indicates this case. If the same reason code applies to all queues, that reason is returned in the **Reason** parameter of the MQPUT or MQPUT1 call, and the response records are not set.

Usually, there are as many response records as there are object records specified by MQOD when the distribution list is opened; when necessary, each response record is set to the completion code and reason code for the put to the queue identified by the corresponding object record. Queues in the distribution list that fail to open must still have response records allocated for them at the appropriate positions in the array, although they are set to the completion code and reason code resulting from the open operation, rather than the put operation.

The number of response records can differ from the number of object records. If there are fewer response records than object records, the application might not be able to identify all the destinations for which the put operation failed, or the reasons for the failures. If there are more response records than object records,

the excess are not used (although it must still be possible to access them). Response records are optional, but if they are supplied there must be *RecsPresent* of them.

Provide the response records in a similar way to the object records in MQOD, either by specifying an offset in *ResponseRecOffset*, or by specifying an address in *ResponseRecPtr* ; for details of how to do this, see the *ObjectRecOffset* field described in "MQOD - Object descriptor" on page 2453. However, use no more than one of *ResponseRecOffset* and *ResponseRecPtr* ; the call fails with reason code MQRC_RESPONSE_RECORDS_ERROR if both are nonzero.

For the MQPUT1 call, this field must be zero. This is because the response information (if requested) is returned in the response records specified by the object descriptor MQOD.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQPMO_VERSION_2.

ResponseRecPtr (MQPTR):

This is the address of the first MQRR response record. *ResponseRecPtr* is used only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero.

Use either *ResponseRecPtr* or *ResponseRecOffset* to specify the response records, but not both; for details, see "ResponseRecOffset (MQLONG)" on page 2493. If you do not use *ResponseRecPtr* set it to the null pointer or null bytes.

For the MQPUT1 call, this field must be the null pointer or null bytes. This is because the response information (if requested) is returned in the response records specified by the object descriptor MQOD.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQPMO_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQPMO_STRUC_ID
Identifier for put-message options structure.

For the C programming language, the constant MQPMO_STRUC_ID_ARRAY is also defined; this has the same value as MQPMO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQPMO_STRUC_ID.

Timeout (MQLONG):

This is a reserved field; its value is not significant. The initial value of this field is -1.

UnknownDestCount (MQLONG):

This is the number of messages that the current MQPUT or MQPUT1 call has sent successfully to queues in the distribution list that resolve to remote queues. Messages that the queue manager retains temporarily in distribution-list form count as the number of individual destinations that those distribution lists contain. This field is also set when putting a message to a single queue that is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is not set if *Version* is less than MQPMO_VERSION_1.

This field is undefined on z/OS because distribution lists are not supported.

Version (MQLONG):

Structure version number.

The value must be one of the following:

MQPMO_VERSION_1
Version-1 put-message options structure.

This version is supported in all environments.

MQPMO_VERSION_2
Version-2 put-message options structure.

This version is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ MQI clients connected to these systems.

MQPMO_VERSION_3
Version-3 put-message options structure.

This version is supported in all environments.

Fields that exist only in the more-recent version of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQPMO_CURRENT_VERSION
Current version of put-message options structure.

This is always an input field. The initial value of this field is MQPMO_VERSION_1.

Initial values and language declarations for MQPMO:

Table 229. Initial values of fields in MQPMO

Field name	Name of constant	Value of constant
StrucId	MQPMO_STRUC_ID	'PMO¬'
Version	MQPMO_VERSION_1	1
Options	MQPMO_NONE	0
Timeout	None	-1
Context	None	0
KnownDestCount	None	0
UnknownDestCount	None	0
InvalidDestCount	None	0
ResolvedQName	None	Null string or blanks
ResolvedQMgrName	None	Null string or blanks
RecsPresent	None	0
PutMsgRecFields	MQPMRF_NONE	0
PutMsgRecOffset	None	0
ResponseRecOffset	None	0
PutMsgRecPtr	None	Null pointer or null bytes
ResponseRecPtr	None	Null pointer or null bytes
OriginalMsgHandle	MQHM_NONE	0
NewMsgHandle	MQHM_NONE	0
Action	MQACTP_NEW	0
PubLevel	None	9

Notes:

1. The symbol ¬ represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQPMO_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure:

```
MQPMO MyPMO = {MQPMO_DEFAULT};
```

C declaration for MQPMO:

```
typedef struct tagMQPMO MQPMO;
struct tagMQPMO {
  MQCHAR4   StrucId;            /* Structure identifier */
  MQLONG    Version;            /* Structure version number */
  MQLONG    Options;            /* Options that control the action of
                                   MQPUT and MQPUT1 */
  MQLONG    Timeout;            /* Reserved */
  MQHOBJ    Context;            /* Object handle of input queue */
  MQLONG    KnownDestCount;     /* Number of messages sent
                                   successfully to local queues */
  MQLONG    UnknownDestCount;   /* Number of messages sent
                                   successfully to remote queues */
  MQLONG    InvalidDestCount;   /* Number of messages that could not
                                   be sent */
  MQCHAR48  ResolvedQName;      /* Resolved name of destination
                                   queue */
  MQCHAR48  ResolvedQMgrName;   /* Resolved name of destination queue
                                   manager */
  /* Ver:1 */
  MQLONG    RecsPresent;        /* Number of put message records or
                                   response records present */
  MQLONG    PutMsgRecFields;    /* Flags indicating which MQPMR fields
                                   are present */
  MQLONG    PutMsgRecOffset;    /* Offset of first put message record
                                   from start of MQPMO */
  MQLONG    ResponseRecOffset;  /* Offset of first response record
                                   from start of MQPMO */
  MQPTR     PutMsgRecPtr;       /* Address of first put message
                                   record */
  MQPTR     ResponseRecPtr;     /* Address of first response record */
  /* Ver:2 */
  MQHMSG    OriginalMsgHandle;  /* Original message handle */
  MQHMSG    NewMsgHandle;       /* New message handle */
  MQLONG    Action;             /* The action being performed */
  MQLONG    PubLevel;           /* Subscription level */
  /* Ver:3 */
};
```

COBOL declaration for MQPMO:

```
**   MQPMO structure
  10 MQPMO.
**    Structure identifier
   15 MQPMO-STRUCID           PIC X(4).
**    Structure version number
   15 MQPMO-VERSION           PIC S9(9) BINARY.
**    Options that control the action of MQPUT and MQPUT1
   15 MQPMO-OPTIONS           PIC S9(9) BINARY.
**    Reserved
   15 MQPMO-TIMEOUT           PIC S9(9) BINARY.
**    Object handle of input queue
   15 MQPMO-CONTEXT           PIC S9(9) BINARY.
**    Number of messages sent successfully to local queues
   15 MQPMO-KNOWNDESTCOUNT    PIC S9(9) BINARY.
**    Number of messages sent successfully to remote queues
   15 MQPMO-UNKNOWNDESTCOUNT  PIC S9(9) BINARY.
**    Number of messages that could not be sent
   15 MQPMO-INVALIDDESTCOUNT  PIC S9(9) BINARY.
**    Resolved name of destination queue
   15 MQPMO-RESOLVEDQNAME     PIC X(48).
**    Resolved name of destination queue manager
   15 MQPMO-RESOLVEDQMGRNAME  PIC X(48).
**    Number of put message records or response records present
   15 MQPMO-RECSPRESENT       PIC S9(9) BINARY.
**    Flags indicating which MQPMR fields are present
   15 MQPMO-PUTMSGRECFIELDS   PIC S9(9) BINARY.
```

```
**    Offset of first put message record from start of MQPMO
   15 MQPMO-PUTMSGRECOFFSET   PIC S9(9) BINARY.
**    Offset of first response record from start of MQPMO
   15 MQPMO-RESPONSERECOFFSET PIC S9(9) BINARY.
**    Address of first put message record
   15 MQPMO-PUTMSGRECPTR      POINTER.
**    Address of first response record
   15 MQPMO-RESPONSERECPTR    POINTER.
**    Original message handle
   15 MQPMO-ORIGINALMSGHANDLE PIC S9(18) BINARY.
**    New message handle
   15 MQPMO-NEWMSGHANDLE      PIC S9(18) BINARY.
**    The action being performed
   15 MQPMO-ACTION            PIC S9(9) BINARY.
**    Publish level
   15 MQPMO-PUBLEVEL          PIC S9(9) BINARY.
```

PL/I declaration for MQPMO:

```
dcl
 1 MQPMO based,
  3 StrucId           char(4),       /* Structure identifier */
  3 Version           fixed bin(31), /* Structure version number */
  3 Options           fixed bin(31), /* Options that control the action
                                        of MQPUT and MQPUT1 */
  3 Timeout           fixed bin(31), /* Reserved */
  3 Context           fixed bin(31), /* Object handle of input queue */
  3 KnownDestCount    fixed bin(31), /* Number of messages sent
                                        successfully to local queues */
  3 UnknownDestCount  fixed bin(31), /* Number of messages sent
                                        successfully to remote queues */
  3 InvalidDestCount  fixed bin(31), /* Number of messages that could
                                        not be sent */
  3 ResolvedQName     char(48),      /* Resolved name of destination
                                        queue */
  3 ResolvedQMgrName  char(48),      /* Resolved name of destination
                                        queue manager */
  3 RecsPresent       fixed bin(31), /* Number of put message records or
                                        response records present */
  3 PutMsgRecFields   fixed bin(31), /* Flags indicating which MQPMR
                                        fields are present */
  3 PutMsgRecOffset   fixed bin(31), /* Offset of first put message
                                        record from start of MQPMO */
  3 ResponseRecOffset fixed bin(31), /* Offset of first response record
                                        from start of MQPMO */
  3 PutMsgRecPtr      pointer,       /* Address of first put message
                                        record */
  3 ResponseRecPtr    pointer,       /* Address of first response
                                        record */
  3 OriginalMsgHandle fixed bin(63), /* Original message handle */
  3 NewMsgHandle      fixed bin(63); /* New message handle */
  3 Action            fixed bin(31); /* The action being performed */
  3 PubLevel          fixed bin(31); /* Publish level */
```

High Level Assembler declaration for MQPMO:

```
MQPMO                   DSECT
MQPMO_STRUCID           DS   CL4   Structure identifier
MQPMO_VERSION           DS   F     Structure version number
MQPMO_OPTIONS           DS   F     Options that control the action of
*                                  MQPUT and MQPUT1
MQPMO_TIMEOUT           DS   F     Reserved
MQPMO_CONTEXT           DS   F     Object handle of input queue
MQPMO_KNOWNDESTCOUNT    DS   F     Number of messages sent successfully
*                                  to local queues
MQPMO_UNKNOWNDESTCOUNT  DS   F     Number of messages sent successfully
*                                  to remote queues
MQPMO_INVALIDDESTCOUNT  DS   F     Number of messages that could not be
*                                  sent
MQPMO_RESOLVEDQNAME     DS   CL48  Resolved name of destination queue
MQPMO_RESOLVEDQMGRNAME  DS   CL48  Resolved name of destination queue
*                                  manager
MQPMO_RECSPRESENT       DS   F     Number of put message records or
*                                  response records present
MQPMO_PUTMSGRECFIELDS   DS   F     Flags indicating which MQPMR
*                                  fields are present
MQPMO_PUTMSGRECOFFSET   DS   F     Offset of first put message record
*                                  from start of MQPMO
MQPMO_RESPONSERECOFFSET DS   F     Offset of first response record
*                                  from start of MQPMO
MQPMO_PUTMSGRECPTR      DS   F     Address of first put message
*                                  record
MQPMO_RESPONSERECPTR    DS   F     Address of first response record
MQPMO_ORIGINALMSGHANDLE DS   D     Original message handle
MQPMO_NEWMSGHANDLE      DS   D     New message handle
MQPMO_ACTION            DS   F     The action being performed
MQPMO_PUBLEVEL          DS   F     Publish level
*
MQPMO_LENGTH            EQU  *-MQPMO
                        ORG  MQPMO
MQPMO_AREA              DS   CL(MQPMO_LENGTH)
```

Visual Basic declaration for MQPMO:

```
Type MQPMO
  StrucId           As String*4  'Structure identifier'
  Version           As Long      'Structure version number'
  Options           As Long      'Options that control the action of'
                                 'MQPUT and MQPUT1'
  Timeout           As Long      'Reserved'
  Context           As Long      'Object handle of input queue'
  KnownDestCount    As Long      'Number of messages sent successfully'
                                 'to local queues'
  UnknownDestCount  As Long      'Number of messages sent successfully'
                                 'to remote queues'
  InvalidDestCount  As Long      'Number of messages that could not be'
                                 'sent'
  ResolvedQName     As String*48 'Resolved name of destination queue'
  ResolvedQMgrName  As String*48 'Resolved name of destination queue'
                                 'manager'
  RecsPresent       As Long      'Number of put message records or'
                                 'response records present'
  PutMsgRecFields   As Long      'Flags indicating which MQPMR fields'
                                 'are present'
  PutMsgRecOffset   As Long      'Offset of first put message record'
                                 'from start of MQPMO'
  ResponseRecOffset As Long      'Offset of first response record from'
                                 'start of MQPMO'
  PutMsgRecPtr      As MQPTR     'Address of first put message record'
  ResponseRecPtr    As MQPTR     'Address of first response record'
End Type
```

MQPMR - Put-message record:

The following table summarizes the fields in the structure.

Table 230. Fields in MQPMR

Field	Description	Topic
MsgId	Message identifier	MsgId
CorrelId	Correlation identifier	CorrelId
GroupId	Group identifier	GroupId
Feedback	Feedback or reason code	Feedback
AccountingToken	Accounting token	AccountingToken

Overview for MQPMR:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ clients connected to these systems.

Purpose: Use the MQPMR structure to specify various message properties for a single destination when putting a message to a distribution list. MQPMR is an input/output structure for the MQPUT and MQPUT1 calls.

Character set and encoding: Data in MQPMR must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ client, the structure must be in the character set and encoding of the client.

Usage: By providing an array of these structures on the MQPUT or MQPUT1 call, you can specify different values for each destination queue in a distribution list. Some of the fields are input only, others are input/output.

Note: This structure is unusual in that it does not have a fixed layout. The fields in this structure are optional, and the presence or absence of each field is indicated by the flags in the *PutMsgRecFields* field in MQPMO. Fields that are present ***must occur in the following order*** :

- *MsgId*
- *CorrelId*
- *GroupId*
- *Feedback*
- *AccountingToken*

Fields that are absent occupy no space in the record.

Because MQPMR does not have a fixed layout, no definition of it is provided in the header, COPY, and INCLUDE files for the supported programming languages. The application programmer must create a declaration containing the fields that are required by the application, and set the flags in *PutMsgRecFields* to indicate the fields that are present.

Fields for MQPMR:

The MQPMR structure contains the following fields; the fields are described in **alphabetical order**:

AccountingToken (MQBYTE32):

This is the accounting token to be used for the message sent to the queue with the name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *AccountingToken* field in MQMD for a put to a single queue. See the description of *AccountingToken* in "MQMD - Message descriptor" on page 2387 for information about the content of this field.

If this field is not present, the value in MQMD is used.

This is an input field.

CorrelId (MQBYTE24):

This is the correlation identifier to be used for the message sent to the queue with a name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *CorrelId* field in MQMD for a put to a single queue.

If this field is not present in the MQPMR record, or there are fewer MQPMR records than destinations, the value in MQMD is used for those destinations that do not have an MQPMR record containing a *CorrelId* field.

If MQPMO_NEW_CORREL_ID is specified, a *single* new correlation identifier is generated and used for all the destinations in the distribution list, regardless of whether they have MQPMR records. This is different from the way that MQPMO_NEW_MSG_ID is processed (see *MsgId* field).

This is an input/output field.

Feedback (MQLONG):

This is the feedback code to be used for the message sent to the queue with the name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *Feedback* field in MQMD for a put to a single queue.

If this field is not present, the value in MQMD is used.

This is an input field.

GroupId (MQBYTE24):

GroupId is the group identifier to be used for the message sent to the queue with the name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *GroupId* field in MQMD for a put to a single queue.

If this field is not present in the MQPMR record, or there are fewer MQPMR records than destinations, the value in MQMD is used for those destinations that do not have an MQPMR record containing a *GroupId* field. The value is processed as documented in Physical order on a queue, but with the following differences:

- GroupId is created from the QMName and a timestamp. Therefore to keep a GroupId unique keep queue manager names unique too. Also do not set the clocks back on the queue managers machine.
- In those cases where a new group identifier would be used, the queue manager generates a different group identifier for each destination (that is, no two destinations have the same group identifier).
- In those cases where the value in the field would be used, the call fails with reason code MQRC_GROUP_ID_ERROR

This is an input/output field.

MsgId (MQBYTE24):

This is the message identifier to be used for the message sent to the queue with a name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *MsgId* field in MQMD for a put to a single queue.

If this field is not present in the MQPMR record, or there are fewer MQPMR records than destinations, the value in MQMD is used for those destinations that do not have an MQPMR record containing a *MsgId* field. If that value is MQMI_NONE, a new message identifier is generated for *each* of those destinations (that is, no two of those destinations have the same message identifier).

If MQPMO_NEW_MSG_ID is specified, new message identifiers are generated for all the destinations in the distribution list, regardless of whether they have MQPMR records. This is different from the way that MQPMO_NEW_CORREL_ID is processed (see *CorrelId* field).

This is an input/output field.

Initial values and language declarations for MQPMR:

There are no initial values defined for this structure, as no structure declarations are provided in the header, COPY, and INCLUDE files for the supported programming languages. The sample declarations show how to declare the structure if all the fields are required.

C declaration for MQPMR:

```
typedef struct tagMQPMR MQPMR;
struct tagMQPMR {
  MQBYTE24  MsgId;            /* Message identifier */
  MQBYTE24  CorrelId;         /* Correlation identifier */
  MQBYTE24  GroupId;          /* Group identifier */
  MQLONG    Feedback;         /* Feedback or reason code */
  MQBYTE32  AccountingToken;  /* Accounting token */
};
```

COBOL declaration for MQPMR:

```
**   MQPMR structure
  10 MQPMR.
**    Message identifier
   15 MQPMR-MSGID           PIC X(24).
**    Correlation identifier
   15 MQPMR-CORRELID        PIC X(24).
**    Group identifier
   15 MQPMR-GROUPID         PIC X(24).
**    Feedback or reason code
   15 MQPMR-FEEDBACK        PIC S9(9) BINARY.
**    Accounting token
   15 MQPMR-ACCOUNTINGTOKEN PIC X(32).
```

PL/I declaration for MQPMR:

```
dcl
 1 MQPMR based,
  3 MsgId           char(24),      /* Message identifier */
  3 CorrelId        char(24),      /* Correlation identifier */
  3 GroupId         char(24),      /* Group identifier */
  3 Feedback        fixed bin(31), /* Feedback or reason code */
  3 AccountingToken char(32);      /* Accounting token */
```

Visual Basic declaration for MQPMR:

```
Type MQPMR
  MsgId           As MQBYTE24 'Message identifier'
  CorrelId        As MQBYTE24 'Correlation identifier'
  GroupId         As MQBYTE24 'Group identifier'
  Feedback        As Long     'Feedback or reason code'
  AccountingToken As MQBYTE32 'Accounting token'
End Type
```

MQRFH - Rules and formatting header:

This section describes the rules and formatting header, what fields it contains, and initial values of those fields.

Overview for MQRFH:

Availability: All IBM MQ systems, plus IBM MQ MQI clients connected to these systems.

Purpose: The MQRFH structure defines the layout of the rules and formatting header. Use this header to send string data in the form of name-value pairs.

Format name: MQFMT_RF_HEADER.

Character set and encoding: The fields in the MQRFH structure (including *NameValueString*) are in the character set and encoding given by the *CodedCharSetId* and *Encoding* fields in the header structure that precedes the MQRFH, or by those fields in the MQMD structure if the MQRFH is at the start of the application message data.

The character set must be one that has single-byte characters for the characters that are valid in queue names.

Fields for MQRFH:

The MQRFH structure contains the following fields; the fields are described in **alphabetical order**:

CodedCharSetId (MQLONG):

This specifies the character set identifier of the data that follows *NameValueString* ; it does not apply to character data in the MQRFH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

MQCCSI_INHERIT
Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

MQCCSI_INHERIT cannot be used if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG):

This specifies the numeric encoding of the data that follows *NameValueString* ; it does not apply to numeric data in the MQRFH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is MQENC_NATIVE.

Flags (MQLONG):

The following can be specified:

MQRFH_NONE
No flags.

The initial value of this field is MQRFH_NONE.

Format (MQCHAR8):

This specifies the format name of the data that follows *NameValueString*.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *Format* field in MQMD.

The initial value of this field is MQFMT_NONE.

NameValueString (MQCHARn):

This is a variable-length character string containing name-value pairs in the form:

```
name1 value1 name2 value2 name3 value3 ...
```

Each name or value must be separated from the adjacent name or value by one or more blank characters; these blanks are not significant. A name or value can contain significant blanks by prefixing and suffixing the name or value with double quotation marks; all characters between the open double quotation mark and the matching closing double quotation mark are treated as significant. In the following example, the name is `FAMOUS_WORDS`, and the value is `Hello World`:

```
FAMOUS_WORDS "Hello World"
```

A name or value can contain any characters other than the null character (which acts as a delimiter for *NameValueString*). However, to assist interoperability an application can restrict names to the following characters:

- First character: upper or lowercase alphabetic (A through Z, or a through z), or underscore.
- Subsequent characters: upper or lowercase alphabetic, decimal digit (0 through 9), underscore, hyphen, or dot.

If a name or value contains one or more double quotation marks, the name or value must be enclosed in double quotation marks, and each double quotation mark within the string must be doubled:

```
Famous_Words "The program displayed ""Hello World"""
```

Names and values are case sensitive, that is, lowercase letters are not considered to be the same as uppercase letters. For example, `FAMOUS_WORDS` and `Famous_Words` are two different names.

The length in bytes of *NameValueString* is equal to *StrucLength* minus MQRFH_STRUC_LENGTH_FIXED. To avoid problems converting the user data in some environments, make this length a multiple of four. Pad *NameValueString* with blanks to this length, or terminate it earlier by placing a null character following the last significant character in the string. The null character and the bytes following it, up to the specified length of *NameValueString*, are ignored.

Note: Because the length of this field is not fixed, the field is omitted from the declarations of the structure that are provided for the supported programming languages.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQRFH_STRUC_ID
Identifier for rules and formatting header structure.

For the C programming language, the constant MQRFH_STRUC_ID_ARRAY is also defined; this has the same value as MQRFH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQRFH_STRUC_ID.

StrucLength (MQLONG):

This is the length in bytes of the MQRFH structure, including the *NameValueString* field at the end of the structure. The length does not include any user data that follows the *NameValueString* field.

To avoid problems converting the user data in some environments, *StrucLength* must be a multiple of four.

The following constant gives the length of the *fixed* part of the structure, that is, the length excluding the *NameValueString* field:

MQRFH_STRUC_LENGTH_FIXED
Length of fixed part of MQRFH structure.

The initial value of this field is MQRFH_STRUC_LENGTH_FIXED.

Version (MQLONG):

This is the structure version number; the value must be:

MQRFH_VERSION_1
Version-1 rules and formatting header structure.

The initial value of this field is MQRFH_VERSION_1.

Initial values and language declarations for MQRFH:

Table 231. Initial values of fields in MQRFH for MQRFH

Field name	Name of constant	Value of constant
StrucId	MQRFH_STRUC_ID	'RFH¬'
Version	MQRFH_VERSION_1	1
StrucLength	MQRFH_STRUC_LENGTH_FIXED	32
Encoding	MQENC_NATIVE	Depends on environment
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
Flags	MQRFH_NONE	0

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQRFH_DEFAULT contains the values that are listed in the table. It can be used in the following way to provide initial values for the fields in the structure:

```
MQRFH MyRFH = {MQRFH_DEFAULT};
```

C declaration for MQRFH:

```
typedef struct tagMQRFH MQRFH;
struct tagMQRFH {
  MQCHAR4  StrucId;         /* Structure identifier */
  MQLONG   Version;         /* Structure version number */
  MQLONG   StrucLength;     /* Total length of MQRFH including
                               NameValueString */
  MQLONG   Encoding;        /* Numeric encoding of data that follows
                               NameValueString */
  MQLONG   CodedCharSetId;  /* Character set identifier of data that
                               follows NameValueString */
  MQCHAR8  Format;          /* Format name of data that follows
                               NameValueString */
  MQLONG   Flags;           /* Flags */
};
```

COBOL declaration for MQRFH:

```
**   MQRFH structure
  10 MQRFH.
**    Structure identifier
   15 MQRFH-STRUCID        PIC X(4).
**    Structure version number
   15 MQRFH-VERSION        PIC S9(9) BINARY.
**    Total length of MQRFH including NAMEVALUESTRING
   15 MQRFH-STRUCLENGTH    PIC S9(9) BINARY.
**    Numeric encoding of data that follows NAMEVALUESTRING
   15 MQRFH-ENCODING       PIC S9(9) BINARY.
**    Character set identifier of data that follows NAMEVALUESTRING
   15 MQRFH-CODEDCHARSETID PIC S9(9) BINARY.
**    Format name of data that follows NAMEVALUESTRING
   15 MQRFH-FORMAT         PIC X(8).
**    Flags
   15 MQRFH-FLAGS          PIC S9(9) BINARY.
```

PL/I declaration for MQRFH:

```
dcl
 1 MQRFH based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 StrucLength    fixed bin(31), /* Total length of MQRFH including
                                     NameValueString */
  3 Encoding       fixed bin(31), /* Numeric encoding of data that
                                     follows NameValueString */
  3 CodedCharSetId fixed bin(31), /* Character set identifier of data
                                     that follows NameValueString */
  3 Format         char(8),       /* Format name of data that follows
                                     NameValueString */
  3 Flags          fixed bin(31); /* Flags */
```

High Level Assembler declaration for MQRFH:

```
MQRFH                 DSECT
MQRFH_STRUCID         DS   CL4  Structure identifier
MQRFH_VERSION         DS   F    Structure version number
MQRFH_STRUCLENGTH     DS   F    Total length of MQRFH including
*                              NAMEVALUESTRING
MQRFH_ENCODING        DS   F    Numeric encoding of data that follows
*                              NAMEVALUESTRING
MQRFH_CODEDCHARSETID  DS   F    Character set identifier of data that
*                              follows NAMEVALUESTRING
MQRFH_FORMAT          DS   CL8  Format name of data that follows
*                              NAMEVALUESTRING
MQRFH_FLAGS           DS   F    Flags
*
MQRFH_LENGTH          EQU  *-MQRFH
                      ORG  MQRFH
MQRFH_AREA            DS   CL(MQRFH_LENGTH)
```

Visual Basic declaration for MQRFH:

```
Type MQRFH
  StrucId        As String*4 'Structure identifier'
  Version        As Long     'Structure version number'
  StrucLength    As Long     'Total length of MQRFH including'
                             'NameValueString'
  Encoding       As Long     'Numeric encoding of data that follows'
                             'NameValueString'
  CodedCharSetId As Long     'Character set identifier of data that'
                             'follows NameValueString'
  Format         As String*8 'Format name of data that follows'
                             'NameValueString'
  Flags          As Long     'Flags'
End Type
```

MQRFH2 - Rules and formatting header 2:

This section describes the rules and formatting header 2, what fields it contains, and initial values of those fields.

Overview for MQRFH2:

Availability

All IBM MQ systems, plus IBM MQ MQI clients connected to these systems.

Purpose

The MQRFH2 header is based on the MQRFH header, but it allows Unicode strings to be transported without translation, and it can carry numeric data types.

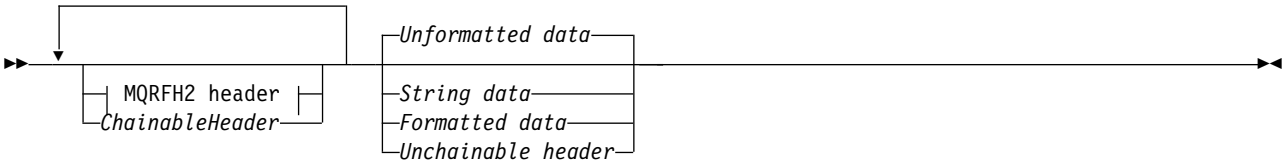
The MQRFH2 structure defines the format of the version-2 rules and formatting header. Use this header to send data that has been encoded using an XML-like syntax. A message can contain two or more MQRFH2 structures in series, with user data optionally following the last MQRFH2 structure in the series.

Format name

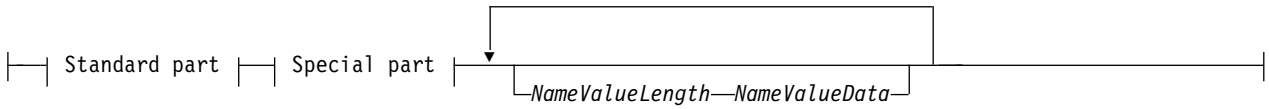
MQFMT_RF_HEADER_2

Syntax

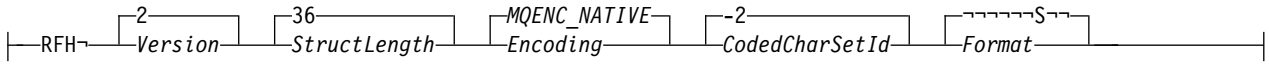
IBM MQ Message



MQRFH2 header:



Standard part:



Special part:

Character set and encoding

Special rules apply to the character set and encoding used for the MQRFH2 structure:

- Fields other than *NameValueData* are in the character set and encoding given by the *CodedCharSetId* and *Encoding* fields in the header structure that precedes MQRFH2, or by those fields in the MQMD structure if the MQRFH2 is at the start of the application message data.

  The character set must be one that has single-byte characters for the characters that are valid in queue names.

  When MQGMO_CONVERT is specified on the MQGET call, the queue manager converts the MQRFH2 fields, other than *NameValueData*, to the requested character set and encoding.
- *NameValueData* is in the character set given by the *NameValueCCSID* field. Only the listed Unicode character sets are valid for *NameValueCCSID* ; see the description of *NameValueCCSID* for details.

  Some character sets have a representation that depends on the encoding. If *NameValueCCSID* is one of these character sets, *NameValueData* must be in the same encoding as the other fields in the MQRFH2.

  When MQGMO_CONVERT is specified on the MQGET call, the queue manager converts *NameValueData* to the requested encoding, but does not change its character set.

Fields for MQRFH2:

The MQRFH2 structure contains the following fields; the fields are described in **alphabetical order**:

CodedCharSetId (MQLONG):

This specifies the character set identifier of the data that follows the last *NameValueData* field; it does not apply to character data in the MQRFH2 structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

MQCCSI_INHERIT
Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

MQCCSI_INHERIT cannot be used if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

The initial value of this field is MQCCSI_INHERIT.

Encoding (MQLONG):

This specifies the numeric encoding of the data that follows the last *NameValueData* field; it does not apply to numeric data in the MQRFH2 structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is MQENC_NATIVE.

Flags (MQLONG):

The initial value of this field is MQRFH_NONE. MQRFH_NONE must be specified.

MQRFH_NONE
No flags.

MQRFH_INTERNAL
The MQRFH2 header contains internally set properties.

MQRFH_INTERNAL is for queue manager use.

The top 16 bits, MQRFH_FLAGS_RESTRICTED_MASK, are reserved for flags the queue manager sets. Flags that a user might set are defined in the bottom 16 bits.

Format (MQCHAR8):

This specifies the format name of the data that follows the last *NameValueData* field.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *Format* field in MQMD.

The initial value of this field is MQFMT_NONE.

NameValueCCSID (MQLONG):

This specifies the coded character set identifier of the data in the *NameValueData* field. This is different from the character set of the other strings in the MQRFH2 structure, and can be different from the character set of the data (if any) that follows the last *NameValueData* field at the end of the structure.

NameValueCCSID must have one of the following values: V 9.0.0

CCSID	Meaning
1200	UTF-16, most recent Unicode version supported
13488	UTF-16, Unicode version 2.0 subset
17584	UTF-16, Unicode version 3.0 subset (includes the Euro symbol)
1208	UTF-8, most recent Unicode version supported

V 9.0.0

For the UTF-16 character sets, the encoding (byte order) of the *NameValueData* must be the same as the encoding of the other fields in the MQRFH2 structure.

Characters beyond the Unicode Basic Multilingual Plane (those above U+FFFF), represented in UTF-16 by surrogate code points (X'D800' through X'DFFF'), or four bytes in UTF-8, are not supported.

Note: If *NameValueCCSID* does not have one of the values listed above, and the MQRFH2 structure requires conversion on the MQGET call, the call completes with reason code MQRC_SOURCE_CCSID_ERROR and the message is returned unconverted.

The initial value of this field is 1208.

NameValueData (MQCHARn):

NameValueData is a variable length field that contains a folder containing name-value pairs or message properties. A folder is a variable-length character string containing data encoded using an XML-like syntax. The length in bytes of the character string is given by the *NameValueLength* field that precedes the *NameValueData* field. The length must be a multiple of four.

The *NameValueLength* and *NameValueData* fields are optional, but if present they must occur as a pair and be adjacent. The pair of fields can be repeated as many times as required, for example:

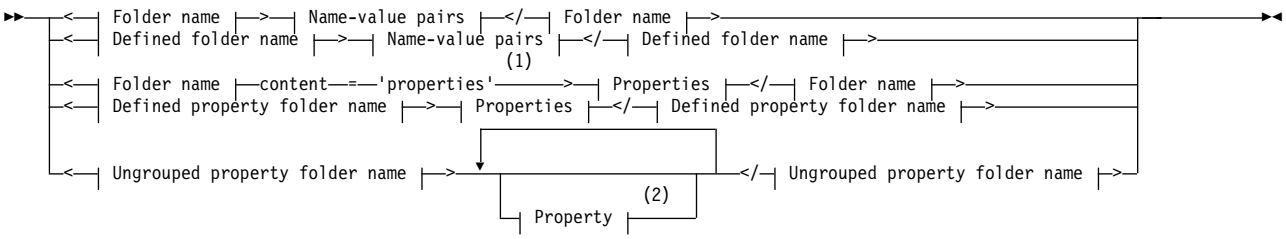
```
length1 data1 length2 data2 length3 data3
```

NameValueData is not converted to the character set specified on the MQGET call. Even if the message is retrieved with the MQGMO_CONVERT option in effect *NameValueData* remains in its original character set. However, *NameValueData* is converted to the encoding specified on the MQGET call.

Notes:

- Because these fields are optional, they are omitted from the declarations of the structure that are provided for the various programming languages supported.
- The terms “defined” and “reserved” are used in the syntax diagram. “Defined” means that the name is used by IBM MQ. “Reserved” means that the name is reserved for future use by IBM MQ.

NameValueData **syntax**



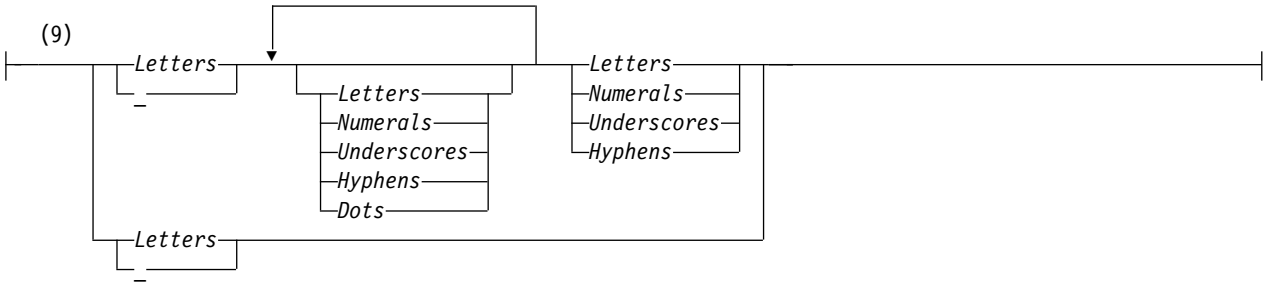
Folder name:

Defined folder name:

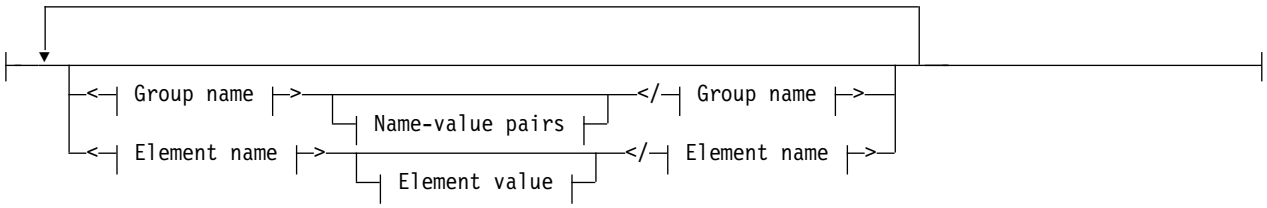
Defined property folder name:

Ungrouped property folder name:

Name:



Name-value pairs:

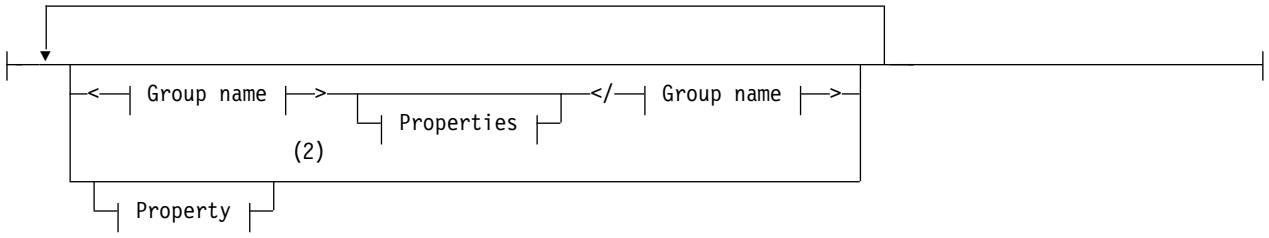


Group name:

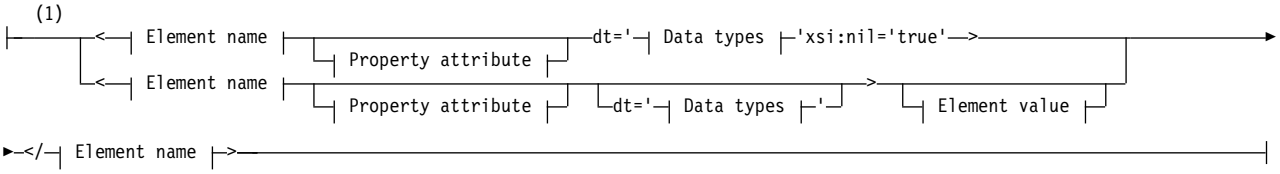
Element name:

Element value:

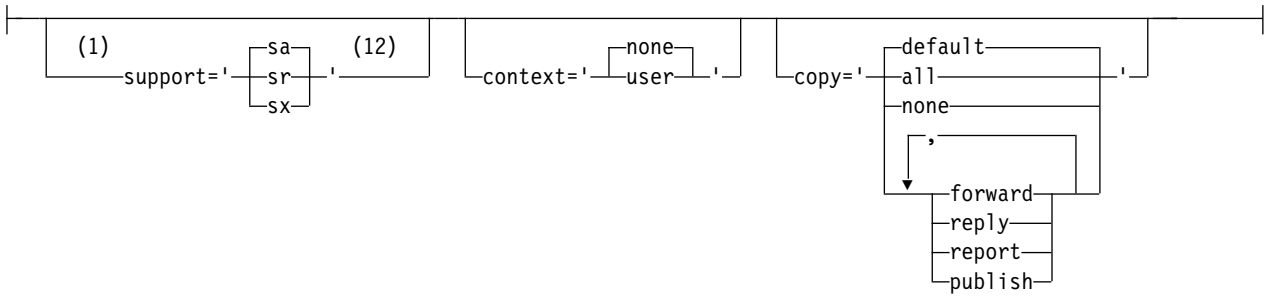
Properties:



Property:



Property attribute:



Data types:

Notes:

1 Double quotation marks or single quotation marks are valid.

2 Do not use an invalid property name; see “Invalid property name” on page 2525. Use a reserved property name only for its defined purpose; see “Defined property names” on page 2525.

3 Do not use an invalid or reserved folder name; see “Invalid path name” on page 2524 and “Reserved folder or property folder name” on page 2524. Use a defined folder name only for its defined purpose; see “Defined folder name” on page 2516.

4 The name must be in lowercase.

5 Only one psc and pscr folder is supported.

6 Only properties in the first MQRFH2 header are significant. WebSphere Application Server Service Integration Bus ignores sib, sib_context, and sib_usr folders in subsequent MQRFH2 headers.

7 Not more than one usr folder must be present in an MQRFH2. Properties in the usr folder must occur no more than once.

8 Only properties in the first mq folder are significant. If the folder is UTF-8, only single byte UTF-8 characters are supported. The only white space character is Unicode U+0020.

9 Valid characters are defined in the W3C XML specification, and consist essentially of Unicode categories Ll, Lu, Lo, Lt, Nl, Mc, Mn, Lm, and Nd ; see Unicode character categories.

10 All characters are significant. Leading and trailing blanks are part of the element value.

11 Do not use an invalid character; see "Invalid characters" on page 2525. Use an escape sequence, rather than these invalid characters.

12 The support property attribute is only valid on the mq folder

Folder name

NameValueData contains a single folder. To create multiple folders, create multiple *NameValueData* fields. You can create multiple *NameValueData* fields in a single MQRFH2 header within a message. Alternatively you can create multiple chained MQRFH2 headers, each containing multiple *NameValueData* fields.

The order of MQRFH2 headers, and the order of *NameValueData* fields makes no difference to the logical contents of a folder. If the same folder is present more than once in a message the folder is parsed as a whole. If the same property occurs in muliple instances of the same folder, it is parsed as a list.

A correct parse of an MQRFH2 is not affected by the alternative ways a folder can be physically stored in a message.

Four folders do not follow this rule. Only the first instance of the mq, sib, sib_context, and sib_usr folder are parsed.

If the same property occurs more than once in the combined contents of the chained MQRFH2 headers, only the first instance of the property is parsed. If a property is set using an API call, such as MQSETMP, and added to an MQRFH2 directly by an application, the API call takes precedence.

A folder name is the name of a folder containing name-value pairs or groups. Groups and name-value pairs can be mixed at the same level in the folder tree; see Figure 51. Do not combine a group name and an element name; see Figure 52

```
<group1><nvp1>value</nvp1></group1><group2><nvp2>value</nvp2></group2>
<group3><nvp1>value</nvp1></group3><nvp3>value</nvp3>
```

Figure 51. Correct uses of groups and name-value pairs

```
<group1><nvp1> value </nvp1> value </group1>
```

Figure 52. Incorrect use of groups and name-value pairs

Do not use an invalid or reserved folder name; see “Invalid path name” on page 2524 and “Reserved folder or property folder name” on page 2524. Use a defined folder name only for its defined purpose; see “Defined folder name.”

If you add the attribute `'content=properties'` to the folder name tag, the folder becomes a property folder; see Figure 53.

```
<myFolder></myfolder>
<myPropertyFolder contents='properties'></myPropertyFolder>
```

Figure 53. Example of a folder and a property folder

Folder names are case-sensitive. Folder names and property folder names share the same namespace. They must have different names. `Folder1` in Figure 54 must be a different name to `Folder2` in Figure 55.

```
< Folder1 ><NVP1> value </NVP1></ Folder1 >
```

Figure 54. `Folder1` namespace

```
< Folder2 content='properties'>< Property1 > value </ Property1 ></ Folder2 >
```

Figure 55. `Folder2` namespace

Groups, properties, and name-value pairs in different folders have different namespaces. `Property1` in Figure 55 is a different property to `Property1` in Figure 56.

```
<Folder3 content='properties'>< Property1 > value </ Property1 ></Folder3>
```

Figure 56. `Folder3` namespace

Property folders are different to non-property folders in two important respects:

1. Property folders contain properties, and non-property folders contain name-value pairs. The folders differ slightly, syntactically.
2. Use the defined interfaces, such as the properties MQI, or JMS message properties, to access message properties. The interfaces ensure the property folders in the `MQRFH2` are well-formed. A well-formed property folder is interoperable between queue managers on different platforms and different releases.

The message property MQI is a robust way to read and write an `MQRFH2`, and avoids the difficulties of parsing an `MQRFH2` correctly.

Defined folder name

A defined folder name is the name of a folder that is reserved for use by IBM MQ, or another product. Do not create a folder of the same name, and do not add your own name-value pairs to the folders. The defined folders are **psc** and **pscr**.

psc and **pscr** are used by queued publish/subscribe.

A segmented message put with either MQMF_SEGMENT or MQMF_SEGMENTATION_ALLOWED cannot contain an MQRFH2 with a defined folder name. The MQPUT fails with reason code 2443, MQRC_SEGMENTATION_NOT_ALLOWED.

Defined property folder name

A defined property folder name is the name of a property folder that is used by IBM MQ, or another product. For the names of the folders and their contents, see Property folders. Defined property folder names are a subset of all the folder names reserved by IBM MQ; see "Reserved folder or property folder name" on page 2524.

Any element stored in a defined property folder is a property. An element stored in a defined property folder must not have a content='properties' attribute.

You can add properties only to the defined property folders **usr**, **mq_usr**, and **sib_usr**. In other property folders, such as **mq** and **sib**, IBM MQ ignores or throws away properties it does not recognize.

The description of each defined property folder lists the properties that IBM MQ has defined that can be used by application programs. Some of the properties are accessed indirectly by setting or getting a JMS property, and some are accessed directly using the MQSETMP and MQINQMP MQI calls.

The defined property folders also contain other properties that IBM MQ has reserved, but which applications do not have access to. The names of the reserved properties are not listed. No reserved properties are present in the **usr**, **mq_usr**, and **sib_usr** property folders. But do not create properties with invalid property names; see "Invalid property name" on page 2525.

Property folders

jms

jms contains JMS header fields, and JMSX properties that cannot be fully expressed in the MQMD. The jms folder is always present in a JMS MQRFH2.

Table 232. jms property name, synonym, data type, and folder

Property synonym	Property name	Data type	Folder
JMSDestination	jms.Dst	string	<jms><Dst> *destination* </Dst></jms>
JMSExpiration	jms.Exp	i8	<jms><Exp> *expiration* </Exp></jms>
JMSCorrelation	jms.Cid	string	<jms><Cid> *correlationId* </Cid></jms>
JMSDelivery	jms.Dlv	i4	<jms><Dlv> *delivery* </Dlv></jms>
JMSPriority	jms.Pri	i4	<jms><Pri> *priority* </Pri></jms>
JMSReplyTo	jms.Rto	string	<jms><Rto> *replyToURI* </Rto></jms>
JMSTimeStamp	jms.Tms	i8	<jms><Tms> *timestamp* </Tms></jms>
JMSXGroupID	jms.Gid	string	<jms><Gid> *groupId* </Gid></jms>
JMSXGroupSeq	jms.Seq	i4	<jms><Seq> *messageSequenceNo* </Seq></jms>

Do not add your own properties in the jms folder.

mcd

mcd contains properties that describe the format of the message. For example, the message service domain Msd property identifies a JMS message as being JMSTextMessage, JMSBytesMessage, JMSStreamMessage, JMSMapMessage, JMSObjectMessage, or null.

The mcd folder is always present in a JMS message containing an MQRFH2.

It is always present in a message containing an MQRFH2 sent from IBM Integration Bus. It describes the domain, format, type, and message set of a message.

Table 233. mcd property name, synonym, data type, and folder

Property synonym	Property name	Data type	Folder
	mcd.Msd	string	<mcd><Msd>*messageDomain*</Msd></mcd>
	mcd.Set	string	<mcd><Set>*messageDomain*</Set></mcd>
	mcd.Type	string	<mcd><Type>*messageDomain*</Type></mcd>
	mcd.Fmt	string	<mcd><Fmt>*messageDomain*</Fmt></mcd>

Do not add your own properties in the mcd folder.

mq_usr

mq_usr contains application-defined properties that are not exposed as JMS user-defined properties. Properties that do not meet JMS requirements can be placed in this folder.

You can create properties in the mq_usr folder. Properties you create in the mq_usr are like properties you create in new folders with the content='properties' attribute.

sib

sib contains WebSphere Application Server service integration bus (WAS/SIB) system message properties. sib properties are not exposed as JMS properties to IBM MQ JMS applications because they are not of the supported types. For example, some sib properties cannot be exposed as JMS properties because they are byte arrays. Some sib properties are exposed to WAS/SIB applications as JMS_IBM_* properties; these include forward and reverse routing paths properties.

Do not add your own properties in the sib folder.

sib_context

sib_context contains WAS/SIB system message properties that are not exposed to WAS/SIB user applications or as JMS properties. sib_context contains security and transactional properties that are used for web services.

Do not add your own properties in the sib_context folder.

sib_usr

sib_usr contains WAS/SIB user message properties that are not exposed as JMS user properties because they are not of supported types. sib_usr is exposed to WAS/SIB applications in the SIMessage interface; see Developing Service Integration.

The type of a sib_usr property must be bin.hex, and the value must be in the correct format. If an IBM MQ application writes a bin.hex typed element to the folder in the wrong format, the application receives an IOException. If the data type of the property is not bin.hex the application receives a ClassCastException.

Do not attempt to make JMS user properties available to WAS/SIB by using this folder; instead use the usr folder.

You can create properties in the sib_usr folder.

usr

usr contains application-defined JMS properties associated with the message. The usr folder is present only if an application has set an application-defined property.

usr is the default property folder. If a property is set without a folder name, it is placed in the usr folder.

Table 234. usr property name, synonym, data type, and folder.

The web services property values are described in MQRFH2 SOAP settings

Property synonym	Property name	Data type	Folder
	usr.contentType	string	<usr><contentType>text/xml; charset=utf-8</contentType></usr>
	usr.endPointURL	string	<usr><endPointURL> *URI* </endPointURL></usr>
	usr.targetService	string	<usr><targetService> *serviceName* </targetService></usr>
	usr.soapAction	string	<usr><soapAction> *name* </soapAction></usr>
	usr.transportVersion	string	<usr><transportVersion> *version* </transportVersion></usr>

You can create properties in the usr folder.

A segmented message put with either MQMF_SEGMENT or MQMF_SEGMENTATION_ALLOWED cannot contain an MQRFH2 with a defined property folder name. The MQPUT fails with reason code 2443, MQRC_SEGMENTATION_NOT_ALLOWED.

Ungrouped property folder name

ibm

ibm contains properties that are used only by IBM MQ.

Table 235. ibm property name, synonym, data type, and folder

Property synonym	Property name	Data type	Folder
	ibm.rfp	string	<ibm><rfp>fingerprint</rfp></ibm>

Do not add your own properties in the ibm folder.

mq

mq contains properties that are used only by IBM MQ.

The following restrictions apply to properties in the mq folder:

- Only properties in the first significant mq folder in the message are acted upon by MQ; properties in any other mq folder in the message are ignored.
- Only single-byte UTF-8 characters are allowed in the folder. A multi-byte character in the folder, can cause parsing to fail, and the message to be rejected.
- Do not use escape strings in the folder. An escape string is treated as the actual value of the element.
- Only Unicode character U+0020 is treated as white space within the folder. All other characters are treated as significant and can cause parsing of the folder to fail, and the message to be rejected.

If parsing of the mq folder fails, or if the folder does not observe these restrictions, the message is rejected with reason code 2527, MQRC_RFH_RESTRICTED_FORMAT_ERR.

Do not add your own properties in the mq folder.

mqema

mqema contains properties that are used only by WebSphere Application Server. The folder has been replaced by mqext.

Do not add your own properties in the mqema folder.

mqext

mqext contains the following types of property:

- Properties that are used only by WebSphere Application Server.
- Properties relating to delayed delivery of messages.

The folder is present if the application has either set at least one of the IBM defined properties or used delivery delay.

Table 236. mqext property name, synonym, data type, and folder

Property synonym	Property name	Data type	Folder
JMSArmCorrelator	mqext.Arm	string	<mqext><Arm>*armCorrelator*</Arm></mqext>
JMSRMCorrelator	mqext.Wrm	string	<mqext><Wrm>*wrmCorrelator*</Wrm></mqext>
JMSDeliveryTime	mqext.Dlt	i8	<mqext><Dlt>*DeliveryTime*</Dlt></mqext>
JMSDeliveryDelay	mqext.Dly	i8	<mqext><Dly>*DeliveryTime*</Dly></mqext>

Do not add your own properties in the mqext folder.

mqps

mqps contains properties that are used only by IBM MQ publish/subscribe. The folder is present only if the application has set at least one of the integrated publish/subscribe properties.

Table 237. mqps property name, synonym, data type, and folder

Property synonym	Property name	Data type	Folder
MQTopicString	mqps.Top	string	<mqps><Top>*topicString*</Top></mqps>
MQSubUserData	mqps.Sud	string	<mqps><Sud>*subscriberUserData*</Sud></mqps>
MQIsRetained	mqps.Ret	boolean	<mqps><Ret>*isRetained*</Ret></mqps>
MQPubOptions	mqps.Pub	i8	<mqps><Pub>*publicationOptions*</Pub></mqps>
MQPubLevel	mqps.Pbl	i8	<mqps><Pbl>*publicationLevel*</Pbl></mqps>
MQPubTime	mqpse.Pts	string	<mqps><Pts>*publicationTime*</Pts></mqps>
MQPubSeqNum	mqpse.Seq	i8	<mqps><Seq>*publicationSequenceNumber*</Seq></mqps>
MQPubStrIntData	mqpse.Sid	string	<mqps><Sid>*publicationData*</Sid></mqps>
MQPubFormat	mqpse.Pfmt	i8	<mqps><Pfmt>*messageFormat*</Pfmt></mqps>

Do not add your own properties in the mqps folder.

mq_svc

mq_svc contains properties used by SupportPac MA93.

Do not add your own properties in the mq_svc folder.

mqtt

mqtt contains properties use by MQ Telemetry

Table 238. `mqtt` property name, synonym, data type, and folder

Property synonym	Property name	Data type	Folder
	`mqtt.clientId`	`string`	`<mqtt><clientId> topicString </clientId></mqtt>`
	`mqtt.qos`	`i4`	`<mqtt><qos> qualityOfService </qos></mqtt>`
	`mqtt.msgid`	`string`	`<mqtt><msgid> messageIdentifier </msgid></mqtt>`

Do not add your own properties in the `mqtt` folder.

A segmented message put with either `MQMF_SEGMENT` or `MQMF_SEGMENTATION_ALLOWED` cannot contain an `MQRFH2` with an ungrouped property folder name. The MQPUT fails with reason code `2443`, `MQRC_SEGMENTATION_NOT_ALLOWED`.

Name-value pairs

In the syntax diagram, "Name-value pairs" describes the content of an ordinary folder. An ordinary folder contains groups, and elements. An element is a name-value pair. A group contains elements and other groups.

In terms of trees, elements are leaf nodes, and groups are internal nodes. An internal node, and the folder, which is the root node, can contain a mixture of internal nodes and leaf nodes. A node cannot be both an internal node and a leaf node at the same time; see Figure 52 on page 2515.

Properties

In the syntax diagram, "Properties" describes the content of a property folder. A property folder contains groups, and properties. A property is a name-value pair with an optional data type attribute. A group contains properties and other groups.

In terms of trees, properties are leaf nodes, and groups are internal nodes. An internal node, and the property folder, which is the root node, can contain a mixture of internal nodes and leaf nodes. A node cannot be both an internal node and a leaf node at the same time; see Figure 52 on page 2515.

Property

A message property is a name-value pair in a property folder. It can optionally include a data type attribute and a property attribute; for an example, see Figure 57. If the data type attribute is omitted, the property type is string.

```
<pf><p1 dt='i8' > value </p1></pf>
```

Figure 57. Data type attribute

The name of a message property is its full path name, with the XML-like, <> syntax, replaced by dots. For example, `myPropertyFolder1.myGroup1.myGroup2.myProperty1` is mapped to a *`NameValueData`* string in Figure 58 on page 2522. The string is formatted for easier reading.

```
<myPropertyFolder1>
    <myGroup1>
        <myGroup2>
            <myProperty1>value</myProperty1>
        </myGroup2>
    </myGroup1>
</myPropertyFolder1>
```

Figure 58. Single property name mapping

A property folder can contain multiple properties. For example the properties in Figure 59 are mapped to the property folder in Figure 60

```
 myPropertyFolder1.myProperty4
myPropertyFolder1.myGroup1.myGroup2.myProperty1
myPropertyFolder1.myGroup1.myGroup2.myProperty2
myPropertyFolder1.myGroup1.myProperty3
```

Figure 59. Multiple properties with the same root name

```
<myPropertyFolder1>
    <myProperty4>value</myProperty4>
    <myGroup1>
        <myGroup2>
            <myProperty1>value</myProperty1>
            <myProperty2>value</myProperty2>
        </myGroup2>
        <myProperty3>value</myProperty3>
    </myGroup1>
</myPropertyFolder1>
```

Figure 60. Multiple property name mapping

Name

A name must begin with a *Letter* or an *Underscore*. It must not contain a *Colon*, not end in a *Period* and contain only *Letters*, *Numerals*, *Underscores*, *Hyphens*, and *Dots*. Valid characters are defined in the W3C XML specification, and consist essentially of Unicode categories `Ll`, `Lu`, `Lo`, `Lt`, `Nl`, `Mc`, `Mn`, `Lm`, and `Nd` ; see Unicode character categories.

The complete path of a property or name-value pair must not break the rule described in "Invalid path name" on page 2524. Paths are restricted to 4095 bytes, must not contain Unicode compatibility characters, and must not start with the string `XML`.

Group name

A group name has the same syntax as a name. Group names are optional. Properties and name-value pairs can be placed in the root of a folder. Use groups if it helps to organize properties and name-value pairs.

Element name

An element name has the same syntax as a name.

Element value

An element value includes all the white space between the < *Element name* > tag and the < */Element name* >. Do not use the two characters < and & in a value. Replace then with < and &.

Property attributes

The property attributes map property descriptor fields: The mappings are as follows:

Support

sa MQPD_SUPPORT_OPTIONAL

sr MQPD_SUPPORT_REQUIRED

sx MQPD_SUPPORT_REQUIRED_IF_LOCAL

Context

none
MQPD_NO_CONTEXT

user
MQPD_USER _CONTEXT

CopyOptions

forward
MQPD_COPY_FORWARD

reply
MQPD_COPY_REPLY

report
MQPD_COPY_REPORT

publish
MQPD_COPY_PUBLISH

all
MQPD_COPY_ALL

Do not use all in combination with other options.

default
MQPD_COPY_DEFAULT

Do not use default in combination with other options. default is the same as forward + report + publish.

none
MQPD_COPY_NONE

Do not use none in combination with other options.

The Support property attributes are only applicable to properties in the **mq** folder.

The Context and CopyOptions property attributes are applicable to all property folders.

Data types

MQRFH2 data types map to message property types as follows:

Table 239. Data type mappings

MQRFH2 data type	**Message property type**
bin.hex	MQBYTE[]
boolean	MQBOOL
i1	MQINT8
i2	MQINT16
i4	MQINT32
i8	MQINT64
r4	MQFLOAT32
r8	MQFLOAT64
string	MQCHAR[]

Any element without a data type is assumed to be of type string.

A null value is indicated by the element attribute `xsi:nil='true'`. Do not use the attribute `xsi:nil='false'` for non-null values. For example, the following property has a null value:

```
<NullProperty
xsi:nil='true'></NullProperty>
```

A byte or character string property can have an empty value. An empty value is represented by an MQRFH2 element with a zero length element value. For example, the following property has an empty value:

```
<EmptyProperty></EmptyProperty>
```

Reserved folder or property folder name

Restrict the name of a folder or property folder not to start with any of the following strings. The prefixes are reserved for folder or property names created by IBM.

Notes:

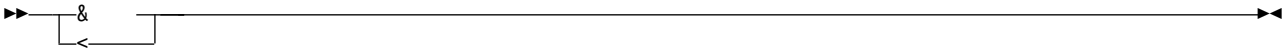
1 A reserved folder or property name contains any mixture of lower and uppercase letters.

Invalid path name

Restrict the complete path of a name-value pair or a property not to include any of the following strings.

Invalid characters

Always use the escape sequences `&` and `<` instead of the literals "&" and "<".

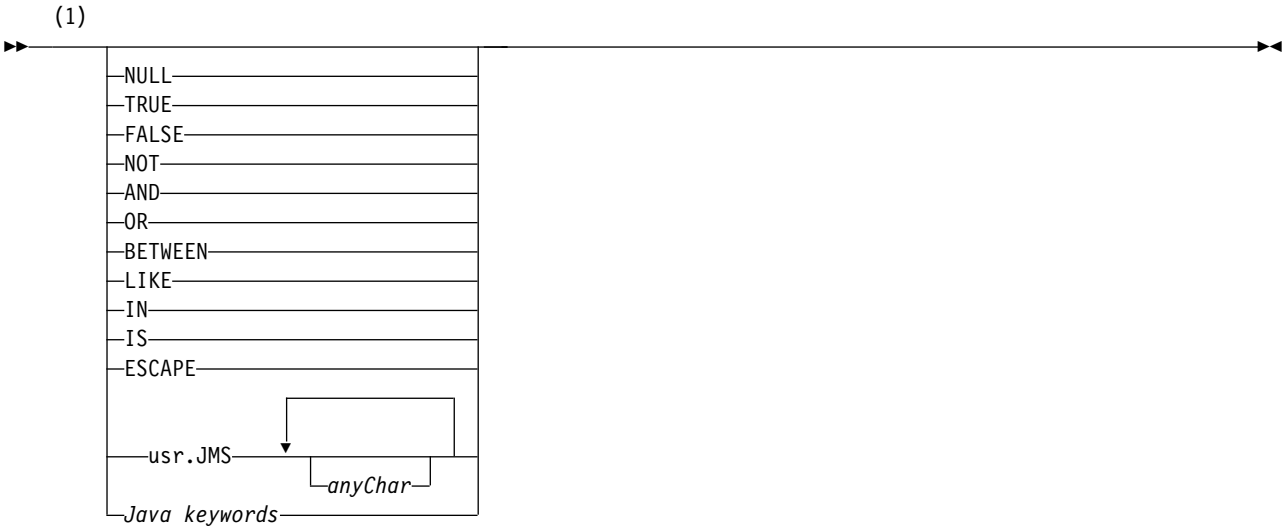


Defined property names

Defined property names are the names of properties that are defined by IBM MQ, or other products, and used by IBM MQ and user applications. Defined properties exist only in defined property folders. Defined property names are described in the description of property folders; see Property folders.

Invalid property name

Do not construct property names that match the following rule. The rule applies to the full property path that names a property, and not only to the property element name.



Notes:

1 An invalid property name can contain any combination of upper and lowercase.

Invalid attributes

Property folders and properties can include only supported "Property attributes" on page 2523 and "Data types" on page 2523.

Any non-supported XML-like attributes, for example, names with quoted string values, that are included in property folders or properties might be removed.

XML-like attributes included in non-property folders or non-property elements that remain in MQRFH2 headers.

NameValueLength (MQLONG):

The length of the corresponding NameValueData field

This specifies the length in bytes of the data in the *NameValueData* field. *NameValueLength* must be a multiple of four.

Note: The *NameValueLength* and *NameValueData* fields are optional, but if present they must occur as a pair and be adjacent. The pair of fields can be repeated as many times as required, for example:

```
length1 data1 length2 data2 length3 data3
```

Because these fields are optional, they are omitted from the declarations of the structure that are provided for the various programming languages supported.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQRFH_STRUC_ID
Identifier for rules and formatting header structure.

For the C programming language, the constant MQRFH_STRUC_ID_ARRAY is also defined; this has the same value as MQRFH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQRFH_STRUC_ID.

StrucLength (MQLONG):

This is the length in bytes of the MQRFH2 structure, including the *NameValueLength* and *NameValueData* fields at the end of the structure. It is valid for there to be multiple pairs of *NameValueLength* and *NameValueData* fields at the end of the structure, in the sequence:

```
length1, data1, length2, data2, ...
```

StrucLength does not include any user data that might follow the last *NameValueData* field at the end of the structure.

To avoid problems with converting the user data in some environments, *StrucLength* must be a multiple of four.

The following constant gives the length of the *fixed* part of the structure, that is, the length excluding the *NameValueLength* and *NameValueData* fields:

MQRFH_STRUC_LENGTH_FIXED_2
Length of fixed part of MQRFH2 structure.

The initial value of this field is MQRFH_STRUC_LENGTH_FIXED_2.

Version (MQLONG):

This is the structure version number; the value must be:

MQRFH_VERSION_2
Version-2 rules and formatting header structure.

The initial value of this field is MQRFH_VERSION_2.

Initial values and language declarations for MQRFH2:

Table 240. Initial values of fields in MQRFH2 for MQRFH2

Field name	Name of constant	Value of constant
StrucId	MQRFH_STRUC_ID	'RFH¬'
Version	MQRFH_VERSION_2	2
StrucLength	MQRFH_STRUC_LENGTH_FIXED_2	36
Encoding	MQENC_NATIVE	Depends on environment
CodedCharSetId	MQCCSI_INHERIT	-2
Format	MQFMT_NONE	Blanks
Flags	MQRFH_NONE	0
NameValueCCSID	None	1208

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQRFH2_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure:

```
MQRFH2 MyRFH2 = {MQRFH2_DEFAULT};
```

C declaration for MQRFH2:

```
typedef struct tagMQRFH2 MQRFH2;
struct tagMQRFH2 {
  MQCHAR4  StrucId;         /* Structure identifier */
  MQLONG   Version;         /* Structure version number */
  MQLONG   StrucLength;     /* Total length of MQRFH2 including all
                               NameValueLength and NameValueData
                               fields */
  MQLONG   Encoding;        /* Numeric encoding of data that follows
                               last NameValueData field */
  MQLONG   CodedCharSetId;  /* Character set identifier of data that
                               follows last NameValueData field */
  MQCHAR8  Format;          /* Format name of data that follows last
                               NameValueData field */
  MQLONG   Flags;           /* Flags */
  MQLONG   NameValueCCSID;  /* Character set identifier of
                               NameValueData */
};
```

COBOL declaration for MQRFH2:

```
**   MQRFH2 structure
  10 MQRFH2.
**    Structure identifier
   15 MQRFH2-STRUCID        PIC X(4).
**    Structure version number
   15 MQRFH2-VERSION        PIC S9(9) BINARY.
**    Total length of MQRFH2 including all NAMEVALUELENGTH and
**    NAMEVALUEDATA fields
   15 MQRFH2-STRUCLENGTH    PIC S9(9) BINARY.
**    Numeric encoding of data that follows last NAMEVALUEDATA field
   15 MQRFH2-ENCODING       PIC S9(9) BINARY.
**    Character set identifier of data that follows last NAMEVALUEDATA
**    field
   15 MQRFH2-CODEDCHARSETID PIC S9(9) BINARY.
**    Format name of data that follows last NAMEVALUEDATA field
   15 MQRFH2-FORMAT         PIC X(8).
**    Flags
   15 MQRFH2-FLAGS          PIC S9(9) BINARY.
**    Character set identifier of NAMEVALUEDATA
   15 MQRFH2-NAMEVALUECCSID PIC S9(9) BINARY.
```

PL/I declaration for MQRFH2:

```
dcl
 1 MQRFH2 based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 StrucLength    fixed bin(31), /* Total length of MQRFH2 including
                                     all NameValueLength and
                                     NameValueData fields */
  3 Encoding       fixed bin(31), /* Numeric encoding of data that
                                     follows last NameValueData field */
  3 CodedCharSetId fixed bin(31), /* Character set identifier of data
                                     that follows last NameValueData
                                     field */
  3 Format         char(8),       /* Format name of data that follows
                                     last NameValueData field */
  3 Flags          fixed bin(31), /* Flags */
  3 NameValueCCSID fixed bin(31); /* Character set identifier of
                                     NameValueData */
```

High Level Assembler declaration for MQRFH2:

```
MQRFH                DSECT
MQRFH_STRUCID        DS   CL4  Structure identifier
MQRFH_VERSION        DS   F    Structure version number
MQRFH_STRUCLENGTH    DS   F    Total length of MQRFH2 including all
*                              NAMEVALUELENGTH and NAMEVALUEDATA fields
MQRFH_ENCODING       DS   F    Numeric encoding of data that follows
*                              last NAMEVALUEDATA field
MQRFH_CODEDCHARSETID DS   F    Character set identifier of data that
*                              follows last NAMEVALUEDATA field
MQRFH_FORMAT         DS   CL8  Format name of data that follows last
*                              NAMEVALUEDATA field
MQRFH_FLAGS          DS   F    Flags
MQRFH_NAMEVALUECCSID DS   F    Character set identifier of
*                              NAMEVALUEDATA
*
MQRFH_LENGTH         EQU  *-MQRFH
                     ORG  MQRFH
MQRFH_AREA           DS   CL(MQRFH_LENGTH)
```

Visual Basic declaration for MQRFH2:

```
Type MQRFH2
  StrucId        As String*4 'Structure identifier'
  Version        As Long     'Structure version number'
  StrucLength    As Long     'Total length of MQRFH2 including all'
                             'NameValueLength and NameValueData fields'
  Encoding       As Long     'Numeric encoding of data that follows'
                             'last NameValueData field'
  CodedCharSetId As Long     'Character set identifier of data that'
                             'follows last NameValueData field'
  Format         As String*8 'Format name of data that follows last'
                             'NameValueData field'
  Flags          As Long     'Flags'
  NameValueCCSID As Long     'Character set identifier of NameValueData'
End Type
```

MQRMH - Reference message header:

The following table summarizes the fields in the structure.

Table 241. Fields in MQRMH

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
StrucLength	Total length of MQRMH, including strings at end of fixed fields, but not the bulk data	StrucLength
Encoding	Numeric encoding of bulk data	Encoding
CodedCharSetId	Character set identifier of bulk data	CodedCharSetId
Format	Format name of bulk data	Format
Flags	Reference message flags	Flags
ObjectType	Object type	ObjectType
ObjectInstanceId	Object instance identifier	ObjectInstanceId
SrcEnvLength	Length of source environment data	SrcEnvLength
SrcEnvOffset	Offset of source environment data	SrcEnvOffset
SrcNameLength	Length of source object name	SrcNameLength
SrcNameOffset	Offset of source object name	SrcNameOffset
DestEnvLength	Length of destination environment data	DestEnvLength
DestEnvOffset	Offset of destination environment data	DestEnvOffset
DestNameLength	Length of destination object name	DestNameLength
DestNameOffset	Offset of destination object name	DestNameOffset
DataLogicalLength	Length of bulk data	DataLogicalLength
DataLogicalOffset	Low offset of bulk data	DataLogicalOffset
DataLogicalOffset2	High offset of bulk data	DataLogicalOffset2

Overview for MQRMH:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ clients connected to these systems.

Purpose: The MQRMH structure defines the format of a reference message header. This header is used with user-written message channel exits to send extremely large amounts of data (called *bulk data*) from one queue manager to another. The difference compared to normal messaging is that the bulk data is not stored on a queue; instead, only a *reference* to the bulk data is stored on the queue. This reduces the possibility of MQ resources being exhausted by a small number of extremely large messages.

Format name: MQFMT_REF_MSG_HEADER.

Character set and encoding: Character data in MQRMH, and the strings addressed by the offset fields, must be in the character set of the local queue manager; this is given by the **CodedCharSetId** queue manager attribute. Numeric data in MQRMH must be in the native machine encoding; this is given by the value of MQENC_NATIVE for the C programming language.

Set the character set and encoding of the MQRMH into the *CodedCharSetId* and *Encoding* fields in:

- The MQMD (if the MQRMH structure is at the start of the message data), or
- The header structure that precedes the MQRMH structure (all other cases).

Usage: An application puts a message consisting of an MQRMH, but omitting the bulk data. When a message channel agent (MCA) reads the message from the transmission queue, a user-supplied message exit is invoked to process the reference message header. The exit can append to the reference message the bulk data identified by the MQRMH structure, before the MCA sends the message through the channel to the next queue manager.

At the receiving end, a message exit that waits for reference messages must exist. When a reference message is received, the exit must create the object from the bulk data that follows the MQRMH in the message, and then pass on the reference message without the bulk data. The reference message can later be retrieved by an application reading the reference message (without the bulk data) from a queue.

Normally, the MQRMH structure is all that is in the message. However, if the message is on a transmission queue, one or more additional headers precede the MQRMH structure.

A reference message can also be sent to a distribution list. In this case, the MQDH structure and its related records precede the MQRMH structure when the message is on a transmission queue.

Note: Do not send a reference message as a segmented message, because the message exit cannot process it correctly.

Data conversion: For data conversion purposes, converting the MQRMH structure includes conversion of the source environment data, source object name, destination environment data, and destination object name. Any other bytes within *StrucLength* bytes of the start of the structure are either discarded or have undefined values after data conversion. The bulk data is converted provided that all the following statements are true:

- The bulk data is present in the message when the data conversion is performed.
- The *Format* field in MQRMH has a value other than MQFMT_NONE.
- A user-written data-conversion exit exists with the format name specified.

Be aware, however, that usually the bulk data is not present in the message when the message is on a queue, and that as a result the bulk data is converted by the MQGMO_CONVERT option.

Fields for MQRMH:

The MQRMH structure contains the following fields; the fields are described in **alphabetical order**:

CodedCharSetId (MQLONG):

This specifies the character set identifier of the bulk data; it does not apply to character data in the MQRMH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

MQCCSI_INHERIT
Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

Do not use MQCCSI_INHERIT if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

DataLogicalLength (MQLONG):

The *DataLogicalLength* field specifies the length of the bulk data referenced by the MQRMH structure.

If the bulk data is actually present in the message, the data begins at an offset of *StrucLength* bytes from the start of the MQRMH structure. The length of the entire message minus *StrucLength* gives the length of the bulk data present.

If data is present in the message, *DataLogicalLength* specifies the amount of that data that is relevant. The normal case is for *DataLogicalLength* to have the same value as the length of data present in the message.

If the MQRMH structure represents the remaining data in the object (starting from the specified logical offset), you can use the value zero for *DataLogicalLength*, provided that the bulk data is not actually present in the message.

If no data is present, the end of MQRMH coincides with the end of the message.

The initial value of this field is 0.

DataLogicalOffset (MQLONG):

This field specifies the low offset of the bulk data from the start of the object of which the bulk data forms part. The offset of the bulk data from the start of the object is called the *logical offset*. This is not the physical offset of the bulk data from the start of the MQRMH structure; that offset is given by *StrucLength*.

To allow large objects to be sent using reference messages, the logical offset is divided into two fields, and the actual logical offset is given by the sum of these two fields:

- *DataLogicalOffset* represents the remainder obtained when the logical offset is divided by 1 000 000 000. It is thus a value in the range 0 through 999 999 999.
- *DataLogicalOffset2* represents the result obtained when the logical offset is divided by 1 000 000 000. It is thus the number of complete multiples of 1 000 000 000 that exist in the logical offset. The number of multiples is in the range 0 through 999 999 999.

The initial value of this field is 0.

DataLogicalOffset2 (MQLONG):

This field specifies the high offset of the bulk data from the start of the object of which the bulk data forms part. It is a value in the range 0 through 999 999 999. See *DataLogicalOffset* for details.

The initial value of this field is 0.

DestEnvLength (MQLONG):

This is the length of the destination environment data. If this field is zero, there is no destination environment data, and *DestEnvOffset* is ignored.

DestEnvOffset (MQLONG):

This field specifies the offset of the destination environment data from the start of the MQRMH structure. Destination environment data can be specified by the creator of the reference message, if that data is known to the creator. For example, on Windows the destination environment data might be the directory path of the object where the bulk data is to be stored. However, if the creator does not know the destination environment data, it is the responsibility of the user-supplied message exit to determine any environment information needed.

The length of the destination environment data is given by *DestEnvLength* ; if this length is zero, there is no destination environment data, and *DestEnvOffset* is ignored. If present, the destination environment data must reside completely within *StrucLength* bytes from the start of the structure.

Applications must not assume that the destination environment data is contiguous with any of the data addressed by the *SrcEnvOffset*, *SrcNameOffset*, and *DestNameOffset* fields.

The initial value of this field is 0.

DestNameLength (MQLONG):

The length of the destination object name. If this field is zero, there is no destination object name, and *DestNameOffset* is ignored.

DestNameOffset (MQLONG):

This field specifies the offset of the destination object name from the start of the MQRMH structure. The destination object name can be specified by the creator of the reference message, if that data is known to the creator. However, if the creator does not know the destination object name, it is the responsibility of the user-supplied message exit to identify the object to be created or modified.

The length of the destination object name is given by *DestNameLength* ; if this length is zero, there is no destination object name, and *DestNameOffset* is ignored. If present, the destination object name must reside completely within *StrucLength* bytes from the start of the structure.

Applications must not assume that the destination object name is contiguous with any of the data addressed by the *SrcEnvOffset*, *SrcNameOffset*, and *DestEnvOffset* fields.

The initial value of this field is 0.

Encoding (MQLONG):

This specifies the numeric encoding of the bulk data; it does not apply to numeric data in the MQRMH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is MQENC_NATIVE.

Flags (MQLONG):

These are reference message flags. The following flags are defined:

MQRMHF_LAST
: This flag indicates that the reference message represents or contains the last part of the referenced object.

MQRMHF_NOT_LAST
: Reference message does not contain or represent last part of object. MQRMHF_NOT_LAST aids program documentation. It is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

The initial value of this field is MQRMHF_NOT_LAST.

Format (MQCHAR8):

This specifies the format name of the bulk data.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *Format* field in MQMD.

The initial value of this field is MQFMT_NONE.

ObjectInstanceId (MQBYTE24):

Use this field to identify a specific instance of an object. If it is not needed, set it to the following value:

MQOII_NONE
No object instance identifier specified. The value is binary zero for the length of the field.

For the C programming language, the constant MQOII_NONE_ARRAY is also defined; this has the same value as MQOII_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_OBJECT_INSTANCE_ID_LENGTH. The initial value of this field is MQOII_NONE.

ObjectType (MQCHAR8):

This is a name that the message exit can use to recognize types of reference message that it supports. The name must conform to the same rules as the *Format* field, see "Format (MQCHAR8)."

The initial value of this field is 8 blanks.

SrcEnvLength (MQLONG):

The length of the source environment data. If this field is zero, there is no source environment data, and *SrcEnvOffset* is ignored.

The initial value of this field is 0.

SrcEnvOffset (MQLONG):

This field specifies the offset of the source environment data from the start of the MQRMH structure. Source environment data can be specified by the creator of the reference message, if that data is known to the creator. For example, on Windows the source environment data might be the directory path of the object containing the bulk data. However, if the creator does not know the source environment data, the user-supplied message exit must determine any environment information needed.

The length of the source environment data is given by *SrcEnvLength* ; if this length is zero, there is no source environment data, and *SrcEnvOffset* is ignored. If present, the source environment data must reside completely within *StrucLength* bytes from the start of the structure.

Applications must not assume that the environment data starts immediately after the last fixed field in the structure or that it is contiguous with any of the data addressed by the *SrcNameOffset*, *DestEnvOffset*, and *DestNameOffset* fields.

The initial value of this field is 0.

SrcNameLength (MQLONG):

The length of the source object name. If this field is zero, there is no source object name, and *SrcNameOffset* is ignored.

The initial value of this field is 0.

SrcNameOffset (MQLONG):

This field specifies the offset of the source object name from the start of the MQRMH structure. The source object name can be specified by the creator of the reference message, if that data is known to the creator. However, if the creator does not know the source object name, the user-supplied message exit must identify the object to be accessed.

The length of the source object name is given by *SrcNameLength* ; if this length is zero, there is no source object name, and *SrcNameOffset* is ignored. If present, the source object name must reside completely within *StrucLength* bytes from the start of the structure.

Applications must not assume that the source object name is contiguous with any of the data addressed by the *SrcEnvOffset*, *DestEnvOffset*, and *DestNameOffset* fields.

The initial value of this field is 0.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQRMH_STRUC_ID
Identifier for reference message header structure.

For the C programming language, the constant MQRMH_STRUC_ID_ARRAY is also defined; this has the same value as MQRMH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQRMH_STRUC_ID.

StrucLength (MQLONG):

The total length of MQRMH, including strings at the end of fixed fields, but not the bulk data.

The initial value of this field is zero.

Version (MQLONG):

The structure version number. The value must be:

MQRMH_VERSION_1
Version-1 reference message header structure.

The following constant specifies the version number of the current version:

MQRMH_CURRENT_VERSION
Current version of reference message header structure.

The initial value of this field is MQRMH_VERSION_1.

Initial values and language declarations for MQRMH:

Table 242. Initial values of fields in MQRMH for MQRMH

Field name	Name of constant	Value of constant
StrucId	MQRMH_STRUC_ID	'RMH¬'
Version	MQRMH_VERSION_1	1
StrucLength	None	0
Encoding	MQENC_NATIVE	Depends on environment
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
Flags	MQRMHF_NOT_LAST	0
ObjectType	None	Blanks
ObjectInstanceId	MQOII_NONE	Nulls
SrcEnvLength	None	0
SrcEnvOffset	None	0
SrcNameLength	None	0
SrcNameOffset	None	0
DestEnvLength	None	0
DestEnvOffset	None	0
DestNameLength	None	0
DestNameOffset	None	0
DataLogicalLength	None	0
DataLogicalOffset	None	0
DataLogicalOffset2	None	0

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQRMH_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure:

```
MQRMH MyRMH = {MQRMH_DEFAULT};
```

C declaration for MQRMH:

```
typedef struct tagMQRMH MQRMH;
struct tagMQRMH {
  MQCHAR4   StrucId;             /* Structure identifier */
  MQLONG    Version;             /* Structure version number */
  MQLONG    StrucLength;         /* Total length of MQRMH, including
                                    strings at end of fixed fields, but
                                    not the bulk data */
  MQLONG    Encoding;            /* Numeric encoding of bulk data */
  MQLONG    CodedCharSetId;      /* Character set identifier of bulk
                                    data */
  MQCHAR8   Format;              /* Format name of bulk data */
  MQLONG    Flags;               /* Reference message flags */
  MQCHAR8   ObjectType;          /* Object type */
  MQBYTE24  ObjectInstanceId;    /* Object instance identifier */
  MQLONG    SrcEnvLength;        /* Length of source environment data */
  MQLONG    SrcEnvOffset;        /* Offset of source environment data */
  MQLONG    SrcNameLength;       /* Length of source object name */
  MQLONG    SrcNameOffset;       /* Offset of source object name */
  MQLONG    DestEnvLength;       /* Length of destination environment
                                    data */
  MQLONG    DestEnvOffset;       /* Offset of destination environment
                                    data */
  MQLONG    DestNameLength;      /* Length of destination object name */
  MQLONG    DestNameOffset;      /* Offset of destination object name */
  MQLONG    DataLogicalLength;   /* Length of bulk data */
  MQLONG    DataLogicalOffset;   /* Low offset of bulk data */
  MQLONG    DataLogicalOffset2;  /* High offset of bulk data */
};
```

COBOL declaration for MQRMH:

```
**   MQRMH structure
  10 MQRMH.
**    Structure identifier
   15 MQRMH-STRUCID            PIC X(4).
**    Structure version number
   15 MQRMH-VERSION            PIC S9(9) BINARY.
**    Total length of MQRMH, including strings at end of fixed fields,
**    but not the bulk data
   15 MQRMH-STRUCLENGTH        PIC S9(9) BINARY.
**    Numeric encoding of bulk data
   15 MQRMH-ENCODING           PIC S9(9) BINARY.
**    Character set identifier of bulk data
   15 MQRMH-CODEDCHARSETID     PIC S9(9) BINARY.
**    Format name of bulk data
   15 MQRMH-FORMAT             PIC X(8).
**    Reference message flags
   15 MQRMH-FLAGS              PIC S9(9) BINARY.
**    Object type
   15 MQRMH-OBJECTTYPE         PIC X(8).
**    Object instance identifier
   15 MQRMH-OBJECTINSTANCEID   PIC X(24).
**    Length of source environment data
   15 MQRMH-SRCENVLENGTH       PIC S9(9) BINARY.
**    Offset of source environment data
   15 MQRMH-SRCENVOFFSET       PIC S9(9) BINARY.
**    Length of source object name
   15 MQRMH-SRCNAMELENGTH      PIC S9(9) BINARY.
**    Offset of source object name
   15 MQRMH-SRCNAMEOFFSET      PIC S9(9) BINARY.
**    Length of destination environment data
   15 MQRMH-DESTENVLENGTH      PIC S9(9) BINARY.
**    Offset of destination environment data
   15 MQRMH-DESTENVOFFSET      PIC S9(9) BINARY.
**    Length of destination object name
   15 MQRMH-DESTNAMELENGTH     PIC S9(9) BINARY.
```

```
**    Offset of destination object name
   15 MQRMH-DESTNAMEOFFSET     PIC S9(9) BINARY.
**    Length of bulk data
   15 MQRMH-DATALOGICALLENGTH  PIC S9(9) BINARY.
**    Low offset of bulk data
   15 MQRMH-DATALOGICALOFFSET  PIC S9(9) BINARY.
**    High offset of bulk data
   15 MQRMH-DATALOGICALOFFSET2 PIC S9(9) BINARY.
```

PL/I declaration for MQRMH:

```
dcl
 1 MQRMH based,
  3 StrucId            char(4),       /* Structure identifier */
  3 Version            fixed bin(31), /* Structure version number */
  3 StrucLength        fixed bin(31), /* Total length of MQRMH,
                                         including strings at end of
                                         fixed fields, but not the bulk
                                         data */
  3 Encoding           fixed bin(31), /* Numeric encoding of bulk
                                         data */
  3 CodedCharSetId     fixed bin(31), /* Character set identifier of
                                         bulk data */
  3 Format             char(8),       /* Format name of bulk data */
  3 Flags              fixed bin(31), /* Reference message flags */
  3 ObjectType         char(8),       /* Object type */
  3 ObjectInstanceId   char(24),      /* Object instance identifier */
  3 SrcEnvLength       fixed bin(31), /* Length of source environment
                                         data */
  3 SrcEnvOffset       fixed bin(31), /* Offset of source environment
                                         data */
  3 SrcNameLength      fixed bin(31), /* Length of source object name */
  3 SrcNameOffset      fixed bin(31), /* Offset of source object name */
  3 DestEnvLength      fixed bin(31), /* Length of destination
                                         environment data */
  3 DestEnvOffset      fixed bin(31), /* Offset of destination
                                         environment data */
  3 DestNameLength     fixed bin(31), /* Length of destination object
                                         name */
  3 DestNameOffset     fixed bin(31), /* Offset of destination object
                                         name */
  3 DataLogicalLength  fixed bin(31), /* Length of bulk data */
  3 DataLogicalOffset  fixed bin(31), /* Low offset of bulk data */
  3 DataLogicalOffset2 fixed bin(31); /* High offset of bulk data */
```

High Level Assembler declaration for MQRMH:

```
MQRMH                   DSECT
MQRMH_STRUCID           DS   CL4   Structure identifier
MQRMH_VERSION           DS   F     Structure version number
MQRMH_STRUCLENGTH       DS   F     Total length of MQRMH, including
*                                  strings at end of fixed fields, but
*                                  not the bulk data
MQRMH_ENCODING          DS   F     Numeric encoding of bulk data
MQRMH_CODEDCHARSETID    DS   F     Character set identifier of bulk
*                                  data
MQRMH_FORMAT            DS   CL8   Format name of bulk data
MQRMH_FLAGS             DS   F     Reference message flags
MQRMH_OBJECTTYPE        DS   CL8   Object type
MQRMH_OBJECTINSTANCEID  DS   XL24  Object instance identifier
MQRMH_SRCENVLENGTH      DS   F     Length of source environment data
MQRMH_SRCENVOFFSET      DS   F     Offset of source environment data
MQRMH_SRCNAMELENGTH     DS   F     Length of source object name
MQRMH_SRCNAMEOFFSET     DS   F     Offset of source object name
MQRMH_DESTENVLENGTH     DS   F     Length of destination environment
*                                  data
MQRMH_DESTENVOFFSET     DS   F     Offset of destination environment
*                                  data
```

```
MQRMH_DESTNAMELENGTH      DS   F     Length of destination object name
MQRMH_DESTNAMEOFFSET      DS   F     Offset of destination object name
MQRMH_DATALOGICALLENGTH   DS   F     Length of bulk data
MQRMH_DATALOGICALOFFSET   DS   F     Low offset of bulk data
MQRMH_DATALOGICALOFFSET2  DS   F     High offset of bulk data
*
MQRMH_LENGTH              EQU  *-MQRMH
                          ORG  MQRMH
MQRMH_AREA                DS   CL(MQRMH_LENGTH)
```

Visual Basic declaration for MQRMH:

```
Type MQRMH
  StrucId            As String*4 'Structure identifier'
  Version            As Long     'Structure version number'
  StrucLength        As Long     'Total length of MQRMH, including'
                                 'strings at end of fixed fields, but'
                                 'not the bulk data'
  Encoding           As Long     'Numeric encoding of bulk data'
  CodedCharSetId     As Long     'Character set identifier of bulk data'
  Format             As String*8 'Format name of bulk data'
  Flags              As Long     'Reference message flags'
  ObjectType         As String*8 'Object type'
  ObjectInstanceId   As MQBYTE24 'Object instance identifier'
  SrcEnvLength       As Long     'Length of source environment data'
  SrcEnvOffset       As Long     'Offset of source environment data'
  SrcNameLength      As Long     'Length of source object name'
  SrcNameOffset      As Long     'Offset of source object name'
  DestEnvLength      As Long     'Length of destination environment'
                                 'data'
  DestEnvOffset      As Long     'Offset of destination environment'
                                 'data'
  DestNameLength     As Long     'Length of destination object name'
  DestNameOffset     As Long     'Offset of destination object name'
  DataLogicalLength  As Long     'Length of bulk data'
  DataLogicalOffset  As Long     'Low offset of bulk data'
  DataLogicalOffset2 As Long     'High offset of bulk data'
End Type
```

MQRR - Response record:

The following table summarizes the fields in the structure.

Table 243. Fields in MQRR

Field	Description	Topic
CompCode	Completion code for queue	CompCode
Reason	Reason code for queue	Reason

Overview for MQRR:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ clients connected to these systems.

Purpose: Use the MQRR structure to receive the completion code and reason code resulting from the open or put operation for a single destination queue, when the destination is a distribution list. MQRR is an output structure for the MQOPEN, MQPUT, and MQPUT1 calls.

Character set and encoding: Data in MQRR must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Usage: By providing an array of these structures on the MQOPEN and MQPUT calls, or on the MQPUT1 call, you can determine the completion codes and reason codes for all the queues in a distribution list when the outcome of the call is mixed, that is, when the call succeeds for some queues in the list but fails for others. Reason code MQRC_MULTIPLE_REASONS from the call indicates that the response records (if provided by the application) have been set by the queue manager.

Fields for MQRR:

The MQRR structure contains the following fields; the fields are described in **alphabetical order**:

CompCode (MQLONG):

This is the completion code resulting from the open or put operation for the queue with the name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call.

This is always an output field. The initial value of this field is MQCC_OK.

Reason (MQLONG):

This is the reason code resulting from the open or put operation for the queue with the name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call.

This is always an output field. The initial value of this field is MQRC_NONE.

Initial values and language declarations for MQRR:

Table 244. Initial values of fields in MQRR for MQRR

Field name	Name of constant	Value of constant
CompCode	MQCC_OK	0
Reason	MQRC_NONE	0

Notes:

1. In the C programming language, the macro variable MQRR_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure:

```
MQRR MyRR = {MQRR_DEFAULT};
```

C declaration:

```
typedef struct tagMQRR MQRR;
struct tagMQRR {
  MQLONG  CompCode;  /* Completion code for queue */
  MQLONG  Reason;    /* Reason code for queue */
};
```

COBOL declaration:

```
**   MQRR structure
  10 MQRR.
**    Completion code for queue
   15 MQRR-COMPCODE PIC S9(9) BINARY.
**    Reason code for queue
   15 MQRR-REASON   PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQRR based,
  3 CompCode fixed bin(31), /* Completion code for queue */
  3 Reason   fixed bin(31); /* Reason code for queue */
```

Visual Basic declaration:

```
Type MQRR
  CompCode As Long 'Completion code for queue'
  Reason   As Long 'Reason code for queue'
End Type
```

MQSCO - SSL/TLS configuration options:

The following table summarizes the fields in the structure.

Table 245. Fields in MQSCO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
KeyRepository	Location of key repository	KeyRepository
CryptoHardware	Details of cryptographic hardware	CryptoHardware
AuthInfoRecCount	Number of MQAIR records present	AuthInfoRecCount
AuthInfoRecOffset	Offset of first MQAIR record from start of MQSCO	AuthInfoRecOffset
AuthInfoRecPtr	Address of first MQAIR record	AuthInfoRecPtr
Note: The following two fields are ignored if *Version* is less than MQSCO_VERSION_2.		
KeyResetCount	TLS secret key reset count	KeyResetCount
FipsRequired	Use FIPS-certified cryptographic algorithms in IBM MQ	"FipsRequired (MQLONG)" on page 2545
Note: The following field is ignored if *Version* is less than MQSCO_VERSION_3.		
EncryptionPolicySuiteB	Use only Suite B cryptographic algorithms	EncryptionPolicySuiteB
Note: The following field is ignored if *Version* is less than MQSCO_VERSION_4.		
CertificateValPolicy	Certificate validation policy	CertificateValPolicy
Note: The following field is ignored if *Version* is less than MQSCO_VERSION_5.		
CertificateLabel	Details the certificate label that is being used.	CertificateLabel

Related reference:

"MQCNO - Connect options" on page 2276
The following table summarizes the fields in the structure.

"Overview for MQSCO"
Availability: AIX, HP-UX, IBM i, Solaris, Linux and Windows clients.

"Fields for MQSCO"

"Initial values and language declarations for MQSCO" on page 2547

Overview for MQSCO:

Availability: AIX, HP-UX, IBM i, Solaris, Linux and Windows clients.

Purpose: The MQSCO structure (in conjunction with the TLS fields in the MQCD structure) allows an application running as an IBM MQ MQI client to specify configuration options that control the use of TLS for the client connection when the channel protocol is TCP/IP. The structure is an input parameter on the MQCONNX call.

If the channel protocol for the client channel is not TCP/IP, the MQSCO structure is ignored.

Character set and encoding: Data in MQSCO must be in the character set given by the **CodedCharSetId** queue manager attribute, and encoding of the local queue manager given by MQENC_NATIVE.

Fields for MQSCO:

The MQSCO structure contains the following fields; the fields are described in **alphabetical order**:

AuthInfoRecCount (MQLONG):

This is the number of authentication information (MQAIR) records addressed by the *AuthInfoRecPtr* or *AuthInfoRecOffset* fields. For more information, see"MQAIR - Authentication information record" on page 2224. The value must be zero or greater. If the value is not valid, the call fails with reason code MQRC_AUTH_INFO_REC_COUNT_ERROR.

This is an input field. The initial value of this field is 0.

AuthInfoRecOffset (MQLONG):

This is the offset in bytes of the first authentication information record from the start of the MQSCO structure. The offset can be positive or negative. The field is ignored if *AuthInfoRecCount* is zero.

You can use either *AuthInfoRecOffset* or *AuthInfoRecPtr* to specify the MQAIR records, but not both; see the description of the *AuthInfoRecPtr* field for details.

This is an input field. The initial value of this field is 0.

AuthInfoRecPtr (PMQAIR):

This is the address of the first authentication information record. The field is ignored if *AuthInfoRecCount* is zero.

You can provide the array of MQAIR records in one of two ways:

- By using the pointer field *AuthInfoRecPtr*

  In this case, the application can declare an array of MQAIR records that is separate from the MQSCO structure, and set *AuthInfoRecPtr* to the address of the array.

  Consider using *AuthInfoRecPtr* for programming languages that support the pointer data type in a fashion that is portable to different environments (for example, the C programming language).
- By using the offset field *AuthInfoRecOffset*

  In this case, the application must declare a compound structure containing an MQSCO followed by the array of MQAIR records, and set *AuthInfoRecOffset* to the offset of the first record in the array from the start of the MQSCO structure. Ensure that this value is correct, and has a value that can be accommodated within an MQLONG (the most restrictive programming language is COBOL, for which the valid range is -999 999 999 through +999 999 999).

  Consider using *AuthInfoRecOffset* for programming languages that do not support the pointer data type, or that implement the pointer data type in a fashion that is not portable to different environments (for example, the COBOL programming language).

Whatever technique you choose, only one of *AuthInfoRecPtr* and *AuthInfoRecOffset* can be used; the call fails with reason code MQRC_AUTH_INFO_REC_ERROR if both are nonzero.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

CertificateLabel (MQCHAR64):

This field gives details of the certificate label being used.

IBM MQ initializes the default value for the *CertificateLabel* field as blanks.

This is interpreted at runtime as the default value, and is backwards compatible.

For example, specifying a MQSCO version less than 5.0, or using the default value of blanks for the *CertificateLabel* field, uses the preexisting default value of `ibmwebspheremq`*user_id*.

CertificateValPolicy (MQLONG):

This field specifies what type of certificate validation policy is used. The field can be set to one of the following values:

MQ_CERT_VAL_POLICY_ANY
Apply each of the certificate validation policies supported by the secure sockets library. Accept the certificate chain if any of the policies considers the certificate chain valid.

MQ_CERT_VAL_POLICY_RFC5280
Apply only the RFC5280 compliant certificate validation policy. This setting provides stricter validation than the ANY setting, but rejects some older digital certificates.

The initial value of this field is MQ_CERT_VAL_POLICY_ANY

CryptoHardware (MQCHAR256):

This field gives configuration details for cryptographic hardware connected to the client system.

Set the field to a string of the following format, or leave it blank or null:

```
GSK_PKCS11=the PKCS #11 driver path and file name;the PKCS #11
token label;the PKCS #11 token password;symmetric cipher setting;
```

To use cryptographic hardware which conforms to the PKCS #11 interface, for example, the IBM 4960 or IBM 4764, the PKCS #11 driver path, PKCS #11 token label, and PKCS #11 token password strings must be specified, each terminated by a semi-colon.

The PKCS #11 driver path is an absolute path to the shared library providing support for the PKCS #11 card. The PKCS #11 driver file name is the name of the shared library. An example of the value required for the PKCS #11 path and file name is:

```
/usr/lib/pkcs11/PKCS11_API.so
```

The PKCS #11 token label must be entirely in lowercase. If you have configured your hardware with a mixed case or uppercase token label, re-configure it with this lowercase label.

If no cryptographic hardware configuration is required, set the field to blank or null.

If the value is shorter than the length of the field, terminate the value with a null character, or pad it with blanks to the length of the field. If the value is not valid, or leads to a failure when used to configure the cryptographic hardware, the call fails with reason code MQRC_CRYPTO_HARDWARE_ERROR.

This is an input field. The length of this field is given by MQ_SSL_CRYPTO_HARDWARE_LENGTH. The initial value of this field is the null string in C, and blank characters in other programming languages.

EncryptionPolicySuiteB(MQLONG):

This field Specifies whether Suite B compliant cryptography is used and what level of strength is employed. The value can be one or more of:

- MQ_SUITE_B_NONE
  Suite B compliant cryptography is not used.
- MQ_SUITE_B_128_BIT
  Suite B 128-bit strength security is used.
- MQ_SUITE_B_192_BIT
  Suite B 192-bit strength security is used.

Note: Using the MQ_SUITE_B_NONE with any other value in this field is invalid.

FipsRequired (MQLONG):

IBM MQ can be configured with cryptographic hardware so that the cryptography modules used are those provided by the hardware product; these can be FIPS-certified to a particular level depending on the cryptographic hardware product in use. Use this field to specify that only FIPS-certified algorithms are used if the cryptography is provided in IBM MQ-provided software.

When IBM MQ is installed an implementation of TLS cryptography is also installed which provides some FIPS-certified modules.

The values can be:

MQSSL_FIPS_NO
This is the default value. When set to this value:

- Any CipherSpec supported on a particular platform can be used.
- If run without use of cryptographic hardware, the following CipherSpecs run using FIPS 140-2 certified cryptography on the IBM MQ platforms:
  - TLS_RSA_WITH_3DES_EDE_CBC_SHA
  - TLS_RSA_WITH_AES_128_CBC_SHA
  - TLS_RSA_WITH_AES_256_CBC_SHA

MQSSL_FIPS_YES
When set to this value, unless you are using cryptographic hardware to perform the cryptography, you can be sure that

- Only FIPS-certified cryptographic algorithms can be used in the CipherSpec applying to this client connection.
- Inbound and outbound TLS channel connections only succeed if one of the following Cipher Specs are used:
  - TLS_RSA_WITH_3DES_EDE_CBC_SHA
  - TLS_RSA_WITH_AES_128_CBC_SHA
  - TLS_RSA_WITH_AES_256_CBC_SHA

Notes:

1. CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA is deprecated.
2. Where possible, if FIPS-only CipherSpecs are configured then the MQI client rejects connections which specify a non-FIPS CipherSpec with MQRC_SSL_INITIALIZATION_ERROR. IBM MQ does not guarantee to reject all such connections and it is your responsibility to determine whether your IBM MQ configuration is FIPS-compliant.

KeyRepository (MQCHAR256):

This field is relevant only for IBM MQ MQI clients running on UNIX and Windows systems. It specifies the location of the key database file in which keys and certificates are stored. The key database file must have a file name of the form `zzz.kdb`, where `zzz` is user-selectable. The *KeyRepository* field contains the path to this file, along with the file name stem (all characters in the file name up to but not including the final `.kdb`). The `.kdb` file suffix is added automatically.

Each key database file has an associated *password stash file*. This holds encoded passwords that are used to allow programmatic access to the key database. The password stash file must reside in the same directory and have the same file stem as the key database, and must end with the suffix `.sth`.

For example, if the *KeyRepository* field has the value `/xxx/yyy/key`, the key database file must be `/xxx/yyy/key.kdb`, and the password stash file must be `/xxx/yyy/key.sth`, where `xxx` and `yyy` represent directory names.

If the value is shorter than the length of the field, terminate the value with a null character, or pad it with blanks to the length of the field. The value is not checked; if there is an error in accessing the key repository, the call fails with reason code MQRC_KEY_REPOSITORY_ERROR.

To run a TLS connection from an IBM MQ MQI client, set *KeyRepository* to a valid key database file name.

This is an input field. The length of this field is given by MQ_SSL_KEY_REPOSITORY_LENGTH. The initial value of this field is the null string in C, and blank characters in other programming languages.

KeyResetCount (MQLONG):

This represents the total number of unencrypted bytes sent and received within a TLS conversation before the secret key is renegotiated.

The number of bytes includes control information sent by the MCA.

If you specify a TLS secret key reset count in the range 1 byte through 32 KB, TLS channels will use a secret key reset count of 32 KB. This is to avoid the processing cost of excessive key resets which would occur for small TLS secret key reset values.

This is an input field. The value is a number in the range 0 through 999 999 999, with a default value of 0. Use a value of 0 to indicate that secret keys are never renegotiated.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQSCO_STRUC_ID
Identifier for TLS configuration options structure.

For the C programming language, the constant MQSCO_STRUC_ID_ARRAY is also defined; this has the same value as MQSCO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQSCO_STRUC_ID.

Version (MQLONG):

This is the structure version number; the value must be:

MQSCO_VERSION_1
Version-1 TLS configuration options structure.

MQSCO_VERSION_2
Version-2 TLS configuration options structure.

MQSCO_VERSION_3
Version-3 TLS configuration options structure.

MQSCO_VERSION_4
Version-4 TLS configuration options structure.

MQSCO_VERSION_5
Version-5 TLS configuration options structure.

The following constant specifies the version number of the current version:

MQSCO_CURRENT_VERSION
Current version of TLS configuration options structure.

This is always an input field. The initial value of this field is MQSCO_VERSION_1.

Initial values and language declarations for MQSCO:

Table 246. Initial values of fields in MQSCO

Field name	Name of constant	Value of constant
StrucId	MQSCO_STRUC_ID	'SCO¬'
Version	MQSCO_CURRENT_VERSION	1
KeyRepository	None	Null string or blanks
CryptoHardware	None	Null string or blanks
AuthInfoRecCount	None	0
AuthInfoRecOffset	None	0
AuthInfoRecPtr	None	Null pointer or null bytes
KeyResetCount	MQSCO_RESET_COUNT_DEFAULT	0
FipsRequired	MQSSL_FIPS_NO	0
EncryptionPolicySuiteB	MQ_SUITE_B_NONE, MQ_SUITE_B_NOT_AVAILABLE, MQ_SUITE_B_NOT_AVAILABLE, MQ_SUITE_B_NOT_AVAILABLE	1, 0, 0, 0
CertificateValPolicy	MQ_CERT_VAL_POLICY_DEFAULT	0

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQSCO_DEFAULT contains the values listed in the table. Use it in the following way to provide initial values for the fields in the structure:

```
MQSCO MySCO = {MQSCO_DEFAULT};
```

C declaration for MQSCO:

```
typedef struct tagMQSCO MQSCO;
struct tagMQSCO {
   MQCHAR4    StrucId;                   /* Structure identifier */
   MQLONG     Version;                   /* Structure version number */
   MQCHAR256  KeyRepository;             /* Location of TLS key */
                                         /* repository */
   MQCHAR256  CryptoHardware;            /* Cryptographic hardware */
                                         /* configuration string */
   MQLONG     AuthInfoRecCount;          /* Number of MQAIR records */
                                         /* present */
   MQLONG     AuthInfoRecOffset;         /* Offset of first MQAIR */
                                         /* record from start of */
                                         /* MQSCO structure */
   PMQAIR     AuthInfoRecPtr;            /* Address of first MQAIR */
                                         /* record */
/* Ver:1 */
   MQLONG     KeyResetCount;             /* Number of unencrypted */
                                         /* bytes sent/received */
                                         /* before secret key is */
                                         /* reset */
   MQLONG     FipsRequired;              /* Using FIPS-certified */
/* Ver:2 */
                                         /* algorithms */
   MQLONG     EncryptionPolicySuiteB[4]; /* Use only Suite B */
/* Ver:3 */
                                         /* cryptographic algorithms */
   MQLONG     CertificateValPolicy;      /* Certificate validation */
                                         /* policy */
/* Ver:4 */
   MQCHAR64   CertificateLabel;          /* Certificate label */
/* Ver:5 */
```

COBOL declaration for MQSCO:

```
**   MQSCO structure
  10 MQSCO.
**    Structure identifier
   15 MQSCO-STRUCID                  PIC X(4).
**    Structure version number
   15 MQSCO-VERSION                  PIC S9(9) BINARY.
**    Location of TLS key repository
   15 MQSCO-KEYREPOSITORY            PIC X(256).
**    Cryptographic hardware configuration string
   15 MQSCO-CRYPTOHARDWARE           PIC X(256).
**    Number of MQAIR records present
   15 MQSCO-AUTHINFORECCOUNT         PIC S9(9) BINARY.
**    Offset of first MQAIR record from start of MQSCO structure
   15 MQSCO-AUTHINFORECOFFSET        PIC S9(9) BINARY.
** Address of first MQAIR record
    15  MQSCO-AUTHINFORECPTR POINTER.
** Version 1 **
** Number of unencrypted bytes sent/received before secret key is
** reset
15  MQSCO-KEYRESETCOUNT PIC S9(9) BINARY.
** Using FIPS-certified algorithms
15  MQSCO-FIPSREQUIRED PIC S9(9) BINARY.
** Version 2 **
** Use only Suite B cryptographic algorithms
15  MQSCO-ENCRYPTIONPOLICYSUITEB PIC S9(9) BINARY OCCURS 4.
** Version 3 **
** Certificate validation policy setting
15  MQSCO-CERTIFICATEVALPOLICY PIC S9(9) BINARY.
** Version 4
```

PL/I declaration for MQSCO:

```
dcl
 1 MQSCO based,
  3 StrucId                   char(4),       /* Structure identifier */
  3 Version                   fixed bin(31), /* Structure version number */
  3 KeyRepository             char(256),     /* Location of TLS key
                                                repository */
  3 CryptoHardware            char(256),     /* Cryptographic hardware
                                                configuration string */
  3 AuthInfoRecCount          fixed bin(31), /* Number of MQAIR records
                                                present */
  3 AuthInfoRecOffset         fixed bin(31), /* Offset of first MQAIR record
                                                from start of MQSCO structure */
  3 AuthInfoRecPtr            pointer,       /* Address of first MQAIR record */
  3 KeyResetCount             fixed bin(31), /* Key reset count */
/* Version 1 */
  3 FipsRequired              fixed bin(31), /* FIPS required */
/* Version 2 */
  3 EncryptionPolicySuiteB (4) fixed bin(31), /* Suite B encryption policy */
/* Version 3 */
  3 CertificateValPolicy      fixed bin(31); /* Certificate validation policy */
/* Version 4 */
```

Visual Basic declaration for MQSCO:

```
Type MQSCO
  StrucId           As String*4   'Structure identifier'
  Version           As Long       'Structure version number'
  KeyRepository     As String*256 'Location of TLS key repository'
  CryptoHardware    As String*256 'Cryptographic hardware configuration'
                                  'string'
  AuthInfoRecCount  As Long       'Number of MQAIR records present'
  AuthInfoRecOffset As Long       'Offset of first MQAIR record from'
                                  'start of MQSCO structure'
  AuthInfoRecPtr    As MQPTR      'Address of first MQAIR record'
  KeyResetCount     As Long       'Number of unencrypted bytes sent/received before secret key is reset'
'Version 1'
  FipsRequired      As Long       'Mandatory FIPS CipherSpecs?'
'Version 2'
End Type
```

MQSD - Subscription descriptor:

The following table summarizes the fields in the structure.

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options	Options
ObjectName	Object name	ObjectName
AlternateUserId	Alternate User Id	AlternateUserId
AlternateSecurityId	Alternate Security ID	AlternateSecurityId
SubExpiry	Subscription Expiry	SubExpiry
ObjectString	Object String	ObjectString
SubName	Subscription Name	SubName
SubUserData	Subscription user data	SubUserData
SubCorrelId	Subscription Correlation ID	SubCorrelId
PubPriority	Publication priority	PubPriority

Field	Description	Topic
PubAccountingToken	Publication Accounting Token	PubAccountingToken
PubAppIdentityData	Publication application identity data	PubAppIdentityData
SelectionString	String providing selection criteria	SelectionString
SubLevel	Subscription Level	SubLevel
ResObjectString	Long object name	ResObjectString

Overview for MQSD:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, z/OS, plus IBM MQ MQI clients connected to these systems.

Purpose: The MQSD structure is used to specify details about the subscription being made.

The structure is an input/output parameter on the MQSUB call. For more information, see MQSUB usage notes.

Managed subscriptions: If an application has no specific need to use a particular queue as the destination for those publications that match its subscription, it can use the managed subscription feature. If an application elects to use a managed subscription, the queue manager informs the subscriber about the destination where published messages are sent, by providing an object handle as an output from the MQSUB call. For more information, see Hobj (MQHOBJ) - input/output.

When the subscription is removed, the queue manager also undertakes to clean up messages that have not been retrieved from the managed destination, in the following situations:

- When the subscription is removed - by use of MQCLOSE with MQCO_REMOVE_SUB - and the managed Hobj is closed.
- By implicit means when the connection is lost to an application using a non-durable subscription (MQSO_NON_DURABLE)
- By expiration when a subscription is removed because it has expired and the managed Hobj is closed.

You must use managed subscriptions with non-durable subscriptions, so that this clean up can occur, and so that messages for closed non-durable subscriptions do not take up space in your queue manager. Durable subscriptions can also use managed destinations.

Version: The current version of MQSD is MQSD_VERSION_1.

Character set and encoding: Data in MQSD must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Fields for MQSD:

The MQSD structure contains the following fields; the fields are described in alphabetical order:

AlternateSecurityId (MQBYTE40):

This is a security identifier that is passed with the AlternateUserId to the authorization service to allow appropriate authorization checks to be performed.

AlternateSecurityId is used only if MQSO_ALTERNATE_USER_AUTHORITY is specified, and the AlternateUserId field is not entirely blank up to the first null character or the end of the field.

On return from an MQSUB call using MQSO_RESUME, this field is unchanged.

See the description of "AlternateSecurityId (MQBYTE40)" on page 2454 in the MQOD data type for more information.

AlternateUserId (MQCHAR12):

If you specify MQSO_ALTERNATE_USER_AUTHORITY, this field contains an alternative user identifier that is used to check the authorization for the subscription and for output to the destination queue (specified in the **Hobj** parameter of the MQSUB call), in place of the user identifier that the application is currently running under.

If successful, the user identifier specified in this field is recorded as the subscription owning user identifier in place of the user identifier that the application is currently running under.

If MQSO_ALTERNATE_USER_AUTHORITY is specified and this field is entirely blank up to the first null character or the end of the field, the subscription can succeed only if no user authorization is needed to subscribe to this topic with the options specified or the destination queue for output.

If MQSO_ALTERNATE_USER_AUTHORITY is not specified, this field is ignored.

The following differences exist in the environments indicated:

- On z/OS, only the first 8 characters of AlternateUserId are used to check the authorization for the subscription. However, the current user identifier must be authorized to specify this particular alternative user identifier; all 12 characters of the alternative user identifier are used for this check. The user identifier must contain only characters allowed by the external security manager.

On return from an MQSUB call using MQSO_RESUME, this field is unchanged.

This is an input field. The length of this field is given by MQ_USER_ID_LENGTH. The initial value of this field is the null string in C, and 12 blank characters in other programming languages.

ObjectName (MQCHAR48):

This is the name of the topic object as defined on the local queue manager.

The name can contain the following characters:

- Uppercase alphabetic characters (A through Z)
- Lowercase alphabetic characters (a through z)
- Numeric digits (0 through 9)
- Period (.), forward slash (/), underscore (_), percent (%)

The name must not contain leading or embedded blanks, but can contain trailing blanks. Use a null character to indicate the end of significant data in the name; the null and any characters following it are treated as blanks. The following restrictions apply in the environments indicated:

- On systems that use EBCDIC Katakana, lowercase characters cannot be used.
- On z/OS:
  - Avoid names that begin or end with an underscore; they cannot be processed by the operations and control panels.
  - The percent character has a special meaning to RACF. If RACF is used as the external security manager, names must not contain the percent. If they do, those names are not included in any security checks when RACF generic profiles are used.
- On IBM i, names containing lowercase characters, forward slash, or percent, must be enclosed in quotation marks when specified on commands. These quotation marks must not be specified for names that occur as fields in structures or as parameters on calls.

The *ObjectName* is used to form the full topic name.

The full topic name can be built from two different fields: *ObjectName* and *ObjectString*. For details of how these two fields are used, see "Using topic strings" on page 2566.

If the object identified by the *ObjectName* field cannot be found, the call fails with reason code MQRC_UNKNOWN_OBJECT_NAME even if there is a string specified in *ObjectString*.

On return from an MQSUB call using the MQSO_RESUME option this field is unchanged.

The length of this field is given by MQ_TOPIC_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

If altering an existing subscription using the MQSO_ALTER option, the name of the topic object subscribed to cannot be changed. This field and the *ObjectString* field can be omitted. If they are provided, they must resolve to the same full topic name. If they do not, the call fails with MQRC_TOPIC_NOT_ALTERABLE.

ObjectString (MQCHARV):

This is the long object name to be used.

The *ObjectString* is used to form the Full topic name.

The full topic name can be built from two different fields: *ObjectName* and *ObjectString*. For details of how these two fields are used, see "Using topic strings" on page 2566.

The maximum length of *ObjectString* is 10240.

If *ObjectString* is not specified correctly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code MQRC_OBJECT_STRING_ERROR.

This is an input field. The initial values of the fields in this structure are the same as those in the MQCHARV structure.

If there are wildcards in the *ObjectString* the interpretation of those wildcards can be controlled using the Wildcard options specified in the Options field of the MQSD.

On return from an MQSUB call using the MQSO_RESUME option this field is unchanged. The full topic name used is returned in the *ResObjectString* field if a buffer is provided.

If altering an existing subscription using the MQSO_ALTER option, the long name of the topic object subscribed to cannot be changed. This field and the *ObjectName* field can be omitted. If they are provided they must resolve to the same full topic name or the call fails with MQRC_TOPIC_NOT_ALTERABLE.

Options (MQLONG):

This provides options to control the action of the MQSUB call.

You must specify at least one of the following options:

- MQSO_ALTER
- MQSO_RESUME
- MQSO_CREATE

To specify more than one option, either add the values together (do not add the same constant more than once), or combine the values using the bitwise OR operation (if the programming language supports bit operations).

Combinations that are not valid are noted in this topic; any other combinations are valid.

Access or creation options: Access and creation options control whether a subscription is created, or whether an existing subscription is returned or altered. You must specify at least one of these options. The table displays valid combinations of access and creation options.

Combination of options	Notes
MQSO_CREATE	Creates a subscription if one does not exist. This combination fails if the subscription exists.
MQSO_RESUME	Resumes an existing subscription. This combination fails if no subscription exists.
MQSO_CREATE + MQSO_RESUME	Creates a subscription if one does not exist and resumes a matching one, if it does exist. This combination is useful when it is used in an application that is run a number of times.
MQSO_ALTER (see note)	Resumes an existing subscription, altering any fields to match that specified in the MQSD. This combination fails if no subscription exists.
MQSO_CREATE + MQSO_ALTER (see note)	Creates a subscription if one does not exist and resumes a matching one, if it does exist, altering any fields to match that specified in the MQSD. This combination is useful combination when used in an application that wants to ensure that its subscription is in a certain state before proceeding.

Note:

Options specifying MQSO_ALTER can also specify MQSO_RESUME, but this combination has no additional effect to specifying MQSO_ALTER alone. MQSO_ALTER implies MQSO_RESUME, because calling MQSUB to alter a subscription implies that the subscription will also be resumed. The opposite is not true, however: resuming a subscription does not imply it is to be altered.

MQSO_CREATE

Create a new subscription for the topic specified. If a subscription using the same *SubName* exists, the call fails with MQRC_SUB_ALREADY_EXISTS. This failure can be avoided by combining the MQSO_CREATE option with MQSO_RESUME. The *SubName* is not always necessary. For more details, see the description of that field.

Combining MQSO_CREATE with MQSO_RESUME returns a handle to a pre-existing subscription for the specified *SubName* if one is found; if there is no existing subscription, a new one is created using all the fields provided in the MQSD.

MQSO_CREATE can also be combined with MQSO_ALTER to similar effect.

MQSO_RESUME

Return a handle to a pre-existing subscription which matches that specified by *SubName*. No changes are made to the matching subscriptions attributes and they are returned on output in the MQSD structure. Only the following MQSD fields are used: StrucId, Version, Options, AlternateUserId and AlternateSecurityId, and SubName.

The call fails with reason code MQRC_NO_SUBSCRIPTION if a subscription does not exist matching the full subscription name. This failure can be avoided by combining the MQSO_CREATE option with MQSO_RESUME.

The user ID of the subscription is the user ID that created the subscription, or if it has been later altered by a different user ID, it is the user ID of the most recent successful alteration. If an AlternateUserId is used, and use of alternate user IDs is allowed for that user, the alternate user ID is recorded as the user ID that created the subscription instead of the user ID under which the subscription was made.

If a matching subscription exists that was created without the MQSO_ANY_USERID option, and the user ID of the subscription is different from that of the application requesting a handle to the subscription, the call fails with reason code MQRC_IDENTITY_MISMATCH.

If a matching subscription exists and is currently in use, the call fails with MQRC_SUBSCRIPTION_IN_USE.

If the subscription named in SubName is not a valid subscription to resume or alter from an application, the call fails with MQRC_INVALID_SUBSCRIPTION.

MQSO_RESUME is implied by MQSO_ALTER so you do not need to combine it with that option. However, combining the two options does not cause an error.

MQSO_ALTER

Return a handle to a pre-existing subscription with the full subscription name matching that specified by the name in *SubName*. Any attributes of the subscription that are different from that specified in the MQSD are altered in the subscription unless alteration is disallowed for that attribute. Details are noted in the description of each attribute and are summarized in the following table. If you try to alter an attribute that cannot be changed, or to alter a subscription that has set the MQSO_IMMUTABLE option, the call fails with the reason code shown in the following table.

The call fails with reason code MQRC_NO_SUBSCRIPTION if a subscription matching the full subscription name does not exist. You can avoid this failure by combining the MQSO_CREATE option with MQSO_ALTER.

Combining MQSO_CREATE with MQSO_ALTER returns a handle to a pre-existing subscription for the specified *SubName* if one is found; if there is no existing subscription, a new one is created using all the fields provided in the MQSD.

The user ID of the subscription is the user ID that created the subscription, or if it is later altered by a different user ID, it is the user ID of the most recent, successful alteration. If an AlternateUserId is used, and use of alternate user IDs is allowed for that user, then the alternate user ID is recorded as the user ID that created the subscription instead of the user ID under which the subscription was made.

If a matching subscription exists that was created without the option MQSO_ANY_USERID and the user ID of the subscription is different from that of the application requesting a handle to the subscription, the call fails with reason code MQRC_IDENTITY_MISMATCH.

If a matching subscription exists and is currently in use, the call fails with MQRC_SUBSCRIPTION_IN_USE.

If the subscription named in SubName is not a valid subscription to resume or alter from an application, the call fails with MQRC_INVALID_SUBSCRIPTION.

The following table shows the ability of MQSO_ALTER to alter attribute values in MQSD and MQSUB.

Table 247. Attributes in MQSD and MQSUB that can be altered

Data type descriptor or function call	Field name	Can this attribute be altered using MQSO_ALTER	Reason code
MQSD	Durability options	No	MQRC_DURABILITY_NOT_ALTERABLE
MQSD	Destination Options	Yes	None
MQSD	Registration options	Yes (see note 1 on page 2556)	MQRC_GROUPING_NOT_ALTERABLE if you try to alter MQSO_GROUP_SUB
MQSD	Publication options	Yes (see note 2 on page 2556)	None
MQSD	Wildcard options	No	MQRC_TOPIC_NOT_ALTERABLE
MQSD	Other options	No (see note 3 on page 2556)	None
MQSD	ObjectName	No	MQRC_TOPIC_NOT_ALTERABLE
MQSD	AlternateUserId	No (see note 4 on page 2556)	None

Table 247. Attributes in MQSD and MQSUB that can be altered (continued)

Data type descriptor or function call	Field name	Can this attribute be altered using MQSO_ALTER	Reason code
MQSD	AlternateSecurityId	No (see note 4)	None
MQSD	SubExpiry	Yes	None
MQSD	ObjectString	No	MQRC_TOPIC_NOT_ALTERABLE
MQSD	SubName	No (see note 5)	None
MQSD	SubUserData	Yes	None
MQSD	SubCorrelId	Yes (see note 6)	MQRC_GROUPING_NOT_ALTERABLE when in a grouped subscription
MQSD	PubPriority	Yes	None
MQSD	PubAccountingToken	Yes	None
MQSD	PubApplIdentityData	Yes	None
MQSD	SubLevel	No	MQRC_SUBLEVEL_NOT_ALTERABLE
MQSUB	Hobj	Yes (see note 6)	MQRC_GROUPING_NOT_ALTERABLE when in a grouped subscription

Notes:

1. MQSO_GROUP_SUB cannot be altered.
2. MQSO_NEW_PUBLICATIONS_ONLY cannot be altered because it is not part of the subscription
3. These options are not part of the subscription
4. This attribute is not part of the subscription
5. This attribute is the identity of the subscription being altered
6. Alterable except when part of a grouped sub (MQSO_GROUP_SUB)

Durability options: The following options control how durable the subscription is. You can specify only one of these options. If you are altering an existing subscription using the MQSO_ALTER option, you cannot change the durability of the subscription. On return from an MQSUB call using MQSO_RESUME, the appropriate durability option is set.

MQSO_DURABLE

Request that the subscription to this topic remains until it is explicitly removed using MQCLOSE with the MQCO_REMOVE_SUB option. If this subscription is not explicitly removed it will remain even after this applications connection to the queue manager is closed.

If a durable subscription is requested to a topic that is defined as not allowing durable subscriptions, the call fails with MQRC_DURABILITY_NOT_ALLOWED.

MQSO_NON_DURABLE

Request that the subscription to this topic is removed when the applications connection to the queue manager is closed, if it is not already explicitly removed. MQSO_NON_DURABLE is the opposite of the MQSO_DURABLE option, and is defined to aid program documentation. It is the default if neither is specified.

Destination options: The following option controls the destination that publications for a topic that has been subscribed to are sent to. If altering an existing subscription using the MQSO_ALTER option, the destination used for publications for the subscription can be changed. On return from an MQSUB call using MQSO_RESUME, this option is set if appropriate.

MQSO_MANAGED

Request that the destination that the publications are sent to is managed by the queue manager.

The object handle returned in *Hobj* represents a queue manager managed queue and is for use with subsequent MQGET, MQCB, MQINQ, or MQCLOSE calls.

An object handle returned from a previous MQSUB call cannot be provided in the **Hobj** parameter when MQSO_MANAGED is not specified.

MQSO_NO_MULTICAST

Request that the destination that the publications are sent to is not a multicast group address. This option is only valid when combined with the MQSO_MANAGED option. When a handle to a queue is provided in the **Hobj** parameter, multicast cannot be used for this subscription, and the option is not valid.

If the topic is defined to only allow multicast subscriptions, using the `MCAST(ONLY)` setting, then the call fails with reason code MQRC_MULTICAST_REQUIRED.

Scope Option: The following option controls the scope of the subscription being made. If altering an existing subscription using the MQSO_ALTER option, this subscription scope option cannot be changed. On returning from an MQSUB call using MQSO-RESUME, the appropriate scope option is set.

MQSO_SCOPE_QMGR

This subscription is made only on the local queue manager. No proxy subscription is distributed to other queue managers in the network. Only publications that are published at this queue manager are sent to this subscriber. This overrides any behavior set using the SUBSCOPE topic attribute.

Note: If not set, the subscription scope is determined by the SUBSCOPE topic attribute.

Registration options: The following options control the details of the registration that is made to the queue manager for this subscription. If altering an existing subscription using the MQSO_ALTER option, these registration options can be changed. On return from an MQSUB call using MQSO_RESUME the appropriate registration options is set.

MQSO_GROUP_SUB

This subscription is to be grouped with other subscriptions of the same SubLevel using the same queue and specifying the same correlation ID so that any publications to topics that would cause more than one publication message to be provided to the group of subscriptions, due to an overlapping set of topic strings being used, only causes one message to be delivered to the queue. If this option is not used, then each unique subscription (identified by SubName) that matches is provided with a copy of the publication which could mean more than one copy of the publication may be placed on the queue shared by a number of subscriptions.

Only the most significant subscription in the group is provided with a copy of the publication. The most significant subscription is based on the Full topic name up to the point where a wildcard is found. If a mixture of wildcard schemes is used within the group, only the position of the wildcard is important. You are advised not to combine different wildcard schemes within a group of subscriptions that share the same queue.

When creating a new grouped subscription it must still have a unique SubName, but if it matches the full topic name of an existing subscription in the group, the call fails with MQRC_DUPLICATE_GROUP_SUB.

If the most significant subscription in group also specifies MQSO_NOT_OWN_PUBS and this is a publication from the same application, then no publication is delivered to the queue.

When altering a subscription made with this option, the fields which imply the grouping, Hobj on the MQSUB call (representing the queue and queue manager name), and the SubCorrelId cannot be changed. Attempting to alter them causes the call to fail with MQRC_GROUPING_NOT_ALTERABLE.

This option must be combined with MQSO_SET_CORREL_ID with a SubCorrelId that is not set to MQCI_NONE, and cannot be combined with MQSO_MANAGED.

MQSO_ANY_USERID

When MQSO_ANY_USERID is specified, the identity of the subscriber is not restricted to a single user ID. This allows any user to alter or resume the subscription when they have suitable authority. Only a single user may have the subscription at any one time. An attempt to resume use of a subscription currently in use by another application causes the call to fail with MQRC_SUBSCRIPTION_IN_USE.

To add this option to an existing subscription the MQSUB call (using MQSO_ALTER) must come from the same user ID as the original subscription itself.

If an MQSUB call refers to an existing subscription with MQSO_ANY_USERID set, and the user ID differs from the original subscription, the call succeeds only if the new user ID has authority to subscribe to the topic. On successful completion, future publications to this subscriber are put to the subscribers queue with the new user ID set in the publication message.

Do not specify both MQSO_ANY_USERID and MQSO_FIXED_USERID. If neither is specified, the default is MQSO_FIXED_USERID.

MQSO_FIXED_USERID

When MQSO_FIXED_USERID is specified, the subscription can be altered or resumed by only the last user ID to alter the subscription. If the subscription has not been altered, it is the user ID that created the subscription.

If an MQSUB verb refers to an existing subscription with MQSO_ANY_USERID set and alters the subscription using MQSO_ALTER to use option MQSO_FIXED_USERID, the user ID of the subscription is now fixed at this new user ID. The call succeeds only if the new user ID has authority to subscribe to the topic.

If a user ID other than the one recorded as owning a subscription tries to resume or alter an MQSO_FIXED_USERID subscription, the call fails with MQRC_IDENTITY_MISMATCH. The owning user ID of a subscription can be viewed using the DISPLAY SBSTATUS command.

Do not specify both MQSO_ANY_USERID and MQSO_FIXED_USERID. If neither is specified, the default is MQSO_FIXED_USERID.

Publication options: The following options control the way publications are sent to this subscriber. If altering an existing subscription using the MQSO_ALTER option, these publication options can be changed.

MQSO_NOT_OWN_PUBS

Tells the broker that the application does not want to see any of its own publications. Publications are considered to originate from the same application if the connection handles are the same. On return from an MQSUB call using MQSO_RESUME, this option is set if appropriate.

MQSO_NEW_PUBLICATIONS_ONLY

No currently retained publications are to be sent, when this subscription is created, only new publications. This option only applies when MQSO_CREATE is specified. Any subsequent changes to a subscription do not alter the flow of publications and so any publications retained on a topic, will have already been sent to the subscriber as new publications.

If this option is specified without MQSO_CREATE the call fails with MQRC_OPTIONS_ERROR. On return from an MQSUB call using MQSO_RESUME, this option is not set even if the subscription was created using this option.

If this option is not used, previously retained messages are sent to the destination queue provided. If this action fails due to an error, either MQRC_RETAINED_MSG_Q_ERROR or MQRC_RETAINED_NOT_DELIVERED, the creation of the subscription fails.

MQSO_PUBLICATIONS_ON_REQUEST

Setting this option indicates that the subscriber will request information specifically when required. The queue manager does not send unsolicited messages to the subscriber. The retained publication (or possibly multiple publications if a wildcard is specified in the topic) is sent to the

subscriber each time an MQSUBRQ call is made using the Hsub handle from a previous MQSUB call. No publications are sent as a result of the MQSUB call using this option. On return from an MQSUB call using MQSO_RESUME, this option is set if appropriate.

This option is not valid in combination with a SubLevel greater than 1.

Read ahead options: The following options control whether non-persistent messages are sent to an application ahead of the application requesting them.

MQSO_READ_AHEAD_AS_Q_DEF
If the MQSUB call uses a managed handle, the default read ahead attribute of the model queue associated with the topic subscribed to determines whether messages are sent to the application before the application requests them.

This is the default value.

MQSO_NO_READ_AHEAD
If the MQSUB call uses a managed handle, messages are not sent to the application before the application requests them.

MQSO_READ_AHEAD
If the MQSUB call uses a managed handle, messages might be sent to the application before the application requests them.

Note:

The following notes apply to the read ahead options:

1. Only one of these options can be specified. If both MQOO_READ_AHEAD and MQOO_NO_READ_AHEAD are specified, reason code MQRC_OPTIONS_ERROR is returned. These options are only applicable if MQSO_MANAGED is specified.
2. They are not applicable for MQSUB when a queue is passed which has been opened previously. Read ahead might not be enabled when requested. The MQGET options used on the first MQGET call might prevent read ahead from being enabled. Also, read ahead is disabled when the client is connecting to a queue manager where read ahead is not supported. If the application is not running as an IBM MQ client, these options are ignored.

Wildcard options: The following options control how wildcards are interpreted in the string provided in the ObjectString field of the MQSD. You can specify only one of these options. If altering an existing subscription using the MQSO_ALTER option, these wildcard options cannot be changed. On return from an MQSUB call using MQSO_RESUME, the appropriate wildcard option is set.

MQSO_WILDCARD_CHAR
Wildcards only operate on characters within the topic string.

The behavior defined by MQSO_WILDCARD_CHAR is shown in the following table.

Special Character	Behavior
Forward slash (/)	No significance, just another character
Asterisk (*)	Wildcard, zero or more characters
Question mark (?)	Wildcard, 1 character
Percent sign (%)	Escape character to allow the characters (*), (?) or (%) to be used in a string and not be interpreted as a special character, for example, (%*), (%?) or (%%).

For example, publishing on the following topic:

```
/level0/level1/level2/level3/level4
```

matches subscribers using the following topics:

```
*
/*
/ level0/level1/level2/level3/*
/ level0/level1/*/level3/level4
/ level0/level1/le?el2/level3/level4
```

Note: This use of wildcards supplies exactly the meaning provided in IBM MQ V6 and WebSphere MB V6 when using MQRFH1 formatted messages for publish/subscribe. It is recommended that this is not used for newly written applications and is only used for applications that were previously running against that version and have not been changed to use the default wildcard behavior as described in MQSO_WILDCARD_TOPIC.

MQSO_WILDCARD_TOPIC

Wildcards only operate on topic elements within the topic string. This is the default behavior if none is chosen.

The behavior required by MQSO_WILDCARD_TOPIC is shown in the following table:

Special Character	Behavior
(/)	Topic level separator
Number sign (#)	Wildcard: multiple topic level
Plus sign (+)	Wildcard: single topic level

Notes:

The (+) and (#) are not treated as wildcards if they are mixed in with other characters (including themselves) within a topic level. In the following string, the (#) and (+) characters are treated as ordinary characters.

```
level0/level1/#+/level3/level#
```

For example, publishing on the following topic:

```
/level0/level1/level2/level3/level4
```

matches subscribers using the following topics:

```
#
/#
/ level0/level1/level2/level3/#
/ level0/level1/+/level3/level4
```

Other options: The following options control the way the API call is issued rather than the subscription. On return from an MQSUB call using MQSO_RESUME, these options are unchanged. See "AlternateUserId (MQCHAR12)" on page 2551 for more details.

MQSO_ALTERNATE_USER_AUTHORITY

The AlternateUserId field contains a user identifier to use to validate this MQSUB call. The call can succeed only if this AlternateUserId is authorized to open the object with the specified access options, regardless of whether the user identifier under which the application is running is authorized to do so.

MQSO_SET_CORREL_ID

The subscription is to use the correlation identifier supplied in the *SubCorrelId* field. If this option is not specified, a correlation identifier is automatically created by the queue manager at subscription time and is returned to the application in the *SubCorrelId* field. For more information, see "SubCorrelId (MQBYTE24)" on page 2564 for more information.

This option cannot be combined with MQSO_MANAGED.

MQSO_SET_IDENTITY_CONTEXT

The subscription is to use the accounting token and application identity data supplied in the *PubAccountingToken* and *PubApplIdentityData* fields.

If this option is specified, the same authorization check is carried out as if the destination queue was accessed using an MQOPEN call with MQOO_SET_IDENTITY_CONTEXT, except in the case where the MQSO_MANAGED option is also used in which case there is no authorization check on the destination queue.

If this option is not specified, the publications sent to this subscriber have default context information associated with them as follows:

Field in MQMD	Value used
UserIdentifier	The user ID associated with the subscription at the time the subscription was made.
AccountingToken	Determined from the environment if possible; Set to MQACT_NONE if not.
ApplIdentityData	Set to blanks

This option is only valid with MQSO_CREATE and MQSO_ALTER. If used with MQSO_RESUME, the *PubAccountingToken* and *PubApplIdentityData* fields are ignored, so this option has no effect.

If a subscription is altered without using this option where previously the subscription supplied identity context information, default context information is generated for the altered subscription.

If a subscription allowing different user IDs to use it with option MQSO_ANY_USERID, is resumed by a different user ID, default identity context is generated for the new user ID now owning the subscription and any subsequent publications are delivered containing the new identity context.

MQSO_FAIL_IF_QUIESCING

The MQSUB call fails if the queue manager is in quiescing state. On z/OS, for a CICS or IMS application, this option also forces the MQSUB call to fail if the connection is in quiescing state.

PubAccountingToken (MQBYTE32):

This is the value that will be in the *AccountingToken* field of the Message Descriptor (MQMD) of all publication messages matching this subscription. *AccountingToken* is part of the identity context of the message. For more information about message context, see Message context. For more information about the *AccountingToken* field in the MQMD, see "AccountingToken (MQBYTE32)" on page 2390

You can use the following special value for the *PubAccountingToken* field:

MQACT_NONE

No accounting token is specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQACT_NONE_ARRAY is also defined; this has the same value as MQACT_NONE, but is an array of characters instead of a string.

If the option MQSO_SET_IDENTITY_CONTEXT is not specified, the accounting token is generated by the queue manager as default context information and this field is an output field which contains the *AccountingToken* which will be set in each message published for this subscription.

If the option MQSO_SET_IDENTITY_CONTEXT is specified, the accounting token is being generated by the user and this field is an input field which contains the *AccountingToken* to be set in each publication for this subscription.

The length of this field is given by MQ_ACCOUNTING_TOKEN_LENGTH. The initial value of this field is MQACT_NONE.

If altering an existing subscription using the MQSO_ALTER option, the value of *AccountingToken* in any future publication messages can be changed.

On return from an MQSUB call using MQSO_RESUME, this field is set to the current *AccountingToken* being used for the subscription.

PubApplIdentityData (MQCHAR32):

This is the value that is in the *ApplIdentityData* field of the Message Descriptor (MQMD) of all publication messages matching this subscription. *ApplIdentityData* is part of the identity context of the message. For more information about message context, see Message context. For more information about the *ApplIdentityData* field in the MQMD, see "ApplIdentityData (MQCHAR32)" on page 2392

If the option MQSO_SET_IDENTITY_CONTEXT is not specified, the *ApplIdentityData* which is set in each message published for this subscription is blanks, as default context information.

If the option MQSO_SET_IDENTITY_CONTEXT is specified, the *PubApplIdentityData* is being generated by the user and this field is an input field which contains the *ApplIdentityData* to be set in each publication for this subscription.

The length of this field is given by MQ_APPL_IDENTITY_DATA_LENGTH. The initial value of this field is the null string in C, and 32 blank characters in other programming languages.

If altering an existing subscription using the MQSO_ALTER option, the *ApplIdentityData* of any future publication messages can be changed.

On return from an MQSUB call using MQSO_RESUME, this field is set to the current *ApplIdentityData* being used for the subscription.

PubPriority (MQLONG):

This is the value that will be in the *Priority* field of the Message Descriptor (MQMD) of all publication messages matching this subscription. For more information about the *Priority* field in the MQMD, see "Priority (MQLONG)" on page 2420.

The value must be greater than or equal to zero; zero is the lowest priority. The following special values can also be used:

MQPRI_PRIORITY_AS_Q_DEF

When a subscription queue is provided in the *Hobj* field in the MQSUB call, and is not a managed handle, then the priority for the message is taken from the **DefPriority** attribute of this queue. If the queue is a cluster queue or there is more than one definition in the queue-name resolution path then the priority is determined when the publication message is put to the queue as described for "Priority (MQLONG)" on page 2420.

If the MQSUB call uses a managed handle, the priority for the message is taken from the **DefPriority** attribute of the model queue associated with the topic subscribed to.

MQPRI_PRIORITY_AS_PUBLISHED

The priority for the message is the priority of the original publication. This is the initial value of the field.

If altering an existing subscription using the MQSO_ALTER option, the *Priority* of any future publication messages can be changed.

On return from an MQSUB call using MQSO_RESUME, this field is set to the current priority being used for the subscription.

ResObjectString (MQCHARV):

This is the long object name after the queue manager resolves the name provided in *ObjectName*.

If the long object name is provided in *ObjectString* and nothing is provided in *ObjectName*, then the value returned in this field is the same as provided in *ObjectString*.

If this field is omitted (that is ResObjectString.VSBufSize is zero) then the *ResObjectString* is not returned, but the length is returned in ResObjectString.VSLength. If the length is shorter than the full ResObjectString then it is truncated and returns as many of the rightmost characters as can fit in the provided length.

If *ResObjectString* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code MQRC_RES_OBJECT_STRING_ERROR.

SelectionString (MQCHARV):

This is the string used to provide the selection criteria used when subscribing for messages from a topic.

This variable length field will be returned on output from an MQSUB call using the MQSO_RESUME option, if a buffer is provided, and also there is a positive buffer length in VSBufSize. If no buffer is provided on the call, only the length of the selection string will be returned in the VSLength field of the MQCHARV. If the buffer provided is smaller than the space required to return the field, only VSBufSize bytes are returned in the provided buffer.

If *SelectionString* is specified incorrectly, according to the description of how to use the "MQCHARV - Variable Length String" on page 2250 structure, or if it exceeds the maximum length, the call fails with reason code MQRC_SELECTION_STRING_ERROR.

SelectionString usage is described in Selectors.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQSD_STRUC_ID
Identifier for Subscription Descriptor structure.

For the C programming language, the constant MQSD_STRUC_ID_ARRAY is also defined; this has the same value as MQSD_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQSD_STRUC_ID.

SubCorrelId (MQBYTE24):

This field contains a correlation identifier common to all publications matching this subscription.

Attention: a correlation identifier can only be passed between queue managers in a publish/subscribe cluster, not a hierarchy.

All publications sent to match this subscription contain this correlation identifier in the message descriptor. If multiple subscriptions get their publications from the same queue, using MQGET by correlation identifier allows only publications for a specific subscription to be obtained. This correlation identifier can either be generated by the queue manager or by the user.

If the option MQSO_SET_CORREL_ID is not specified, the correlation identifier is generated by the queue manager and this field is an output field containing the correlation identifier that will be set in each message published for this subscription. The generated correlation identifier consists of a 4-byte product identifier (AMQX or CSQM in either ASCII or EBCDIC) followed by a product specific implementation of a unique string.

If the option MQSO_SET_CORREL_ID is specified, the correlation identifier is generated by the user and this field is an input field containing the correlation identifier to be set in each publication for this subscription. In this case, if the field contains MQCI_NONE, the correlation identifier that is set in each message published for this subscription is the correlation identifier created by the original put of the message.

If the option MQSO_GROUP_SUB is specified and the correlation identifier specified is the same as an existing grouped subscription using the same queue and an overlapping topic string, only the most significant subscription in the group is provided with a copy of the publication.

The length of this field is given by MQ_CORREL_ID_LENGTH. The initial value of this field is MQCI_NONE.

If you are altering an existing subscription using the MQSO_ALTER option, and this field is an input field, then the subscription correlation identifier can be changed, unless the subscription is a grouped subscription, that is, it has been created using the option MQSO_GROUP_SUB, in which case the subscription correlation identifier cannot be changed.

On return from an MQSUB call using MQSO_RESUME, this field is set to the current correlation identifier for the subscription.

SubExpiry (MQLONG):

This is the time expressed in tenths of a second after which the subscription expires. No more publications will match this subscription after this interval has passed. As soon as a subscription expires, publications are no longer sent to the queue. However, the publications that are already there are not affected in any way. *SubExpiry* has no effect on publication expiry.

The following special value is recognized:

MQEI_UNLIMITED
The subscription has an unlimited expiration time.

If altering an existing subscription using the MQSO_ALTER option, the expiry of the subscription can be changed.

On return from an MQSUB call using the MQSO_RESUME option this field is set to the original expiry of the subscription and not the remaining expiry time.

SubLevel (MQLONG):

This is the level associated with the subscription. Publications are only delivered to this subscription if it is in the set of subscriptions with the highest SubLevel value less than or equal to the PubLevel used at publication time. However, if a publication has been retained, it is no longer available to subscribers at higher levels because it is republished at PubLevel 1.

The value must be in the range zero to 9. Zero is the lowest level.

The initial value of this field is 1.

For more information see Intercepting publications.

If altering an existing subscription using the MQSO_ALTER option, then the SubLevel cannot be changed.

Combining a SubLevel with a value greater than 1 with the option MQSO_PUBLICATIONS_ON_REQUEST is not allowed.

On return from an MQSUB call using MQSO_RESUME, this field is set to the current level being used for the subscription.

SubUserData (MQCHARV):

This specifies the subscription user data. The data provided on the subscription in this field will be included as the MQSubUserData message property of every publication sent to this subscription.

The maximum length of *SubUserData* is 10240.

If *SubUserData* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code MQRC_SUB_USER_DATA_ERROR.

This is an input field. The initial values of the fields in this structure are the same as those in the MQCHARV structure.

If altering an existing subscription using the MQSO_ALTER option, the subscription user data can be changed.

This variable length field is returned on output from an MQSUB call using the MQSO_RESUME option, if a buffer is provided and there is a positive buffer length in *VSBufLen*. If no buffer is provided on the call, only the length of the subscription user date is returned in the *VSLength* field of the MQCHARV. If the buffer provided is smaller than the space required to return the field, only *VSBufLen* bytes are returned in the provided buffer.

SubName (MQCHARV):

This specifies the subscription name. This field is only required if *Options* specifies the option MQSO_DURABLE, but if provided will be used by the queue manager for MQSO_NON_DURABLE as well.

If specified, *SubName* must be unique within the queue manager, because it is the method used to identify the subscription.

The maximum length of *SubName* is 10240.

This field serves two purposes. For an MQSO_DURABLE subscription, you use this field to identify a subscription so you can resume it after it has been created if you have either closed the handle to the subscription (using the MQCO_KEEP_SUB option) or have been disconnected from the queue manager. This is done using the MQSUB call with the MQSO_RESUME option. It is also displayed in the administrative view of subscriptions in the SUBNAME field in DISPLAY SBSTATUS.

If *SubName* is specified incorrectly, according to the description of how to use the MQCHARV structure, is left out when it is required (that is *SubName*. *VSLength* is zero), or if it exceeds the maximum length, the call fails with reason code MQRC_SUB_NAME_ERROR.

This is an input field. The initial values of the fields in this structure are the same as those in the MQCHARV structure.

If altering an existing subscription using the MQSO_ALTER option, the subscription name cannot be changed, because it is the identifying field used to find the referenced subscription. It is not changed on output from an MQSUB call with the MQSO_RESUME option.

Version (MQLONG):

This is the structure version number; the value must be:

MQSD_VERSION_1
Version-1 Subscription Descriptor structure.

The following constant specifies the version number of the current version:

MQSD_CURRENT_VERSION
Current version of Subscription Descriptor structure.

This is always an input field. The initial value of this field is MQSD_VERSION_1.

Using topic strings:

A topic is constructed from the subtopic identified in a topic object, and a subtopic provided by an application. You can use either subtopic as the topic name, or combine them to form a new topic name.

In an MQI program the full topic name is created by MQOPEN. It is composed of two fields used in publish/subscribe MQI calls, in the order listed:

1. The **TOPICSTR** attribute of the topic object, named in the **ObjectName** field.
2. The **ObjectString** parameter defining the subtopic provided by the application.

The resulting topic string is returned in the **ResObjectString** parameter.

These fields are considered to be present if the first character of each field is not a blank or null character, and the field length is greater than zero. If only one of the fields is present, it is used unchanged as the

topic name. If neither field has a value, the call fails with reason code MQRC_UNKNOWN_OBJECT_NAME, or MQRC_TOPIC_STRING_ERROR if the full topic name is not valid.

If both fields are present, a ' / ' character is inserted between the two elements of the resultant combined topic name.

Table 248 shows examples of topic string concatenation:

Table 248. Topic string concatenation examples

TOPICSTR	ObjectString	Full topic name	Comment
Football/Scores	''	Football/Scores	The TOPICSTR is used alone
''	Football/Scores	Football/Scores	The ObjectString is used alone
Football	Scores	Football/Scores	A '/' character is added at the concatenation point
Football	/Scores	Football//Scores	An 'empty node' is produced between the two strings
/Football	Scores	/Football/Scores	The topic starts with an 'empty node'

The ' / ' character is considered as a special character, providing structure to the full topic name in Topic trees, and must not be used for any other reason as the structure of the topic tree is affected. The topic " `/Football` " is not the same as the topic " `Football` ".

The following wildcard characters are special characters:

- plus sign ' + '
- number sign ' # '
- asterisk ' * '
- question mark ' ? '

These characters are not considered as invalid, however you must ensure to understand how they are used. You might prefer not to use these characters in your topic strings when publishing. Publishing on a topic string with ' # ' or ' + ' mixed in with other characters (including themselves) within a topic level, can be subscribed to with either wildcard scheme. Publishing on a topic string with ' # ' or ' + ' as the only character between two ' / ' characters produces a topic string that cannot be subscribed to explicitly by an application using the wildcard scheme MQSO_WILDCARD_TOPIC. This situation results in the application getting more publications than expected.

Example code snippet

This code snippet, extracted from the example program Example 2: Publisher to a variable topic, combines a topic object with a variable topic string.

```
MQOD     td = {MQOD_DEFAULT};  /* Object Descriptor          */
td.ObjectType = MQOT_TOPIC;    /* Object is a topic          */
td.Version = MQOD_VERSION_4;   /* Descriptor needs to be V4  */
strncpy(td.ObjectName, topicName,  MQ_TOPIC_NAME_LENGTH);
td.ObjectString.VSPtr = topicString;
td.ObjectString.VSLength = (MQLONG)strlen(topicString);
td.ResObjectString.VSPtr = resTopicStr;
td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
```

Initial values and language declarations for MQSD:

Field name	Name of constant	Value of constant
StrucId	MQSD_STRUC_ID	'SD¬¬'
Version	MQSD_VERSION_1	1
Options	MQSO_NON_DURABLE	0
ObjectName	None	Null string or blanks
AlternateUserId	None	Null string or blanks
AlternateSecurityId	MQSID_NONE	Nulls
SubExpiry	MQEI_UNLIMITED	-1
ObjectString	None	Names and values as defined for MQCHARV
SubName	None	Names and values as defined for MQCHARV
SubUserData	None	Names and values as defined for MQCHARV
SubCorrelId	MQCI_NONE	Nulls
PubPriority	MQPRI_PRIORITY_AS_Q_DEF	-3
PubAccountingToken	MQACT_NONE	Nulls
PubApplIdentityData	None	Null string or blanks
SelectionString	None	Names and values as defined for MQCHARV
SubLevel	None	1
ResObjectString	None	Names and values as defined for MQCHARV

Notes:

1. The symbol ¬ represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQSD_DEFAULT contains the values that are listed in the table. It can be used in the following way to provide initial values for the fields in the structure:

```
MQSD MySD = {MQSD_DEFAULT};
```

C declaration for MQSD:

```
typedef struct tagMQSD MQSD;
struct tagMQSD {
  MQCHAR4   StrucId;              /* Structure identifier */
  MQLONG    Version;              /* Structure version number */
  MQLONG    Options;              /* Options associated with subscribing */
  MQCHAR48  ObjectName;           /* Object name */
  MQCHAR12  AlternateUserId;      /* Alternate user identifier */
  MQBYTE40  AlternateSecurityId;  /* Alternate security identifier */
  MQLONG    SubExpiry;            /* Expiry of Subscription */
  MQCHARV   ObjectString;         /* Object Long name */
  MQCHARV   SubName;              /* Subscription name */
  MQCHARV   SubUserData;          /* Subscription User data */
  MQBYTE24  SubCorrelId;          /* Correlation Id related to this subscription */
  MQLONG    PubPriority;          /* Priority set in publications */
  MQBYTE32  PubAccountingToken;   /* Accounting Token set in publications */
  MQCHAR32  PubApplIdentityData;  /* Appl Identity Data set in publications */
  MQCHARV   SelectionString;      /* Message selector structure */
  MQLONG    SubLevel;             /* Subscription level */
  MQCHARV   ResObjectString;      /* Resolved Long object name*/
  /* Ver:1 */
};
```

COBOL declaration for MQSD:

```
** Address of variable length string
   20  MQSD-OBJECTSTRING-VSPTR         POINTER.
** Offset of variable length string
   20  MQSD-OBJECTSTRING-VSOFFSET      PIC S9(9) BINARY.
** size of buffer
   20  MQSD-OBJECTSTRING-VSBUFSIZE     PIC S9(9) BINARY.
** Length of variable length string
   20  MQSD-OBJECTSTRING-VSLENGTH      PIC S9(9) BINARY.
** CCSID of variable length string
   20  MQSD-OBJECTSTRING-VSCCSID       PIC S9(9) BINARY.
** Subscription name
 15  MQSD-SUBNAME.
** Address of variable length string
   20  MQSD-SUBNAME-VSPTR              POINTER.
** Offset of variable length string
   20  MQSD-SUBNAME-VSOFFSET           PIC S9(9) BINARY.
** size of buffer
   20  MQSD-SUBNAME-VSBUFSIZE          PIC S9(9) BINARY.
** Length of variable length string
   20  MQSD-SUBNAME-VSLENGTH           PIC S9(9) BINARY.
** CCSID of variable length string
   20  MQSD-SUBNAME-VSCCSID            PIC S9(9) BINARY.
** Subscription User data
 15  MQSD-SUBUSERDATA.
** Address of variable length string
   20  MQSD-SUBUSERDATA-VSPTR          POINTER.
** Offset of variable length string
   20  MQSD-SUBUSERDATA-VSOFFSET       PIC S9(9) BINARY.
** size of buffer
   20  MQSD-SUBUSERDATA-VSBUFSIZE      PIC S9(9) BINARY.
** Length of variable length string
   20  MQSD-SUBUSERDATA-VSLENGTH       PIC S9(9) BINARY.
** CCSID of variable length string
   20  MQSD-SUBUSERDATA-VSCCSID        PIC S9(9) BINARY.
** Correlation Id related to this subscription
 15  MQSD-SUBCORRELID                 PIC X(24).
** Priority set in publications
 15  MQSD-PUBPRIORITY                 PIC S9(9) BINARY.
** Accounting Token set in publications
 15  MQSD-PUBACCOUNTINGTOKEN          PIC X(32).
** Appl Identity Data set in publications
 15  MQSD-PUBAPPLIDENTITYDATA         PIC X(32).
```

```
** Message Selector
 15 MQSD-SELECTIONSTRING.
** Address of variable length string
   20  MQSD-SELECTIONSTRING-VSPTR     POINTER.
** Offset of variable length string
   20  MQSD-SELECTIONSTRING-VSOFFSET       PIC S9(9) BINARY.
** size of buffer
   20  MQSD-SELECTIONSTRING-VSBUFSIZE      PIC S9(9) BINARY.
** Length of variable length string
   20  MQSD-SELECTIONSTRING-VSLENGTH       PIC S9(9) BINARY.
** CCSID of variable length string
   20  MQSD-SELECTIONSTRING-VSCCSID        PIC S9(9) BINARY.
** Selection criteria
   20  MQSD-SELECTIONSTRING-SUBLEVEL       PIC S9(9) BINARY.
** Long object name
   20  MQSD-SELECTIONSTRING-RESOBJSTRING   PIC S9(9) BINARY.
```

PL/I declaration for MQSD:

```
dcl
1 MQSD based,
3 StrucId       char(4),    /* Structure identifier */
3 Version       fixed bin(31), /* Structure version number */
3 Options       fixed bin(31), /* Options associated with subscribing */
3 ObjectName     char(48),   /* Object name */
3 AlternateUserId   char(12),   /* Alternate user identifier */
3 AlternateSecurityId char(40),   /* Alternate security identifier */
3 SubExpiry      fixed bin(31), /* Expiry of Subscription */
3 ObjectString,           /* Object Long name */
5 VSPtr       pointer,    /* Address of variable length string */
5 VSOffset      fixed bin(31), /* Offset of variable length string */
5 VSBufSize     fixed bin(31), /* size of buffer */
5 VSLength      fixed bin(31), /* Length of variable length string */
5 VSCCSID      fixed bin(31); /* CCSID of variable length string */
3 SubName,              /* Subscription name */
5 VSPtr       pointer,    /* Address of variable length string */
5 VSOffset      fixed bin(31), /* Offset of variable length string */
5 VSBufSize     fixed bin(31), /* size of buffer */
5 VSLength      fixed bin(31), /* Length of variable length string */
5 VSCCSID      fixed bin(31); /* CCSID of variable length string */
3 SubUserData,            /* Subscription User data */
5 VSPtr       pointer,    /* Address of variable length string */
5 VSOffset      fixed bin(31), /* Offset of variable length string */
5 VSBufSize     fixed bin(31), /* size of buffer */
5 VSLength      fixed bin(31), /* Length of variable length string */
5 VSCCSID      fixed bin(31), /* CCSID of variable length string */
3 SubCorrelId     char(24),   /* Correlation Id related to this subscription */
3 PubPriority     fixed bin(31), /* Priority set in publications */
3 PubAccountingToken char(32),   /* Accounting Token set in publications */
3 PubApplIdentityData char(32),   /* Appl Identity Data set in publications */
3 SelectionString,          /* Message Selection */
5 VSPtr       pointer,    /* Address of variable length string */
5 VSOffset      fixed bin(31), /* Offset of variable length string */
5 VSBufSize     fixed bin(31), /* size of buffer */
5 VSLength      fixed bin(31), /* Length of variable length string */
5 VSCCSID      fixed bin(31), /* CCSID of variable length string */
3 SubLevel      fixed bin(31), /* Subscription level */
3 ResObjectString,          /* Resolved Long object name */
5 VSPtr       pointer,    /* Address of variable length string */
5 VSOffset      fixed bin(31), /* Offset of variable length string */
5 VSBufSize     fixed bin(31), /* size of buffer */
5 VSLength      fixed bin(31), /* Length of variable length string */
5 VSCCSID      fixed bin(31); /* CCSID of variable length string */
```

High Level Assembler declaration for MQSD:

```
MQSD              DSECT
MQSD_STRUCID          DS  CL4  Structure identifier
MQSD_VERSION          DS  F   Structure version number
MQSD-OPTIONS          DS  F   Options associated with subscribing
MQSD_OBJECTNAME        DS  CL48 Object name
MQSD_ALTERNATEUSERID      DS  CL12 Alternate user identifier
MQSD_ALTERNATESECURITYID    DS  CL40 Alternate security identifier
MQSD_SUBEXPIRY         DS  F   Expiry of Subscription
MQSD_OBJECTSTRING       DS  0F  Object Long name
MQSD_OBJECTSTRING_VSPTR    DS  F   Address of variable length string
MQSD_OBJECTSTRING_VSOFFSET   DS  F   Offset of variable length string
MQSD_OBJECTSTRING_VSBUFSIZE  DS  F   size of buffer
MQSD_OBJECTSTRING_VSLENGTH   DS  F   Length of variable length string
MQSD_OBJECTSTRING_VSCCSID   DS  F   CCSID of variable length string
MQSD_OBJECTSTRING_LENGTH    EQU  *-MQSD_OBJECTSTRING
ORG  MQSD_OBJECTSTRING
MQSD_OBJECTSTRING_AREA     DS  CL(MQSD_OBJECTSTRING_LENGTH)
*
MQSD_SUBNAME          DS  0F  Subscription name
MQSD_SUBNAME_VSPTR       DS  F   Address of variable length string
MQSD_SUBNAME_VSOFFSET     DS  F   Offset of variable length string
MQSD_SUBNAME_VSBUFSIZE     DS  F   size of buffer
MQSD_SUBNAME_VSLENGTH     DS  F   Length of variable length string
MQSD_SUBNAME_VSCCSID      DS  F   CCSID of variable length string
MQSD_SUBNAME_LENGTH      EQU  *-MQSD_SUBNAME
ORG  MQSD_SUBNAME
MQSD_SUBNAME_AREA       DS  CL(MQSD_SUBNAME_LENGTH)
*
MQSD_SUBUSERDATA        DS  0F  Subscription User data
MQSD_SUBUSERDATA_VSPTR     DS  F   Address of variable length string
MQSD_SUBUSERDATA_VSOFFSET   DS  F   Offset of variable length string
MQSD_SUBUSERDATA_VSBUFSIZE   DS  F   size of buffer
MQSD_SUBUSERDATA_VSLENGTH   DS  F   Length of variable length string
MQSD_SUBUSERDATA_VSCCSID    DS  F   CCSID of variable length string
MQSD_SUBUSERDATA_LENGTH    EQU  *-MQSD_SUBUSERDATA
ORG  MQSD_SUBUSERDATA
MQSD_SUBUSERDATA_AREA     DS  CL(MQSD_SUBUSERDATA_LENGTH)
*
MQSD_SUBCORRELID        DS  CL24 Correlation Id related to this subscription
MQSD_PUBPRIORITY        DS  F   Priority set in publications
MQSD_PUBACCOUNTINGTOKEN    DS  CL32 Accounting Token set in publications
MQSD_PUBAPPLIDENTITYDATA    DS  CL32 Appl Identity Data set in publications
*
MQSD_SELECTIONSTRING      DS  F   Message Selector
MQSD_SELECTIONSTRING_VSPTR   DS  F   Address of variable length string
MQSD_SELECTIONSTRING_VSOFFSET DS  F   Offset of variable length string
MQSD_SELECTIONSTRING_VSBUFSIZE DS  F   size of buffer
MQSD_SELECTIONSTRING_VSLENGTH DS  F   Length of variable length string
MQSD_SELECTIONSTRING_VSCCSID  DS  F   CCSID of variable length string
MQSD_SELECTIONSTRING_LENGTH  EQU *- MQSD_SELECTIONSTRING
ORG MQSD_SELECTIONSTRING
MQSD_SELECTIONSTRING_AREA   DS  CL(MQSD_SELECTIONSTRING_LENGTH)
*
MQSD-SUBLEVEL         DS  F   Subscription level
*
MQSD_RESOBJECTSTRING      DS  F   Resolved Long object name
MQSD_RESOBJECTSTRING_VSPTR   DS  F   Address of variable length string
MQSD_RESOBJECTSTRING_VSOFFSET DS  F   Offset of variable length string
MQSD_RESOBJECTSTRING_VSBUFSIZE DS  F   size of buffer
MQSD_RESOBJECTSTRING_VSLENGTH DS  F   Length of variable length string
MQSD_RESOBJECTSTRING_VSCCSID  DS  F   CCSID of variable length string
MQSD_RESOBJECTSTRING_LENGTH  EQU *- MQSD_RESOBJECTSTRING
ORG MQSD_RESOBJECTSTRING
MQSD_RESOBJECTSTRING_AREA   DS  CL(MQSD_RESOBJECTSTRING_LENGTH)
*
MQSD_LENGTH          EQU  *-MQSD
ORG  MQSD
MQSD_AREA           DS  CL(MQSD_LENGTH)
```

MQSMPO - Set message property options:

The following table summarizes the fields in the structure.

Table 249. Fields in MQSMPO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options	Options
ValueEncoding	Property value encoding	ValueEncoding
ValueCCSID	Property value character set	ValueCCSID

Overview for MQSMPO:

Availability: All IBM MQ systems and IBM MQ clients.

Purpose: The **MQSMPO** structure allows applications to specify options that control how properties of messages are set. The structure is an input parameter on the **MQSETMP** call.

Character set and encoding: Data in **MQSMPO** must be in the character set of the application and encoding of the application (**MQENC_NATIVE**).

Fields for MQSMPO:

The MQSMPO structure contains the following fields; the fields are described in **alphabetical order**:

Options (MQLONG):

Location options: The following options relate to the relative location of the property compared to the property cursor:

MQSMPO_SET_FIRST
Sets the value of the first property that matches the specified name, or if it does not exist, adds a new property after all other properties with a matching hierarchy.

MQSMPO_SET_PROP_UNDER_CURSOR
Sets the value of the property pointed to by the property cursor. The property pointed to by the property cursor is the one that was last inquired using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused on an MQGET call, or when the message handle is specified in the *MsgHandle* field of the MQGMO or MQPMO structure on an MQPUT call.

If this option is used when the property cursor has not yet been established or if the property pointer to by the property cursor has been deleted, the call fails with completion code MQCC_FAILED and reason code MQRC_PROPERTY_NOT_AVAILABLE.

MQSMPO_SET_PROP_BEFORE_CURSOR
Sets a new property before the property pointed to by the property cursor. The property pointed to by the property cursor is the one that was last inquired using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused on an MQGET call, or when the message handle is specified in the *MsgHandle* field of the MQGMO or MQPMO structure on an MQPUT call.

If this option is used when the property cursor has not yet been established or if the property pointer to by the property cursor has been deleted, the call fails with completion code MQCC_FAILED and reason code MQRC_PROPERTY_NOT_AVAILABLE.

MQSMPO_SET_PROP_AFTER_CURSOR
Sets a new property after the property pointed to by the property cursor. The property pointed to by the property cursor is the one that was last inquired using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused on an MQGET call, or when the message handle is specified in the *MsgHandle* field of the MQGMO or MQPMO structure on an MQPUT call.

If this option is used when the property cursor has not yet been established or if the property pointer to by the property cursor has been deleted, the call fails with completion code MQCC_FAILED and reason code MQRC_PROPERTY_NOT_AVAILABLE.

If you need none of the options described, use the following option:

MQSMPO_NONE
No options specified.

This is always an input field. The initial value of this field is MQSMPO_SET_FIRST.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQSMPO_STRUC_ID

Identifier for set message property options structure.

For the C programming language, the constant **MQSMPO_STRUC_ID_ARRAY** is also defined; this has the same value as **MQSMPO_STRUC_ID**, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is **MQSMPO_STRUC_ID**.

ValueCCSID (MQLONG):

The character set of the property value to be set if the value is a character string.

This is always an input field. The initial value of this field is **MQCCSI_APPL**.

ValueEncoding (MQLONG):

The encoding of the property value to be set if the value is numeric.

This is always an input field. The initial value of this field is **MQENC_NATIVE**.

Version (MQLONG):

This is the structure version number; the value must be:

MQSMPO_VERSION_1
Version-1 set message property options structure.

The following constant specifies the version number of the current version:

MQSMPO_CURRENT_VERSION
Current version of set message property options structure.

This is always an input field. The initial value of this field is **MQSMPO_VERSION_1**.

Initial values and language declarations for MQSMPO:

Table 250. Initial values of fields in MQSMPO

Field name	Name of constant	Value of constant
StrucId	MQSMPO_STRUC_ID	'SMPO'
Version	MQSMPO_VERSION_1	1
Options	MQSMPO_NONE	0
ValueEncoding	MQENC_NATIVE	Depends on environment
ValueCCSID	MQCCSI_APPL	-3

Notes:

1. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
2. In the C programming language, the macro variable MQSMPO_DEFAULT contains the values that are listed in the table. It can be used in the following way to provide initial values for the fields in the structure:

```
MQSMPO MySMPO = {MQSMPO_DEFAULT};
```

C declaration for MQSMPO:

```
typedef struct tagMQSMPO MQSMPO;
struct tagMQSMPO {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    Options;         /* Options that control the action of MQSETMP */
  MQLONG    ValueEncoding;   /* Encoding of Value */
  MQLONG    ValueCCSID;      /* Character set identifier of Value */
};
```

COBOL declaration for MQSMPO:

```
**   MQSMPO structure
  10 MQSMPO.
**    Structure identifier
   15 MQSMPO-STRUCID       PIC X(4).
**    Structure version number
   15 MQSMPO-VERSION       PIC S9(9) BINARY.
**    Options that control the action of MQSETMP
   15 MQSMPO-OPTIONS       PIC S9(9) BINARY.
**    Encoding of VALUE
   15 MQSMPO-VALUEENCODING PIC S9(9) BINARY.
**    Character set identifier of VALUE
   15 MQSMPO-VALUECCSID    PIC S9(9) BINARY.
```

PL/I declaration for MQSMPO:

```
dcl
 1 MQSMPO based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 Options        fixed bin(31), /* Options that control the action of MQSETMP */
  3 ValueEncoding  fixed bin(31), /* Encoding of Value */
  3 ValueCCSID     fixed bin(31), /* Character set identifier of Value */
```

High Level Assembler declaration for MQSMPO:

```
MQSMPO                DSECT
MQSMPO_STRUCID        DS   CL4   Structure identifier
MQSMPO_VERSION        DS   F     Structure version number
MQSMPO_OPTIONS        DS   F     Options that control the action of
*                                MQSETMP
MQSMPO_VALUEENCODING  DS   F     Encoding of VALUE
MQSMPO_VALUECCSID     DS   F     Character set identifier of VALUE
MQSMPO_LENGTH         EQU  *-MQSMPO
MQSMPO_AREA           DS   CL(MQSMPO_LENGTH)
```

MQSRO - Subscription request options:

This section describes subscription request options, what fields it contains, and initial values of those fields.

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options	Options
NumPubs	Number of publications	NumPubs

Overview for MQSRO:
Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, z/OS plus IBM MQ MQI clients connected to these systems.

Purpose: The MQSRO structure allows the application to specify options that control how a subscription request is made. The structure is an input/output parameter on the MQSUBRQ call.

Version: The current version of MQSRO is MQSRO_VERSION_1.

Character set and encoding: Data in MQSRO must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Fields for MQSRO:

The MQSRO structure contains the following fields; the fields are described in alphabetical order:

NumPubs (MQLONG):

This is an output field, returned to the application to indicate the number of publications sent to the subscription queue as a result of this call. Although this number of publications have been sent as a result of this call, there is no guarantee that this many messages will be available for the application to get, especially if they are non-persistent messages.

There might be more than one publication if the topic subscribed to contained a wildcard. If no wildcards were present in the topic string when the subscription represented by *Hsub* was created, then at most one publication is sent as a result of this call.

Options (MQLONG):

One of the following options must be specified. Only one option can be specified.

MQSRO_FAIL_IF_QUIESCING

The MQSUBRQ call fails if the queue manager is in the quiescing state. On z/OS, for a CICS or IMS application, this option also forces the MQSUBRQ call to fail if the connection is in a quiescing state.

Default option: If the option described previously is not required, the following option must be used:

MQSRO_NONE

Use this value to indicate that no other options have been specified; all options assume their default values.

MQSRO_NONE helps program documentation. Although it is not intended that this option be used with any other, because its value is zero, this use cannot be detected.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQSRO_STRUC_ID
Identifier for Subscription Request Options structure.

For the C programming language, the constant MQSRO_STRUC_ID_ARRAY is also defined; this has the same value as MQSRO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQSRO_STRUC_ID.

Version (MQLONG):

This is the structure version number; the value must be:

MQSRO_VERSION_1
Version-1 Subscription Request Options structure.

The following constant specifies the version number of the current version:

MQSRO_CURRENT_VERSION
Current version of Subscription Request Options structure.

This is always an input field. The initial value of this field is MQSRO_VERSION_1.

Initial values and language declarations for MQSRO:

Field name	Name of constant	Value of constant
StrucId	MQSRO_STRUC_ID	'SRO¬'
Version	MQSRO_VERSION_1	1
Options	MQSRO_NONE	0
NumPubs	None	0

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQSRO_DEFAULT contains the values that are listed in the table. It can be used in the following way to provide initial values for the fields in the structure:

```
MQSRO MySRO = {MQSRO_DEFAULT};
```

C declaration for MQSRO:

```
typedef struct tagMQSRO MQSRO;
struct tagMQSRO {
  MQCHAR4   StrucId;              /* Structure identifier */
  MQLONG    Version;              /* Structure version number */
  MQLONG    Options;              /* Options that control the action of MQSUBRQ */
  MQLONG    NumPubs;              /* Number of publications sent */
  /* Ver:1 */
};
```

COBOL declaration for MQSRO:

```
** MQSRO structure
 10  MQSRO.
** Structure identifier
 15  MQSRO-STRUCID             PIC X(4).
** Structure version number
 15  MQSRO-VERSION             PIC S9(9) BINARY.
** Options that control the action of MQSUBRQ
 15  MQSRO-OPTIONS             PIC S9(9) BINARY.
** Number of publications sent
 15  MQSRO-NUMPUBS             PIC S9(9) BINARY.
```

PL/I declaration for MQSRO:

```
dcl
 1 MQSRO based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 Options        fixed bin(31), /* Options that control the action of MQSUBRQ */
  3 NumPubs        fixed bin(31); /* Number of publications sent */
```

High Level Assembler declaration for MQSRO:

```
MQSRO                   DSECT
MQSRO_STRUCID           DS    CL4   Structure identifier
MQSRO_VERSION           DS    F     Structure version number
MQSRO_OPTIONS           DS    F     Options that control the action of MQSUBRQ
MQSRO_NUMPUBS           DS    F     Number of publications sent
*
MQSRO_LENGTH            EQU   *-MQSRO
                        ORG   MQSRO
MQSRO_AREA              DS    CL(MQSRO_LENGTH)
```

MQSTS - Status reporting structure:

The following table summarizes the fields in the structure.

Table 251. Fields in MQSTS

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
CompCode	Completion code of first error	CompCode
Reason	Reason code of first error	Reason
PutSuccessCount	Number of successful asynchronous put calls	SuccessCount
PutWarningcount	Number of asynchronous put calls which had warnings	WarningCount
PutFailureCount	Number of failed asynchronous put calls	FailureCount
ObjectType	Type of failing object	ObjectType
ObjectName	Name of failing object	ObjectName

Table 251. Fields in MQSTS (continued)

Field	Description	Topic
ObjectQMgrName	Name of queue manager owning the failing object	ObjectQMgrName
ResolvedObjectName	Resolved name of destination queue	ResolvedObjectName
ResolvedQMgrName	Resolved name of destination queue manager	ResolvedQMgrName
Note: The remaining fields are ignored if Version is less than MQSTS_VERSION_2.		
ObjectString	Long object name of failing object	ObjectString
SubName	Subscription name of failing subscription	SubName
OpenOptions	Open options associated with the failure	OpenOptions
SubOptions	Subscription options associated with the failure	SubOptions

Overview for MQSTS:

Purpose: The MQSTS structure is an output parameter from the MQSTAT command.

Character set and encoding: Character data in MQSTS is in the character set of the local queue manager; this is given by the *CodedCharSetId* queue manager attribute. Numeric data in MQSTS is in the native machine encoding; this is given by *Encoding*.

Usage: The MQSTAT command is used to retrieve status information. This information is returned in an MQSTS structure. For information about MQSTAT, see "MQSTAT - Retrieve status information" on page 2778.

Fields for MQSTS:

The MQSTS structure contains the following fields; the fields are described in **alphabetical order**:

CompCode (MQLONG):

The completion code of the operation being reported on.

The interpretation of `CompCode` depends on the value of the `MQSTAT` **`Type`** parameter.

MQSTAT_TYPE_ASYNC_ERROR

This is the completion code resulting from a previous asynchronous put operation on the object specified in `ObjectName`.

MQSTAT_TYPE_RECONNECTION

If the connection is reconnecting or failed to reconnect this is the completion code that caused the connection to begin reconnecting.

If the connection is currently connected the value is `MQCC_OK`.

MQSTAT_TYPE_RECONNECTION_ERROR

If the connection failed to reconnect this is the completion code that caused the reconnection to fail.

If the connection is currently connected, or reconnecting, the value is `MQCC_OK`.

`CompCode` is always an output field. Its initial value is `MQCC_OK`.

ObjectName (MQCHAR48):

The name of the object being reported on.

The interpretation of `ObjectName` depends on the value of the `MQSTAT` **`Type`** parameter.

MQSTAT_TYPE_ASYNC_ERROR

This is the name of the queue or topic used in the put operation, the failure of which is reported in the *`CompCode`* and *`Reason`* fields in the `MQSTS` structure.

MQSTAT_TYPE_RECONNECTION

If the connection is reconnecting, this is the name of the queue manager associated with the connection.

MQSTAT_TYPE_RECONNECTION_ERROR

If the connection failed to reconnect, this is the name of the object which caused reconnection to fail. The reason for the failure is reported in the *`CompCode`* and *`Reason`* fields in the `MQSTS` structure.

`ObjectName` is an output field. Its initial value is the null string in C, and 48 blank characters in other programming languages.

ObjectQMgrName (MQCHAR48):

The name of the queue manager being reported on.

The interpretation of `ObjectQMgrName` depends on the value of the `MQSTAT` **`Type`** parameter.

MQSTAT_TYPE_ASYNC_ERROR

This is the name of the queue manager on which the *`ObjectName`* object is defined. A name that is entirely blank up to the first null character or the end of the field denotes the queue manager to which the application is connected (the local queue manager).

MQSTAT_TYPE_RECONNECTION

Blank.

MQSTAT_TYPE_RECONNECTION_ERROR

If the connection failed to reconnect, this is the name of the object which caused reconnection to fail. The reason for the failure is reported in the *`CompCode`* and *`Reason`* fields in the `MQSTS` structure.

`ObjectQMgrName` is an output field. Its value is the null string in C, and 48 blank characters in other programming languages.

ObjectString (MQCHARV):

Long object name of failing object being reported on. Present only in Version 2 of MQSTS or higher.

The interpretation of ObjectString depends on the value of the MQSTAT **Type** parameter.

MQSTAT_TYPE_ASYNC_ERROR

This is the long object name of the queue or topic used in the MQPUT operation, which failed.

MQSTAT_TYPE_RECONNECTION

Zero length string

MQSTAT_TYPE_RECONNECTION_ERROR

This is the long object name of the object that caused the reconnection to fail.

ObjectString is an output field. Its initial value is a zero length string.

ObjectType (MQLONG):

The type of the object named in *ObjectName* being reported on.

Possible values of ObjectType are listed in "MQOT_* (Object Types and Extended Object Types)" on page 2135.

ObjectType is an output field. Its initial value is MQOT_Q.

OpenOptions (MQLONG):

The OpenOptions used to open the object being reported upon. Present only in Version 2 of MQSTS or higher.

The value of OpenOptions depends on the value of the MQSTAT **Type** parameter.

MQSTAT_TYPE_ASYNC_ERROR

Zero.

MQSTAT_TYPE_RECONNECTION

Zero.

MQSTAT_TYPE_RECONNECTION_ERROR

The OpenOptions used when the failure occurred. The reason for the failure is reported in the *CompCode* and *Reason* fields in the MQSTS structure.

OpenOptions is an output field. Its initial value is zero.

PutFailureCount (MQLONG):

The number of asynchronous put operations that failed.

The value of `PutFailureCount` depends on the value of the `MQSTAT` **Type** parameter.

MQSTAT_TYPE_ASYNC_ERROR

The number of asynchronous put operations to the object named in the `MQSTS` structure that completed with `MQCC_FAILED`.

MQSTAT_TYPE_RECONNECTION

Zero.

MQSTAT_TYPE_RECONNECTION_ERROR

Zero.

`PutFailureCount` is an output field. Its initial value is zero.

PutSuccessCount (MQLONG):

The number of asynchronous put operations that succeeded.

The value of `PutSuccessCount` depends on the value of the `MQSTAT` **Type** parameter.

MQSTAT_TYPE_ASYNC_ERROR

The number of asynchronous put operations to the object named in the `MQSTS` structure that completed with `MQCC_OK`.

MQSTAT_TYPE_RECONNECTION

Zero.

MQSTAT_TYPE_RECONNECTION_ERROR

Zero.

`PutSuccessCount` is an output field. Its initial value is zero.

PutWarningCount (MQLONG):

The number of asynchronous put operations that ended with a warning.

The value of `PutWarningCount` depends on the value of the `MQSTAT` **Type** parameter.

MQSTAT_TYPE_ASYNC_ERROR

The number of asynchronous put operations to the object named in the `MQSTS` structure that completed with `MQCC_WARNING`.

MQSTAT_TYPE_RECONNECTION

Zero.

MQSTAT_TYPE_RECONNECTION_ERROR

Zero.

`PutWarningCount` is an output field. Its initial value is zero.

SubName (MQCHARV):

The name of the failing subscription. Present only in Version 2 of MQSTS or higher.

The interpretation of SubName depends on the value of the MQSTAT **Type** parameter.

MQSTAT_TYPE_ASYNC_ERROR

Zero length string.

MQSTAT_TYPE_RECONNECTION

Zero length string.

MQSTAT_TYPE_RECONNECTION_ERROR

The name of the subscription that caused reconnection to fail. If no subscription name is available, or the failure is not related to a subscription, this is a zero-length string.

SubName is an output field. Its initial value is a zero length string.

SubOptions (MQLONG):

The SubOptions used to open the failing subscription. Present only in Version 2 of MQSTS or higher.

The interpretation of SubOptions depends on the value of the MQSTAT **Type** parameter.

MQSTAT_TYPE_ASYNC_ERROR

Zero.

MQSTAT_TYPE_RECONNECTION

Zero.

MQSTAT_TYPE_RECONNECTION_ERROR

The SubOptions used when the failure occurred. If the failure is not related to subscribing to a topic, the value returned is zero.

SubOptions is an output field. Its initial value is zero.

Reason (MQLONG):

The reason code of the operation being reported on.

The interpretation of Reason depends on the value of the MQSTAT **Type** parameter.

MQSTAT_TYPE_ASYNC_ERROR

This is the reason code resulting from a previous asynchronous put operation on the object specified in ObjectName.

MQSTAT_TYPE_RECONNECTION

If the connection is reconnecting or failed to reconnect this is the reason code that caused the reconnection to begin reconnecting.

If the connection is currently connected the value is MQRC_NONE.

MQSTAT_TYPE_RECONNECTION_ERROR

If the connection failed to reconnect this is the reason code that caused the reconnection to fail.

If the connection is currently connected, or reconnecting, the value is MQRC_NONE.

Reason is an output field. Its initial value is MQRC_NONE.

ResolvedObjectName (MQCHAR48):

The name of the object named in *ObjectName* after the local queue manager resolves the name.

The interpretation of ResolvedObjectName depends on the value of the MQSTAT **Type** parameter.

MQSTAT_TYPE_ASYNC_ERROR

ResolvedObjectName is the name of the object named in *ObjectName* after the local queue manager resolves the name. The name returned is the name of an object that exists on the queue manager identified by *ResolvedQMgrName*.

MQSTAT_TYPE_RECONNECTION

Blank.

MQSTAT_TYPE_RECONNECTION_ERROR

Blank.

ResolvedObjectName is an output field. Its initial value is the null string in C, and 48 blank characters in other programming languages.

ResolvedQMgrName (MQCHAR48):

The name of the destination queue manager after the local queue manager resolves the name.

The interpretation of ResolvedQMgrName depends on the value of the MQSTAT **Type** parameter.

MQSTAT_TYPE_ASYNC_ERROR

ResolvedQMgrName is the name of the destination queue manager after the local queue manager resolves the name. The name returned is the name of the queue manager that owns the object identified by *ResolvedObjectName*. *ResolvedQMgrName* might be the name of the local queue manager.

MQSTAT_TYPE_RECONNECTION

Blank.

MQSTAT_TYPE_RECONNECTION_ERROR

Blank.

ResolvedQMgrName is always an output field. Its initial value is the null string in C, and 48 blank characters in other programming languages.

StrucId (MQCHAR4):

The identifier for the status reporting structure, MQSTS.

StrucId is the structure identifier. The value must be:

MQSTS_STRUC_ID
Identifier for status reporting structure.

For the C programming language, the constant MQSTS_STRUC_ID_ARRAY is also defined; this has the same value as MQSTS_STRUC_ID, but is an array of characters instead of a string.

StrucId is always an input field. Its initial value is MQSTS_STRUC_ID.

Version (MQLONG):

The structure version number.

The value must be either:

MQSTS_VERSION_1
Version 1 status reporting structure.

MQSTS_VERSION_2
Version 2 status reporting structure.

The following constant specifies the version number of the current version:

MQSTS_CURRENT_VERSION
Current version of status reporting structure. The current version is MQSTS_VERSION_2.

Version is always an input field. Its initial value is MQSTS_VERSION_1.

Initial values and language declarations for MQSTS:

Table 252. Initial values of fields in MQSTS

Field name	Name of constant	Value of constant
StrucId	MQSTS_STRUC_ID	'STAT¬'
Version	MQSTS_VERSION_1	1
CompCode	MQCC_OK	0
Reason	MQRC_NONE	0
PutSuccessCount	None	0
PutWarningCount	None	0
PutFailureCount	None	0
ObjectType	MQOT_Q	1
ObjectName	None	Null string or blanks
ObjectQMgrName	None	Null string or blanks
ResolvedObjectName	None	Null string or blanks
ResolvedQMgrName	None	Null string or blanks
ObjectString	MQCHARV_DEFAULT	{NULL,0,0,0,-3}
SubName	MQCHARV_DEFAULT	{NULL,0,0,0,-3}
OpenOptions	None	0
SubOptions	None	0

Table 252. Initial values of fields in MQSTS (continued)

Field name	Name of constant	Value of constant
Notes: 1. The symbol ¬ represents a single blank character. 2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages. 3. In the C programming language, the macro variable MQSTS_DEFAULT contains the values that are listed in the table. It can be used in the following way to provide initial values for the fields in the structure: `MQSTS MySTS = {MQSTS_DEFAULT};`		

C declaration for MQSTS:

```
typedef struct tagMQSTS MQSTS;
struct tagMQSTS {
 MQCHAR4   StrucId;            /* Structure identifier */
 MQLONG    Version;            /* Structure version number */
 MQLONG    CompCode;           /* Completion Code of first error */
 MQLONG    Reason;             /* Reason Code of first error */
 MQLONG    PutSuccessCount;    /* Number of Async calls succeeded */
 MQLONG    PutWarningCount;    /* Number of Async calls had warnings */
 MQLONG    PutFailureCount;    /* Number of Async calls had failures */
 MQLONG    ObjectType;         /* Failing object type */
 MQCHAR48  ObjectName;         /* Failing object name */
 MQCHAR48  ObjectQMgrName;     /* Failing object queue manager name */
 MQCHAR48  ResolvedObjectName; /* Resolved name of destination queue */
 MQCHAR48  ResolvedQMgrName;   /* Resolved name of destination qmgr */
/* Ver:1 */
 MQCHARV   ObjectString;       /* Failing object long name */
 MQCHARV   SubName;            /* Failing subscription name */
 MQLONG    OpenOptions;        /* Failing open options */
 MQLONG    SubOptions;         /* Failing subscription options */
/* Ver:2 */
};
```

COBOL declaration for MQSTS:

```
      ** MQSTS structure
       10  MQSTS.
      ** Structure identifier
       15  MQSTS-STRUCID PIC X(4).
      ** Structure version number
       15  MQSTS-VERSION PIC S9(9) BINARY.
      ** Completion Code of first error
       15  MQSTS-COMPCODE PIC S9(9) BINARY.
      ** Reason Code of first error
       15  MQSTS-REASON PIC S9(9) BINARY.
      ** Number of Async put calls succeeded
       15  MQSTS-PUTSUCCESSCOUNT PIC S9(9) BINARY.
      ** Number of Async put calls had warnings
       15  MQSTS-PUTWARNINGCOUNT PIC S9(9) BINARY.
      ** Number of Async put calls had failures
       15  MQSTS-PUTFAILURECOUNT PIC S9(9) BINARY.
      ** Failing object type
       15  MQSTS-OBJECTTYPE PIC S9(9) BINARY.
      ** Failing object name
       15  MQSTS-OBJECTNAME PIC X(48).
      ** Failing object queue manager
       15  MQSTS-OBJECTQMGRNAME PIC X(48).
      ** Resolved name of destination queue
       15  MQSTS-RESOLVEDOBJECTNAME PIC X(48).
      ** Resolved name of destination qmgr
```

```
      15  MQSTS-RESOLVEDQMGRNAME PIC X(48).
     ** Ver:1 **
     ** Failing object long name
      15  MQSTS-OBJECTSTRING.
     ** Address of variable length string
      20  MQSTS-OBJECTSTRING-VSPTR POINTER.
     ** Offset of variable length string
      20  MQSTS-OBJECTSTRING-VSOFFSET PIC S9(9) BINARY.
     ** Size of buffer
      20  MQSTS-OBJECTSTRING-VSBUFSIZE PIC S9(9) BINARY.
     ** Length of variable length string
      20  MQSTS-OBJECTSTRING-VSLENGTH PIC S9(9) BINARY.
     ** CCSID of variable length string
      20  MQSTS-OBJECTSTRING-VSCCSID PIC S9(9) BINARY.
     ** Failing subscription name
      15  MQSTS-SUBNAME.
     ** Address of variable length string
      20  MQSTS-SUBNAME-VSPTR POINTER.
     ** Offset of variable length string
      20  MQSTS-SUBNAME-VSOFFSET PIC S9(9) BINARY.
     ** Size of buffer
      20  MQSTS-SUBNAME-VSBUFSIZE PIC S9(9) BINARY.
     ** Length of variable length string
      20  MQSTS-SUBNAME-VSLENGTH PIC S9(9) BINARY.
     ** CCSID of variable length string
      20  MQSTS-SUBNAME-VSCCSID PIC S9(9) BINARY.
     ** Failing open options
      15  MQSTS-OPENOPTIONS PIC S9(9) BINARY.
     ** Failing subscription options
      15  MQSTS-SUBOPTIONS PIC S9(9) BINARY.
     ** Ver:2 **
```

PL/I declaration for MQSTS:

```
dcl
   1 MQSTS based,
     3 StrucId             char(4),       /* Structure identifier */
      3 Version            fixed bin(31), /* Structure version number */
      3 CompCode           fixed bin(31), /* Completion code */
      3 Reason             fixed bin(31), /* Reason code */
      3 PutSuccessCount    fixed bin(31), /* Put success count */
      3 PutWarningCount    fixed bin(31), /* Put warning count */
      3 PutFailureCount    fixed bin(31), /* Put failure count */
      3 ObjectType         fixed bin(31), /* Object type */
      3 ObjectName         char(48), /* Object name */
      3 ObjectQmgrName     char(48), /* Object queue manager */
      3 ResolvedObjectName char(48), /* Resolved Object name */
      3 ResolvedQmgrName   char(48); /* Resolved Object queue manager */
  /* Ver:1 */
      3 ObjectString,                /* Failing object long name */
         5 VSPtr pointer,            /* Address of variable length string */
         5 VSOffset fixed bin(31),   /* Offset of variable length string */
         5 VSBufSize fixed bin(31),  /* Size of buffer */
         5 VSLength fixed bin(31),   /* Length of variable length string */
         5 VSCCSID fixed bin(31);    /* CCSID of variable length string */
      3 SubName,                     /* Failing subscription name */
         5 VSPtr pointer,            /* Address of variable length string */
         5 VSOffset fixed bin(31),   /* Offset of variable length string */
         5 VSBufSize fixed bin(31),  /* Size of buffer */
         5 VSLength fixed bin(31),   /* Length of variable length string */
         5 VSCCSID fixed bin(31);    /* CCSID of variable length string */
     3 OpenOptions fixed bin(31),    /* Failing open options */
     3 SubOptions fixed bin(31);     /* Failing subscription options */
  /* Ver:2 */
```

High Level Assembler declaration for MQSTS:

```
MQSTS                          DSECT
MQSTS_STRUCID                  DS    CL4   Structure identifier
MQSTS_VERSION                  DS    F     Structure version number
MQSTS_COMPCODE                 DS    F     Completion code
MQSTS_REASON                   DS    F     Reason code
MQSTS_PUTSUCCESSCOUNT          DS    F     Success count
MQSTS_PUTWARNINGCOUNT          DS    F     Warning count
MQSTS_PUTFAILURECOUNT          DS    F     Failure count
MQSTS_OBJTYPE                  DS    F     Object type
MQSTS_OBJNAME                  DS    CL48  Object name
MQSTS_OBJQMGR                  DS    CL48  Object queue manager
MQSTS_ROBJNAME                 DS    CL48  Resolved object name
MQSTS_ROBJQMGR                 DS    CL48  Resolved object queue manager
MQSTS_OBJECTSTRING             DS    0F   Force fullword alignment
MQSTS_OBJECTSTRING_VSPTR       DS    A  Address of variable length string
MQSTS_OBJECTSTRING_VSOFFSET    DS    F  Offset of variable length string
MQSTS_OBJECTSTRING_VSBUFSIZE   DS    F  Size of buffer
MQSTS_OBJECTSTRING_VSLENGTH    DS    F  Length of variable length string
MQSTS_OBJECTSTRING_VSCCSID     DS    F  CCSID of variable length string
MQSTS_OBJECTSTRING_LENGTH      EQU   *-MQSTS_OBJECTSTRING
                               ORG   MQSTS_OBJECTSTRING
MQSTS_OBJECTSTRING_AREA        DS    CL(MQSTS_OBJECTSTRING_LENGTH)
*
MQSTS_SUBNAME                  DS    0F Force fullword alignment
MQSTS_SUBNAME_VSPTR            DS    A  Address of variable length string
MQSTS_SUBNAME_VSOFFSET         DS    F  Offset of variable length string
MQSTS_SUBNAME_VSBUFSIZE        DS    F  Size of buffer
MQSTS_SUBNAME_VSLENGTH         DS    F  Length of variable length string
MQSTS_SUBNAME_VSCCSID          DS    F  CCSID of variable length string
MQSTS_SUBNAME_LENGTH           EQ    *-MQSTS_SUBNAME
                               ORG   MQSTS_SUBNAME
MQSTS_SUBNAME_AREA             DS    CL(MQSTS_SUBNAME_LENGTH)
*
MQSTS_OPENOPTIONS              DS    F  Failing open options
MQSTS_SUBOPTIONS               DS    F  Failing subscription option
MQSTS_LENGTH                   EQU   *-MQSTS
                               ORG   MQSTS
MQSTS_AREA                     DS    CL(MQSTS_LENGTH)
```

MQTM - Trigger message:

The following table summarizes the fields in the structure.

Table 253. Fields in MQTM

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
QName	Name of triggered queue	QName
ProcessName	Name of process object	ProcessName
TriggerData	Trigger data	TriggerData
ApplType	Application type	ApplType
ApplId	Application identifier	ApplId
EnvData	Environment data	EnvData
UserData	User data	UserData

Overview for MQTM:

Purpose: The MQTM structure describes the data in the trigger message that is sent by the queue manager to a trigger-monitor application when a trigger event occurs for a queue.

This structure is part of the IBM MQ Trigger Monitor Interface (TMI), which is one of the IBM MQ framework interfaces.

Format name: MQFMT_TRIGGER.

Character set and encoding: Character data in MQTM is in the character set of the queue manager that generates the MQTM. Numeric data in MQTM is in the machine encoding of the queue manager that generates the MQTM.

The character set and encoding of the MQTM are given by the *CodedCharSetId* and *Encoding* fields in:

- The MQMD (if the MQTM structure is at the start of the message data), or
- The header structure that precedes the MQTM structure (all other cases).

Usage: A trigger-monitor application might need to pass some or all of the information in the trigger message to the application that the trigger-monitor application starts. Information that might be needed by the started application includes *QName*, *TriggerData*, and *UserData*. The trigger-monitor application can pass the MQTM structure directly to the started application, or pass an MQTMC2 structure instead, depending on what is permitted by the environment and convenient for the started application. For information about MQTMC2, see "MQTMC2 - Trigger message 2 (character format)" on page 2596.

- On z/OS, for an MQAT_CICS application that is started using the CKTI transaction, the entire trigger message structure MQTM is made available to the started transaction; the information can be retrieved by using the EXEC CICS RETRIEVE command.
- On IBM i, the trigger-monitor application provided with IBM MQ passes an MQTMC2 structure to the started application.

For information about using triggers, see Starting IBM MQ applications using triggers.

MQMD for a trigger message: The fields in the MQMD of a trigger message generated by the queue manager are set as follows:

Field in MQMD	Value used
StrucId	MQMD_STRUC_ID
Version	MQMD_VERSION_1
Report	MQRO_NONE
MsgType	MQMT_DATAGRAM
Expiry	MQEI_UNLIMITED
Feedback	MQFB_NONE
Encoding	MQENC_NATIVE
CodedCharSetId	Queue manager's **CodedCharSetId** attribute
Format	MQFMT_TRIGGER
Priority	Initiation queue's **DefPriority** attribute
Persistence	MQPER_NOT_PERSISTENT
MsgId	A unique value
CorrelId	MQCI_NONE
BackoutCount	0
ReplyToQ	Blanks
ReplyToQMgr	Name of queue manager
UserIdentifier	Blanks
AccountingToken	MQACT_NONE
ApplIdentityData	Blanks

Field in MQMD	Value used
PutApplType	MQAT_QMGR, or as appropriate for the message channel agent
PutApplName	First 28 bytes of the queue manager name
PutDate	Date when trigger message is sent
PutTime	Time when trigger message is sent
ApplOriginData	Blanks

An application that generates a trigger message is recommended to set similar values, except for the following:

- The *Priority* field can be set to MQPRI_PRIORITY_AS_Q_DEF (the queue manager will change this to the default priority for the initiation queue when the message is put).
- The *ReplyToQMgr* field can be set to blanks (the queue manager will change this to the name of the local queue manager when the message it put).
- Set the context fields as appropriate for the application.

Fields for MQTM:

The MQTM structure contains the following fields; the fields are described in **alphabetical order**:

ApplId (MQCHAR256):

This is a character string that identifies the application to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the **ApplId** attribute of the process object identified by the *ProcessName* field; see "Attributes for process definitions" on page 2872 for details of this attribute. The content of this data is of no significance to the queue manager.

The meaning of *ApplId* is determined by the trigger-monitor application. The trigger monitor provided by IBM MQ requires *ApplId* to be the name of an executable program. The following notes apply to the environments indicated:

- On z/OS, *ApplId* is:
  - A CICS transaction identifier, for applications started using the CICS trigger-monitor transaction CKTI
  - An IMS transaction identifier, for applications started using the IMS trigger monitor CSQQTRMN
- On Windows systems, the program name can be prefixed with a drive and directory path.
- On IBM i, the program name can be prefixed with a library name and / character.
- On UNIX, the program name can be prefixed with a directory path.

The length of this field is given by MQ_PROCESS_APPL_ID_LENGTH. The initial value of this field is the null string in C, and 256 blank characters in other programming languages.

ApplType (MQLONG):

This identifies the nature of the program to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the **ApplType** attribute of the process object identified by the *ProcessName* field; see "Attributes for process definitions" on page 2872 for details of this attribute. The content of this data is of no significance to the queue manager.

ApplType can have one of the following standard values. User-defined types can also be used, but should be restricted to values in the range MQAT_USER_FIRST through MQAT_USER_LAST:

MQAT_AIX
AIX application (same value as MQAT_UNIX).

MQAT_BATCH
Batch application

MQAT_BROKER
Broker application

MQAT_CICS
CICS transaction.

MQAT_CICS_BRIDGE
CICS bridge application.

MQAT_CICS_VSE
CICS/VSE transaction.

MQAT_DOS
IBM MQ MQI client application on PC DOS.

MQAT_IMS
IMS application.

MQAT_IMS_BRIDGE
IMS bridge application.

MQAT_JAVA
Java application.

MQAT_MVS
MVS or TSO application (same value as MQAT_ZOS).

MQAT_NOTES_AGENT
Lotus Notes Agent application.

MQAT_NSK
HP Integrity NonStop Server application.

MQAT_OS390
OS/390 application (same value as MQAT_ZOS).

MQAT_OS400
IBM i application.

MQAT_RRS_BATCH
RRS batch application.

MQAT_UNIX
UNIX application.

MQAT_UNKNOWN
Application of unknown type.

MQAT_USER
User-defined application type.

MQAT_VOS
Stratus VOS application.

MQAT_WINDOWS
16-bit Windows application.

MQAT_WINDOWS_NT
32-bit Windows application.

MQAT_WLM
z/OS workload manager application.

MQAT_XCF
XCF.

MQAT_ZOS
z/OS application.

MQAT_USER_FIRST
Lowest value for user-defined application type.

MQAT_USER_LAST
Highest value for user-defined application type.

The initial value of this field is `0`.

EnvData (MQCHAR128):

This is a character string that contains environment-related information pertaining to the application to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the **EnvData** attribute of the process object identified by the *ProcessName* field; see "Attributes for process definitions" on page 2872 for details of this attribute. The content of this data is of no significance to the queue manager.

On z/OS, for a CICS application started using the CKTI transaction, or an IMS application to be started using the CSQQTRMN transaction, this information is not used.

The length of this field is given by MQ_PROCESS_ENV_DATA_LENGTH. The initial value of this field is the null string in C, and 128 blank characters in other programming languages.

ProcessName (MQCHAR48):

This is the name of the queue manager process object specified for the triggered queue, and can be used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the **ProcessName** attribute of the queue identified by the *QName* field; see"Attributes for queues" on page 2833for details of this attribute.

Names that are shorter than the defined length of the field are always padded to the right with blanks; they are not ended prematurely by a null character.

The length of this field is given by MQ_PROCESS_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

QName (MQCHAR48):

This is the name of the queue for which a trigger event occurred, and is used by the application started by the trigger-monitor application. The queue manager initializes this field with the value of the **QName** attribute of the triggered queue; see"Attributes for queues" on page 2833for details of this attribute.

Names that are shorter than the defined length of the field are padded to the right with blanks; they are not ended prematurely by a null character.

The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

StrucId (MQCHAR4):

This is the structure identifier. The value must be:

MQTM_STRUC_ID
Identifier for trigger message structure.

For the C programming language, the constant MQTM_STRUC_ID_ARRAY is also defined; this has the same value as MQTM_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQTM_STRUC_ID.

TriggerData (MQCHAR64):

This is free-format data for use by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the **TriggerData** attribute of the queue identified by the *QName* field; see "Attributes for queues" on page 2833 for details of this attribute. The content of this data is of no significance to the queue manager.

On z/OS, for a CICS application started using the CKTI transaction, this information is not used.

The length of this field is given by MQ_TRIGGER_DATA_LENGTH. The initial value of this field is the null string in C, and 64 blank characters in other programming languages.

UserData (MQCHAR128):

This is a character string that contains user information relevant to the application to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the **UserData** attribute of the process object identified by the *ProcessName* field; see "Attributes for process definitions" on page 2872 for details of this attribute. The content of this data is of no significance to the queue manager.

For Microsoft Windows, the character string must not contain double quotation marks if the process definition is going to be passed to **runmqtrm**.

The length of this field is given by MQ_PROCESS_USER_DATA_LENGTH. The initial value of this field is the null string in C, and 128 blank characters in other programming languages.

Version (MQLONG):

This is the structure version number. The value must be:

MQTM_VERSION_1
Version number for trigger message structure.

The following constant specifies the version number of the current version:

MQTM_CURRENT_VERSION
Current version of trigger message structure.

The initial value of this field is MQTM_VERSION_1.

Initial values and language declarations for MQTM:

Table 254. Initial values of fields in MQTM for MQTM

Field name	Name of constant	Value of constant
StrucId	MQTM_STRUC_ID	'TM¬¬'
Version	MQTM_VERSION_1	1
QName	None	Null string or blanks
ProcessName	None	Null string or blanks
TriggerData	None	Null string or blanks
ApplType	None	0
ApplId	None	Null string or blanks
EnvData	None	Null string or blanks
UserData	None	Null string or blanks

Notes:

1. The symbol ¬ represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQTM_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure:

```
MQTM MyTM = {MQTM_DEFAULT};
```

C declaration for MQTM:

```
typedef struct tagMQTM MQTM;
struct tagMQTM {
  MQCHAR4    StrucId;      /* Structure identifier */
  MQLONG     Version;      /* Structure version number */
  MQCHAR48   QName;        /* Name of triggered queue */
  MQCHAR48   ProcessName;  /* Name of process object */
  MQCHAR64   TriggerData;  /* Trigger data */
  MQLONG     ApplType;     /* Application type */
  MQCHAR256  ApplId;       /* Application identifier */
  MQCHAR128  EnvData;      /* Environment data */
  MQCHAR128  UserData;     /* User data */
};
```

COBOL declaration for MQTM:

```
**   MQTM structure
  10 MQTM.
**    Structure identifier
   15 MQTM-STRUCID     PIC X(4).
**    Structure version number
   15 MQTM-VERSION     PIC S9(9) BINARY.
**    Name of triggered queue
   15 MQTM-QNAME       PIC X(48).
**    Name of process object
   15 MQTM-PROCESSNAME PIC X(48).
**    Trigger data
   15 MQTM-TRIGGERDATA PIC X(64).
**    Application type
   15 MQTM-APPLTYPE    PIC S9(9) BINARY.
**    Application identifier
   15 MQTM-APPLID      PIC X(256).
**    Environment data
   15 MQTM-ENVDATA     PIC X(128).
**    User data
   15 MQTM-USERDATA    PIC X(128).
```

PL/I declaration for MQTM:

```
dcl
 1 MQTM based,
  3 StrucId     char(4),       /* Structure identifier */
  3 Version     fixed bin(31), /* Structure version number */
  3 QName       char(48),      /* Name of triggered queue */
  3 ProcessName char(48),      /* Name of process object */
  3 TriggerData char(64),      /* Trigger data */
  3 ApplType    fixed bin(31), /* Application type */
  3 ApplId      char(256),     /* Application identifier */
  3 EnvData     char(128),     /* Environment data */
  3 UserData    char(128);     /* User data */
```

High Level Assembler declaration for MQTM:

```
MQTM              DSECT
MQTM_STRUCID      DS   CL4    Structure identifier
MQTM_VERSION      DS   F      Structure version number
MQTM_QNAME        DS   CL48   Name of triggered queue
MQTM_PROCESSNAME  DS   CL48   Name of process object
MQTM_TRIGGERDATA  DS   CL64   Trigger data
MQTM_APPLTYPE     DS   F      Application type
MQTM_APPLID       DS   CL256  Application identifier
MQTM_ENVDATA      DS   CL128  Environment data
MQTM_USERDATA     DS   CL128  User data
*
MQTM_LENGTH       EQU  *-MQTM
                  ORG  MQTM
MQTM_AREA         DS   CL(MQTM_LENGTH)
```

Visual Basic declaration for MQTM:

```
Type MQTM
  StrucId     As String*4   'Structure identifier'
  Version     As Long       'Structure version number'
  QName       As String*48  'Name of triggered queue'
  ProcessName As String*48  'Name of process object'
  TriggerData As String*64  'Trigger data'
  ApplType    As Long       'Application type'
  ApplId      As String*256 'Application identifier'
  EnvData     As String*128 'Environment data'
  UserData    As String*128 'User data'
End Type
```

MQTMC2 - Trigger message 2 (character format):

The following table summarizes the fields in the structure.

Table 255. Fields in MQTMC2

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
QName	Name of triggered queue	QName
ProcessName	Name of process object	ProcessName
TriggerData	Trigger data	TriggerData
ApplType	Application type	ApplType
ApplId	Application identifier	ApplId
EnvData	Environment data	EnvData
UserData	User data	UserData
QMgrName	Queue manager name	QMgrName

Overview for MQTMC2:

Purpose: When a trigger-monitor application retrieves a trigger message (MQTM) from an initiation queue, the trigger monitor might need to pass some or all of the information in the trigger message to the application that the trigger monitor starts.

Information that the started application might need includes *QName*, *TriggerData*, and *UserData*. The trigger monitor application can pass the MQTM structure directly to the started application, or pass an MQTMC2 structure instead, depending on what is permitted by the environment and convenient for the started application.

This structure is part of the IBM MQ Trigger Monitor Interface (TMI), which is one of the IBM MQ framework interfaces.

Character set and encoding: Character data in MQTMC2 is in the character set of the local queue manager; this is given by the **CodedCharSetId** queue manager attribute.

Usage: The MQTMC2 structure is very similar to the format of the MQTM structure. The difference is that the non-character fields in MQTM are changed in MQTMC2 to character fields of the same length, and the queue manager name is added at the end of the structure.

- On z/OS, for an MQAT_IMS application that is started using the CSQQTRMN application, an MQTMC2 structure is made available to the started application.
- On IBM i, the trigger monitor application provided with IBM MQ passes an MQTMC2 structure to the started application.

Fields for MQTMC2:

The MQTMC2 structure contains the following fields; the fields are described in **alphabetical order**:

ApplId (MQCHAR256):

Application identifier.

See the *ApplId* field in the MQTM structure.

ApplType (MQCHAR4):

Application type.

This field always contains blanks, whatever the value in the *ApplType* field in the MQTM structure of the original trigger message.

EnvData (MQCHAR128):

Environment data.

See the *EnvData* field in the MQTM structure.

ProcessName (MQCHAR48):

Name of process object.

See the *ProcessName* field in the MQTM structure.

QMgrName (MQCHAR48):

Queue manager name.

This is the name of the queue manager at which the trigger event occurred.

QName (MQCHAR48):

Name of triggered queue.

See the *QName* field in the MQTM structure.

StrucId (MQCHAR4):

Structure identifier.

The value must be:

MQTMC_STRUC_ID
Identifier for trigger message (character format) structure.

For the C programming language, the constant MQTMC_STRUC_ID_ARRAY is also defined; this has the same value as MQTMC_STRUC_ID, but is an array of characters instead of a string.

TriggerData (MQCHAR64):

Trigger data.

See the *TriggerData* field in the MQTM structure.

UserData (MQCHAR128):

User data.

See the *UserData* field in the MQTM structure.

Version (MQCHAR4):

Structure version number.

The value must be:

MQTMC_VERSION_2
Version 2 trigger message (character format) structure.

For the C programming language, the constant MQTMC_VERSION_2_ARRAY is also defined; this has the same value as MQTMC_VERSION_2, but is an array of characters instead of a string.

The following constant specifies the version number of the current version:

MQTMC_CURRENT_VERSION
Current version of trigger message (character format) structure.

Initial values and language declarations for MQTMC2:

Table 256. Initial values of fields in MQTMC2 for MQTMC2

Field name	Name of constant	Value of constant
StrucId	MQTMC_STRUC_ID	'TMC⌂'
Version	MQTMC_VERSION_2	'⌂⌂2'
QName	None	Null string or blanks
ProcessName	None	Null string or blanks
TriggerData	None	Null string or blanks
ApplType	None	Blanks
ApplId	None	Null string or blanks
EnvData	None	Null string or blanks
UserData	None	Null string or blanks
QMgrName	None	Null string or blanks

Notes:

1. The symbol ⌂ represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQTMC2_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

```
MQTMC2 MyTMC = {MQTMC2_DEFAULT};
```

C declaration for MQTMC2:

```
typedef struct tagMQTMC2 MQTMC2;
struct tagMQTMC2 {
  MQCHAR4    StrucId;      /* Structure identifier */
  MQCHAR4    Version;      /* Structure version number */
  MQCHAR48   QName;        /* Name of triggered queue */
  MQCHAR48   ProcessName;  /* Name of process object */
  MQCHAR64   TriggerData;  /* Trigger data */
  MQCHAR4    ApplType;     /* Application type */
  MQCHAR256  ApplId;       /* Application identifier */
  MQCHAR128  EnvData;      /* Environment data */
  MQCHAR128  UserData;     /* User data */
  MQCHAR48   QMgrName;     /* Queue manager name */
};
```

COBOL declaration for MQTMC2:

```
**   MQTMC2 structure
  10 MQTMC2.
**    Structure identifier
   15 MQTMC2-STRUCID     PIC X(4).
**    Structure version number
   15 MQTMC2-VERSION     PIC X(4).
**    Name of triggered queue
   15 MQTMC2-QNAME       PIC X(48).
**    Name of process object
   15 MQTMC2-PROCESSNAME PIC X(48).
**    Trigger data
   15 MQTMC2-TRIGGERDATA PIC X(64).
**    Application type
   15 MQTMC2-APPLTYPE    PIC X(4).
**    Application identifier
   15 MQTMC2-APPLID      PIC X(256).
**    Environment data
   15 MQTMC2-ENVDATA     PIC X(128).
**    User data
   15 MQTMC2-USERDATA    PIC X(128).
**    Queue manager name
   15 MQTMC2-QMGRNAME    PIC X(48).
```

PL/I declaration for MQTMC2:

```
dcl
 1 MQTMC2 based,
  3 StrucId     char(4),   /* Structure identifier */
  3 Version     char(4),   /* Structure version number */
  3 QName       char(48),  /* Name of triggered queue */
  3 ProcessName char(48),  /* Name of process object */
  3 TriggerData char(64),  /* Trigger data */
  3 ApplType    char(4),   /* Application type */
  3 ApplId      char(256), /* Application identifier */
  3 EnvData     char(128), /* Environment data */
  3 UserData    char(128), /* User data */
  3 QMgrName    char(48);  /* Queue manager name */
```

High Level Assembler declaration for MQTMC2:

```
MQTMC2              DSECT
MQTMC2_STRUCID      DS   CL4    Structure identifier
MQTMC2_VERSION      DS   CL4    Structure version number
MQTMC2_QNAME        DS   CL48   Name of triggered queue
MQTMC2_PROCESSNAME  DS   CL48   Name of process object
MQTMC2_TRIGGERDATA  DS   CL64   Trigger data
MQTMC2_APPLTYPE     DS   CL4    Application type
MQTMC2_APPLID       DS   CL256  Application identifier
MQTMC2_ENVDATA      DS   CL128  Environment data
MQTMC2_USERDATA     DS   CL128  User data
MQTMC2_QMGRNAME     DS   CL48   Queue manager name
*
MQTMC2_LENGTH       EQU  *-MQTMC2
                    ORG  MQTMC2
MQTMC2_AREA         DS   CL(MQTMC2_LENGTH)
```

Visual Basic declaration for MQTMC2:

```
Type MQTMC2
  StrucId     As String*4   'Structure identifier'
  Version     As String*4   'Structure version number'
  QName       As String*48  'Name of triggered queue'
  ProcessName As String*48  'Name of process object'
  TriggerData As String*64  'Trigger data'
  ApplType    As String*4   'Application type'
  ApplId      As String*256 'Application identifier'
  EnvData     As String*128 'Environment data'
  UserData    As String*128 'User data'
  QMgrName    As String*48  'Queue manager name'
End Type
```

MQWIH - Work information header:

The following table summarizes the fields in the structure.

Table 257. Fields in MQWIH

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
StrucLength	Length of MQWIH structure	StrucLength
Encoding	Numeric encoding of data that follows MQWIH	Encoding
CodedCharSetId	Character-set identifier of data that follows MQWIH	CodedCharSetId
Format	Format name of data that follows MQWIH	Format
Flags	Flags	Flags
ServiceName	Service name	ServiceName
ServiceStep	Service step name	ServiceStep
MsgToken	Message token	MsgToken
Reserved	Reserved	Reserved

Overview for MQWIH:

The MQWIH structure describes the header information for a message sent to z/OS workload management (WLM). For any IBM MQ supported platform you can create and transmit a message that includes the MQWIH structure, but only an IBM MQ for z/OS queue manager can interact with WLM. Therefore, for the message to get to WLM from a non-z/OS queue manager, your queue manager network must include at least one z/OS queue manager through which the message can be routed.

Availability: All IBM MQ systems, plus IBM MQ clients connected to these systems.

Purpose: The MQWIH structure describes the information that must be present at the start of a message that is to be handled by the z/OS workload manager.

Format name: MQFMT_WORK_INFO_HEADER.

Character set and encoding: The fields in the MQWIH structure are in the character set and encoding given by the *CodedCharSetId* and *Encoding* fields in the header structure that precedes MQWIH, or by those fields in the MQMD structure if the MQWIH is at the start of the application message data.

The character set must be one that has single-byte characters for the characters that are valid in queue names.

Usage: If a message is to be processed by the z/OS workload manager, the message must begin with an MQWIH structure.

Fields for MQWIH:

The MQWIH structure contains the following fields; the fields are described in **alphabetical order**:

CodedCharSetId (MQLONG):

This specifies the character set identifier of the data that follows the MQWIH structure; it does not apply to character data in the MQWIH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. You can use the following special value:

MQCCSI_INHERIT
Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

MQCCSI_INHERIT cannot be used if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG):

This specifies the numeric encoding of the data that follows the MQWIH structure; it does not apply to numeric data in the MQWIH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is 0.

Flags (MQLONG):

The value must be:

MQWIH_NONE
No flags.

The initial value of this field is MQWIH_NONE.

Format (MQCHAR8):

This specifies the format name of the data that follows the MQWIH structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *Format* field in MQMD.

The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

MsgToken (MQBYTE16):

This is a message token that uniquely identifies the message.

For the MQPUT and MQPUT1 calls, this field is ignored. The length of this field is given by MQ_MSG_TOKEN_LENGTH. The initial value of this field is MQMTOK_NONE.

Reserved (MQCHAR32):

This is a reserved field; it must be blank.

ServiceName (MQCHAR32):

This is the name of the service that is to process the message.

The length of this field is given by MQ_SERVICE_NAME_LENGTH. The initial value of this field is 32 blank characters.

ServiceStep (MQCHAR8):

This is the name of the step of *ServiceName* to which the message relates.

The length of this field is given by MQ_SERVICE_STEP_LENGTH. The initial value of this field is 8 blank characters.

StrucId (MQCHAR4):

This is the structure identifier. The value must be:

MQWIH_STRUC_ID
Identifier for work information header structure.

For the C programming language, the constant MQWIH_STRUC_ID_ARRAY is also defined; this has the same value as MQWIH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQWIH_STRUC_ID.

StrucLength (MQLONG):

This is the length of the MQWIH structure. The value must be:

MQWIH_LENGTH_1
Length of version-1 work information header structure.

The following constant specifies the length of the current version:

MQWIH_CURRENT_LENGTH
Length of current version of work information header structure.

The initial value of this field is MQWIH_LENGTH_1.

Version (MQLONG):

This is the structure version number. The value must be:

MQWIH_VERSION_1
Version-1 work information header structure.

The following constant specifies the version number of the current version:

MQWIH_CURRENT_VERSION
Current version of work information header structure.

The initial value of this field is MQWIH_VERSION_1.

Initial values and language declarations for MQWIH:

Table 258. Initial values of fields in MQWIH for MQWIH

Field name	Name of constant	Value of constant
StrucId	MQWIH_STRUC_ID	'WIH¬'
Version	MQWIH_VERSION_1	1
StrucLength	MQWIH_LENGTH_1	120
Encoding	None	0
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
Flags	MQWIH_NONE	0
ServiceName	None	Blanks
ServiceStep	None	Blanks
MsgToken	MQMTOK_NONE	Nulls
Reserved	None	Blanks

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQWIH_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure:

   ```
   MQWIH MyWIH = {MQWIH_DEFAULT};
   ```

C declaration for MQWIH:

```
typedef struct tagMQWIH MQWIH;
struct tagMQWIH {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    StrucLength;     /* Length of MQWIH structure */
  MQLONG    Encoding;        /* Numeric encoding of data that follows
                                MQWIH */
  MQLONG    CodedCharSetId;  /* Character-set identifier of data that
                                follows MQWIH */
  MQCHAR8   Format;          /* Format name of data that follows
                                MQWIH */
  MQLONG    Flags;           /* Flags */
  MQCHAR32  ServiceName;     /* Service name */
  MQCHAR8   ServiceStep;     /* Service step name */
  MQBYTE16  MsgToken;        /* Message token */
  MQCHAR32  Reserved;        /* Reserved */
};
```

COBOL declaration for MQWIH:

```
**   MQWIH structure
  10 MQWIH.
**    Structure identifier
   15 MQWIH-STRUCID        PIC X(4).
**    Structure version number
   15 MQWIH-VERSION        PIC S9(9) BINARY.
**    Length of MQWIH structure
   15 MQWIH-STRUCLENGTH    PIC S9(9) BINARY.
**    Numeric encoding of data that follows MQWIH
   15 MQWIH-ENCODING       PIC S9(9) BINARY.
**    Character-set identifier of data that follows MQWIH
   15 MQWIH-CODEDCHARSETID PIC S9(9) BINARY.
**    Format name of data that follows MQWIH
   15 MQWIH-FORMAT         PIC X(8).
**    Flags
   15 MQWIH-FLAGS          PIC S9(9) BINARY.
**    Service name
   15 MQWIH-SERVICENAME    PIC X(32).
**    Service step name
   15 MQWIH-SERVICESTEP    PIC X(8).
**    Message token
   15 MQWIH-MSGTOKEN       PIC X(16).
**    Reserved
   15 MQWIH-RESERVED       PIC X(32).
```

PL/I declaration for MQWIH:

```
dcl
 1 MQWIH based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 StrucLength    fixed bin(31), /* Length of MQWIH structure */
  3 Encoding       fixed bin(31), /* Numeric encoding of data that
                                     follows MQWIH */
  3 CodedCharSetId fixed bin(31), /* Character-set identifier of data
                                     that follows MQWIH */
  3 Format         char(8),       /* Format name of data that follows
                                     MQWIH */
  3 Flags          fixed bin(31), /* Flags */
  3 ServiceName    char(32),      /* Service name */
  3 ServiceStep    char(8),       /* Service step name */
  3 MsgToken       char(16),      /* Message token */
  3 Reserved       char(32);      /* Reserved */
```

High Level Assembler declaration for MQWIH:

```
MQWIH                 DSECT
MQWIH_STRUCID         DS   CL4   Structure identifier
MQWIH_VERSION         DS   F     Structure version number
MQWIH_STRUCLENGTH     DS   F     Length of MQWIH structure
MQWIH_ENCODING        DS   F     Numeric encoding of data that follows
*                                MQWIH
MQWIH_CODEDCHARSETID  DS   F     Character-set identifier of data that
*                                follows MQWIH
MQWIH_FORMAT          DS   CL8   Format name of data that follows MQWIH
MQWIH_FLAGS           DS   F     Flags
MQWIH_SERVICENAME     DS   CL32  Service name
MQWIH_SERVICESTEP     DS   CL8   Service step name
MQWIH_MSGTOKEN        DS   XL16  Message token
MQWIH_RESERVED        DS   CL32  Reserved
*
MQWIH_LENGTH          EQU  *-MQWIH
                      ORG  MQWIH
MQWIH_AREA            DS   CL(MQWIH_LENGTH)
```

Visual Basic declaration for MQWIH:

```
Type MQWIH
  StrucId        As String*4  'Structure identifier'
  Version        As Long      'Structure version number'
  StrucLength    As Long      'Length of MQWIH structure'
  Encoding       As Long      'Numeric encoding of data that follows'
                              'MQWIH'
  CodedCharSetId As Long      'Character-set identifier of data that'
                              'follows MQWIH'
  Format         As String*8  'Format name of data that follows MQWIH'
  Flags          As Long      'Flags'
  ServiceName    As String*32 'Service name'
  ServiceStep    As String*8  'Service step name'
  MsgToken       As MQBYTE16  'Message token'
  Reserved       As String*32 'Reserved'
End Type
```

MQXP - Exit parameter block:

The following table summarizes the fields in the structure.

Table 259. Fields in MQXP

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
ExitId	Exit identifier	ExitId
ExitReason	Reason for invocation of exit	ExitReason
ExitResponse	Response from exit	ExitResponse
ExitCommand	API call code	ExitCommand
ExitParmCount	Parameter count	ExitParmCount
ExitUserArea	User area	ExitUserArea

Overview for MQXP: z/OS

Availability: z/OS.

Purpose: The MQXP structure is used as an input/output parameter to the API-crossing exit. For more information about this exit, see The API-crossing exit.

Character set and encoding: Character data in MQXP is in the character set of the local queue manager; this is given by the **CodedCharSetId** queue manager attribute. Numeric data in MQXP is in the native machine encoding; this is given by MQENC_NATIVE.

Fields for MQXP:

The MQXP structure contains the following fields; the fields are described in **alphabetical order**:

ExitCommand (MQLONG):

This field is set on entry to the exit routine. It identifies the API call that caused the exit to be invoked:

MQXC_CALLBACK
The CALLBACK call.

MQXC_MQBACK
The MQBACK call.

MQXC_MQCB
The MQCB call.

MQXC_MQCLOSE
The MQCLOSE call.

MQXC_MQCMIT
The MQCMIT call.

MQXC_MQCTL
The MQCTL call.

MQXC_MQGET
The MQGET call.

MQXC_MQINQ
The MQINQ call.

MQXC_MQOPEN
The MQOPEN call.

MQXC_MQPUT
The MQPUT call.

MQXC_MQPUT1
The MQPUT1 call.

MQXC_MQSET
The MQSET call.

MQXC_MQSTAT
The MQSTAT call.

MQXC_MQSUB
The MQSUB call.

MQXC_MQSUBRQ
The MQSUBRQ call.

This is an input field to the exit.

ExitId (MQLONG):

This is set on entry to the exit routine, and indicates the type of exit:

MQXT_API_CROSSING_EXIT
API-crossing exit for CICS.

This is an input field to the exit.

ExitParmCount (MQLONG):

This field is set on entry to the exit routine. It contains the number of parameters that the MQ call takes. These are:

Call name	Number of parameters
MQBACK	3
MQCLOSE	5
MQCMIT	3
MQGET	9
MQINQ	10
MQOPEN	6
MQPUT	8
MQPUT1	8
MQSET	10

This is an input field to the exit.

ExitReason (MQLONG):

This is set on entry to the exit routine. For the API-crossing exit it indicates whether the routine is called before or after execution of the API call:

MQXR_BEFORE
Before API execution.

MQXR_AFTER
After API execution.

This is an input field to the exit.

ExitResponse (MQLONG):

The value is set by the exit to communicate with the caller. The following values are defined:

MQXCC_OK
Exit completed successfully.

MQXCC_SUPPRESS_FUNCTION
Suppress function.

When this value is set by an API-crossing exit called *before* the API call, the API call is not performed. The *CompCode* for the call is set to MQCC_FAILED, the *Reason* is set to MQRC_SUPPRESSED_BY_EXIT, and all other parameters remain as the exit left them.

When this value is set by an API-crossing exit called *after* the API call, it is ignored by the queue manager.

MQXCC_SKIP_FUNCTION
Skip function.

When this value is set by an API-crossing exit called *before* the API call, the API call is not performed; the *CompCode* and *Reason* and all other parameters remain as the exit left them.

When this value is set by an API-crossing exit called *after* the API call, it is ignored by the queue manager.

This is an output field from the exit.

ExitUserArea (MQBYTE16):

This is a field that is available for the exit to use. It is initialized to binary zero for the length of the field before the first invocation of the exit for the task, and thereafter any changes made to this field by the exit are preserved across invocations of the exit. The following value is defined:

MQXUA_NONE
No user information.

The value is binary zero for the length of the field.

For the C programming language, the constant MQXUA_NONE_ARRAY is also defined; this has the same value as MQXUA_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_EXIT_USER_AREA_LENGTH. This is an input/output field to the exit.

Reserved (MQLONG):

This is a reserved field. Its value is not significant to the exit.

StrucId (MQCHAR4):

This is the structure identifier. The value must be:

MQXP_STRUC_ID
Identifier for exit parameter structure.

For the C programming language, the constant MQXP_STRUC_ID_ARRAY is also defined; this has the same value as MQXP_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

Version (MQLONG):

This is the structure version number. The value must be:

MQXP_VERSION_1
Version number for exit parameter-block structure.

Note: When a new version of this structure is introduced, the layout of the existing part is not changed. The exit must therefore check that the version number is equal to or greater than the lowest version that contains the fields that the exit needs to use.

This is an input field to the exit.

Language declarations:

This structure is supported in the following programming languages.

C declaration for MQXP:

```
typedef struct tagMQXP MQXP;
struct tagMQXP {
  MQCHAR4   StrucId;        /* Structure identifier */
  MQLONG    Version;        /* Structure version number */
  MQLONG    ExitId;         /* Exit identifier */
  MQLONG    ExitReason;     /* Reason for invocation of exit */
  MQLONG    ExitResponse;   /* Response from exit */
  MQLONG    ExitCommand;    /* API call code */
  MQLONG    ExitParmCount;  /* Parameter count */
  MQLONG    Reserved;       /* Reserved */
  MQBYTE16  ExitUserArea;   /* User area */
};
```

COBOL declaration for MQXP:

```
**   MQXP structure
  10 MQXP.
**    Structure identifier
   15 MQXP-STRUCID       PIC X(4).
**    Structure version number
   15 MQXP-VERSION       PIC S9(9) BINARY.
**    Exit identifier
   15 MQXP-EXITID        PIC S9(9) BINARY.
**    Reason for invocation of exit
   15 MQXP-EXITREASON    PIC S9(9) BINARY.
**    Response from exit
   15 MQXP-EXITRESPONSE  PIC S9(9) BINARY.
**    API call code
   15 MQXP-EXITCOMMAND   PIC S9(9) BINARY.
**    Parameter count
   15 MQXP-EXITPARMCOUNT PIC S9(9) BINARY.
**    Reserved
   15 MQXP-RESERVED      PIC S9(9) BINARY.
**    User area
   15 MQXP-EXITUSERAREA  PIC X(16).
```

PL/I declaration for MQXP:

```
dcl
 1 MQXP based,
  3 StrucId       char(4),       /* Structure identifier */
  3 Version       fixed bin(31), /* Structure version number */
  3 ExitId        fixed bin(31), /* Exit identifier */
  3 ExitReason    fixed bin(31), /* Reason for invocation of exit */
  3 ExitResponse  fixed bin(31), /* Response from exit */
  3 ExitCommand   fixed bin(31), /* API call code */
  3 ExitParmCount fixed bin(31), /* Parameter count */
  3 Reserved      fixed bin(31), /* Reserved */
  3 ExitUserArea  char(16);      /* User area */
```

High Level Assembler declaration for MQXP:

```
MQXP                DSECT
MQXP_STRUCID        DS   CL4   Structure identifier
MQXP_VERSION        DS   F     Structure version number
MQXP_EXITID         DS   F     Exit identifier
MQXP_EXITREASON     DS   F     Reason for invocation of exit
MQXP_EXITRESPONSE   DS   F     Response from exit
MQXP_EXITCOMMAND    DS   F     API call code
MQXP_EXITPARMCOUNT  DS   F     Parameter count
MQXP_RESERVED       DS   F     Reserved
MQXP_EXITUSERAREA   DS   XL16  User area
*
MQXP_LENGTH         EQU  *-MQXP
                    ORG  MQXP
MQXP_AREA           DS   CL(MQXP_LENGTH)
```

MQXQH - Transmission-queue header:

The following table summarizes the fields in the structure.

Table 260. Fields in MQXQH

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
RemoteQName	Name of destination queue	RemoteQName
RemoteQMgrName	Name of destination queue manager	RemoteQMgrName
MsgDesc	Original message descriptor	MsgDesc

Overview for MQXQH:

Availability: All IBM MQ systems and IBM MQ clients.

Purpose: The MQXQH structure describes the information that is prefixed to the application message data of messages when they are on transmission queues. A transmission queue is a special type of local queue that temporarily holds messages destined for remote queues (that is, destined for queues that do not belong to the local queue manager). A transmission queue is denoted by the **Usage** queue attribute having the value MQUS_TRANSMISSION.

Format name: MQFMT_XMIT_Q_HEADER.

Character set and encoding: Data in MQXOH must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by MQENC_NATIVE.

Set the character set and encoding of the MQXQH into the *CodedCharSetId* and *Encoding* fields in:

- The separate MQMD (if the MQXQH structure is at the start of the message data), or
- The header structure that precedes the MQXQH structure (all other cases).

Usage: A message that is on a transmission queue has *two* message descriptors:

- One message descriptor is stored separately from the message data; this is called the *separate message descriptor*, and is generated by the queue manager when the message is placed on the transmission queue. Some of the fields in the separate message descriptor are copied from the message descriptor provided by the application on the MQPUT or MQPUT1 call.

  The separate message descriptor is the one that is returned to the application in the **MsgDesc** parameter of the MQGET call when the message is removed from the transmission queue.

- A second message descriptor is stored within the MQXQH structure as part of the message data; this is called the *embedded message descriptor*, and is a copy of the message descriptor that was provided by the application on the MQPUT or MQPUT1 call (with minor variations).

  The embedded message descriptor is always a version-1 MQMD. If the message put by the application has nondefault values for one or more of the version-2 fields in the MQMD, an MQMDE structure follows the MQXQH, and is in turn followed by the application message data (if any). The MQMDE is either:

  – Generated by the queue manager (if the application uses a version-2 MQMD to put the message), or
  – Already present at the start of the application message data (if the application uses a version-1 MQMD to put the message).

  The embedded message descriptor is the one that is returned to the application in the **MsgDesc** parameter of the MQGET call when the message is removed from the final destination queue.

Fields in the separate message descriptor: The fields in the separate message descriptor are set by the queue manager as shown. If the queue manager does not support the version-2 MQMD, a version-1 MQMD is used without loss of function.

Field in separate MQMD	Value used
StrucId	MQMD_STRUC_ID
Version	MQMD_VERSION_2
Report	Copied from the embedded message descriptor, but with the bits identified by MQRO_ACCEPT_UNSUP_IF_XMIT_MASK set to zero. (This prevents a COA or COD report message being generated when a message is placed on or removed from a transmission queue.)
MsgType	Copied from the embedded message descriptor.
Expiry	Copied from the embedded message descriptor.
Feedback	Copied from the embedded message descriptor.
Encoding	MQENC_NATIVE (see note)
CodedCharSetId	Queue manager's **CodedCharSetId** attribute.
Format	MQFMT_XMIT_Q_HEADER
Priority	Copied from the embedded message descriptor.
Persistence	Copied from the embedded message descriptor.
MsgId	A new value is generated by the queue manager. This message identifier is different from the *MsgId* that the queue manager may have generated for the embedded message descriptor described previously.
CorrelId	The *MsgId* from the embedded message descriptor. For messages being put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE, *CorrelId* is reserved for internal use.
BackoutCount	0
ReplyToQ	Copied from the embedded message descriptor.
ReplyToQMgr	Copied from the embedded message descriptor.
UserIdentifier	Copied from the embedded message descriptor.
AccountingToken	Copied from the embedded message descriptor. For messages being put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE, *AccountingToken* is reserved for internal use.
ApplIdentityData	Copied from the embedded message descriptor.
PutApplType	MQAT_QMGR
PutApplName	First 28 bytes of the queue manager name.
PutDate	Date when message was put on transmission queue.
PutTime	Time when message was put on transmission queue.
ApplOriginData	Blanks
GroupId	MQGI_NONE
MsgSeqNumber	1
Offset	0
MsgFlags	MQMF_NONE
OriginalLength	MQOL_UNDEFINED

- On Windows, the value of MQENC_NATIVE for Micro Focus COBOL differs from the value for C. The value in the *Encoding* field in the separate message descriptor is always the value for C in these environments; this value is 546 in decimal. Also, the integer fields in the MQXQH structure are in the encoding that corresponds to this value (the native Intel encoding).

Fields in the embedded message descriptor: The fields in the embedded message descriptor have the same values as those in the **MsgDesc** parameter of the MQPUT or MQPUT1 call, except for the following:

- The *Version* field always has the value MQMD_VERSION_1.
- If the *Priority* field has the value MQPRI_PRIORITY_AS_Q_DEF, it is replaced by the value of the queue's **DefPriority** attribute.
- If the *Persistence* field has the value MQPER_PERSISTENCE_AS_Q_DEF, it is replaced by the value of the queue's **DefPersistence** attribute.
- If the *MsgId* field has the value MQMI_NONE, or the MQPMO_NEW_MSG_ID option was specified, or the message is a distribution-list message, *MsgId* is replaced by a new message identifier generated by the queue manager.

  When a distribution-list message is split into smaller distribution-list messages placed on different transmission queues, the *MsgId* field in each of the new embedded message descriptors is the same as that in the original distribution-list message.
- If the MQPMO_NEW_CORREL_ID option was specified, *CorrelId* is replaced by a new correlation identifier generated by the queue manager.
- The context fields are set as indicated by the MQPMO_*_CONTEXT options specified in the **PutMsgOpts** parameter; the context fields are:
  - *AccountingToken*
  - *ApplIdentityData*
  - *ApplOriginData*
  - *PutApplName*
  - *PutApplType*
  - *PutDate*
  - *PutTime*
  - *UserIdentifier*
- The version-2 fields (if they were present) are removed from the MQMD, and moved into an MQMDE structure, if one or more of the version-2 fields has a nondefault value.

Putting messages on remote queues: When an application puts a message on a remote queue (either by specifying the name of the remote queue directly, or by using a local definition of the remote queue), the local queue manager:

- Creates an MQXQH structure containing the embedded message descriptor
- Appends an MQMDE if one is needed and is not already present
- Appends the application message data
- Places the message on an appropriate transmission queue

Putting messages directly on transmission queues: An application can also put a message directly on a transmission queue. In this case the application must prefix the application message data with an MQXQH structure, and initialize the fields with appropriate values. In addition, the *Format* field in the **MsgDesc** parameter of the MQPUT or MQPUT1 call must have the value MQFMT_XMIT_Q_HEADER.

Character data in the MQXQH structure created by the application must be in the character set of the local queue manager (defined by the **CodedCharSetId** queue manager attribute), and integer data must be in the native machine encoding. In addition, character data in the MQXQH structure must be padded

with blanks to the defined length of the field; the data must not be ended prematurely by using a null character, because the queue manager does not convert the null and subsequent characters to blanks in the MQXQH structure.

However, the queue manager does not check that an MQXQH structure is present, or that valid values have been specified for the fields.

Applications should not put their messages directly to the SYSTEM.CLUSTER.TRANSMIT.QUEUE.

Getting messages from transmission queues: Applications that get messages from a transmission queue must process the information in the MQXQH structure in an appropriate fashion. The presence of the MQXQH structure at the beginning of the application message data is indicated by the value MQFMT_XMIT_Q_HEADER being returned in the *Format* field in the **MsgDesc** parameter of the MQGET call. The values returned in the *CodedCharSetId* and *Encoding* fields in the **MsgDesc** parameter indicate the character set and encoding of the character and integer data in the MQXQH structure. The character set and encoding of the application message data are defined by the *CodedCharSetId* and *Encoding* fields in the embedded message descriptor.

Fields for MQXQH:

The MQXQH structure contains the following fields; the fields are described in **alphabetical order**:

MsgDesc (MQMD1):

This is the embedded message descriptor, and is a close copy of the message descriptor MQMD that was specified as the **MsgDesc** parameter on the MQPUT or MQPUT1 call when the message was originally put to the remote queue.

Note: This is a version-1 MQMD.

The initial values of the fields in this structure are the same as those in the MQMD structure.

RemoteQMgrName (MQCHAR48):

This is the name of the queue manager or queue-sharing group that owns the queue that is the apparent eventual destination for the message.

If the message is a distribution-list message, *RemoteQMgrName* is blank.

The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

RemoteQName (MQCHAR48):

This is the name of the message queue that is the apparent eventual destination for the message (this might prove not to be the eventual destination if, for example, this queue is defined at *RemoteQMgrName* to be a local definition of another remote queue).

If the message is a distribution-list message (that is, the *Format* field in the embedded message descriptor is MQFMT_DIST_HEADER), *RemoteQName* is blank.

The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

StrucId (MQCHAR4):

This is the structure identifier. The value must be:

MQXQH_STRUC_ID
Identifier for transmission-queue header structure.

For the C programming language, the constant MQXQH_STRUC_ID_ARRAY is also defined; this has the same value as MQXQH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQXQH_STRUC_ID.

Version (MQLONG):

This is the structure version number. The value must be:

MQXQH_VERSION_1
Version number for transmission-queue header structure.

The following constant specifies the version number of the current version:

MQXQH_CURRENT_VERSION
Current version of transmission-queue header structure.

The initial value of this field is MQXQH_VERSION_1.

Initial values and language declarations for MQXQH:

Table 261. Initial values of fields in MQXQH for MQXQH

Field name	Name of constant	Value of constant
StrucId	MQXQH_STRUC_ID	'XQH¬'
Version	MQXQH_VERSION_1	1
RemoteQName	None	Null string or blanks
RemoteQMgrName	None	Null string or blanks
MsgDesc	Same names and values as MQMD; see Table 216 on page 2438	-

Table 261. Initial values of fields in MQXQH for MQXQH (continued)

Field name	Name of constant	Value of constant
Notes: 1. The symbol ¬ represents a single blank character. 2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages. 3. In the C programming language, the macro variable MQXQH_DEFAULT contains the values that are listed in the table. Use it in the following way to provide initial values for the fields in the structure: `MQXQH MyXQH = {MQXQH_DEFAULT};`		

C declaration for MQXQH:

```
typedef struct tagMQXQH MQXQH;
struct tagMQXQH {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQCHAR48  RemoteQName;     /* Name of destination queue */
  MQCHAR48  RemoteQMgrName;  /* Name of destination queue manager */
  MQMD1     MsgDesc;         /* Original message descriptor */
};
```

COBOL declaration for MQXQH:

```
**   MQXQH structure
  10 MQXQH.
**    Structure identifier
   15 MQXQH-STRUCID                  PIC X(4).
**    Structure version number
   15 MQXQH-VERSION                  PIC S9(9) BINARY.
**    Name of destination queue
   15 MQXQH-REMOTEQNAME              PIC X(48).
**    Name of destination queue manager
   15 MQXQH-REMOTEQMGRNAME           PIC X(48).
**    Original message descriptor
   15 MQXQH-MSGDESC.
**     Structure identifier
    20 MQXQH-MSGDESC-STRUCID          PIC X(4).
**     Structure version number
    20 MQXQH-MSGDESC-VERSION          PIC S9(9) BINARY.
**     Report options
    20 MQXQH-MSGDESC-REPORT           PIC S9(9) BINARY.
**     Message type
    20 MQXQH-MSGDESC-MSGTYPE          PIC S9(9) BINARY.
**     Expiry time
    20 MQXQH-MSGDESC-EXPIRY           PIC S9(9) BINARY.
**     Feedback or reason code
    20 MQXQH-MSGDESC-FEEDBACK         PIC S9(9) BINARY.
**     Numeric encoding of message data
    20 MQXQH-MSGDESC-ENCODING         PIC S9(9) BINARY.
**     Character set identifier of message data
    20 MQXQH-MSGDESC-CODEDCHARSETID   PIC S9(9) BINARY.
**     Format name of message data
    20 MQXQH-MSGDESC-FORMAT           PIC X(8).
**     Message priority
    20 MQXQH-MSGDESC-PRIORITY         PIC S9(9) BINARY.
**     Message persistence
    20 MQXQH-MSGDESC-PERSISTENCE      PIC S9(9) BINARY.
**     Message identifier
    20 MQXQH-MSGDESC-MSGID            PIC X(24).
**     Correlation identifier
    20 MQXQH-MSGDESC-CORRELID         PIC X(24).
```

```
**     Backout counter
    20 MQXQH-MSGDESC-BACKOUTCOUNT     PIC S9(9) BINARY.
**     Name of reply-to queue
    20 MQXQH-MSGDESC-REPLYTOQ         PIC X(48).
**     Name of reply queue manager
    20 MQXQH-MSGDESC-REPLYTOQMGR      PIC X(48).
**     User identifier
    20 MQXQH-MSGDESC-USERIDENTIFIER   PIC X(12).
**     Accounting token
    20 MQXQH-MSGDESC-ACCOUNTINGTOKEN  PIC X(32).
**     Application data relating to identity
    20 MQXQH-MSGDESC-APPLIDENTITYDATA PIC X(32).
**     Type of application that put the message
    20 MQXQH-MSGDESC-PUTAPPLTYPE      PIC S9(9) BINARY.
**     Name of application that put the message
    20 MQXQH-MSGDESC-PUTAPPLNAME      PIC X(28).
**     Date when message was put
    20 MQXQH-MSGDESC-PUTDATE          PIC X(8).
**     Time when message was put
    20 MQXQH-MSGDESC-PUTTIME          PIC X(8).
**     Application data relating to origin
    20 MQXQH-MSGDESC-APPLORIGINDATA   PIC X(4).
```

PL/I declaration for MQXQH:

```
dcl
 1 MQXQH based,
  3 StrucId           char(4),       /* Structure identifier */
  3 Version           fixed bin(31), /* Structure version number */
  3 RemoteQName       char(48),      /* Name of destination queue */
  3 RemoteQMgrName    char(48),      /* Name of destination queue
                                        manager */
  3 MsgDesc,                         /* Original message descriptor */
   5 StrucId          char(4),       /* Structure identifier */
   5 Version          fixed bin(31), /* Structure version number */
   5 Report           fixed bin(31), /* Report options */
   5 MsgType          fixed bin(31), /* Message type */
   5 Expiry           fixed bin(31), /* Expiry time */
   5 Feedback         fixed bin(31), /* Feedback or reason code */
   5 Encoding         fixed bin(31), /* Numeric encoding of message
                                        data */
   5 CodedCharSetId   fixed bin(31), /* Character set identifier of
                                        message data */
   5 Format           char(8),       /* Format name of message data */
   5 Priority         fixed bin(31), /* Message priority */
   5 Persistence      fixed bin(31), /* Message persistence */
   5 MsgId            char(24),      /* Message identifier */
   5 CorrelId         char(24),      /* Correlation identifier */
   5 BackoutCount     fixed bin(31), /* Backout counter */
   5 ReplyToQ         char(48),      /* Name of reply-to queue */
   5 ReplyToQMgr      char(48),      /* Name of reply queue manager */
   5 UserIdentifier   char(12),      /* User identifier */
   5 AccountingToken  char(32),      /* Accounting token */
   5 ApplIdentityData char(32),      /* Application data relating to
                                        identity */
   5 PutApplType      fixed bin(31), /* Type of application that put the
                                        message */
   5 PutApplName      char(28),      /* Name of application that put the
                                        message */
   5 PutDate          char(8),       /* Date when message was put */
   5 PutTime          char(8),       /* Time when message was put */
   5 ApplOriginData   char(4);       /* Application data relating to
                                        origin */
```

High Level Assembler declaration for MQXQH:

```
MQXQH                           DSECT
MQXQH_STRUCID                   DS   CL4   Structure identifier
MQXQH_VERSION                   DS   F     Structure version number
MQXQH_REMOTEQNAME               DS   CL48  Name of destination queue
MQXQH_REMOTEQMGRNAME            DS   CL48  Name of destination queue
*                                          manager
MQXQH_MSGDESC                   DS   0F    Force fullword alignment
MQXQH_MSGDESC_STRUCID           DS   CL4   Structure identifier
MQXQH_MSGDESC_VERSION           DS   F     Structure version number
MQXQH_MSGDESC_REPORT            DS   F     Report options
MQXQH_MSGDESC_MSGTYPE           DS   F     Message type
MQXQH_MSGDESC_EXPIRY            DS   F     Expiry time
MQXQH_MSGDESC_FEEDBACK          DS   F     Feedback or reason code
MQXQH_MSGDESC_ENCODING          DS   F     Numeric encoding of message
*                                          data
MQXQH_MSGDESC_CODEDCHARSETID    DS   F     Character set identifier of
*                                          message data
MQXQH_MSGDESC_FORMAT            DS   CL8   Format name of message data
MQXQH_MSGDESC_PRIORITY          DS   F     Message priority
MQXQH_MSGDESC_PERSISTENCE       DS   F     Message persistence
MQXQH_MSGDESC_MSGID             DS   XL24  Message identifier
MQXQH_MSGDESC_CORRELID          DS   XL24  Correlation identifier
MQXQH_MSGDESC_BACKOUTCOUNT      DS   F     Backout counter
MQXQH_MSGDESC_REPLYTOQ          DS   CL48  Name of reply-to queue
MQXQH_MSGDESC_REPLYTOQMGR       DS   CL48  Name of reply queue manager
MQXQH_MSGDESC_USERIDENTIFIER    DS   CL12  User identifier
MQXQH_MSGDESC_ACCOUNTINGTOKEN   DS   XL32  Accounting token
MQXQH_MSGDESC_APPLIDENTITYDATA  DS   CL32  Application data relating to
*                                          identity
MQXQH_MSGDESC_PUTAPPLTYPE       DS   F     Type of application that put
*                                          the message
MQXQH_MSGDESC_PUTAPPLNAME       DS   CL28  Name of application that put
*                                          the message
MQXQH_MSGDESC_PUTDATE           DS   CL8   Date when message was put
MQXQH_MSGDESC_PUTTIME           DS   CL8   Time when message was put
MQXQH_MSGDESC_APPLORIGINDATA    DS   CL4   Application data relating to
*                                          origin
MQXQH_MSGDESC_LENGTH            EQU  *-MQXQH_MSGDESC
                                ORG  MQXQH_MSGDESC
MQXQH_MSGDESC_AREA              DS   CL(MQXQH_MSGDESC_LENGTH)
*
MQXQH_LENGTH                    EQU  *-MQXQH
                                ORG  MQXQH
MQXQH_AREA                      DS   CL(MQXQH_LENGTH)
```

Visual Basic declaration for MQXQH:

```
Type MQXQH
  StrucId        As String*4  'Structure identifier'
  Version        As Long      'Structure version number'
  RemoteQName    As String*48 'Name of destination queue'
  RemoteQMgrName As String*48 'Name of destination queue manager'
  MsgDesc        As MQMD1     'Original message descriptor'
End Type
```

Function calls

This section gives information on all of the MQI calls that are possible. Descriptions, syntax, parameter information, usage notes, and language invocations for each possible language are given for each of the different calls.

Call descriptions:

This section describes MQI calls.

Online help on the UNIX platform, in the form of *man* pages, is available for these calls.

Note: The calls associated with data conversion, MQXCNVC and MQ_DATA_CONV_EXIT, are in "Data conversion" on page 2909.

Conventions used in the call descriptions:

For each call, this collection of topics gives a description of the parameters and usage of the call in a format that is independent of programming language. This is followed by typical invocations of the call, and typical declarations of its parameters, in each of the supported programming languages.

Important: When coding IBM MQ API calls you must ensure that all relevant parameters (as described in the following sections) are provided. Failure to do so can produce unpredictable results.

The description of each call contains the following sections:

Call name
The call name, followed by a brief description of the purpose of the call.

Parameters
For each parameter, the name is followed by its data type in parentheses ( ) and one of the following:

input You supply information in the parameter when you make the call.

output
The queue manager returns information in the parameter when the call completes or fails.

input/output
You supply information in the parameter when you make the call, and the queue manager changes the information when the call completes or fails.

For example:

Compcode (MQLONG) - output

In some cases, the data type is a structure. In all cases, there is more information about the data type or structure in "Elementary data types" on page 2196.

The last two parameters in each call are a completion code and a reason code. The completion code indicates whether the call completed successfully, partially, or not at all. Further information about the partial success or the failure of the call is given in the reason code. For more information about each completion and reason code, see "Return codes" on page 2876.

Usage notes
Additional information about the call, describing how to use it and any restrictions on its use.

Assembler language invocation
Typical invocation of the call, and declaration of its parameters, in assembler language.

C invocation
Typical invocation of the call, and declaration of its parameters, in C.

COBOL invocation
Typical invocation of the call, and declaration of its parameters, in COBOL.

PL/I invocation
Typical invocation of the call, and declaration of its parameters, in PL/I.

All parameters are passed by reference.

Visual Basic invocation
Typical invocation of the call, and declaration of its parameters, in Visual Basic.

Other notation conventions are:

Constants

Names of constants are shown in uppercase; for example, MQOO_OUTPUT. A set of constants having the same prefix is shown as follows: MQIA_*. See “Constants” on page 2055 for the value of a constant.

Arrays

In some calls, parameters are arrays of character strings that do not have fixed sizes. In the descriptions of these parameters, a lowercase n represents a numeric constant. When you code the declaration for that parameter, replace the n with the numeric value that you require.

Using the calls in the C language:

Parameters that are *input only* and of type MQHCONN, MQHOBJ, MQHMSG, or MQLONG are passed by value. For all other parameters, the *address* of the parameter is passed by value.

You do not need to specify all parameters that are passed by address every time that you invoke a function. Where you do not need a particular parameter, specify a null pointer as the parameter on the function invocation, in place of the address of parameter data. Parameters for which this is possible are identified in the call descriptions.

No parameter is returned as the value of the call; in C terminology, this means that all calls return `void`.

Declaring the Buffer parameter:

The **MQGET**, **MQPUT**, and **MQPUT1** calls each have one parameter that has an undefined data type: the *Buffer* parameter. Use this parameter to send and receive the application's message data.

Parameters of this sort are shown in the C examples as arrays of MQBYTE. You can declare the parameters in this way, but it is usually more convenient to declare them as the particular structure that describes the layout of the data in the message. The function prototype declares the parameter as a pointer-to-void, so that you can specify the address of any sort of data as the parameter on the call invocation.

Pointer-to-void is a pointer to data of undefined format. It is defined as:

```
typedef void *PMQVOID;
```

MQBACK - Back out changes:

The MQBACK call indicates to the queue manager that all the message gets and puts that have occurred since the last sync point are to be backed out.

Messages put as part of a unit of work are deleted; messages retrieved as part of a unit of work are reinstated on the queue.

- On z/OS, this call is used only by batch programs (including IMS batch DL/I programs).

Syntax

MQBACK (*Hconn*, *Compcode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

Compcode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_OUTCOME_PENDING
(2124, X'84C') Result of back-out operation is pending.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_ENVIRONMENT_ERROR
(2012, X'7DC') Call not valid in environment.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OUTCOME_MIXED
(2123, X'84B') Result of commit or back-out operation is mixed.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_STORAGE_MEDIUM_FULL
(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes

Usage notes

1. You can use this call only when the queue manager itself coordinates the unit of work. This can be:
   - A local unit of work, where the changes affect only MQ resources.
   - A global unit of work, where the changes can affect resources belonging to other resource managers, as well as affecting MQ resources.

   For further details about local and global units of work, see "MQBEGIN - Begin unit of work" on page 2626.
2. In environments where the queue manager does not coordinate the unit of work, use the appropriate back-out call instead of MQBACK. The environment might also support an implicit back out caused by the application terminating abnormally.
   - On z/OS, use the following calls:
     - Batch programs (including IMS batch DL/I programs) can use the MQBACK call if the unit of work affects only MQ resources. However, if the unit of work affects both MQ resources and resources belonging to other resource managers (for example, Db2), use the SRRBACK call provided by the z/OS Recoverable Resource Service (RRS). The SRRBACK call backs out changes to resources belonging to the resource managers that have been enabled for RRS coordination.
     - CICS applications must use the `EXEC CICS SYNCPOINT ROLLBACK` command to back out the unit of work. Do not use the MQBACK call for CICS applications.
     - IMS applications (other than batch DL/I programs) must use IMS calls such as `ROLB` to back out the unit of work. Do not use the MQBACK call for IMS applications (other than batch DL/I programs).
   - On IBM i, use this call for local units of work coordinated by the queue manager. This means that a commitment definition must not exist at job level, that is, the `STRCMTCTL` command with the **`CMTSCOPE(*JOB)`** parameter must not have been issued for the job.
3. If an application ends with uncommitted changes in a unit of work, the disposition of those changes depends on whether the application ends normally or abnormally. See the usage notes in "MQDISC - Disconnect queue manager" on page 2679 for further details.
4. When an application puts or gets messages in groups or segments of logical messages, the queue manager retains information relating to the message group and logical message for the last successful MQPUT and MQGET calls. This information is associated with the queue handle, and includes such things as:
   - The values of the *GroupId*, *MsgSeqNumber*, *Offset*, and *MsgFlags* fields in MQMD.
   - Whether the message is part of a unit of work.
   - For the MQPUT call: whether the message is persistent or nonpersistent.

   The queue manager keeps *three* sets of group and segment information, one set for each of the following:
   - The last successful MQPUT call (this can be part of a unit of work).
   - The last successful MQGET call that removed a message from the queue (this can be part of a unit of work).
   - The last successful MQGET call that browsed a message on the queue (this cannot be part of a unit of work).

5. The information associated with the MQGET call is restored to the value that it had before the first successful MQGET call for that queue handle in the current unit of work.

   Queues that were updated by the application after the unit of work started, but outside the scope of the unit of work, do not have their group and segment information restored if the unit of work is backed out.

   Restoring the group and segment information to its previous value when a unit of work is backed out allows the application to spread a large message group or large logical message consisting of many segments across several units of work, and to restart at the correct point in the message group or logical message if one of the units of work fails.

   Using several units of work might be advantageous if the local queue manager has only limited queue storage. However, the application must maintain sufficient information to be able to restart putting or getting messages at the correct point if a system failure occurs.

   For details of how to restart at the correct point after a system failure, see the MQPMO_LOGICAL_ORDER option described in "MQPMO - Put-message options" on page 2477, and the MQGMO_LOGICAL_ORDER option described in "MQGMO - Get-message options" on page 2330.

   The remaining usage notes apply only when the queue manager coordinates the units of work.
6. A unit of work has the same scope as a connection handle. All MQ calls that affect a particular unit of work must be performed using the same connection handle. Calls issued using a different connection handle (for example, calls issued by another application) affect a different unit of work. See the **Hconn** parameter described in "MQCONN - Connect queue manager" on page 2656 for information about the scope of connection handles.
7. Only messages that were put or retrieved as part of the current unit of work are affected by this call.
8. A long-running application that issues MQGET, MQPUT, or MQPUT1 calls within a unit of work, but that never issues a commit or backout call, can fill queues with messages that are not available to other applications. To guard against this possibility, the administrator must set the **MaxUncommittedMsgs** queue manager attribute to a value that is low enough to prevent runaway applications filling the queues, but high enough to allow the expected messaging applications to work correctly.

C invocation

```
MQBACK (Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;     /* Connection handle */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQBACK' USING HCONN, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN     PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE  PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON    PIC S9(9) BINARY.
```

PL/I invocation

```
call MQBACK (Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn     fixed bin(31);  /* Connection handle */
dcl CompCode  fixed bin(31);  /* Completion code */
dcl Reason    fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQBACK,(HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN     DS  F  Connection handle
COMPCODE  DS  F  Completion code
REASON    DS  F  Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQBACK Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn    As Long 'Connection handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

MQBEGIN - Begin unit of work:

The MQBEGIN call begins a unit of work that is coordinated by the queue manager, and that can involve external resource managers.

Syntax

MQBEGIN (*Hconn*, *BeginOptions*, *Compcode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

Hconn must be a nonshared connection handle. If a shared connection handle is specified, the call fails with reason code MQRC_HCONN_ERROR. See the description of the MQCNO_HANDLE_SHARE_* options in "MQCNO - Connect options" on page 2276 for more information about shared and nonshared handles.

BeginOptions

Type: MQBO - input/output

These are options that control the action of MQBEGIN, as described in "MQBEGIN - Begin unit of work."

If no options are required, programs written in C or S/390 assembler can specify a null parameter address, instead of specifying the address of an MQBO structure.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_NO_EXTERNAL_PARTICIPANTS
(2121, X'849') No participating resource managers registered.

MQRC_PARTICIPANT_NOT_AVAILABLE
(2122, X'84A') Participating resource manager not available.

If *CompCode* is MQCC_FAILED:

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_BO_ERROR
(2134, X'856') Begin-options structure not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_ENVIRONMENT_ERROR
(2012, X'7DC') Call not valid in environment.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

MQRC_UOW_IN_PROGRESS
(2128, X'850') Unit of work already started.

For more information about these reason codes, see Reason codes.

Usage notes

1. Use the MQBEGIN call to start a unit of work that is coordinated by the queue manager and that might involve changes to resources owned by other resource managers. The queue manager supports three types of unit-of-work:

- **Queue manager-coordinated local unit of work**: A unit of work in which the queue manager is the only resource manager participating, and so the queue manager acts as the unit-of-work coordinator.
  - To start this type of unit of work, specify the MQPMO_SYNCPOINT or MQGMO_SYNCPOINT option on the first MQPUT, MQPUT1, or MQGET call in the unit of work.
  - To commit or back out this type of unit of work, use the MQCMIT or MQBACK call.
- **Queue manager-coordinated global unit of work**: A unit of work in which the queue manager acts as the unit-of-work coordinator, both for MQ resources *and* for resources belonging to other resource managers. Those resource managers cooperate with the queue manager to ensure that all changes to resources in the unit of work are committed or backed out together.
  - To start this type of unit of work, use the MQBEGIN call.
  - To commit or back out this type of unit of work, use the MQCMIT and MQBACK calls.
- **Externally-coordinated global unit of work**: A unit of work in which the queue manager is a participant, but the queue manager does not act as the unit-of-work coordinator. Instead, there is an external unit-of-work coordinator with which the queue manager cooperates.
  - To start this type of unit of work, use the relevant call provided by the external unit-of-work coordinator.

    If the MQBEGIN call is used to try to start the unit of work, the call fails with reason code MQRC_ENVIRONMENT_ERROR.
  - To commit or back out this type of unit of work, use the commit and back-out calls provided by the external unit-of-work coordinator.

    If you use the MQCMIT or MQBACK call to commit or back out the unit of work, the call fails with reason code MQRC_ENVIRONMENT_ERROR.

2. If the application ends with uncommitted changes in a unit of work, the disposition of those changes depends on whether the application ends normally or abnormally. See the usage notes in "MQDISC - Disconnect queue manager" on page 2679 for further details.
3. An application can participate in only one unit of work at a time. The MQBEGIN call fails with reason code MQRC_UOW_IN_PROGRESS if there is already a unit of work in existence for the application, regardless of which type of unit of work it is.
4. The MQBEGIN call is not valid in an MQ MQI client environment. An attempt to use the call fails with reason code MQRC_ENVIRONMENT_ERROR.
5. When the queue manager is acting as the unit-of-work coordinator for global units of work, the resource managers that can participate in the unit of work are defined in the queue manager configuration file.
6. On IBM i, the three types of unit of work are supported as follows:
   - **Queue manager-coordinated local unit of work** can be used only when a commitment definition does not exist at the job level, that is, the `STRCMTCTL` command with the **`CMTSCOPE(*JOB)`** parameter must not have been issued for the job.
   - **Queue manager-coordinated global unit of work** is not supported.
   - **Externally-coordinated global unit of work** can be used only when a commitment definition exists at job level, that is, the `STRCMTCTL` command with the **`CMTSCOPE(*JOB)`** parameter must have been issued for the job. If this has been done, the IBM i `COMMIT` and `ROLLBACK` operations apply to MQ resources as well as to resources belonging to other participating resource managers.

C invocation

```
MQBEGIN (Hconn, &BeginOptions, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;         /* Connection handle */
MQBO     BeginOptions;  /* Options that control the action of MQBEGIN */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQBEGIN' USING HCONN, BEGINOPTIONS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN         PIC S9(9) BINARY.
**   Options that control the action of MQBEGIN
 01  BEGINOPTIONS.
     COPY CMQBOV.
**   Completion code
 01  COMPCODE      PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON        PIC S9(9) BINARY.
```

PL/I invocation

```
call MQBEGIN (Hconn, BeginOptions, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn         fixed bin(31);  /* Connection handle */
dcl BeginOptions  like MQBO;      /* Options that control the action of
                                     MQBEGIN */
dcl CompCode      fixed bin(31);  /* Completion code */
dcl Reason        fixed bin(31);  /* Reason code qualifying CompCode */
```

Visual Basic invocation

```
MQBEGIN Hconn, BeginOptions, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn        As Long 'Connection handle'
Dim BeginOptions As MQBO 'Options that control the action of MQBEGIN'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

MQBUFMH - Convert buffer into message handle:

The MQBUFMH function call converts a buffer into a message handle and is the inverse of the MQMHBUF call.

This call takes a message descriptor and MQRFH2 properties in the buffer and makes them available through a message handle. The MQRFH2 properties in the message data are, optionally, removed. The *Encoding*, *CodedCharSetId*, and *Format* fields of the message descriptor are updated, if necessary, to correctly describe the contents of the buffer after the properties have been removed.

Syntax

MQBUFMH (*Hconn*, *Hmsg*, *BufMsgHOpts*, *MsgDesc*, *BufferLength*, *Buffer*, *DataLength*, *Compcode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of **Hconn** must match the connection handle that was used to create the message handle specified in the **Hmsg** parameter.

If the message handle was created using MQHC_UNASSOCIATED_HCONN, a valid connection must be established on the thread converting a buffer into a message handle. If a valid connection is not established, the call fails with MQRC_CONNECTION_BROKEN.

Hmsg
Type: MQHMQSG - input

This is the message handle for which a buffer is required. The value was returned by a previous MQCRTMH call.

BufMsgHOpts
Type: MQBMHO - input

The MQBMHO structure allows applications to specify options that control how message handles are produced from buffers.

See "MQBMHO - Buffer to message handle options" on page 2229 for details.

MsgDesc
Type: MQMD - input/output

The *MsgDesc* structure contains the message descriptor properties and describes the contents of the buffer area.

On output from the call, the properties are optionally removed from the buffer area and, in this case, the message descriptor is updated to correctly describe the buffer area.

Data in this structure must be in the character set and encoding of the application.

BufferLength
Type: MQLONG - input

BufferLength is the length of the Buffer area, in bytes.

A *BufferLength* of zero bytes is valid, and indicates that the buffer area contains no data.

Buffer
Type: MQBYTExBufferLength - input/output

These are options that control the action of MQBEGIN, as described in "MQBEGIN - Begin unit of work" on page 2626.

Buffer defines the area containing the message buffer. For most data, you should align the buffer on a 4-byte boundary.

If **Buffer** contains character or numeric data, set the *CodedCharSetId* and *Encoding* fields in the **MsgDesc** parameter to the values appropriate to the data; this enables the data to be converted, if necessary.

If properties are found in the message buffer they are optionally removed; they later become available from the message handle on return from the call.

In the C programming language, the parameter is declared as a pointer-to-void, which means the address of any type of data can be specified as the parameter.

If the **BufferLength** parameter is zero, **Buffer** is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler can be null.

DataLength
Type: MQLONG - output

The length, in bytes, of the buffer which might have the properties removed.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'089C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BMHO_ERROR
(2489, X'09B9') Buffer to message handle options structure not valid.

MQRC_BUFFER_ERROR
(2004, X'07D4') Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'07D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'08AB') MQI call entered before previous call completed.

MQRC_CONNECTION_BROKEN
(2009, X'07D9') Connection to queue manager lost.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle not valid.

MQRC_MD_ERROR
(2026, X'07EA') Message descriptor not valid.

MQRC_MSG_HANDLE_IN_USE
(2499, X'09C3') Message handle already in use.

MQRC_OPTIONS_ERROR
(2046, X'07FE') Options not valid or not consistent.

MQRC_RFH_ERROR
(2334, X'091E') MQRFH2 structure not valid.

MQRC_RFH_FORMAT_ERROR
(2421, X'0975') An MQRFH2 folder containing properties could not be parsed.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

Usage notes

MQBUFMH calls cannot be intercepted by API exits - a buffer is converted into a message handle in the application space; the call does not reach the queue manager.

C invocation

```
MQBUFMH (Hconn, Hmsg, &BufMsgHOpts, &MsgDesc, BufferLength, Buffer,
         &DataLength, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;        /* Connection handle */
MQHMSG  Hmsg;         /* Message handle */
MQBMHO  BufMsgHOpts;  /* Options that control the action of MQBUFMH */
MQMD    MsgDesc;      /* Message descriptor */
MQLONG  BufferLength; /* Length in bytes of the Buffer area */
MQBYTE  Buffer[n];    /* Area to contain the message buffer */
MQLONG  DataLength;   /* Length of the output buffer */
MQLONG  CompCode;     /* Completion code */
MQLONG  Reason;       /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQBUFMH' USING HCONN, HMSG, BUFMSGHOPTS, MSGDESC, BUFFERLENGTH,
                     BUFFER, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN         PIC S9(9) BINARY.
**   Message handle
 01  HMSG          PIC S9(18) BINARY.
**   Options that control the action of MQBUFMH
 01  BUFMSGHOPTS.
     COPY CMQBMHOV.
**   Message descriptor
 01  MSGDESC.
     COPY CMQMD.
**   Length in bytes of the Buffer area
 01  BUFFERLENGTH  PIC S9(9) BINARY.
**   Area to contain the message buffer
 01  BUFFER        PIC X(n).
**   Length of the output buffer
 01  DATALENGTH    PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE      PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON        PIC S9(9) BINARY.
```

PL/I invocation

```
call MQBUFMH (Hconn, Hmsg, BufMsgHOpts, MsgDesc, BufferLength, Buffer,
DataLength, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn        fixed bin(31); /* Connection handle */
dcl Hmsg         fixed bin(63); /* Message handle */
dcl BufMsgHOpts  like MQBMHO;   /* Options that control the action of
                                   MQBUFMH */
dcl MsgDesc      like MQMD;     /* Message descriptor */
dcl BufferLength fixed bin(31); /* Length in bytes of the Buffer area */
dcl Buffer       char(n);       /* Area to contain the message buffer */
dcl DataLength   fixed bin(31); /* Length of the output buffer */
dcl CompCode     fixed bin(31); /* Completion code */
dcl Reason       fixed bin(31); /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQBUFMH,(HCONN,HMSG,BUFMSGHOPTS,MSGDESC,BUFFERLENGTH,BUFFER,
               DATALENGTH,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN        DS       F      Connection handle
HMSG         DS       D      Message handle
BUFMSGHOPTS  CMQBMHOA ,      Options that control the action of MQBUFMH
MSGDESC      CMQMDA   ,      Message descriptor
BUFFERLENGTH DS       F      Length in bytes of the BUFFER area
BUFFER       DS       CL(n)  Area to contain the properties
DATALENGTH   DS       F      Length of the output buffer
COMPCODE     DS       F      Completion code
REASON       DS       F      Reason code qualifying COMPCODE
```

MQCB - Manage callback:

The MQCB call registers a callback for the specified object handle and controls activation and changes to the callback.

A callback is a piece of code (specified as either the name of a function that can be dynamically linked or as function pointer) that is called by IBM MQ when certain events occur.

To use MQCB and MQCTL on a V7 client you must be connected to a V7 server and the **SHARECNV** parameter of the channel must have a non-zero value.

The types of callback that can be defined are:

Message consumer
A message consumer callback function is called when a message, meeting the selection criteria specified, is available on an object handle.

Only one callback function can be registered against each object handle. If a single queue is to be read with multiple selection criteria then the queue must be opened multiple times and a consumer function registered on each handle.

Event handler
The event handler is called for conditions that affect the whole callback environment.

The function is called when an event condition occurs, for example, a queue manager or connection stopping or quiescing.

The function is not called for conditions that are specific to a single message consumer, for example MQRC_GET_INHIBITED; it is called however if a callback function does not end normally.

Syntax

MQCB (*Hconn*, *Operation*, *CallbackDesc*, *Hobj*, *MsgDesc*, *GetMsgOpts*, *CompCode*, *Reason*)

Parameters

Hconn
Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications you can specify the following special value for *MQHC_DEF_HCONN* to use the connection handle associated with this execution unit.

Operation
Type: MQLONG - input

The operation being processed on the callback defined for the specified object handle. You must specify one of the following options. To specify more than one option, either add the values together

(do not add the same constant more than once), or combine the values using the bitwise OR operation (if the programming language supports bit operations).

MQOP_REGISTER
Define the callback function for the specified object handle. This operation defines the function to be called and the selection criteria to be used.

If a callback function is already defined for the object handle the definition is replaced. If an error is detected while replacing the callback, the function is deregistered.

If a callback is registered in the same callback function in which it was previously deregistered, this is treated as a replace operation; any initial or final calls are not invoked.

You can use MQOP_REGISTER with MQOP_SUSPEND or MQOP_RESUME.

MQOP_DEREGISTER
Stop the consuming of messages for the object handle and removes the handle from those eligible for a callback.

A callback is automatically deregistered if the associated handle is closed.

If MQOP_DEREGISTER is called from within a consumer, and the callback has a stop call defined, it is invoked upon return from the consumer.

If this operation is issued against an *Hobj* with no registered consumer, the call returns with MQRC_CALLBACK_NOT_REGISTERED.

MQOP_SUSPEND
Suspends the consuming of messages for the object handle.

If this operation is applied to an event handler, the event handler does not get events while suspended, and any events missed while in the suspended state are not provided to the operation when it is resumed.

While suspended, the consumer function continues to get the control type callbacks.

MQOP_RESUME
Resume the consuming of messages for the object handle.

If this operation is applied to an event handler, the event handler does not get events while suspended, and any events missed while in the suspended state are not provided to the operation when it is resumed.

CallbackDesc
Type: MQCBD - input

This is a structure that identifies the callback function that is being registered by the application and the options used when registering it.

See MQCBD for details of the structure.

Callback descriptor is required only for the MQOP_REGISTER option; if the descriptor is not required, the parameter address passed can be null.

Hobj
Type: MQHOBJ - input

This handle represents the access that has been established to the object from which a message is to be consumed. This is a handle that has been returned from a previous MQOPEN or MQSUB call (in the **Hobj** parameter).

Hobj is not required when defining an event handler routine (MQCBT_EVENT_HANDLER) and should be specified as MQHO_NONE.

If *Hobj* has been returned from an MQOPEN call, the queue must have been opened with one or more of the following options:

- MQOO_INPUT_SHARED
- MQOO_INPUT_EXCLUSIVE
- MQOO_INPUT_AS_Q_DEF
- MQOO_BROWSE

MsgDesc
Type: MQMD - input

This structure describes the attributes of the message required, and the attributes of the message retrieved.

The **MsgDesc** parameter defines the attributes of the messages required by the consumer, and the version of the MQMD to be passed to the message consumer.

The *MsgId*, *CorrelId*, *GroupId*, *MsgSeqNumber*, and *Offset* in the MQMD are used for message selection, depending on the options specified in the **GetMsgOpts** parameter.

The *Encoding* and *CodedCharSetId* are used for message conversion if you specify the MQGMO_CONVERT option.

See MQMD for details.

MsgDesc is used for MQOP_REGISTER and if you require values other than the default for any fields. *MsgDesc* is not used for an event handler.

If the descriptor is not required the parameter address passed can be null.

Note, that if multiple consumers are registered against the same queue with overlapping selectors, the chosen consumer for each message is undefined.

GetMsgOpts
Type: MQGMO - input

The **GetMsgOpts** parameter controls how the message consumer gets messages. All options of this parameter have meanings as described in "MQGMO - Get-message options" on page 2330, when used on an MQGET call, except:

MQGMO_SET_SIGNAL
This option is not permitted.

MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT, MQGMO_MARK_*
The order of messages delivered to a browsing consumer is dictated by the combinations of these options. Significant combinations are:

MQGMO_BROWSE_FIRST
The first message on the queue is delivered repeatedly to the consumer. This is useful when the consumer destructively consumes the message in the callback. Use this option with care.

MQGMO_BROWSE_NEXT
The consumer is given each message on the queue, from the current cursor position until the end of the queue is reached.

MQGMO_BROWSE_FIRST + MQGMO_BROWSE_NEXT
The cursor is reset to the start of the queue. The consumer is then given each message until the cursor reaches the end of the queue.

MQGMO_BROWSE_FIRST + MQGMO_MARK_*
Starting at the beginning of the queue, the consumer is given the first nonmarked message on the queue, which is then marked for this consumer. This combination ensures that the consumer can receive new messages added behind the current cursor point.

MQGMO_BROWSE_NEXT + MQGMO_MARK_*
Starting at the cursor position, the consumer is given the next nonmarked message on the queue, which is then marked for this consumer. Use this combination with care because messages can be added to the queue behind the current cursor position.

MQGMO_BROWSE_FIRST + MQGMO_BROWSE_NEXT + MQGMO_MARK_*
This combination is not permitted. If used the call returns MQRC_OPTIONS_ERROR.

MQGMO_NO_WAIT, MQGMO_WAIT, and WaitInterval
These options control how the consumer is invoked.

MQGMO_NO_WAIT
The consumer is never called with MQRC_NO_MSG_AVAILABLE. The consumer is only called for messages and events.

MQGMO_WAIT with a zero WaitInterval
The MQRC_NO_MSG_AVAILABLE code is passed to the consumer when there are no messages available and either the consumer has been started or the consumer has been delivered at least one message since the last "no messages" reason code.

This prevents the consumer from polling in a busy loop when a zero wait interval is specified.

MQGMO_WAIT and a positive WaitInterval
The consumer is called after the specified wait interval with reason code MQRC_NO_MSG_AVAILABLE. This call is made regardless of whether any messages have been delivered to the consumer. This allows the user to perform heartbeat or batch type processing.

MQGMO_WAIT and WaitInterval of MQWI_UNLIMITED
This specifies an infinite wait before returning MQRC_NO_MSG_AVAILABLE. The consumer is never called with MQRC_NO_MSG_AVAILABLE.

GetMsgOpts is used only for MQOP_REGISTER and if you require values other than the default for any fields. *GetMsgOpts* is not used for an event handler.

If the *GetMsgOpts* are not required, the parameter address passed can be null. Using this parameter is the same as specifying MQGMO_DEFAULT together with MQGMO_FAIL_IF_QUIESCING.

If a message properties handle is provided in the MQGMO structure, a copy is provided in the MQGMO structure that is passed into the consumer callback. On return from the MQCB call, the application can delete the message properties handle.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

The reason codes in the following list are the ones that the queue manager can return for the **Reason** parameter.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_CONV_LOAD_ERROR
(2133, X'855') Unable to load data conversion services modules.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CALLBACK_LINK_ERROR
(2487, X'9B7') Incorrect callback type field.

MQRC_CALLBACK_NOT_REGISTERED
(2448, X'990') Unable to unregister, suspend, or resume because there is no registered callback.

MQRC_CALLBACK_ROUTINE_ERROR
(2486, X'9B6') Either *CallbackFunction* or *CallbackName* must be specified but not both.

MQRC_CALLBACK_TYPE_ERROR
(2483, X'9B3') Incorrect callback type field.

MQRC_CBD_OPTIONS_ERROR
(2484, X'9B4') Incorrect MQCBD options field.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_CORREL_ID_ERROR
(2207, X'89F') Correlation-identifier error.

MQRC_DATA_LENGTH_ERROR
(2010, X'7DA') Data length parameter not valid.

MQRC_FUNCTION_NOT_SUPPORTED
(2298, X'8FA') The function requested is not available in the current environment.

MQRC_GET_INHIBITED
(2016, X'7E0') Gets inhibited for the queue.

MQRC_GLOBAL_UOW_CONFLICT
(2351, X'92F') Global units of work conflict.

MQRC_GMO_ERROR
(2186, X'88A') Get-message options structure not valid.

MQRC_HANDLE_IN_USE_FOR_UOW
(2353, X'931') Handle in use for global unit of work.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_INCONSISTENT_BROWSE
(2259, X'8D3') Inconsistent browse specification.

MQRC_INCONSISTENT_UOW
(2245, X'8C5') Inconsistent unit-of-work specification.

MQRC_INVALID_MSG_UNDER_CURSOR
(2246, X'8C6') Message under cursor not valid for retrieval.

MQRC_LOCAL_UOW_CONFLICT
(2352, X'930') Global unit of work conflicts with local unit of work.

MQRC_MATCH_OPTIONS_ERROR
(2247, X'8C7') Match options not valid.

MQRC_MAX_MSG_LENGTH_ERROR
(2485, X'9B4') Incorrect *MaxMsgLength* field.

MQRC_MD_ERROR
(2026, X'7EA') Message descriptor not valid.

MQRC_MODULE_ENTRY_NOT_FOUND
(2497, X'9C1') The specified function entry point could not be found in the module.

MQRC_MODULE_INVALID
(2496, X'9C0') Module found, however it is of the wrong type; not 32 bit, 64 bit, or a valid dynamic link library.

MQRC_MODULE_NOT_FOUND
(2495, X'9BF') Module not found in the search path or not authorized to load.

MQRC_MSG_SEQ_NUMBER_ERROR
(2250, X'8CA') Message sequence number not valid.

MQRC_MSG_TOKEN_ERROR
(2331, X'91B') Use of message token not valid.

MQRC_NO_MSG_AVAILABLE
(2033, X'7F1') No message available.

MQRC_NO_MSG_UNDER_CURSOR
(2034, X'7F2') Browse cursor not positioned on message.

MQRC_NOT_OPEN_FOR_BROWSE
(2036, X'7F4') Queue not open for browse.

MQRC_NOT_OPEN_FOR_INPUT
(2037, X'7F5') Queue not open for input.

MQRC_OBJECT_CHANGED
(2041, X'7F9') Object definition changed since opened.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OPERATION_ERROR
(2206, X'89E') Incorrect operation code on API Call.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_INDEX_TYPE_ERROR
(2394, X'95A') Queue has wrong index type.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SIGNAL_OUTSTANDING
(2069, X'815') Signal outstanding for this handle.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_SYNCPOINT_LIMIT_REACHED
(2024, X'7E8') No more messages can be handled within current unit of work.

MQRC_SYNCPOINT_NOT_AVAILABLE
(2072, X'818') Sync point support not available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

MQRC_UOW_ENLISTMENT_ERROR
(2354, X'932') Enlistment in global unit of work failed.

MQRC_UOW_MIX_NOT_SUPPORTED
(2355, X'933') Mixture of unit-of-work calls not supported.

MQRC_UOW_NOT_AVAILABLE
(2255, X'8CF') Unit of work not available for the queue manager to use.

MQRC_WAIT_INTERVAL_ERROR
(2090, X'82A') Wait interval in MQGMO not valid.

MQRC_WRONG_GMO_VERSION
(2256, X'8D0') Wrong version of MQGMO supplied.

MQRC_WRONG_MD_VERSION
(2257, X'8D1') Wrong version of MQMD supplied.

For detailed information about these codes, see Reason codes.

Usage notes

1. MQCB is used to define the action to be invoked for each message, matching the specified criteria, available on the queue. When the action is processed, either the message is removed from the queue and passed to the defined message consumer, or a message token is provided, which is used to retrieve the message.
2. MQCB can be used to define callback routines before starting consumption with MQCTL or it can be used from within a callback routine.
3. To use MQCB from outside of a callback routine, you must first suspend message consumption by using MQCTL and resume consumption afterward.
4. MQCB is not supported within the IMS adapter.

Message consumer callback sequence

You can configure a consumer to invoke callback at key points during the lifecycle of the consumer. For example:

- when the consumer is first registered,
- when the connection is started,
- when the connection is stopped and
- when the consumer is deregistered, either explicitly, or implicitly by an MQCLOSE.

Table 262. MQCTL verb definitions

Verb	Meaning
MQCTL(START)	MQCTL call using the MQOP_START Operation
MQCTL(STOP)	MQCTL call using the MQOP_STOP Operation
MQCTL(WAIT)	MQCTL call using the MQOP_START_WAIT Operation

This allows the consumer to maintain state associated with the consumer. When a callback is requested by an application, the rules for consumer invocation are as follows:

REGISTER
Is always the first type of invocation of the callback.

Is always called on the same thread, as the MQCB(REGISTER) call.

START
Is always called synchronously with the MQCTL(START) verb.

- All START callbacks are completed before the MQCTL(START) verb returns.

Is on the same thread as the message delivery if THREAD_AFFINITY is requested.

The call with start is not guaranteed if, for example, a previous callback issues MQCTL(STOP) during the MQCTL(START).

STOP No further messages or events are delivered after this call until the connection is restarted.

A STOP is guaranteed if the application was previously called for START, or a message, or an event.

DEREGISTER
Is always the last type of invocation of the callback.

Ensure that your application performs thread-based initialization and cleanup in the START and STOP callbacks. You can do non-thread based initialization and cleanup with REGISTER and DEREGISTER callbacks.

Do not make any assumptions about the life and availability of the thread other than what is stated. For example, do not rely on a thread staying alive beyond the last call to DEREGISTER. Similarly, when you have chosen not to use THREAD_AFFINITY, do not assume that the thread exists whenever the connection is started.

If your application has particular requirements for thread characteristics, it can always create a thread accordingly, then use MQCTL(WAIT). This has the effect of 'donating' the thread to IBM MQ for asynchronous message delivery.

Message consumer connection usage

You can configure a consumer to invoke callback at key points during the lifecycle of the consumer. For example:

- when the consumer is first registered,
- when the connection is started,
- when the connection is stopped and
- when the consumer is deregistered, either explicitly, or implicitly by an MQCLOSE.

Table 263. MQCTL verb definitions

Verb	Meaning
MQCTL(START)	MQCTL call using the MQOP_START Operation
MQCTL(STOP)	MQCTL call using the MQOP_STOP Operation
MQCTL(WAIT)	MQCTL call using the MQOP_START_WAIT Operation

This allows the consumer to maintain state associated with the consumer. When a callback is requested by an application, the rules for consumer invocation are as follows:

REGISTER
Is always the first type of invocation of the callback.

Is always called on the same thread, as the MQCB(REGISTER) call.

START
Is always called synchronously with the MQCTL(START) verb.

- All START callbacks are completed before the MQCTL(START) verb returns.

Is on the same thread as the message delivery if THREAD_AFFINITY is requested.

The call with start is not guaranteed if, for example, a previous callback issues MQCTL(STOP) during the MQCTL(START).

STOP No further messages or events are delivered after this call until the connection is restarted.

A STOP is guaranteed if the application was previously called for START, or a message, or an event.

DEREGISTER
Is always the last type of invocation of the callback.

Ensure that your application performs thread-based initialization and cleanup in the START and STOP callbacks. You can do non-thread based initialization and cleanup with REGISTER and DEREGISTER callbacks.

Do not make any assumptions about the life and availability of the thread other than what is stated. For example, do not rely on a thread staying alive beyond the last call to DEREGISTER. Similarly, when you have chosen not to use THREAD_AFFINITY, do not assume that the thread exists whenever the connection is started.

If your application has particular requirements for thread characteristics, it can always create a thread accordingly, then use MQCTL(WAIT). This has the effect of 'donating' the thread to IBM MQ for asynchronous message delivery.

C invocation

```
MQCB (Hconn, Operation, CallbackDesc, Hobj, MsgDesc,
GetMsgOpts, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;         /* Connection handle */
MQLONG   Operation;     /* Operation being processed */
MQCBD    CallbackDesc;  /* Callback descriptor */
MQHOBJ   HObj           /* Object handle */
MQMD     MsgDesc        /* Message descriptor attributes */
MQGMO    GetMsgOpts     /* Message options */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCB' USING HCONN, OPERATION, CBDESC, HOBJ, MSGDESC,
            GETMSGOPTS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN    PIC S9(9) BINARY.
**   Operation
 01  OPERATION PIC S9(9) BINARY.
**   Callback Descriptior
 01  CBDESC.
     COPY CMQCBDV.
01  HOBJ     PIC S9(9) BINARY.
**   Message Descriptior
 01  MSGDESC.
     COPY CMQMDV.
**   Get Message Options
 01  GETMSGOPTS.
     COPY CMQGMOV.
**   Completion code
 01  COMPCODE PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON   PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCB(Hconn, Operation, CallbackDesc, Hobj, MsgDesc, GetMsgOpts,
          CompCode, Reason)
```

Declare the parameters as follows:

```
dcl Hconn        fixed bin(31); /* Connection handle */
dcl Operation    fixed bin(31); /* Operation */
dcl CallbackDesc like MQCBD;    /* Callback Descriptor */
dcl Hobj         fixed bin(31); /* Object Handle */
dcl MsgDesc      like MQMD;     /* Message Descriptor */
dcl GetMsgOpts   like MQGMO;    /* Get Message Options */
dcl CompCode     fixed bin(31); /* Completion code */
dcl Reason       fixed bin(31); /* Reason code qualifying CompCode */
```

MQCB_FUNCTION - Callback function:

The MQCB_FUNCTION function call is the callback function for event handling and asynchronous message consumption.

The MQCB_FUNCTION call definition is provided solely to describe the parameters that are passed to the callback function. No entry point called MQCB_FUNCTION is provided by the queue manager.

The specification of the actual function to be called is an input to the MQCB call and is passed in through the MQCBD structure.

Syntax

MQCB_FUNCTION (*Hconn*, *MsgDesc*, *GetMsgOpts*, *Buffer*, *Context*)

Parameters

Hconn
Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call. On z/OS for CICS applications the MQCONN call can be omitted, and the following value specified for Hconn:

MQHC_DEF_CONN

Default connection handle.

MsgDesc
Type: MQMD - input

This structure describes the attributes of the message retrieved.

See "MQMD - Message descriptor" on page 2387 for details.

The version of MQMD passed is the same version as passed on the MQCB call that defined the consumer function.

The address of the MQMD is passed as null characters if a version 4 MQGMO was used to request that a Message Handle be returned instead of an MQMD.

This is an input field to the message consumer function; it is not relevant to an event handler function.

GetMsgOpts
Type: MQGMO - input

Options used to control the actions of the message consumer. This parameter also contains additional information about the message returned.

See MQGMO for details.

The version of MQGMO passed is the latest version supported.

This is an input field to the message consumer function; it is not relevant to an event handler function.

Buffer
Type: MQBYTExBufferLength - input

This is the area containing the message data.

If no message is available for this call, or if the message contains no message data, the address of the *Buffer* is passed as nulls.

This is an input field to the message consumer function; it is not relevant to an event handler function.

Context

Type: MQCBC - input/output

This structure provides context information to the callback functions. See "MQCBC - Callback context" on page 2235 for details.

Usage notes

1. Be aware that if your callback routines use services that could delay or block the thread, for example, MQGET with wait, could delay the dispatch of other callbacks.
2. A separate unit of work is not automatically established for each invocation of a callback routine, so routines can either issue a commit call, or defer committing, until a logical batch of work has been processed. When the batch of work is committed, it commits the messages for all callback functions that have been invoked since the last sync point.
3. Programs invoked by CICS LINK or CICS START retrieve parameters using CICS services through named objects known as channel containers. The container names are the same as the parameter names. For more information, see your CICS documentation.
4. Callback routines can issue an MQDISC call, but not for their own connection. For example, if a callback routine has created a connection, then it can also disconnect the connection.
5. A callback routine should not, in general, rely on being invoked from the same thread each time. If required, use the MQCTLO_THREAD_AFFINITY when the connection is started.
6. When a callback routine receives a nonzero reason code, it must take appropriate action.
7. MQCB_FUNCTION is not supported within the IMS adapter.

MQCLOSE - Close object:

The MQCLOSE call relinquishes access to an object, and is the inverse of the MQOPEN and MQSUB calls.

Syntax

MQCLOSE (*Hconn*, *Hobj*, *Options*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications you can omit the MQCONN call, and specify the following value for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hobj

Type: MQHOBJ - input/output

This handle represents the object that is being closed. The object can be of any type. The value of *Hobj* was returned by a previous MQOPEN call.

On successful completion of the call, the queue manager sets this parameter to a value that is not a valid handle for the environment. This value is:

MQHO_UNUSABLE_HOBJ
Unusable object handle.

On z/OS, *Hobj* is set to a value that is undefined.

Options
Type: MQLONG - input

This parameter controls how the object is closed.

Only permanent dynamic queues and subscriptions can be closed in more than one way, because they must be either retained or deleted; these are queues with the **DefinitionType** attribute that has the value MQQDT_PERMANENT_DYNAMIC (see the **DefinitionType** attribute described in "Attributes for queues" on page 2833). The close options are summarized in this topic.

Durable subscriptions can either be kept or removed; these are created using the MQSUB call with the MQSO_DURABLE option.

When closing the handle to a managed destination (that is the **Hobj** parameter returned on an MQSUB call which used the MQSO_MANAGED option) the queue manager cleans up any publications that have not been retrieved when the associated subscription has also been removed. The subscription is removed using the MQCO_REMOVE_SUB option on the **Hsub** parameter returned on an MQSUB call. Note MQCO_REMOVE_SUB is the default behavior on MQCLOSE for a non-durable subscription.

When closing a handle to a non-managed destination you are responsible for cleaning up the queue where publications are sent. Close the subscription using MQCO_REMOVE_SUB first and then process messages off the queue until none remain.

You must specify one option only from the following:

Dynamic queue options: These options control how permanent dynamic queues are closed.

MQCO_DELETE
The queue is deleted if either of the following is true:

- It is a permanent dynamic queue, created by a previous MQOPEN call, and there are no messages on the queue and no uncommitted get or put requests outstanding for the queue (either for the current task or any other task).
- It is the temporary dynamic queue that was created by the MQOPEN call that returned *Hobj*. In this case, all the messages on the queue are purged.

In all other cases, including the case where the *Hobj* was returned on an MQSUB call, the call fails with reason code MQRC_OPTION_NOT_VALID_FOR_TYPE, and the object is not deleted.

On z/OS, if the queue is a dynamic queue that has been logically deleted, and this is the last handle for it, the queue is physically deleted. See "Usage notes" on page 2650 for further details.

MQCO_DELETE_PURGE
The queue is deleted, and any messages on it purged, if either of the following is true:

- It is a permanent dynamic queue, created by a previous MQOPEN call, and there are no uncommitted get or put requests outstanding for the queue (either for the current task or any other task).
- It is the temporary dynamic queue that was created by the MQOPEN call that returned *Hobj*.

In all other cases, including the case where the *Hobj* was returned on an MQSUB call, the call fails with reason code MQRC_OPTION_NOT_VALID_FOR_TYPE, and the object is not deleted.

The table shows which close options are valid, and whether the object is retained or deleted.

Type of object or queue	MQCO_NONE	MQCO_DELETE	MQCO_DELETE_PURGE
Object other than a queue	Retained	Not valid	Not valid
Predefined queue	Retained	Not valid	Not valid
Permanent dynamic queue	Retained	Deleted if empty and no pending updates	Messages deleted; queue deleted if no pending updates
Temporary dynamic queue (call issued by creator of queue)	Deleted	Deleted	Deleted
Temporary dynamic queue (call not issued by creator of queue)	Retained	Not valid	Not valid
Distribution list	Retained	Not valid	Not valid
Managed subscription destination	Retained	Not valid	Not valid
Distribution list (subscription has been removed)	Messages deleted; queue deleted	Not valid	Not valid

Subscription closure options: These options control whether durable subscriptions are removed when the handle is closed, and whether publications still waiting to be read by the application are cleaned up. These options are only valid for use with an object handle returned in the **Hsub** parameter of an MQSUB call.

MQCO_KEEP_SUB
The handle to the subscription is closed but the subscription made is kept. Publications continue to be sent to the destination specified in the subscription. This option is only valid if the subscription was made with the option MQSO_DURABLE.

MQCO_KEEP_SUB is the default if the subscription is durable

MQCO_REMOVE_SUB
The subscription is removed and the handle to the subscription is closed.

The **Hobj** parameter of the MQSUB call is not invalidated by closure of the **Hsub** parameter and might continue to be used for MQGET or MQCB to receive the remaining publications. When the **Hobj** parameter of the MQSUB call is also closed, if it was a managed destination any unretrieved publications are removed.

MQCO_REMOVE_SUB is the default if the subscription is non-durable.

Successful completion of MQCO_REMOVE_SUB does not mean that the action completed. To check that this call has completed, see the DELETE SUB step in Checking that async commands for distributed networks have finished.

These subscription closure options are summarized in the following tables.

To close a durable subscription handle but retain the subscription, use the following subscription closure options:

Task	Subscription closure option
Keep publications on an MQOPENed handle	MQCO_KEEP_SUB
Remove publications on an MQOPENed handle	Action not allowed
Keep publications on an MQSO_MANAGED handle	MQCO_KEEP_SUB
Remove publications on an MQSO_MANAGED handle	Action not allowed

To unsubscribe, either by closing a durable subscription handle and unsubscribing it or closing a non-durable subscription handle, use the following subscription closure options:

Task	Subscription closure option
Keep publications on an MQOPENed handle	MQCO_REMOVE_SUB
Remove publications on an MQOPENed handle	Action not allowed
Keep publications on an MQSO_MANAGED handle	MQCO_REMOVE_SUB

Read ahead options: The following options control what happens to non-persistent messages which have been sent to the client before an application requested them and have not yet been consumed by the application. These messages are stored in the client read ahead buffer waiting to be requested by the application and can either be discarded or consumed from the queue before the MQCLOSE is completed.

MQCO_IMMEDIATE
The object is closed immediately and any messages which have been sent to the client before an application requested them are discarded and are not available to be consumed by any application. This is the default value.

MQCO_QUIESCE
A request to close the object is made, but if any messages which have been sent to the client before an application requested them, still reside in the client read ahead buffer, the MQCLOSE call returns with a warning of MQRC_READ_AHEAD_MSGS and the object handle remains valid.

The application can then continue to use the object handle to retrieve messages until no more are available, and then close the object again. No more messages are sent to the client ahead of an application requesting them, read ahead is now turned off.

Applications are advised to use MQCO_QUIESCE rather than trying to reach a point where there are no more messages in the client read ahead buffer, because a message could arrive between the last MQGET call and the following MQCLOSE which would be discarded if MQCO_IMMEDIATE was used.

If an MQCLOSE with MQCO_QUIESCE is issued from within an asynchronous callback function, the same behavior of reading ahead messages applies. If the warning MQRC_READ_AHEAD_MSGS is returned, then the callback function is called at least one more time. When the last remaining message that was read ahead has been passed to the callback function the MQCBC ConsumerFlags field is set to MQCBCF_READA_BUFFER_EMPTY.

Default option: If you require none of the options described previously, you can use the following option:

MQCO_NONE
No optional close processing required.

This must be specified for:

- Objects other than queues
- Predefined queues
- Temporary dynamic queues (but only in those cases where *Hobj* is not the handle returned by the MQOPEN call that created the queue).
- Distribution lists

In all the above cases, the object is retained and not deleted.

If this option is specified for a temporary dynamic queue:

- The queue is deleted, if it was created by the MQOPEN call that returned *Hobj* ; any messages that are on the queue are purged.
- In all other cases the queue (and any messages on it) are retained.

If this option is specified for a permanent dynamic queue, the queue is retained and not deleted.

On z/OS, if the queue is a dynamic queue that has been logically deleted, and this is the last handle for it, the queue is physically deleted. See "Usage notes" on page 2650 for further details.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

The reason codes listed are the ones that the queue manager can return for the **Reason** parameter.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_INCOMPLETE_GROUP
(2241, X'8C1') Message group not complete.

MQRC_INCOMPLETE_MSG
(2242, X'8C2') Logical message not complete.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_NOT_AVAILABLE
(2345, X'929') Coupling facility not available.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC__DB2_NOT_AVAILABLE
(2342, X'926') Db2 subsystem not available.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OPTION_NOT_VALID_FOR_TYPE
(2045, X'7FD') On an MQOPEN or MQCLOSE call: option not valid for object type.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_Q_NOT_EMPTY
(2055, X'807') Queue contains one or more messages or uncommitted put or get requests.

MQRC_READ_AHEAD_MSGS
(nnnn, X'xxx') The client has read ahead messages that have not yet been consumed by the application.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SECURITY_ERROR
(2063, X'80F') Security error occurred.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

Usage notes

1. When an application issues the MQDISC call, or ends either normally or abnormally, any objects that were opened by the application and are still open are closed automatically with the MQCO_NONE option.
2. The following points apply if the object being closed is a *queue*:
   - If operations on the queue are performed as part of a unit of work, the queue can be closed before or after the sync point occurs without affecting the outcome of the sync point. If the queue is triggered, performing a rollback before closing the queue can cause a trigger message to be issued. For more information about trigger messages, see Properties of trigger messages.
   - If the queue was opened with the MQOO_BROWSE option, the browse cursor is destroyed. If the queue is then reopened with the MQOO_BROWSE option, a new browse cursor is created (see MQOO_BROWSE).
   - If a message is currently locked for this handle at the time of the MQCLOSE call, the lock is released (see MQGMO_LOCK).
   - On z/OS, if there is an MQGET request with the MQGMO_SET_SIGNAL option outstanding against the queue handle being closed, the request is canceled (see MQGMO_SET_SIGNAL). Signal requests for the same queue but lodged against different handles (*Hobj*) are not affected (unless a dynamic queue is being deleted, in which case they are also canceled).
3. The following points apply if the object being closed is a *dynamic queue* (either permanent or temporary):
   - For a dynamic queue, you can specify the MQCO_DELETE and MQCO_DELETE_PURGE options regardless of the options specified on the corresponding MQOPEN call.
   - When a dynamic queue is deleted, all MQGET calls with the MQGMO_WAIT option that are outstanding against the queue are canceled and reason code MQRC_Q_DELETED is returned. See MQGMO_WAIT.

     Although applications cannot access a deleted queue, the queue is not removed from the system, and associated resources are not freed, until all handles that reference the queue have been closed, and all units of work that affect the queue have been either committed or backed out.

     On z/OS, a queue that has been logically deleted but not yet removed from the system prevents the creation of a new queue with the same name as the deleted queue; the MQOPEN call fails with reason code MQRC_NAME_IN_USE in this case. Also, such a queue can still be displayed using MQSC commands, even though it cannot be accessed by applications.
   - When a permanent dynamic queue is deleted, if the *Hobj* handle specified on the MQCLOSE call is not the one that was returned by the MQOPEN call that created the queue, a check is made that the user identifier that was used to validate the MQOPEN call is authorized to delete the queue. If the MQOO_ALTERNATE_USER_AUTHORITY option was specified on the MQOPEN call, the user identifier checked is the *AlternateUserId*.

     This check is not performed if:
     - The handle specified is the one returned by the MQOPEN call that created the queue.
     - The queue being deleted is a temporary dynamic queue.
   - When a temporary dynamic queue is closed, if the *Hobj* handle specified on the MQCLOSE call is the one that was returned by the MQOPEN call that created the queue, the queue is deleted. This occurs regardless of the close options specified on the MQCLOSE call. If there are messages on the queue, they are discarded; no report messages are generated.

     If there are uncommitted units of work that affect the queue, the queue and its messages are still deleted, but the units of work do not fail. However, as described previously, the resources associated with the units of work are not freed until each of the units of work has been either committed or backed out.
4. The following points apply if the object being closed is a *distribution list*:

- The only valid close option for a distribution list is MQCO_NONE; the call fails with reason code MQRC_OPTIONS_ERROR or MQRC_OPTION_NOT_VALID_FOR_TYPE if any other options are specified.
- When a distribution list is closed, individual completion codes and reason codes are not returned for the queues in the list; only the **CompCode** and **Reason** parameters of the call are available for diagnostic purposes.

  If a failure occurs closing one of the queues, the queue manager continues processing and attempts to close the remaining queues in the distribution list. The **CompCode** and **Reason** parameters of the call are set to return information describing the failure. It is possible for the completion code to be MQCC_FAILED, even though most of the queues were closed successfully. The queue that encountered the error is not identified.

  If there is a failure on more than one queue, it is not defined which failure is reported in the **CompCode** and **Reason** parameters.

C invocation

```
MQCLOSE (Hconn, &Hobj, Options, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;     /* Connection handle */
MQHOBJ   Hobj;      /* Object handle */
MQLONG   Options;   /* Options that control the action of MQCLOSE */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

COBOL invocation

```
     CALL 'MQCLOSE' USING HCONN, HOBJ, OPTIONS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN     PIC S9(9) BINARY.
**   Object handle
 01  HOBJ      PIC S9(9) BINARY.
**   Options that control the action of MQCLOSE
 01  OPTIONS   PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE  PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON    PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCLOSE (Hconn, Hobj, Options, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn     fixed bin(31);  /* Connection handle */
dcl Hobj      fixed bin(31);  /* Object handle */
dcl Options   fixed bin(31);  /* Options that control the action of
                                 MQCLOSE */
dcl CompCode  fixed bin(31);  /* Completion code */
dcl Reason    fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQCLOSE,(HCONN,HOBJ,OPTIONS,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN     DS  F  Connection handle
HOBJ      DS  F  Object handle
OPTIONS   DS  F  Options that control the action of MQCLOSE
COMPCODE  DS  F  Completion code
REASON    DS  F  Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQCLOSE Hconn, Hobj, Options, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn    As Long 'Connection handle'
Dim Hobj     As Long 'Object handle'
Dim Options  As Long 'Options that control the action of MQCLOSE'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

MQCMIT - Commit changes:

The MQCMIT call indicates to the queue manager that the application has reached a sync point, and that all the message gets and puts that have occurred since the last sync point are to be made permanent.

Messages put as part of a unit of work are made available to other applications; messages retrieved as part of a unit of work are deleted.

- z/OS On z/OS, the call is used only by batch programs (including IMS batch DL/I programs).

Syntax

MQCMIT (*Hconn*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

The reason codes listed are the ones that the queue manager can return for the **Reason** parameter.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_BACKED_OUT
(2003, X'7D3') Unit of work backed out.

MQRC_OUTCOME_PENDING
(2124, X'84C') Result of commit operation is pending.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CALL_INTERRUPTED
(2549, X'9F5') MQPUT or MQCMIT was interrupted and reconnection processing cannot reestablish a definite outcome.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_ENVIRONMENT_ERROR
(2012, X'7DC') Call not valid in environment.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OUTCOME_MIXED
(2123, X'84B') Result of commit or back-out operation is mixed.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RECONNECT_FAILED
(2548, X'9F4') After reconnecting, an error occurred reinstating the handles for a reconnectable connection.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_STORAGE_MEDIUM_FULL
(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

Usage notes

1. Use this call only when the queue manager itself coordinates the unit of work. This can be:
   - A local unit of work, where the changes affect only IBM MQ resources.
   - A global unit of work, where the changes can affect resources belonging to other resource managers, as well as affecting IBM MQ resources.

   For further details about local and global units of work, see "MQBEGIN - Begin unit of work" on page 2626.
2. In environments where the queue manager does not coordinate the unit of work, the appropriate commit call must be used instead of MQCMIT. The environment might also support an implicit commit caused by the application terminating normally.
   - On z/OS, use the following calls:
     - Batch programs (including IMS batch DL/I programs) can use the MQCMIT call if the unit of work affects only IBM MQ resources. However, if the unit of work affects both IBM MQ resources and resources belonging to other resource managers (for example, Db2), use the SRRCMIT call provided by the z/OS Recoverable Resource Service (RRS). The SRRCMIT call commits changes to resources belonging to the resource managers that have been enabled for RRS coordination.
     - CICS applications must use the `EXEC CICS SYNCPOINT` command to commit the unit of work explicitly. Alternatively, ending the transaction results in an implicit commit of the unit of work. The MQCMIT call cannot be used for CICS applications.
     - IMS applications (other than batch DL/I programs) must use IMS calls such as `GU` and `CHKP` to commit the unit of work. The MQCMIT call cannot be used for IMS applications (other than batch DL/I programs).
   - On IBM i, use this call for local units of work coordinated by the queue manager. This means that a commitment definition must not exist at job level, that is, the `STRCMTCTL` command with the **`CMTSCOPE(*JOB)`** parameter must not have been issued for the job.
3. If an application ends with uncommitted changes in a unit of work, the disposition of those changes depends on whether the application ends normally or abnormally. See MQDISC usage notes for further details.
4. When an application puts or gets messages in groups or segments of logical messages, the queue manager retains information relating to the message group and logical message for the last successful MQPUT and MQGET calls. This information is associated with the queue handle, and includes such things as:
   - The values of the *`GroupId`*, *`MsgSeqNumber`*, *`Offset`*, and *`MsgFlags`* fields in MQMD.
   - Whether the message is part of a unit of work.
   - For the MQPUT call: whether the message is persistent or nonpersistent.

   When a unit of work is committed, the queue manager retains the group and segment information, and the application can continue putting or getting messages in the current message group or logical message.

   Retaining the group and segment information when a unit of work is committed allows the application to spread a large message group or large logical message consisting of many segments across several units of work. Using several units of work is advantageous if the local queue manager has only limited queue storage. However, the application must maintain sufficient information to restart putting or getting messages at the correct point if a system failure occurs. For details of how to restart at the correct point after a system failure, see MQPMO_LOGICAL_ORDER and MQGMO_LOGICAL_ORDER.

   The remaining usage notes apply only when the queue manager coordinates the units of work:
5. A unit of work has the same scope as a connection handle; all IBM MQ calls that affect a particular unit of work must be performed using the same connection handle. Calls issued using a different

connection handle (for example, calls issued by another application) affect a different unit of work. See the **Hconn** parameter described in MQCONN for information about the scope of connection handles.

6. Only messages that were put or retrieved as part of the current unit of work are affected by this call.
7. A long-running application that issues MQGET, MQPUT, or MQPUT1 calls within a unit of work, but that never issues a commit or back-out call, can fill queues with messages that are not available to other applications. To guard against this, the administrator must set the **MaxUncommittedMsgs** queue manager attribute to a value that is low enough to prevent runaway applications filling the queues, but high enough to allow the expected messaging applications to work correctly.
8. UNIX Windows On UNIX and Windows systems, if the **Reason** parameter is MQRC_CONNECTION_BROKEN (with a *CompCode* of MQCC_FAILED), or MQRC_UNEXPECTED_ERROR it is possible that the unit of work was successfully committed.

C invocation

```
MQCMIT (Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;     /* Connection handle */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCMIT' USING HCONN, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN     PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE  PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON    PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCMIT (Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn     fixed bin(31);  /* Connection handle */
dcl CompCode  fixed bin(31);  /* Completion code */
dcl Reason    fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQCMIT,(HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN     DS  F  Connection handle
COMPCODE  DS  F  Completion code
REASON    DS  F  Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQCMIT Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn    As Long 'Connection handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

MQCONN - Connect queue manager:

The MQCONN call connects an application program to a queue manager.

It provides a queue manager connection handle, which the application uses on subsequent message queuing calls.

- On z/OS, CICS applications do not have to issue this call. These applications are connected automatically to the queue manager to which the CICS system is connected. However, the MQCONN and MQDISC calls are still accepted from CICS applications.
- On IBM i, applications must use the MQCONN or MQCONNX call to connect to the queue manager, and the MQDISC call to disconnect from the queue manager.

A client connection cannot be made on a server only installation, and a local connection cannot be made on a client only installation.

Syntax

MQCONN (*QMgrName*, *Hconn*, *CompCode*, *Reason*)

Parameters

QMgrName

Type: MQCHAR48 - input

This is the name of the queue manager to which the application wants to connect. The name can contain the following characters:

- Uppercase alphabetic characters (A through Z)
- Lowercase alphabetic characters (a through z)
- Numeric digits (0 through 9)
- Period (.), forward slash (/), underscore (_), percent (%)

The name must not contain leading or embedded blanks, but can contain trailing blanks. A null character can be used to indicate the end of significant data in the name; the null and any characters following it are treated as blanks. The following restrictions apply in the environments indicated:

- On systems that use EBCDIC Katakana, lowercase characters cannot be used.
- On z/OS, names that begin or end with an underscore cannot be processed by the operations and control panels. For this reason, avoid such names.
- On IBM i, enclose names containing lowercase characters, forward slash, or percent in quotation marks when specified on commands. Do not specify these quotation marks in the **QMgrName** parameter.

If the name consists entirely of blanks, the name of the *default* queue manager is used.

The name specified for *QMgrName* must be the name of a *connectable* queue manager.

On z/OS, the queue managers to which it is possible to connect are determined by the environment:

- For CICS, you can use only the queue manager to which the CICS system is connected. The **QMgrName** parameter must still be specified, but its value is ignored; blank characters are a suitable option.
- For IMS, only queue managers that are listed in the subsystem definition table (CSQQDEFV), *and* listed in the SSM table in IMS, are connectable (see usage note 6).
- For z/OS batch and TSO, only queue managers that reside on the same system as the application are connectable (see usage note 6).

Queue-sharing groups: On systems where several queue managers exist and are configured to form a queue-sharing group, the name of the queue-sharing group can be specified for *QMgrName* in place of the name of a queue manager. This allows the application to connect to *any* queue manager that is

available in the queue-sharing group and that is on the same z/OS image as the application. The system can also be configured so that using a blank *QMgrName* connects to the queue-sharing group instead of to the default queue manager.

If *QMgrName* specifies the name of the queue-sharing group, but there is also a queue manager with that name on the system, connection is made to the latter in preference to the former. Only if that connection fails is connection to one of the queue managers in the queue-sharing group attempted.

If the connection is successful, you can use the handle returned by the MQCONN or MQCONNX call to access *all* the resources (both shared and nonshared) that belong to the queue manager to which connection has been made. Access to these resources is subject to the typical authorization controls.

If the application issues two MQCONN or MQCONNX calls to establish concurrent connections, and one or both calls specifies the name of the queue-sharing group, the second call returns completion code MQCC_WARNING and reason code MQRC_ALREADY_CONNECTED when it connects to the same queue manager as the first call.

Queue-sharing groups are supported only on z/OS. Connection to a queue-sharing group is supported only in the batch, RRS batch, CICS, and TSO environments. For CICS, you can use only the queue-sharing group to which the CICS system is connected. You must still specify the **QMgrName** parameter, but its value is ignored; blank characters are a suitable option.

IBM MQ MQI client applications: For IBM MQ MQI client applications, a connection is attempted for each client-connection channel definition with the specified queue manager name, until one is successful. The queue manager, however, must have the same name as the specified name. If an all-blank name is specified, each client-connection channel with an all-blank queue manager name is tried until one is successful; in this case there is no check against the actual name of the queue manager.

IBM MQ client applications are not supported in z/OS, but z/OS can act as an IBM MQ server, to which IBM MQ client applications can connect.

IBM MQ MQI client queue manager groups: If the specified name starts with an asterisk (*), the queue manager to which connection is made might have a different name from that specified by the application. The specified name (without the asterisk) defines a *group* of queue managers that are eligible for connection. The implementation selects one from the group by trying each one in turn until one is found to which a connection can be made. The order in which connections are attempted is influenced by the client channel weight and connection affinity values of the candidate channels. If none of the queue managers in the group is available for connection, the call fails. Each queue manager is tried once only. If an asterisk alone is specified for the name, an implementation-defined default queue manager group is used.

Queue manager groups are supported only for applications running in an MQ-client environment; the call fails if a non-client application specifies a queue manager name beginning with an asterisk. A group is defined by providing several client connection channel definitions with the same queue manager name (the specified name without the asterisk), to communicate with each of the queue managers in the group. The default group is defined by providing one or more client connection channel definitions, each with a blank queue manager name (specifying an all-blank name therefore has the same effect as specifying a single asterisk for the name for a client application).

After connecting to one queue manager of a group, an application can specify blanks in the typical way in the queue manager name fields in the message and object descriptors to mean the name of the queue manager to which the application has connected (the *local queue manager*). If the application needs to know this name, use the MQINQ call to inquire the **QMgrName** queue manager attribute.

Prefixing an asterisk to the connection name implies that the application does not depend on connecting to a particular queue manager in the group. Suitable applications are:

- Applications that put messages but do not get messages.

- Applications that put request messages and then get the reply messages from a *temporary dynamic* queue.

Unsuitable applications are ones that need to get messages from a particular queue at a particular queue manager; such applications must not prefix the name with an asterisk.

If you specify an asterisk, the maximum length of the remainder of the name is 47 characters.

Queue manager groups are not supported on z/OS.

The length of this parameter is given by MQ_Q_MGR_NAME_LENGTH.

Hconn

Type: MQHCONN - output

This handle represents the connection to the queue manager. Specify it on all subsequent message queuing calls issued by the application. It ceases to be valid when the MQDISC call is issued, or when the unit of processing that defines the scope of the handle terminates.

IBM MQ now supplies the mqm library with client packages as well as server packages. This means that when an MQI call that is found in the mqm library is made, the connection type is checked to see if it is a client or server connection, and then the correct underlying call is made. Therefore an exit which is passed an *Hconn* can now be linked against the mqm library, but used on a client installation.

Handle scope: The scope of the handle returned depends on the call used to connect to the queue manager (MQCONN or MQCONNX). If the call used is MQCONNX, the scope of the handle also depends on the MQCNO_HANDLE_SHARE_* option specified in the *Options* field of the MQCNO structure.

- If the call is MQCONN, or the MQCNO_HANDLE_SHARE_NONE option is specified, the handle returned is a *nonshared* handle.

  The scope of a nonshared handle is the smallest unit of parallel processing supported by the platform on which the application is running (see Table 264 for details); the handle is not valid outside the unit of parallel processing from which the call was issued.
- If you specify the MQCNO_HANDLE_SHARE_BLOCK or MQCNO_HANDLE_SHARE_NO_BLOCK option, the handle returned is a *shared* handle.

  The scope of a shared handle is the process that owns the thread from which the call was issued; the handle can be used from any thread that belongs to that process. Not all platforms support threads.
- If the MQCONN or MQCONNX call fails with completion code equal to MQCC_FAILED, then the Hconn value is undefined.

Table 264. Scope of nonshared handles on various platforms

Platform	Scope of nonshared handle
z/OS	• CICS: the CICS task • IMS: the task, up to the next sync point (excluding subtasks of the task) • z/OS batch and TSO: the task (excluding subtasks of the task)
IBM i	Job
UNIX	Thread
16 bit Windows applications	Process
32 bit Windows applications	Thread

On z/OS for CICS applications the value returned is:

MQHC_DEF_HCONN
Default connection handle.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_ALREADY_CONNECTED
(2002, X'7D2') Application already connected.

MQRC_CLUSTER_EXIT_LOAD_ERROR
(2267, X'8DB') Unable to load cluster workload exit.

MQRC_SSL_ALREADY_INITIALIZED
(2391, X'957') SSL already initialized.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_CONN_LOAD_ERROR
(2129, X'851') Unable to load adapter connection module.

MQRC_ADAPTER_DEFS_ERROR
(2131, X'853') Adapter subsystem definition module not valid.

MQRC_ADAPTER_DEFS_LOAD_ERROR
(2132, X'854') Unable to load adapter subsystem definition module.

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ADAPTER_STORAGE_SHORTAGE
(2127, X'84F') Insufficient storage for adapter.

MQRC_ANOTHER_Q_MGR_CONNECTED
(2103, X'837') Another queue manager already connected.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_INIT_ERROR
(2375, X'947') API exit initialization failed.

MQRC_API_EXIT_TERM_ERROR
(2376, X'948') API exit termination failed.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CONN_ID_IN_USE
(2160, X'870') Connection identifier already in use.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_ERROR
(2273, X'8E1') Error processing MQCONN call.

MQRC_CONNECTION_NOT_AVAILABLE
(2568, X'A08') Occurs on an MQCONN or MQCONNX call when the queue manager is unable to provide a connection of the requested connection type on the current installation. A client connection cannot be made on a server only installation. A local connection cannot be made on a client only installation.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_CRYPTO_HARDWARE_ERROR
(2382, X'94E') Cryptographic hardware configuration error.

MQRC_DUPLICATE_RECOV_COORD
(2163, X'873') Recovery coordinator exists.

MQRC_ENVIRONMENT_ERROR
(2012, X'7DC') Call not valid in environment.

Additionally, on the MQCONNX call, passing the “MQCSP - Security parameters” on page 2292 control block from a CICS or IMS application.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOST_NOT_AVAILABLE
(2538, X'9EA') An MQCONN call was issued from a client to connect to a queue manager but the attempt to allocate a conversation to the remote system failed.

MQRC_INSTALLATION_MISMATCH
(2583, X'A17') Mismatch between queue manager installation and selected library.

MQRC_KEY_REPOSITORY_ERROR
(2381, X'94D') Key repository not valid.

MQRC_MAX_CONNS_LIMIT_REACHED
(2025, X'7E9') Maximum number of connections reached.

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_OPEN_FAILED
(2137, X'859') Object not opened successfully.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SECURITY_ERROR
(2063, X'80F') Security error occurred.

MQRC_SSL_INITIALIZATION_ERROR
(2393, X'959') SSL initialization error.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

Usage notes

1. The queue manager to which connection is made using the MQCONN call is called the *local queue manager*.
2. Queues that are owned by the local queue manager appear to the application as local queues. It is possible to put messages on and get messages from these queues.

   Shared queues that are owned by the queue-sharing group to which the local queue manager belongs appear to the application as local queues. It is possible to put messages on and get messages from these queues.

   Queues that are owned by remote queue managers appear as remote queues. It is possible to put messages on these queues, but not to get messages from these queues.
3. If the queue manager fails while an application is running, the application must issue the MQCONN call again to obtain a new connection handle to use on subsequent IBM MQ calls. The application can issue the MQCONN call periodically until the call succeeds.

   If an application is not sure whether it is connected to the queue manager, the application can safely issue an MQCONN call to obtain a connection handle. If the application is already connected, the handle returned is the same as that returned by the previous MQCONN call, but with completion code MQCC_WARNING and reason code MQRC_ALREADY_CONNECTED.
4. When the application has finished using IBM MQ calls, the application must use the MQDISC call to disconnect from the queue manager.
5. If the MQCONN call fails with completion code equal to MQCC_FAILED, then the Hconn value is undefined.
6. On z/OS:
   - Batch, TSO, and IMS applications must issue the MQCONN call to use the other IBM MQ calls. These applications can connect to more than one queue manager concurrently.

     If the queue manager fails, the application must issue the call again after the queue manager has restarted to obtain a new connection handle.

     Although IMS applications can issue the MQCONN call repeatedly, even when already connected, this is not recommended for online message processing programs (MPPs).
   - CICS applications do not have to issue the MQCONN call to use the other IBM MQ calls, but can do so if they want; both the MQCONN call and the MQDISC call are accepted. However, it is not possible to connect to more than one queue manager concurrently.

If the queue manager fails, these applications are automatically reconnected when the queue manager restarts, and so do not need to issue the MQCONN call.

7. On z/OS, to define the available queue managers:
   - For batch applications, system programmers can use the CSQBDEF macro to create a module (CSQBDEFV) that defines the default queue manager name, or queue-sharing group name.
   - For IMS applications, system programmers can use the CSQQDEFX macro to create a module (CSQQDEFV) that defines the names of the available queue managers and specifies the default queue manager.

     In addition, each queue manager must be defined to the IMS control region and to each dependent region accessing that queue manager. To do this, you must create a subsystem member in the IMS.PROCLIB library and identify the subsystem member to the applicable IMS regions. If an application attempts to connect to a queue manager that is not defined in the subsystem member for its IMS region, the application abends.

   z/OS For more information about using these macros, see Macros intended for customer use.
8. On IBM i, programs that end abnormally are not automatically disconnected from the queue manager. Write applications to allow for the possibility of the MQCONN or MQCONNX call returning completion code MQCC_WARNING and reason code MQRC_ALREADY_CONNECTED. Use the connection handle returned in this situation as normal.

C invocation

```
MQCONN (QMgrName, &Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;  /* Name of queue manager */
MQHCONN   Hconn;     /* Connection handle */
MQLONG    CompCode;  /* Completion code */
MQLONG    Reason;    /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCONN' USING QMGRNAME, HCONN, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Name of queue manager
 01  QMGRNAME  PIC X(48).
**   Connection handle
 01  HCONN     PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE  PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON    PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCONN (QMgrName, Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl QMgrName  char(48);       /* Name of queue manager */
dcl Hconn     fixed bin(31);  /* Connection handle */
dcl CompCode  fixed bin(31);  /* Completion code */
dcl Reason    fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQCONN,(QMGRNAME,HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

```
QMGRNAME  DS  CL48  Name of queue manager
HCONN     DS  F     Connection handle
COMPCODE  DS  F     Completion code
REASON    DS  F     Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQCONN QMgrName, Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim QMgrName As String*48 'Name of queue manager'
Dim Hconn    As Long      'Connection handle'
Dim CompCode As Long      'Completion code'
Dim Reason   As Long      'Reason code qualifying CompCode'
```

MQCONNX - Connect queue manager (extended):

The MQCONNX call connects an application program to a queue manager. It provides a queue manager connection handle, which is used by the application on subsequent IBM MQ calls.

The MQCONNX call is like the MQCONN call, except that MQCONNX allows options to be specified to control the way that the call works.

- This call is supported on all IBM MQ systems, and IBM MQ clients connected to these systems.

A client connection cannot be made on a server only installation, and a local connection cannot be made on a client only installation.

Syntax

MQCONNX (*QMgrName*, *ConnectOpts*, *Hconn*, *CompCode*, *Reason*)

Parameters

QMgrName

Type: MQCHAR48 - input

See the **QMgrName** parameter described in "MQCONN - Connect queue manager" on page 2656 for details.

ConnectOpts

Type: MQCNO - input/output

See "MQCNO - Connect options" on page 2276 for details.

Hconn

Type: MQHCONN - output

This handle represents the connection to the queue manager. Specify it on all subsequent message queuing calls issued by the application. It ceases to be valid when the MQDISC call is issued, or when the unit of processing that defines the scope of the handle terminates.

IBM MQ now supplies the mqm library with client packages as well as server packages. This means that when an MQI call that is found in the mqm library is made, the connection type is checked to see if it is a client or server connection, and then the correct underlying call is made. Therefore an exit which is passed an *Hconn* can now be linked against the mqm library, but used on a client installation.

Handle scope: The scope of the handle returned depends on the call used to connect to the queue manager (MQCONN or MQCONNX). If the call used is MQCONNX, the scope of the handle also depends on the MQCNO_HANDLE_SHARE_* option specified in the *Options* field of the MQCNO structure.

- If the call is MQCONN, or the MQCNO_HANDLE_SHARE_NONE option is specified, the handle returned is a *nonshared* handle.

  The scope of a nonshared handle is the smallest unit of parallel processing supported by the platform on which the application is running (see Table 265 for details); the handle is not valid outside the unit of parallel processing from which the call was issued.
- If you specify the MQCNO_HANDLE_SHARE_BLOCK or MQCNO_HANDLE_SHARE_NO_BLOCK option, the handle returned is a *shared* handle.

  The scope of a shared handle is the process that owns the thread from which the call was issued; the handle can be used from any thread that belongs to that process. Not all platforms support threads.
- If the MQCONN or MQCONNX call fails with completion code equal to MQCC_FAILED, then the Hconn value is undefined.

Table 265. Scope of nonshared handles on various platforms

Platform	Scope of nonshared handle
z/OS	• CICS: the CICS task • IMS: the task, up to the next sync point (excluding subtasks of the task) • z/OS batch and TSO: the task (excluding subtasks of the task)
IBM i	Job
UNIX	Thread
16 bit Windows applications	Process
32 bit Windows applications	Thread

On z/OS for CICS applications the value returned is:

MQHC_DEF_HCONN
Default connection handle.

CompCode
Type: MQLONG - output

See the **CompCode** parameter described in "MQCONN - Connect queue manager" on page 2656 for details.

Reason
Type: MQLONG - output

The following codes can be returned by the MQCONN and MQCONNX calls. For a list of additional codes that can be returned by the MQCONNX call, see the following codes.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_ALREADY_CONNECTED
(2002, X'7D2') Application already connected.

MQRC_CLUSTER_EXIT_LOAD_ERROR
(2267, X'8DB') Unable to load cluster workload exit.

MQRC_SSL_ALREADY_INITIALIZED
(2391, X'957') SSL already initialized.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_CONN_LOAD_ERROR
(2129, X'851') Unable to load adapter connection module.

MQRC_ADAPTER_DEFS_ERROR
(2131, X'853') Adapter subsystem definition module not valid.

MQRC_ADAPTER_DEFS_LOAD_ERROR
(2132, X'854') Unable to load adapter subsystem definition module.

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ADAPTER_STORAGE_SHORTAGE
(2127, X'84F') Insufficient storage for adapter.

MQRC_ANOTHER_Q_MGR_CONNECTED
(2103, X'837') Another queue manager already connected.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_INIT_ERROR
(2375, X'947') API exit initialization failed.

MQRC_API_EXIT_TERM_ERROR
(2376, X'948') API exit termination failed.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CONN_ID_IN_USE
(2160, X'870') Connection identifier already in use.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_ERROR
(2273, X'8E1') Error processing MQCONN call.

MQRC_CONNECTION_NOT_AVAILABLE
(2568, X'A08') Occurs on an MQCONN or MQCONNX call when the queue manager is unable to provide a connection of the requested connection type on the current installation. A client connection cannot be made on a server only installation. A local connection cannot be made on a client only installation.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_CRYPTO_HARDWARE_ERROR
(2382, X'94E') Cryptographic hardware configuration error.

MQRC_DUPLICATE_RECOV_COORD
(2163, X'873') Recovery coordinator exists.

MQRC_ENVIRONMENT_ERROR
(2012, X'7DC') Call not valid in environment.

Additionally, on the MQCONNX call, passing the "MQCSP - Security parameters" on page 2292 control block from a CICS or IMS application.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOST_NOT_AVAILABLE
(2538, X'9EA') An MQCONN call was issued from a client to connect to a queue manager but the attempt to allocate a conversation to the remote system failed.

MQRC_INSTALLATION_MISMATCH
(2583, X'A17') Mismatch between queue manager installation and selected library.

MQRC_KEY_REPOSITORY_ERROR
(2381, X'94D') Key repository not valid.

MQRC_MAX_CONNS_LIMIT_REACHED
(2025, X'7E9') Maximum number of connections reached.

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_OPEN_FAILED
(2137, X'859') Object not opened successfully.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SECURITY_ERROR
(2063, X'80F') Security error occurred.

MQRC_SSL_INITIALIZATION_ERROR
(2393, X'959') SSL initialization error.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

The following additional reason codes can be returned by the MQCONNX call:

If *CompCode* is MQCC_FAILED:

MQRC_AIR_ERROR
(2385, X'951') Authentication information record not valid.

MQRC_AUTH_INFO_CONN_NAME_ERROR
(2387, X'953') Authentication information connection name not valid.

MQRC_AUTH_INFO_REC_COUNT_ERROR
(2383, X'94F') Authentication information record count not valid.

MQRC_AUTH_INFO_REC_ERROR
(2384, X'950') Authentication information record fields not valid.

MQRC_AUTH_INFO_TYPE_ERROR
(2386, X'952') Authentication information type not valid.

MQRC_CD_ERROR
(2277, X'8E5') Channel definition not valid.

MQRC_CLIENT_CONN_ERROR
(2278, X'8E6') Client connection fields not valid.

MQRC_CNO_ERROR
(2139, X'85B') Connect-options structure not valid.

MQRC_CONN_TAG_IN_USE
(2271, X'8DF') Connection tag in use.

MQRC_CONN_TAG_NOT_USABLE
(2350, X'92E') Connection tag not usable.

MQRC_LDAP_PASSWORD_ERROR
(2390, X'956') LDAP password not valid.

MQRC_LDAP_USER_NAME_ERROR
(2388, X'954') LDAP user name fields not valid.

MQRC_LDAP_USER_NAME_LENGTH_ERR
(2389, X'955') LDAP user name length not valid.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_SCO_ERROR
(2380, X'94C') SSL configuration options structure not valid.

MQRC_SSL_CONFIG_ERROR
(2392, X'958') SSL configuration error.

For detailed information about these codes, see Reason codes.

Usage notes

For the Visual Basic programming language, the following point applies:

- The **ConnectOpts** parameter is declared as being of type MQCNO. If the application is running as an IBM MQ MQI client, and you want to specify the parameters of the client-connection channel, declare the **ConnectOpts** parameter as being of type Any, so that the application can specify an MQCNOCD structure on the call in place of an MQCNO structure. However, this means that the **ConnectOpts** parameter cannot be checked to ensure that it is the correct data type.

C invocation

```
MQCONNX (QMgrName, &ConnectOpts, &Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;     /* Name of queue manager */
MQCNO     ConnectOpts;  /* Options that control the action of MQCONNX */
MQHCONN   Hconn;        /* Connection handle */
MQLONG    CompCode;     /* Completion code */
MQLONG    Reason;       /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCONNX' USING QMGRNAME, CONNECTOPTS, HCONN, COMPCODE,
REASON.
```

Declare the parameters as follows:

```
**   Name of queue manager
 01  QMGRNAME     PIC X(48).
**   Options that control the action of MQCONNX
 01  CONNECTOPTS.
     COPY CMQCNOV.
**   Connection handle
 01  HCONN        PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE     PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON       PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCONNX (QMgrName, ConnectOpts, Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl QMgrName     char(48);       /* Name of queue manager */
dcl ConnectOpts  like MQCNO;     /* Options that control the action of
                                    MQCONNX */
dcl Hconn        fixed bin(31);  /* Connection handle */
dcl CompCode     fixed bin(31);  /* Completion code */
dcl Reason       fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQCONNX,(QMGRNAME,CONNECTOPTS,HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

```
QMGRNAME     DS       CL48  Name of queue manager
CONNECTOPTS  CMQCNOA  ,     Options that control the action of MQCONNX
HCONN        DS       F     Connection handle
COMPCODE     DS       F     Completion code
REASON       DS       F     Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQCONNX QMgrName, ConnectOpts, Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim QMgrName    As String*48 'Name of queue manager'
Dim ConnectOpts As MQCNO     'Options that control the action of'
                             'MQCONNX'
Dim Hconn       As Long      'Connection handle'
Dim CompCode    As Long      'Completion code'
Dim Reason      As Long      'Reason code qualifying CompCode'
```

MQCRTMH - Create message handle:

The MQCRTMH call returns a message handle.

An application can use the MQCRTMH call on subsequent message queuing calls:

- Use the MQSETMP call to set a property of the message handle.
- Use the MQINQMP call to inquire on the value of a property of the message handle.
- Use the MQDLTMP call to delete a property of the message handle.

The message handle can be used on the MQPUT and MQPUT1 calls to associate the properties of the message handle with those of the message being put. Similarly by specifying a message handle on the MQGET call, the properties of the message being retrieved can be accessed using the message handle when the MQGET call completes.

Use MQDLTMH to delete the message handle.

Syntax

MQCRTMH (*Hconn*, *CrtMsgHOpts*, *Hmsg*, *CompCode*, *Reason*)

Parameters

`Hconn`

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *`Hconn`* was returned by a previous MQCONN or MQCONNX call. If the connection to the queue manager ceases to be valid and no IBM MQ call is operating on the message handle, MQDLTMH is implicitly called to delete the message.

Alternatively, you can specify the following value:

MQHC_UNASSOCIATED_HCONN
The connection handle does not represent a connection to any particular queue manager.

When this value is used, the message handle must be deleted with an explicit call to MQDLTMH in order to release any storage allocated to it; IBM MQ never implicitly deletes the message handle.

There must be at least one valid connection to a queue manager established on the thread creating the message handle, otherwise the call fails with MQRC_HCONN_ERROR.

In an environment with multiple installations on a single system, the MQHC_UNASSOCIATED_HCONN value is limited to use with the first installation loaded into the process. The reason code MQRC_HMSG_NOT_AVAILABLE is returned if the message handle is supplied to a different installation.

On z/OS for CICS applications the MQCONN call can be omitted, and you can specify the following value for *`Hconn`*:

MQHC_DEF_CONN
Default connection handle

`CrtMsgHOpts`

Type: MQCMHO - input

The options that control the action of MQCRTMH. See MQCMHO for details.

`Hmsg`

Type: MQHMSG - output

On output a message handle is returned that can be used to set, inquire, and delete properties of the message handle. Initially the message handle contains no properties.

A message handle also has an associated message descriptor. Initially this contains the default values. The values of the associated message descriptor fields can be set and inquired using the MQSETMP and MQINQMP calls. The MQDLTMP call resets a field of the message descriptor back to its default value.

If the *Hconn* parameter is specified as the value MQHC_UNASSOCIATED_HCONN then the returned message handle can be used on MQGET, MQPUT, or MQPUT1 calls with any connection within the unit of processing, but can only be in use by one IBM MQ call at a time. If the handle is in use when a second IBM MQ call attempts to use the same message handle, the second IBM MQ call fails with reason code MQRC_MSG_HANDLE_IN_USE.

If the *Hconn* parameter is not MQHC_UNASSOCIATED_HCONN then the returned message handle can only be used on the specified connection.

The same *Hconn* parameter value must be used on the subsequent MQI calls where this message handle is used:

- MQDLTMH
- MQSETMP
- MQINQMP
- MQDLTMP
- MQMHBUF
- MQBUFMH

The returned message handle ceases to be valid when the MQDLTMH call is issued for the message handle, or when the unit of processing that defines the scope of the handle terminates. MQDLTMH is called implicitly if a specific connection is supplied when the message handle is created and the connection to the queue manager ceases to be valid, for example, if MQDBC is called.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'089C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'08AB') MQI call entered before previous call completed.

MQRC_CMHO_ERROR
(2461, X'099D') Create message handle options structure not valid.

MQRC_CONNECTION_BROKEN
(2273, X'7D9') Connection to queue manager lost.

MQRC_HANDLE_NOT_AVAILABLE
(2017, X'07E1') No more handles available.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle pointer not valid.

MQRC_OPTIONS_ERROR
(2046, X'07FE') Options not valid or not consistent.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

C

```
MQCRTMH (Hconn, &CrtMsgHOpts, &Hmsg, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;       /* Connection handle */
MQCMHO   CrtMsgHOpts; /* Options that control the action of MQCRTMH */
MQHMSG   Hmsg;        /* Message handle */
MQLONG   CompCode;    /* Completion code */
MQLONG   Reason;      /* Reason code qualifying CompCode */
```

COBOL

```
CALL 'MQCRTMH' USING HCONN, CRTMSGOPTS, HMSG, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN     PIC S9(9) BINARY.
**   Options that control the action of MQCRTMH
 01  CRTMSGHOPTS.
     COPY CMQCMHOV.
**   Message handle
 01  HMSG    PIC S9(18) BINARY.
**   Completion code
 01  COMPCODE  PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON    PIC S9(9) BINARY.
```

PL/I

```
call MQCRTMH (Hconn, CrtMsgHOpts, Hmsg, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn       fixed bin(31);  /* Connection handle */
dcl CrtMsgHOpts like MQCMHO;    /* Options that control the action of MQCRTMH */
dcl Hmsg        fixed bin(63);  /* Message handle */
dcl CompCode    fixed bin(31);  /* Completion code */
dcl Reason      fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler

```
CALL MQCRTMH,(HCONN,CRTMSGHOPTS,HMSG,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN          DS        F  Connection handle
CRTMSGHOPTS    CMQCMHOA  ,  Options that control the action of MQCRTMH
HMSG           DS        D  Message handle
COMPCODE       DS        F  Completion code
REASON         DS        F  Reason code qualifying COMPCODE
```

MQCTL - Control callbacks:

The MQCTL call performs controlling actions on callbacks and the object handles opened for a connection.

Syntax

MQCTL (*Hconn*, *Operation*, *ControlOpts*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications the MQCONN call can be omitted, and you can specify the following special value for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Operation

Type: MQLONG - input

The operation being processed on the callback defined for the specified object handle. You must specify one, and one only, of the following options:

MQOP_START

Start the consuming of messages for all defined message consumer functions for the specified connection handle.

Callbacks run on a thread started by the system, which is different from any of the application threads.

This operation gives control of the provided connection handle to system. The only MQI calls which can be issued by a thread other than the consumer thread are:

- MQCTL with Operation MQOP_STOP
- MQCTL with Operation MQOP_SUSPEND
- MQDISC - Performs MQCTL with Operation MQOP_STOP before disconnection the HConn.

MQRC_HCONN_ASYNC_ACTIVE is returned if an IBM MQ API call is issued while the connection handle is started, and the call does not originate from a message consumer function.

If a message consumer stops the connection during the MQCBCT_START_CALL then the MQCTL call returns with a failure reason code of MQRC_CONNECTION_STOPPED.

This can be issued in a consumer function. For the same connection as the callback routine, its only purpose is to cancel a previously issued MQOP_STOP operation.

This option is not supported in the following environments: CICS on z/OS or if the application is bound with a nonthreaded IBM MQ library.

MQOP_START_WAIT
Start the consuming of messages for all defined message consumer functions for the specified connection handle.

Message consumers run on the same thread and control is not returned to the caller of MQCTL until:

- Released by the use of the MQCTL MQOP_STOP or MQOP_SUSPEND operations, or
- All consumer routines have been deregistered or suspended.

If all consumers are deregistered or suspended, an implicit MQOP_STOP operation is issued.

This option cannot be used from within a callback routine, either for the current connection handle or any other connection handle. If the call is attempted it returns with MQRC_ENVIRONMENT_ERROR.

If, at any time during an MQOP_START_WAIT operation there are no registered, non-suspended consumers the call fails with a reason code of MQRC_NO_CALLBACKS_ACTIVE.

If, during an MQOP_START_WAIT operation, the connection is suspended, the MQCTL call returns a warning reason code of MQRC_CONNECTION_SUSPENDED; the connection remains 'started'.

The application can choose to issue MQOP_STOP or MQOP_RESUME. In this instance, the MQOP_RESUME operation blocks.

This option is not supported in a single threaded client.

MQOP_STOP
Stop the consuming of messages, and wait for all consumers to complete their operations before this option completes. This operation releases the connection handle.

If issued from within a callback routine, this option does not take effect until the routine exits. No more message consumer routines are called after the consumer routines for messages already read have completed, and after stop calls (if requested) to callback routines have been made.

If issued outside a callback routine, control does not return to the caller until the consumer routines for messages already read have completed, and after stop calls (if requested) to callbacks have been made. The callbacks themselves, however, remain registered.

This function has no effect on read ahead messages. You must ensure that consumers run MQCLOSE(MQCO_QUIESCE), from within the callback function, to determine whether there are any further messages available to be delivered.

MQOP_SUSPEND
Pause the consuming of messages. This operation releases the connection handle.

This does not have any effect on the reading ahead of messages for the application. If you intend to stop consuming messages for a long time, consider closing the queue and reopening it when consumption continues.

If issued from within a callback routine, it does not take effect until the routine exits. No more message consumer routines will be called after the current routine exits.

If issued outside a callback, control does not return to the caller until the current consumer routine has completed and no more are called.

MQOP_RESUME
Resume the consuming of messages.

This option is normally issued from the main application thread, but it can also be used from within a callback routine to cancel an earlier suspension request issued in the same routine.

If the MQOP_RESUME is used to resume an MQOP_START_WAIT then the operation blocks.

ControlOpts
Type: MQCTLO - input

Options that control the action of MQCTL

See MQCTLO for details of the structure.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_CONV_LOAD_ERROR
(2133, X'855') Unable to load data conversion services modules.

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALLBACK_LINK_ERROR
(2487, X'9B7') Unable to call the callback routine

MQRC_CALLBACK_NOT_ REGISTERED
(2448, X'990') Unable to Deregister, Suspend, or Resume because there is no registered callback

MQRC_CALLBACK_ROUTINE_ERROR
(2486, X'9B6') Either, both CallbackFunction and CallbackName have been specified on an MQOP_REGISTER call.

Or either CallbackFunction or CallbackName have been specified but does not match the currently registered callback function.

MQRC_CALLBACK_TYPE_ERROR
(2483, X'9B3') Incorrect CallBackType field.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CBD_ERROR
(2444, X'98C') Option block is incorrect.

MQRC_CBD_OPTIONS_ERROR
(2484, X'9B4') Incorrect MQCBD options field.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_CORREL_ID_ERROR
(2207, X'89F') Correlation-identifier error.

MQRC_FUNCTION_NOT_SUPPORTED
(2298, X'8FA') The function requested is not available in the current environment.

MQRC_GET_INHIBITED
(2016, X'7E0') Gets inhibited for the queue.

MQRC_GLOBAL_UOW_CONFLICT
(2351, X'92F') Global units of work conflict.

MQRC_GMO_ERROR
(2186, X'88A') Get-message options structure not valid.

MQRC_HANDLE_IN_USE_FOR_UOW
(2353, X'931') Handle in use for global unit of work.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_INCONSISTENT_BROWSE
(2259, X'8D3') Inconsistent browse specification.

MQRC_INCONSISTENT_UOW
(2245, X'8C5') Inconsistent unit-of-work specification.

MQRC_INVALID_MSG_UNDER_CURSOR
(2246, X'8C6') Message under cursor not valid for retrieval.

MQRC_LOCAL_UOW_CONFLICT
(2352, X'930') Global unit of work conflicts with local unit of work.

MQRC_MATCH_OPTIONS_ERROR
(2247, X'8C7') Match options not valid.

MQRC_MAX_MSG_LENGTH_ERROR
(2485, X'9B5') Incorrect MaxMsgLength field

MQRC_MD_ERROR
(2026, X'7EA') Message descriptor not valid.

MQRC_MODULE_ENTRY_NOT_FOUND
(2497, X'9C1')The specified function entry point could not be found in the module.

MQRC_MODULE_INVALID
(2496, X'9C0') Module is found but is of the wrong type (32 bit/64 bit) or is not a valid dll.

MQRC_MODULE_NOT_FOUND
(2495, X'9BF') Module not found in the search path or not authorized to load.

MQRC_MSG_ID_ERROR
(2206, X'89E') Message-identifier error.

MQRC_MSG_SEQ_NUMBER_ERROR
(2250, X'8CA') Message sequence number not valid.

MQRC_MSG_TOKEN_ERROR
(2331, X'91B') Use of message token not valid.

MQRC_NOT_OPEN_FOR_BROWSE
(2036, X'7F4') Queue not open for browse.

MQRC_NOT_OPEN_FOR_INPUT
(2037, X'7F5') Queue not open for input.

MQRC_OBJECT_CHANGED
(2041, X'7F9') Object definition changed since opened.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OPERATION_ERROR
(2488, X'9B8') Incorrect Operation code on API Call

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_INDEX_TYPE_ERROR
(2394, X'95A') Queue has wrong index type.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SIGNAL_OUTSTANDING
(2069, X'815') Signal outstanding for this handle.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_SYNCPOINT_NOT_AVAILABLE
(2072, X'818') Syncpoint support not available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

MQRC_UOW_ENLISTMENT_ERROR
(2354, X'932') Enlistment in global unit of work failed.

MQRC_UOW_MIX_NOT_SUPPORTED
(2355, X'933') Mixture of unit-of-work calls not supported.

MQRC_UOW_NOT_AVAILABLE
(2255, X'8CF') Unit of work not available for the queue manager to use.

MQRC_WAIT_INTERVAL_ERROR
(2090, X'82A') Wait interval in MQGMO not valid.

MQRC_WRONG_GMO_VERSION
(2256, X'8D0') Wrong version of MQGMO supplied.

MQRC_WRONG_MD_VERSION
(2257, X'8D1') Wrong version of MQMD supplied.

For detailed information about these codes, see Reason codes.

Usage notes

1. Callback routines must check the responses from all services they invoke, and if the routine detects a condition that cannot be resolved, it must issue an MQCB MQOP_DEREGISTER command to prevent repeated calls to the callback routine.
2. If you are using asynchronous consume in an application where an XA Transaction Manager is managing global transactions, including updates to IBM MQ, you need to consider the following additional points:
   a. It is not valid to call MQCTL(MQOP_START) for an **HConn**, after it has been created, after calling **xa_open**.

      The reason is, that the **HConn** has become attached to an XA context, and so cannot then be accessed on the separate thread, or threads, in use by the asynchronous consume mechanism.
   b. If you call MQCTL(MQOP_START) in that scenario the call fails with reason code MQRC_ASYNC_XA_CONFLICT (2350).
   c. It is valid to call MQCTL(MQOP_START_WAIT) for an **HConn**, after it has been created, after calling **xa_open**.

      The reason is, that this method of starting the asynchronous consume mechanism causes all further callbacks for the **HConn** to run on the thread where the MQCTL call is made. Therefore, the link between the **HConn** and the thread is not lost.
3. z/OS On z/OS, when Operation is MQOP_START:

- Programs which use asynchronous callback routines must be authorized to use z/OS UNIX System Services (USS).
- Language Environment (LE) programs which use asynchronous callback routines must use the LE runtime option POSIX(ON).
- Non-LE programs which use asynchronous callback routines must not use the USS pthread_create interface (callable service BPX1PTC).

4. z/OS MQCTL is not supported within the IMS adapter.

Note: In CICS, MQOP_START is not supported. Instead, use the MQOP_START_WAIT function call.

C invocation

```
MQCTL (Hconn, Operation, &ControlOpts, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHCONN  Hconn;          /* Connection handle */
MQLONG   Operation;      /* Operation being processed */
MQCTLO   ControlOpts     /* Options that control the action of MQCTL */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCTL' USING HCONN, OPERATION, CTLOPTS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN    PIC S9(9) BINARY.
**   Operation
 01  OPERATION PIC S9(9) BINARY.
**   Control Options
 01  CTLOPTS.
     COPY CMQCTLOV.
**   Completion code
 01  COMPCODE PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON   PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCTL(Hconn, Operation, CtlOpts, CompCode, Reason)
```

Declare the parameters as follows:

```
dcl Hconn        fixed bin(31); /* Connection handle */
dcl Operation    fixed bin(31); /* Operation */
dcl CtlOpts like MQCTLO;        /* Options that control the action of MQCTL */
dcl CompCode     fixed bin(31); /* Completion code */
dcl Reason       fixed bin(31); /* Reason code qualifying CompCode */
```

MQDISC - Disconnect queue manager:

The MQDISC call breaks the connection between the queue manager and the application program, and is the inverse of the MQCONN or MQCONNX call.

- On z/OS, all applications that use asynchronous message consumption, event handling or callback, the main control thread must issue an MQDISC call before ending. See Asynchronous consumption of IBM MQ messages for more details.
- On z/OS, CICS applications do not need to issue this call to disconnect from the queue manager, but might need to issue it to end the use of a connection tag.

Syntax

MQDISC (*Hconn*, *CompCode*, *Reason*)

Parameters

Hconn
Type: MQHCONN - input/output

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications you can omit the MQCONN call, and specify the following value for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

On successful completion of the call, the queue manager sets *Hconn* to a value that is not a valid handle for the environment. This value is:

MQHC_UNUSABLE_HCONN
Unusable connection handle.

On z/OS, *Hconn* is set to a value that is undefined.

CompCode
Type: MQLONG - output

The completion code; it is one of the following codes:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_BACKED_OUT
(2003, X'7D3') Unit of work backed out.

MQRC_CONN_TAG_NOT_RELEASED
(2344, X'928') Connection tag not released.

MQRC_OUTCOME_PENDING
(2124, X'84C') Result of commit operation is pending.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_DISC_LOAD_ERROR
(2138, X'85A') Unable to load adapter disconnection module.

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_INIT_ERROR
(2375, X'947') API exit initialization failed.

MQRC_API_EXIT_TERM_ERROR
(2376, X'948') API exit termination failed.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_OUTCOME_MIXED
(2123, X'84B') Result of commit or back-out operation is mixed.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

Usage notes

1. If an MQDISC call is issued when the connection still has objects open under that connection, the queue manager closes those objects, with the close options set to MQCO_NONE.
2. If the application ends with uncommitted changes in a unit of work, the disposition of those changes depends on how the application ends:
   a. If the application issues the MQDISC call before ending:
      - For a queue manager-coordinated unit of work, the queue manager issues the MQCMIT call on behalf of the application. The unit of work is committed if possible, and backed out if not.
      - For an externally coordinated unit of work, there is no change in the status of the unit of work; however, the queue manager typically indicates that the unit of work must be committed when asked by the unit-of-work coordinator.

        On z/OS, CICS, IMS (other than batch DL/1 programs), and RRS applications are like this.

   b. If the application ends normally but without issuing the MQDISC call, the action taken depends on the environment:
      - On z/OS, except for MQ Java or MQ JMS applications, the actions described in note 2a occur.
      - In all other cases, the actions described in note 2c occur.

      Because of the differences between environments, ensure that applications that you want to port either commit or back out the unit of work before they end.
   c. If the application ends *abnormally* without issuing the MQDISC call, the unit of work is backed out.
3. On z/OS, the following points apply:
   - CICS applications do not have to issue the MQDISC call to disconnect from the queue manager, because the CICS system itself connects to the queue manager, and the MQDISC call has no effect on this connection.
   - CICS, IMS (other than batch DL/1 programs), and RRS applications use units of work that are coordinated by an external unit-of-work coordinator. As a result, the MQDISC call does not affect the status of the unit of work (if any) that exists when the call is issued.

     However the MQDISC call *does* indicate the end of use of the connection tag *ConnTag* that was associated with the connection by an earlier MQCONNX call issued by the application. If there is an active unit of work that references the connection tag when the MQDISC call is issued, the call completes with completion code MQCC_WARNING and reason code MQRC_CONN_TAG_NOT_RELEASED. The connection tag does not become available for reuse until the external unit-of-work coordinator has resolved the unit of work.

Note: In CICS, MQOP_START is not supported. Instead, use the MQOP_START_WAIT function call.

C invocation

```
MQDISC (&Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;     /* Connection handle */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQDISC' USING HCONN, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN     PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE  PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON    PIC S9(9) BINARY.
```

PL/I invocation

```
call MQDISC (Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn     fixed bin(31);  /* Connection handle */
dcl CompCode  fixed bin(31);  /* Completion code */
dcl Reason    fixed bin(31);  /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQDISC,(HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN     DS  F  Connection handle
COMPCODE  DS  F  Completion code
REASON    DS  F  Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQDISC Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn    As Long 'Connection handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

MQDLTMH - Delete message handle:

The MQDLTMH call deletes a message handle and is the inverse of the MQCRTMH call.

Syntax

MQDLTMH (*Hconn*, *Hmsg*, *DltMsgHOpts*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager.

The value must match the connection handle that was used to create the message handle specified in the **Hmsg** parameter.

If the message handle was created using MQHC_UNASSOCIATED_HCONN then a valid connection must be established on the thread deleting the message handle, otherwise the call fails with MQRC_CONNECTION_BROKEN.

Hmsg

Type: MQHMSG - input/output

This is the message handle to be deleted. The value was returned by a previous MQCRTMH call.

On successful completion of the call, the handle is set to an invalid value for the environment. This value is:

MQHM_UNUSABLE_HMSG
Unusable message handle.

The message handle cannot be deleted if another IBM MQ call is in progress that was passed the same message handle.

DltMsgHOpts
Type: MQDMHO - input

See MQDMHO for details.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'089C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'08AB') MQI call entered before previous call completed.

MQRC_CONNECTION_BROKEN
(2009, X'07D9') Connection to queue manager lost.

MQRC_DMHO_ERROR
(2462, X'099E') Delete message handle options structure not valid.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle pointer not valid.

MQRC_MSG_HANDLE_IN_USE
(2499, X'09C3') Message handle already in use.

MQRC_OPTIONS_ERROR
(2046, X'07FE') Options not valid or not consistent.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

C invocation

```
MQDLTMH (Hconn, &Hmsg, &DltMsgHOpts, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;       /* Connection handle */
MQHMSG   Hmsg;        /* Message handle */
MQDMHO   DltMsgHOpts; /* Options that control the action of MQDLTMH */
MQLONG   CompCode;    /* Completion code */
MQLONG   Reason;      /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQDLTMH' USING HCONN, HMSG, DLTMSGOPTS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**  Connection handle
01   HCONN   PIC S9(9) BINARY.

**  Options that control the action of MQDLTMH
01   DLTMSGHOPTS.
COPY CMQDLMHOV.

**  Completion code
01  COMPCODE   PIC S9(9) BINARY.

** Reason code qualifying COMPCODE
01   REASON      PIC S9(9) BINARY.
```

PL/I invocation

```
call MQDLTMH (Hconn, Hmsg, DltMsgHOpts, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn          /* Connection handle */
dcl Hmsg           /* Message handle */
dcl DltMsgHOpts like MQDMHO;    /* Options that control the action of MQDLTMH */
dcl CompCode       /* Completion code */
dcl Reason         /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQDLTMH,(HCONN,HMSG,DLTMSGHOPTS,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN          DS        F  Connection handle
HMSG           DS        D  Message handle
DLTMSGHOPTS    CMQDMHOA  ,  Options that control the action of MQDLTMH
COMPCODE       DS        F  Completion code
REASON         DS        F  Reason code qualifying COMPCODE
```

MQDLTMP - Delete message property:

The MQDLTMP call deletes a property from a message handle and is the inverse of the MQSETMP call.

Syntax

MQDLTMP (*Hconn*, *Hmsg*, *DltPropOpts*, *Name*, *CompCode*, *Reason*)

Parameters

Hconn
Type: MQHCONN - input

This handle represents the connection to the queue manager. The value must match the connection handle that was used to create the message handle specified in the **Hmsg** parameter.

If the message handle was created using MQHC_UNASSOCIATED_HCONN then a valid connection must be established on the thread deleting the message handle otherwise the call fails with MQRC_CONNECTION_BROKEN.

Hmsg
Type: MQHMSG - input

This is the message handle containing the property to be deleted. The value was returned by a previous MQCRTMH call.

DltPropOpts
Type: MQDMPO - input

See the MQDMPO data type for details.

Name
Type: MQCHARV - input

The name of the property to delete. See Property names for further information about property names.

Wildcards are not allowed in the property name.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_PROPERTY_NOT_AVAILABLE
(2471, X'09A7') Property not available.

MQRC_RFH_FORMAT_ERROR
(2421, X'0975') An MQRFH2 folder containing properties could not be parsed.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'089C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'0852') Unable to load adapter service module.

MQRC_ASID_MISMATCH
(2157, X'086D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'08AB') MQI call entered before previous call completed.

MQRC_CONNECTION_BROKEN
(2009, X'07D9') Connection to queue manager lost.

MQRC_DMPO_ERROR
(2481, X'09B1') Delete message property options structure not valid.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle not valid.

MQRC_MSG_HANDLE_IN_USE
(2499, X'09C3') Message handle already in use.

MQRC_OPTIONS_ERROR
(2046, X'07FE') Options not valid or not consistent.

MQRC_PROPERTY_NAME_ERROR
(2442, X'098A') Invalid property name.

MQRC_SOURCE_CCSID_ERROR
(2111, X'083F') Property name coded character set identifier not valid.

MQRC_UNEXPECTED_ERROR
(2195, X'0893') Unexpected error occurred.

For detailed information about these codes, see:

- Reason codes for IBM MQ for z/OS
- API reason codes for other IBM MQ platforms

.

C invocation

```
MQDLTMP (Hconn, Hmsg, &DltPropOpts, &Name, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHCONN Hconn;       /* Connection handle */
MQHMSG  Hmsg;        /* Message handle */
MQDMPO  DltPropOpts; /* Options that control the action of MQDLTMP */
MQCHARV Name;        /* Property name */
MQLONG  CompCode;    /* Completion code */
MQLONG  Reason;      /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQDLTMP' USING HCONN, HMSG, DLTPROPOPTS, NAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN    PIC S9(9) BINARY.
**   Message handle
 01  HMSG   PIC S9(18) BINARY.
**   Options that control the action of MQDLTMP
 01  DLTPROPOPTS.
     COPY CMQDMPOV.
**   Property name
 01  NAME
     COPY CMQCHRVV.
**   Completion code
 01  COMPCODE PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON   PIC S9(9) BINARY.
```

PL/I invocation

```
call MQDLTMP (Hconn, Hmsg, DltPropOpts, Name, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn       fixed bin(31); /* Connection handle */
dcl Hmsg        fixed bin(63); /* Message handle */
dcl DltPropOpts like MQDMPO;   /* Options that control the action of MQDLTMP */
dcl Name        like MQCHARV;  /* Property name */
dcl CompCode    fixed bin(31); /* Completion code */
dcl Reason      fixed bin(31); /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQDLTMP,(HCONN,HMSG,DLTPROPOPTS,NAME,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN        DS         F     Connection handle
HMSG         DS         D     Message handle
DLTPROPOPTS  CMQDMPOA   ,     Options that control the action of MQDLTMP
NAME         CMQCHRVA   ,     Property name
COMPCODE     DS         F     Completion code
REASON       DS         F     Reason code qualifying COMPCODE
```

MQGET - Get message:

The MQGET call retrieves a message from a local queue that has been opened using the MQOPEN call.

Syntax

MQGET (*Hconn*, *Hobj*, *MsgDesc*, *GetMsgOpts*, *BufferLength*, *Buffer*, *DataLength*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hobj

Type: MQHOBJ - input

This handle represents the queue from which a message is to be retrieved. The value of *Hobj* was returned by a previous MQOPEN call. The queue must have been opened with one or more of the following options (see "MQOPEN - Open object" on page 2725 for details):

- MQOO_INPUT_SHARED
- MQOO_INPUT_EXCLUSIVE
- MQOO_INPUT_AS_Q_DEF
- MQOO_BROWSE

MsgDesc

Type: MQMD - input/output

This structure describes the attributes of the message required, and the attributes of the message retrieved. See "MQMD - Message descriptor" on page 2387 for details.

If *BufferLength* is less than the message length, *MsgDesc* is filled by the queue manager, whether MQGMO_ACCEPT_TRUNCATED_MSG is specified on the **GetMsgOpts** parameter (see MQGMO - Options field).

If the application provides a version-1 MQMD, the message returned has an MQMDE prefixed to the application message data, but only if one or more of the fields in the MQMDE has a nondefault value. If all the fields in the MQMDE have default values, the MQMDE is omitted. A format name of MQFMT_MD_EXTENSION in the *Format* field in MQMD indicates that an MQMDE is present.

The application does not need to provide an MQMD structure if a valid message handle is supplied in the *MsgHandle* field. If nothing is provided in this field, the descriptor of the message is taken from the descriptor associated with the message handles.

If the application provides a message handle rather than an MQMD structure, and specifies MQGMO_PROPERTIES_FORCE_MQRFH2, the call fails with reason code MQRC_MD_ERROR. The call also fails, with reason code MQRC_MD_ERROR, if the application does not provide an MQMD structure and specifies MQGMO_PROPERTIES_AS_Q_DEF, and the **PropertyControl** queue attribute is MQPROP_FORCE_MQRFH2.

If match options are specified and the message descriptor associated with the message handle is being used, the input fields used for matching come from the message handle.

GetMsgOpts

Type: MQGMO - input/output

See "MQGMO - Get-message options" on page 2330 for details.

BufferLength

Type: MQLONG - input

This is the length in bytes of the *Buffer* area. Specify zero for messages that have no data, or if the message is to be removed from the queue and the data discarded (you must specify MQGMO_ACCEPT_TRUNCATED_MSG in this case).

Note: The length of the longest message that it is possible to read from the queue is given by the **MaxMsgLength** queue attribute; see "Attributes for queues" on page 2833.

Buffer

Type: MQBYTExBufferLength - output

This is the area to contain the message data. Align the buffer on a boundary appropriate to the nature of the data in the message. 4 byte alignment is suitable for most messages (including messages containing IBM MQ header structures), but some messages might require more stringent alignment. For example, a message containing a 64 bit binary integer might require 8-byte alignment.

If *BufferLength* is less than the message length, as much of the message as possible is moved into **Buffer**. This happens whether MQGMO_ACCEPT_TRUNCATED_MSG is specified on the **GetMsgOpts** parameter (see MQGMO - Options field for more information).

The character set and encoding of the data in **Buffer** are given by the *CodedCharSetId* and *Encoding* fields returned in the **MsgDesc** parameter. If these values are different from the values required by the receiver, the receiver must convert the application message data to the character set and encoding required. The MQGMO_CONVERT option can be used (with a user-written exit if necessary) to convert the message data; see "MQGMO - Get-message options" on page 2330 for details of this option.

Note: All the other parameters on the MQGET call are in the character set and encoding of the local queue manager (given by the **CodedCharSetId** queue manager attribute and MQENC_NATIVE).

If the call fails, the contents of the buffer might still have changed.

In the C programming language, the parameter is declared as a pointer-to-void: the address of any type of data can be specified as the parameter.

If the **BufferLength** parameter is zero, *Buffer* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler can be null.

DataLength

Type: MQLONG - output

This is the length in bytes of the application data *in the message*. If this value is greater than *BufferLength*, only *BufferLength* bytes are returned in the **Buffer** parameter (that is, the message is truncated). If the value is zero, the message contains no application data.

If *BufferLength* is less than the message length, *DataLength* is still completed by the queue manager, whether MQGMO_ACCEPT_TRUNCATED_MSG is specified on the **GetMsgOpts** parameter (see MQGMO - Options field for more information). This allows the application to determine the size of the buffer required to accommodate the message data, and then reissue the call with a buffer of the appropriate size.

However, if the MQGMO_CONVERT option is specified, and the converted message data is too long to fit in *Buffer*, the value returned for *DataLength* is:

- The length of the *unconverted* data, for queue manager defined formats.

  In this case, if the nature of the data causes it to expand during conversion, the application must allocate a buffer bigger than the value returned by the queue manager for *DataLength*.
- The value returned by the data-conversion exit, for application-defined formats.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

The reason codes listed are the ones that the queue manager can return for the **Reason** parameter. If the application specifies the MQGMO_CONVERT option, and a user-written exit is invoked to convert some or all the message data, the exit decides what value is returned for the **Reason** parameter. As a result, values other than those values documented are possible.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_CONVERTED_MSG_TOO_BIG
(2120, X'848') Converted data too large for buffer.

MQRC_CONVERTED_STRING_TOO_BIG
(2190, X'88E') Converted string too large for field.

MQRC_DBCS_ERROR
(2150, X'866') DBCS string not valid.

MQRC_FORMAT_ERROR
(2110, X'83E') Message format not valid.

MQRC_INCOMPLETE_GROUP
(2241, X'8C1') Message group not complete.

MQRC_INCOMPLETE_MSG
(2242, X'8C2') Logical message not complete.

MQRC_INCONSISTENT_CCSIDS
(2243, X'8C3') Message segments have differing CCSIDs.

MQRC_INCONSISTENT_ENCODINGS
(2244, X'8C4') Message segments have differing encodings.

MQRC_INCONSISTENT_UOW
(2245, X'8C5') Inconsistent unit-of-work specification.

MQRC_MSG_TOKEN_ERROR
(2331, X'91B') Invalid use of message token.

MQRC_NO_MSG_LOCKED
(2209, X'8A1') No message locked.

MQRC_NOT_CONVERTED
(2119, X'847') Message data not converted.

MQRC_OPTIONS_CHANGED
(nnnn, X'xxx') Options that were required to be consistent have been changed.

MQRC_PARTIALLY_CONVERTED
(2272, X'8E0') Message data partially converted.

MQRC_SIGNAL_REQUEST_ACCEPTED
(2070, X'816') No message returned (but signal request accepted).

MQRC_SOURCE_BUFFER_ERROR
(2145, X'861') Source buffer parameter not valid.

MQRC_SOURCE_CCSID_ERROR
(2111, X'83F') Source coded character set identifier not valid.

MQRC_SOURCE_DECIMAL_ENC_ERROR
(2113, X'841') Packed-decimal encoding in message not recognized.

MQRC_SOURCE_FLOAT_ENC_ERROR
(2114, X'842') Floating-point encoding in message not recognized.

MQRC_SOURCE_INTEGER_ENC_ERROR
(2112, X'840') Source integer encoding not recognized.

MQRC_SOURCE_LENGTH_ERROR
(2143, X'85F') Source length parameter not valid.

MQRC_TARGET_BUFFER_ERROR
(2146, X'862') Target buffer parameter not valid.

MQRC_TARGET_CCSID_ERROR
(2115, X'843') Target coded character set identifier not valid.

MQRC_TARGET_DECIMAL_ENC_ERROR
(2117, X'845') Packed-decimal encoding specified by receiver not recognized.

MQRC_TARGET_FLOAT_ENC_ERROR
(2118, X'846') Floating-point encoding specified by receiver not recognized.

MQRC_TARGET_INTEGER_ENC_ERROR
(2116, X'844') Target integer encoding not recognized.

MQRC_TRUNCATED_MSG_ACCEPTED
(2079, X'81F') Truncated message returned (processing completed).

MQRC_TRUNCATED_MSG_FAILED
(2080, X'820') Truncated message returned (processing not completed).

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_CONV_LOAD_ERROR
(2133, X'855') Unable to load data conversion services modules.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BACKED_OUT
(2003, X'7D3') Unit of work backed out.

MQRC_BUFFER_ERROR
(2004, X'7D4') Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_NOT_AVAILABLE
(2345, X'929') Coupling facility not available.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CF_STRUC_LIST_HDR_IN_USE
(2347, X'92B') Coupling-facility structure list-header in use.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_CORREL_ID_ERROR
(2207, X'89F') Correlation-identifier error.

MQRC_DATA_LENGTH_ERROR
(2010, X'7DA') Data length parameter not valid.

MQRC_DB2_NOT_AVAILABLE
(2342, X'926') Db2 subsystem not available.

MQRC_GET_INHIBITED
(2016, X'7E0') Gets inhibited for the queue.

MQRC_GLOBAL_UOW_CONFLICT
(2351, X'92F') Global units of work conflict.

MQRC_GMO_ERROR
(2186, X'88A') Get-message options structure not valid.

MQRC_HANDLE_IN_USE_FOR_UOW
(2353, X'931') Handle in use for global unit of work.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_INCONSISTENT_BROWSE
(2259, X'8D3') Inconsistent browse specification.

MQRC_INCONSISTENT_UOW
(2245, X'8C5') Inconsistent unit-of-work specification.

MQRC_INVALID_MSG_UNDER_CURSOR
(2246, X'8C6') Message under cursor not valid for retrieval.

MQRC_LOCAL_UOW_CONFLICT
(2352, X'930') Global unit of work conflicts with local unit of work.

MQRC_MATCH_OPTIONS_ERROR
(2247, X'8C7') Match options not valid.

MQRC_MD_ERROR
(2026, X'7EA') Message descriptor not valid.

MQRC_MSG_ID_ERROR
(2206, X'89E') Message-identifier error.

MQRC_MSG_SEQ_NUMBER_ERROR
(2250, X'8CA') Message sequence number not valid.

MQRC_MSG_TOKEN_ERROR
(2331, X'91B') Use of message token not valid.

MQRC_NO_MSG_AVAILABLE
(2033, X'7F1') No message available.

MQRC_NO_MSG_UNDER_CURSOR
(2034, X'7F2') Browse cursor not positioned on message.

MQRC_NOT_OPEN_FOR_BROWSE
(2036, X'7F4') Queue not open for browse.

MQRC_NOT_OPEN_FOR_INPUT
(2037, X'7F5') Queue not open for input.

MQRC_OBJECT_CHANGED
(2041, X'7F9') Object definition changed since opened.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_INDEX_TYPE_ERROR
(2394, X'95A') Queue has wrong index type.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SECOND_MARK_NOT_ALLOWED
(2062, X'80E') A message is already marked.

MQRC_SIGNAL_OUTSTANDING
(2069, X'815') Signal outstanding for this handle.

MQRC_SIGNAL1_ERROR
(2099, X'833') Signal field not valid.

MQRC_STORAGE_MEDIUM_FULL
(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_SYNCPOINT_LIMIT_REACHED
(2024, X'7E8') No more messages can be handled within current unit of work.

MQRC_SYNCPOINT_NOT_AVAILABLE
(2072, X'818') sync point support not available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

MQRC_UOW_ENLISTMENT_ERROR
(2354, X'932') Enlistment in global unit of work failed.

MQRC_UOW_MIX_NOT_SUPPORTED
(2355, X'933') Mixture of unit-of-work calls not supported.

MQRC_UOW_NOT_AVAILABLE
(2255, X'8CF') Unit of work not available for the queue manager to use.

MQRC_WAIT_INTERVAL_ERROR
(2090, X'82A') Wait interval in MQGMO not valid.

MQRC_WRONG_GMO_VERSION
(2256, X'8D0') Wrong version of MQGMO supplied.

MQRC_WRONG_MD_VERSION
(2257, X'8D1') Wrong version of MQMD supplied.

For detailed information about these codes, see Reason codes.

Usage notes

1. The message retrieved is normally deleted from the queue. This deletion can occur as part of the MQGET call itself, or as part of a sync point.

   The browse options are: MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT, and MQGMO_BROWSE_MSG_UNDER_CURSOR.
2. If the MQGMO_LOCK option is specified with one of the browse options, the browsed message is locked so that it is visible only to this handle.

   If the MQGMO_UNLOCK option is specified, a previously locked message is unlocked. No message is retrieved in this case, and the **MsgDesc**, **BufferLength**, **Buffer**, and **DataLength** parameters are not checked or altered.
3. For applications issuing an MQGET call, the message retrieved can be lost if the application terminates abnormally or the connection is severed while processing the call. This issue arises because the surrogate running on the same platform as the queue manager that issues the MQGET call on behalf of the application cannot detect the loss of the application until the surrogate is about to return the message to the application, *after* the message has been removed from the queue. This issue can occur for both persistent messages and nonpersistent messages.

   To eliminate the risk of losing messages in this way, always retrieve messages within units of work. That is, by specifying the MQGMO_SYNCPOINT option on the MQGET call, and using the MQCMIT or MQBACK calls to commit or back out the unit of work when message processing is complete. If MQGMO_SYNCPOINT is specified, and the client terminates abnormally or the connection is severed, the surrogate backs out the unit of work on the queue manager and the message is reinstated on the queue. For more information about sync points, see Syncpoint considerations in IBM MQ applications.

   This situation can arise with IBM MQ clients as well as with applications that are running on the same platform as the queue manager.
4. If an application puts a sequence of messages on a particular queue within a single unit of work, and then commits that unit of work successfully, the messages become available for retrieval as follows:
   - If the queue is a *nonshared* queue (that is, a local queue), all messages within the unit of work become available at the same time.

- If the queue is a *shared* queue, messages within the unit of work become available in the order in which they were put, but not all at the same time. When the system is heavily laden, it is possible for the first message in the unit of work to be retrieved successfully, but for the MQGET call for the second or subsequent message in the unit of work to fail with MQRC_NO_MSG_AVAILABLE. If this issue occurs, the application must wait a short while and then try the operation again.

5. If an application puts a sequence of messages on the same queue without using message groups, the order of those messages is preserved if certain conditions are satisfied. See MQPUT usage notes for details. If the conditions are satisfied, the messages are presented to the receiving application in the order in which they were sent, if:
   - Only one receiver is getting messages from the queue.

     If there are two or more applications getting messages from the queue, they must agree with the sender the mechanism to be used to identify messages that belong to a sequence. For example, the sender might set all the *CorrelId* fields in the messages in a sequence to a value that was unique to that sequence of messages.
   - The receiver does not deliberately change the order of retrieval, for example by specifying a particular *MsgId* or *CorrelId*.

   If the sending application puts the messages as a message group, the messages are presented to the receiving application in the correct order if the receiving application specifies the MQGMO_LOGICAL_ORDER option on the MQGET call. For more information about message groups, see:
   - MQMD - MsgFlags field
   - MQPMO_LOGICAL_ORDER
   - MQGMO_LOGICAL_ORDER

   If the user is getting messages in a group under sync point, they must ensure that the complete group is processed before attempting to finish the transaction.
6. Applications must test for the feedback code MQFB_QUIT in the *Feedback* field of the **MsgDesc** parameter, and end if they find this value. See MQMD - Feedback field for more information.
7. If the queue identified by *Hobj* was opened with the MQOO_SAVE_ALL_CONTEXT option, and the completion code from the MQGET call is MQCC_OK or MQCC_WARNING, the context associated with the queue handle *Hobj* is set to the context of the message that has been retrieved (unless the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT, or MQGMO_BROWSE_MSG_UNDER_CURSOR option is set, in which case the context is marked as not available).

   You can use the saved context on a subsequent MQPUT or MQPUT1 call by specifying the MQPMO_PASS_IDENTITY_CONTEXT or MQPMO_PASS_ALL_CONTEXT options. This enables the context of the message received to be transferred in whole or in part to another message (for example, when the message is forwarded to another queue). For more information about message context, see Message context.
8. If you include the MQGMO_CONVERT option in the **GetMsgOpts** parameter, the application message data is converted to the representation requested by the receiving application, before the data is placed in the **Buffer** parameter:
   - The *Format* field in the control information in the message identifies the structure of the application data, and the *CodedCharSetId* and *Encoding* fields in the control information in the message specify its character-set identifier and encoding.
   - The application issuing the MQGET call specifies in the *CodedCharSetId* and *Encoding* fields in the **MsgDesc** parameter the character-set identifier and encoding to which to convert the application message data.

   When conversion of the message data is necessary, the conversion is performed either by the queue manager itself or by a user-written exit, depending on the value of the *Format* field in the control information in the message:
   - The following format names are formats that are converted by the queue manager; these formats are called "built-in" formats:

  - MQFMT_ADMIN
  - MQFMT_CICS (z/OS only)
  - MQFMT_COMMAND_1
  - MQFMT_COMMAND_2
  - MQFMT_DEAD_LETTER_HEADER
  - MQFMT_DIST_HEADER
  - MQFMT_EVENT version 1
  - MQFMT_EVENT version 2 (z/OS only)
  - MQFMT_IMS
  - MQFMT_IMS_VAR_STRING
  - MQFMT_MD_EXTENSION
  - MQFMT_PCF
  - MQFMT_REF_MSG_HEADER
  - MQFMT_RF_HEADER
  - MQFMT_RF_HEADER_2
  - MQFMT_STRING
  - MQFMT_TRIGGER
  - MQFMT_WORK_INFO_HEADER (z/OS only)
  - MQFMT_XMIT_Q_HEADER
- The format name MQFMT_NONE is a special value that indicates that the nature of the data in the message is undefined. As a consequence, the queue manager does not attempt conversion when the message is retrieved from the queue.

  Note: If MQGMO_CONVERT is specified on the MQGET call for a message that has a format name of MQFMT_NONE, and the character set or encoding of the message differs from that specified in the **MsgDesc** parameter, the message is returned in the **Buffer** parameter (assuming no other errors), but the call completes with completion code MQCC_WARNING and reason code MQRC_FORMAT_ERROR.

  You can use MQFMT_NONE either when the nature of the message data means that it does not require conversion, or when the sending and receiving applications have agreed between themselves the form in which to send the message data.
- All other format names pass the message to a user-written exit for conversion. The exit has the same name as the format, apart from environment-specific additions. User-specified format names must not begin with the letters IBM MQ.

  See “Data conversion” on page 2909 for details of the data-conversion exit.

User data in the message can be converted between any supported character sets and encodings. However, be aware that, if the message contains one or more IBM MQ header structures, the message cannot be converted from or to a character set that has double-byte or multi-byte characters for any of the characters that are valid in queue names. Reason code MQRC_SOURCE_CCSID_ERROR or MQRC_TARGET_CCSID_ERROR results if this is attempted, and the message is returned unconverted. Unicode character set V 9.0.0 UTF-16 is an example of such a character set.

On return from MQGET, the following reason code indicates that the message was converted successfully:

- MQRC_NONE

The following reason code indicates that the message *might* have been converted successfully; the application must check the *CodedCharSetId* and *Encoding* fields in the **MsgDesc** parameter to find out:

- MQRC_TRUNCATED_MSG_ACCEPTED

All other reason codes indicate that the message was not converted.

Note: The interpretation of this reason code is true for conversions performed by a user-written exit only if the exit conforms to the processing guidelines described in "Data conversion" on page 2909.

9. When using the object-oriented interface to get messages, you can choose not to specify a buffer to hold the message data for an MQGET call. However, in versions of IBM MQ, prior to version 7, it was possible for MQGET to fail with reason code MQRC_CONVERTED_MSG_TO_BIG, even when a buffer was not specified. From IBM MQ Version 7, when you get a message using an object-oriented application without restricting the size of the receive message buffer, the application does not fail with MQRC_CONVERTED_MSG_TOO_BIG, and receives the converted message. This is true of the following environments:
   - .NET, including fully managed applications
   - C++
   - Java (IBM MQ classes for Java)

   Note: For all clients, if the value of *sharingConversations* is zero, the channel operates as it did before IBM WebSphere MQ Version 7.0, and message handling reverts to Version 6 behavior. In this situation, if the buffer is too small to receive the converted message, the unconverted message is returned, with reason code MQRC_CONVERTED_MSG_TOO_BIG. For more information about *sharingConversations*, see Using sharing conversations in a client application.
10. For the built-in formats, the queue manager can perform *default conversion* of character strings in the message when the MQGMO_CONVERT option is specified. Default conversion allows the queue manager to use an installation-specified default character set that approximates the actual character set, when converting string data. As a result, the MQGET call can succeed with completion code MQCC_OK, instead of completing with MQCC_WARNING and reason code MQRC_SOURCE_CCSID_ERROR or MQRC_TARGET_CCSID_ERROR.

    Note: The result of using an approximate character set to convert string data is that some characters might be converted incorrectly. To avoid this, use characters in the string that are common to both the actual character set and the default character set.

    Default conversion applies both to the application message data and to character fields in the MQMD and MQMDE structures:
    - Default conversion of the application message data occurs only when *all* the following statements are true:
      - The application specifies MQGMO_CONVERT.
      - The message contains data that must be converted either from or to a character set that is not supported.
      - Default conversion was enabled when the queue manager was installed or restarted.
    - Default conversion of the character fields in the MQMD and MQMDE structures occurs as necessary, if default conversion is enabled for the queue manager. The conversion is performed even if the MQGMO_CONVERT option is not specified by the application on the MQGET call.
11. For the Visual Basic programming language, the following points apply:
    - If the size of the **Buffer** parameter is less than the length specified by the **BufferLength** parameter, the call fails with reason code MQRC_STORAGE_NOT_AVAILABLE.
    - The **Buffer** parameter is declared as being of type `String`. If the data to be retrieved from the queue is not of type `String`, use the MQGETAny call in place of MQGET.

      The MQGETAny call has the same parameters as the MQGET call, except that the **Buffer** parameter is declared as being of type `Any`, allowing any type of data to be retrieved. However, this means that *Buffer* cannot be checked to ensure that it is at least *BufferLength* bytes in size.
12. Not all MQGET options are supported when read ahead is enabled. The following table indicated which options are allowed and whether they can be altered between MQGET calls.

Table 266. MQGET options permitted when read ahead is enabled

	Permitted when read ahead is enabled and can be altered between MQGET calls	Permitted when read ahead is enabled but cannot be altered between MQGET calls [a]	MQGET options that are not permitted when read ahead is enabled [b]
MQGET MD values	MsgId [c] CorrelId [c]	Encoding CodedCharSetId	
MQGET MQGMO options	MQGMO_WAIT MQGMO_NO_WAIT MQGMO_FAIL_IF_QUIESCING MQGMO_BROWSE_FIRST [d] MQGMO_BROWSE_NEXT [d] MQGMO_BROWSE_MESSAGE _UNDER_CURSOR [d]	MQGMO_SYNCPOINT_IF_PERSISTENT MQGMO_NO_SYNCPOINT MQGMO_ACCEPT_TRUNCATED_MSG MQGMO_CONVERT MQGMO_LOGICAL_ORDER MQGMO_COMPLETE_MSG MQGMO_ALL_MSGS_AVAILABLE MQGMO_ALL_SEGMENTS_AVAILABLE MQGMO_MARK_BROWSE_HANDLE MQGMO_MARK_BROWSE_CO_OP MQGMO_UNMARK_BROWSE_CO_OP MQGMO_UNMARK_BROWSE_HANDLE MQGMO_UNMARKED_BROWSE_MSG MQGMO_PROPERTIES_FORCE_MQRFH2 MQGMO_NO_PROPERTIES MQGMO_PROPERTIES_IN_HANDLE MQGMO_PROPERTIES_COMPATIBILITY	MQGMO_SET_SIGNAL MQGMO_SYNCPOINT MQGMO_MARK_SKIP _BACKOUT MQGMO_MSG_UNDER _CURSOR [d] MQGMO_LOCK MQGMO_UNLOCK
MQGMO values		MsgHandle	

a. If these options are altered between MQGET calls an MQRC_OPTIONS_CHANGED reason code is returned.

b. If these options are specified on the first MQGET call then read ahead is disabled. If these options are specified on a subsequent MQGET call a reason code MQRC_OPTIONS_ERROR is returned.

c. The client applications need to be aware that if the MsgId and CorrelId values are altered between MQGET calls messages with the previous values might have already been sent to the client and remain in the client read ahead buffer until consumed (or automatically purged).

d. The first MQGET call determines whether messages are to be browsed or got from a queue when read ahead is enabled. If the application attempts to use a combination of browse and get an MQRC_OPTIONS_CHANGED reason code is returned.

e. MQGMO_MSG_UNDER_CURSOR is not possible with read ahead. Messages can be browsed or got when read ahead is enabled but not a combination of both.

13. Applications can destructively get uncommitted messages only if those messages are put in the same local unit of work as the get. Applications cannot get uncommitted messages nondestructively.

14. Messages under a browse cursor can be retrieved in a unit of work. It is not possible to retrieve an uncommitted message in this way.

C invocation

```
MQGET (Hconn, Hobj, &MsgDesc, &GetMsgOpts, BufferLength, Buffer,
      &DataLength, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;         /* Connection handle */
MQHOBJ   Hobj;          /* Object handle */
MQMD     MsgDesc;       /* Message descriptor */
MQGMO    GetMsgOpts;    /* Options that control the action of MQGET */
MQLONG   BufferLength;  /* Length in bytes of the Buffer area */
MQBYTE   Buffer[n];     /* Area to contain the message data */
MQLONG   DataLength;    /* Length of the message */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQGET' USING HCONN, HOBJ, MSGDESC, GETMSGOPTS, BUFFERLENGTH,
BUFFER, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN         PIC S9(9) BINARY.
**   Object handle
 01  HOBJ          PIC S9(9) BINARY.
```

```
**   Message descriptor
 01  MSGDESC.
     COPY CMQMDV.
**   Options that control the action of MQGET
 01  GETMSGOPTS.
     COPY CMQGMOV.
**   Length in bytes of the BUFFER area
 01  BUFFERLENGTH  PIC S9(9) BINARY.
**   Area to contain the message data
 01  BUFFER        PIC X(n).
**   Length of the message
 01  DATALENGTH    PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE      PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON        PIC S9(9) BINARY.
```

PL/I invocation

```
call MQGET (Hconn, Hobj, MsgDesc, GetMsgOpts, BufferLength, Buffer,
           DataLength, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn         fixed bin(31);  /* Connection handle */
dcl Hobj          fixed bin(31);  /* Object handle */
dcl MsgDesc       like MQMD;      /* Message descriptor */
dcl GetMsgOpts    like MQGMO;     /* Options that control the action of
                                     MQGET */
dcl BufferLength  fixed bin(31);  /* Length in bytes of the Buffer
                                     area */
dcl Buffer        char(n);        /* Area to contain the message data */
dcl DataLength    fixed bin(31);  /* Length of the message */
dcl CompCode      fixed bin(31);  /* Completion code */
dcl Reason        fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
         CALL MQGET,(HCONN,HOBJ,MSGDESC,GETMSGOPTS,BUFFERLENGTH,
               BUFFER,DATALENGTH,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN         DS       F      Connection handle
HOBJ          DS       F      Object handle
MSGDESC       CMQMDA   ,      Message descriptor
GETMSGOPTS    CMQGMOA  ,      Options that control the action of MQGET
BUFFERLENGTH  DS       F      Length in bytes of the BUFFER area
BUFFER        DS       CL(n)  Area to contain the message data
DATALENGTH    DS       F      Length of the message
COMPCODE      DS       F      Completion code
REASON        DS       F      Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQGET Hconn, Hobj, MsgDesc, GetMsgOpts, BufferLength, Buffer,
DataLength, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn        As Long   'Connection handle'
Dim Hobj         As Long   'Object handle'
Dim MsgDesc      As MQMD   'Message descriptor'
Dim GetMsgOpts   As MQGMO  'Options that control the action of MQGET'
Dim BufferLength As Long   'Length in bytes of the Buffer area'
Dim Buffer       As String 'Area to contain the message data'
```

```
Dim DataLength   As Long   'Length of the message'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

MQINQ - Inquire object attributes:

The MQINQ call returns an array of integers and a set of character strings containing the attributes of an object.

The following types of object are valid:

- Queue manager
- Queue
- Namelist
- Process definition

Syntax

MQINQ (*Hconn*, *Hobj*, *SelectorCount*, *Selectors*, *IntAttrCount*, *IntAttrs*, *CharAttrLength*, *CharAttrs*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hobj

Type: MQHOBJ - input

This handle represents the object (of any type) with attributes that are required. The handle must be returned by a previous MQOPEN call that specified the MQOO_INQUIRE option.

SelectorCount

Type: MQLONG - input

This is the count of selectors that are supplied in the *Selectors* array. It is the number of attributes that are to be returned. Zero is a valid value. The maximum number allowed is 256.

Selectors

Type: MQLONG x *SelectorCount* - input

This is an array of **SelectorCount** attribute selectors; each selector identifies an attribute (integer or character) with a value that is required.

Each selector must be valid for the type of object that *Hobj* represents, otherwise the call fails with completion code MQCC_FAILED and reason code MQRC_SELECTOR_ERROR.

In the special case of queues:

- If the selector is not valid for queues of any type, the call fails with completion code MQCC_FAILED and reason code MQRC_SELECTOR_ERROR.
- If the selector applies only to queues of types other than the type of the object, the call succeeds with completion code MQCC_WARNING and reason code MQRC_SELECTOR_NOT_FOR_TYPE.

- If the queue being inquired is a cluster queue, the selectors that are valid depend on how the queue was resolved; see "Usage notes" on page 2712 for further details.

You can specify selectors in any order. Attribute values that correspond to integer attribute selectors (MQIA_* selectors) are returned in *IntAttrs* in the same order in which these selectors occur in *Selectors*. Attribute values that correspond to character attribute selectors (MQCA_* selectors) are returned in *CharAttrs* in the same order in which those selectors occur. MQIA_* selectors can be interleaved with the MQCA_* selectors; only the relative order within each type is important.

Note:

1. The integer and character attribute selectors are allocated within two different ranges; the MQIA_* selectors reside within the range MQIA_FIRST through MQIA_LAST, and the MQCA_* selectors within the range MQCA_FIRST through MQCA_LAST.

   For each range, the constants MQIA_LAST_USED and MQCA_LAST_USED define the highest value that the queue manager accepts.
2. If all the MQIA_* selectors occur first, the same element numbers can be used to address corresponding elements in the *Selectors* and *IntAttrs* arrays.
3. If the **SelectorCount** parameter is zero, *Selectors* is not referred to. In this case, the parameter address passed by programs written in C or S/390 assembler might be null.

The attributes that can be inquired are listed in the following tables. For the MQCA_* selectors, the constant that defines the length in bytes of the resulting string in *CharAttrs* is provided in parentheses.

The tables that follow list the selectors, by object, in alphabetical order, as follows:

- Table 267 MQINQ attribute selectors for queues
- Table 268 on page 2703 MQINQ attribute selectors for namelists
- Table 269 on page 2704 MQINQ attribute selectors for process definitions
- Table 270 on page 2704 MQINQ attribute selectors for the queue manager

All selectors are supported on all IBM MQ platforms, except where indicated in the **Note** column as follows:

Not z/OS
: Supported on all platforms **except** z/OS

z/OS Supported **only** on z/OS

Table 267. MQINQ attribute selectors for queues

Selector	Length of field	Description	Note
MQCA_ALTERATION_DATE	MQ_DATE_LENGTH	Date of most-recent alteration	
MQCA_ALTERATION_TIME	MQ_TIME_LENGTH	Time of most-recent alteration	
MQCA_BACKOUT_REQ_Q_NAME	MQ_Q_NAME_LENGTH	Excessive backout requeue name	
MQCA_BASE_Q_NAME	MQ_Q_NAME_LENGTH	Name of queue that alias resolves to	
MQCA_CF_STRUC_NAME	MQ_CF_STRUC_NAME_LENGTH	Coupling-facility structure name	z/OS
MQCA_CLUS_CHL_NAME	MQ_CHANNEL_NAME_LENGTH	Name of the cluster-sender channel that uses this queue as a transmission queue.	
MQCA_CLUSTER_NAME	MQ_CLUSTER_NAME_LENGTH	Cluster name	
MQCA_CLUSTER_NAMELIST	MQ_NAMELIST_NAME_LENGTH	Cluster namelist	
MQCA_CREATION_DATE	MQ_CREATION_DATE_LENGTH	Queue creation date	
MQCA_CREATION_TIME	MQ_CREATION_TIME_LENGTH	Queue creation time	
MQCA_INITIATION_Q_NAME	MQ_Q_NAME_LENGTH	Initiation queue name	
MQCA_PROCESS_NAME	MQ_PROCESS_NAME_LENGTH	Name of process definition	

Table 267. MQINQ attribute selectors for queues (continued)

Selector	Length of field	Description	Note
MQCA_Q_DESC	MQ_Q_DESC_LENGTH	Queue description	
MQCA_Q_NAME	MQ_Q_NAME_LENGTH	Queue name	
MQCA_REMOTE_Q_MGR_NAME	MQ_Q_MGR_NAME_LENGTH	Name of remote queue manager	
MQCA_REMOTE_Q_NAME	MQ_Q_NAME_LENGTH	Name of remote queue as known on remote queue manager	
MQCA_STORAGE_CLASS	MQ_STORAGE_CLASS_LENGTH	Name of storage class	z/OS
MQCA_TRIGGER_DATA	MQ_TRIGGER_DATA_LENGTH	Trigger data	
MQCA_XMIT_Q_NAME	MQ_Q_NAME_LENGTH	Transmission queue name	
MQIA_ACCOUNTING_Q	MQLONG	Controls collection of accounting data for queue	Not z/OS
MQIA_BACKOUT_THRESHOLD	MQLONG	Backout threshold	
MQIA_CLWL_Q_PRIORITY	MQLONG	Priority of queue	
MQIA_CLWL_Q_RANK	MQLONG	Rank of queue	
MQIA_CLWL_USEQ	MQLONG	Use remote queues	
MQIA_CURRENT_Q_DEPTH	MQLONG	Number of messages on queue	
MQIA_DEF_BIND	MQLONG	Default binding	
MQIA_DEF_INPUT_OPEN_OPTION	MQLONG	Default open-for-input option	
MQIA_DEF_PERSISTENCE	MQLONG	Default message persistence	
MQIA_DEF_PRIORITY	MQLONG	Default message priority	
MQIA_DEFINITION_TYPE	MQLONG	Queue definition type	
MQIA_DIST_LISTS	MQLONG	Distribution list support	Not z/OS
MQIA_HARDEN_GET_BACKOUT	MQLONG	Whether to harden backout count	
MQIA_INDEX_TYPE	MQLONG	Type of index maintained for queue	z/OS
MQIA_INHIBIT_GET	MQLONG	Whether get operations are allowed	
MQIA_INHIBIT_PUT	MQLONG	Whether put operations are allowed	
MQIA_MAX_MSG_LENGTH	MQLONG	Maximum message length	
MQIA_MAX_Q_DEPTH	MQLONG	Maximum number of messages allowed on queue	
MQIA_MSG_DELIVERY_SEQUENCE	MQLONG	Whether message priority is relevant	
MQIA_NPM_CLASS	MQLONG	Level of reliability for nonpersistent messages	
MQIA_OPEN_INPUT_COUNT	MQLONG	Number of MQOPEN calls that have the queue open for input	
MQIA_OPEN_OUTPUT_COUNT	MQLONG	Number of MQOPEN calls that have the queue open for output	
MQIA_PROPERTY_CONTROL	MQLONG	Property control attribute	
MQIA_Q_DEPTH_HIGH_EVENT	MQLONG	Control attribute for queue depth high events	Not z/OS
MQIA_Q_DEPTH_HIGH_LIMIT	MQLONG	High limit for queue depth	Not z/OS
MQIA_Q_DEPTH_LOW_EVENT	MQLONG	Control attribute for queue depth low events	Not z/OS

Table 267. MQINQ attribute selectors for queues (continued)

Selector	Length of field	Description	Note
MQIA_Q_DEPTH_LOW_LIMIT	MQLONG	Low limit for queue depth	Not z/OS
MQIA_Q_DEPTH_MAX_EVENT	MQLONG	Control attribute for queue depth max events	Not z/OS
MQIA_Q_SERVICE_INTERVAL	MQLONG	Limit for queue service interval	Not z/OS
MQIA_Q_SERVICE_INTERVAL_EVENT	MQLONG	Control attribute for queue service interval events	Not z/OS
MQIA_Q_TYPE	MQLONG	Queue type	
MQIA_QSG_DISP	MQLONG	Queue-sharing group disposition	z/OS
MQIA_RETENTION_INTERVAL	MQLONG	Queue retention interval	
MQIA_SCOPE	MQLONG	Queue definition scope	Not z/OS
MQIA_SHAREABILITY	MQLONG	Whether queue can be shared for input	
MQIA_STATISTICS_Q	MQLONG	Controls collection of statistics data for queue	Not z/OS
MQIA_TRIGGER_CONTROL	MQLONG	Trigger control	
MQIA_TRIGGER_DEPTH	MQLONG	Trigger depth	
MQIA_TRIGGER_MSG_PRIORITY	MQLONG	Threshold message priority for triggers	
MQIA_TRIGGER_TYPE	MQLONG	Trigger type	
MQIA_USAGE	MQLONG	Usage	

Table 268. MQINQ attribute selectors for namelists

Selector	Length of field	Description	Note
MQCA_ALTERATION_DATE	MQ_DATE_LENGTH	Date of most-recent alteration	
MQCA_ALTERATION_TIME	MQ_TIME_LENGTH	Time of most-recent alteration	
MQCA_NAMELIST_DESC	MQ_NAMELIST_DESC_LENGTH	Namelist description	
MQCA_NAMELIST_NAME	MQ_NAMELIST_NAME_LENGTH	Name of namelist object	
MQIA_NAMELIST_TYPE	MQLONG	Namelist type	z/OS
MQCA_NAMES	MQ_Q_NAME_LENGTH x *Number of names in the list*	Names in the namelist	
MQIA_NAME_COUNT	MQLONG	Number of names in the namelist	
MQIA_QSG_DISP	MQLONG	Queue-sharing group disposition	z/OS

Table 269. MQINQ attribute selectors for process definitions

Selector	Length of field	Description	Note
MQCA_ALTERATION_DATE	MQ_DATE_LENGTH	Date of most-recent alteration	
MQCA_ALTERATION_TIME	MQ_TIME_LENGTH	Time of most-recent alteration	
MQCA_APPL_ID	MQ_PROCESS_APPL_ID_LENGTH	Application identifier	
MQCA_ENV_DATA	MQ_PROCESS_ENV_DATA_LENGTH	Environment data	
MQCA_PROCESS_DESC	MQ_PROCESS_DESC_LENGTH	Description of process definition	
MQCA_PROCESS_NAME	MQ_PROCESS_NAME_LENGTH	Name of process definition	
MQCA_USER_DATA	MQ_PROCESS_USER_DATA_LENGTH	User data	
MQIA_APPL_TYPE	MQLONG	Application type	
MQIA_QSG_DISP	MQLONG	Queue-sharing group disposition	z/OS

Table 270. MQINQ attribute selectors for the queue manager

Selector	Length of field	Description	Note
MQCA_ALTERATION_DATE	MQ_DATE_LENGTH	Date of most-recent alteration	
MQCA_ALTERATION_TIME	MQ_TIME_LENGTH	Time of most-recent alteration	
MQCA_CHANNEL_AUTO_DEF_EXIT	MQ_EXIT_NAME_LENGTH	Automatic channel definition exit name	
MQCA_CHINIT_SERVICE_PARM		Reserved for use by IBM	
MQCA_CLUSTER_WORKLOAD_DATA	MQ_EXIT_DATA_LENGTH	Data passed to cluster workload exit	
MQCA_CLUSTER_WORKLOAD_EXIT	MQ_EXIT_NAME_LENGTH	Name of cluster workload exit	
MQCA_COMMAND_INPUT_Q_NAME	MQ_Q_NAME_LENGTH	System command input queue name	
MQCA_DEAD_LETTER_Q_NAME	MQ_Q_NAME_LENGTH	Name of dead-letter queue	
MQCA_DEF_XMIT_Q_NAME	MQ_Q_NAME_LENGTH	Default transmission queue name	
MQCA_DNS_GROUP	MQ_DNS_GROUP_NAME_LENGTH	Name of the group for the TCP listener that handles inbound transmissions for the queue-sharing group to join. The name applies when using Workload Manager Dynamic Domain Name Services.	z/OS
MQCA_IGQ_USER_ID	MQ_USER_ID_LENGTH	Intra-group queuing user identifier	z/OS
MQCA_INSTALLATION_DESC	MQ_INSTALLATION_DESC_LENGTH	Description of the associated installation	Not z/OS. Not IBM i
MQCA_INSTALLATION_NAME	MQ_INSTALLATION_NAME_LENGTH	Name of the installation associated with the queue manager	Not z/OS. Not IBM i
MQCA_INSTALLATION_PATH	MQ_INSTALLATION_PATH_LENGTH	Path where the associated IBM MQ is installed	Not z/OS. Not IBM i
MQCA_LU_GROUP_NAME	MQ_LU_NAME_LENGTH	Generic LU name for the LU 6.2 listener that handles inbound transmissions for the queue-sharing group to use	z/OS

Table 270. MQINQ attribute selectors for the queue manager (continued)

Selector	Length of field	Description	Note
MQCA_LU_NAME	MQ_LU_NAME_LENGTH	Name of the LU to use for outbound LU 6.2 transmissions. Set this name to the same LU that the listener uses for inbound transmissions	z/OS
MQCA_LU62_ARM_SUFFIX	MQ_ARM_SUFFIX_LENGTH	Suffix of the SYS1.PARMLIB member APPCPM *xx*, that nominates the LUADD for this channel initiator	z/OS
MQCA_PARENT	MQ_Q_MGR_NAME_LENGTH	Name of a hierarchically connected queue manager that is nominated as the parent of this queue manager	
MQCA_Q_MGR_DESC	MQ_Q_MGR_DESC_LENGTH	Queue manager description	
MQCA_Q_MGR_IDENTIFIER	MQ_Q_MGR_IDENTIFIER_LENGT	Queue manager identifier (H)	
MQCA_Q_MGR_NAME	MQ_Q_MGR_NAME_LENGTH	Name of local queue manager	
MQCA_QSG_NAME	MQ_QSG_NAME_LENGTH	Queue-sharing group name	z/OS
MQCA_REPOSITORY_NAME	MQ_CLUSTER_NAME_LENGTH	Name of cluster for which queue manager provides repository services	
MQCA_REPOSITORY_NAMELIST	MQ_NAMELIST_NAME_LENGTH	Name of namelist object containing names of clusters for which queue manager provides repository services	
MQCA_TCP_NAME	MQ_TCP_NAME_LENGTH	Name of the TCP/IP system that you are using	z/OS
MQIA_ACCOUNTING_CONN_OVERRIDE	MQLONG	Override accounting settings	Not z/OS
MQIA_ACCOUNTING_INTERVAL	MQLONG	How often to write intermediate accounting records	Not z/OS
MQIA_ACCOUNTING_MQI	MQLONG	Controls collection of accounting information for MQI data	Not z/OS
MQIA_ACCOUNTING_Q	MQLONG	Controls collection of accounting information for queues	Not z/OS
MQIA_ACTIVE_CHANNELS	MQLONG	Maximum number of channels that can be active at any time	z/OS
MQIA_ADOPTNEWMCA_CHECK	MQLONG	Elements that are checked to determine whether to adopt an MCA. The check is performed when a new inbound channel is detected that has the same name as an MCA that is already active.	z/OS
MQIA_ADOPTNEWMCA_INTERVAL	MQLONG	Amount of time, in seconds, that the new channel waits for the orphaned channel to end	Not z/OS
MQIA_ADOPTNEWMCA_TYPE	MQLONG	Whether to restart an orphaned instance of an MCA of a particular channel type automatically when a new inbound channel request matching the AdoptNewMCACheck parameters is detected	z/OS
MQIA_AUTHORITY_EVENT	MQLONG	Control attribute for authority events	Not z/OS
MQIA_BRIDGE_EVENT	MQLONG	Control attribute for IMS bridge events	z/OS

Table 270. MQINQ attribute selectors for the queue manager (continued)

Selector	Length of field	Description	Note
MQIA_CHANNEL_AUTO_DEF	MQLONG	Control attribute for automatic channel definition	Not z/OS
MQIA_CHANNEL_AUTO_DEF_EVENT	MQLONG	Control attribute for automatic channel definition events	Not z/OS
MQIA_CHANNEL_EVENT	MQLONG	Control attribute for channel events	
MQIA_CHINIT_ADAPTERS	MQLONG	Number of adapter subtasks to use for processing IBM MQ calls	z/OS
MQIA_CHINIT_DISPATCHERS	MQLONG	Number of dispatchers to use for the channel initiator	z/OS
MQIA_CHINIT_TRACE_AUTO_START	MQLONG	Whether to start channel initiator trace automatically	z/OS
MQIA_CHINIT_TRACE_TABLE_SIZE	MQLONG	Size of the trace data space (in MB) of the channel initiator	z/OS
MQIA_CLUSTER_WORKLOAD_LENGTH	MQLONG	Cluster workload length.	
MQIA_CLWL_MRU_CHANNELS	MQLONG	Number of most recently used channels for cluster workload balancing	
MQIA_CLWL_USEQ	MQLONG	Use remote queues	
MQIA_CODED_CHAR_SET_ID	MQLONG	Coded character set identifier	
MQIA_COMMAND_EVENT	MQLONG	Control attribute for command events	
MQIA_COMMAND_LEVEL	MQLONG	Command level supported by queue manager	
MQIA_CONFIGURATION_EVENT	MQLONG	Control attribute for configuration events	Not z/OS
MQIA_DEF_CLUSTER_XMIT_Q_TYPE	MQLONG	Default transmission queue type to be used for cluster-sender channels.	
MQIA_DIST_LISTS	MQLONG	Distribution list support	Not z/OS
MQIA_DNS_WLM	MQLONG	Whether the TCP listener that handles inbound transmissions for the queue-sharing group registers with Workload Manager for Dynamic Domain Name Services	z/OS
MQIA_EXPIRY_INTERVAL	MQLONG	Interval between scans for expired messages	z/OS
MQIA_GROUP_UR	MQLONG	Control attribute for whether GROUP units of recovery are enabled for this queue manager. The GROUP unit of recovery disposition is only available if the queue manager is a member of a queue-sharing group	z/OS
MQIA_IGQ_PUT_AUTHORITY	MQLONG	Intra-group queuing put authority	z/OS
MQIA_INHIBIT_EVENT	MQLONG	Control attribute for inhibit events	Not z/OS
MQIA_INTRA_GROUP_queuing	MQLONG	Intra-group queuing support	z/OS
MQIA_LISTENER_TIMER	MQLONG	Time interval (in seconds) between IBM MQ attempts to restart the listener if APPC or TCP/IP failed.	z/OS

Table 270. MQINQ attribute selectors for the queue manager (continued)

Selector	Length of field	Description	Note
MQIA_LOCAL_EVENT	MQLONG	Control attribute for local events	Not z/OS
MQIA_LOGGER_EVENT	MQLONG	Control attribute for inhibit events	Not z/OS
MQIA_LU62_CHANNELS	MQLONG	Maximum number of channels that can be current, or clients that can be connected, using the LU 6.2 transmission protocol	z/OS
MQIA_MSG_MARK_BROWSE_INTERVAL	MQLONG	Time interval (in milliseconds) after which the queue manager can automatically remove a mark from browse messages	
MQIA_MAX_CHANNELS	MQLONG	Maximum number of channels that can be current (including server-connection channels with connected clients)	z/OS
MQIA_MAX_HANDLES	MQLONG	Maximum number of handles	
MQIA_MAX_MSG_LENGTH	MQLONG	Maximum message length	
MQIA_MAX_PRIORITY	MQLONG	Maximum priority	
MQIA_MAX_UNCOMMITTED_MSGS	MQLONG	Maximum number of uncommitted messages within a unit of work	
MQIA_OUTBOUND_PORT_MAX	MQLONG	With MQIA_OUTBOUND_PORT_MIN, defines range of port numbers to use when binding outgoing channels	z/OS
MQIA_OUTBOUND_PORT_MIN	MQLONG	With MQIA_OUTBOUND_PORT_MAX, defines range of port numbers to use when binding outgoing channels	z/OS
MQIA_PERFORMANCE_EVENT	MQLONG	Control attribute for performance events	Not z/OS
MQIA_PLATFORM	MQLONG	Platform on which the queue manager resides	
MQIA_PROT_POLICY_CAPABILITY	MQLONG	Indicates whether security capabilities of Advanced Message Security are available for a queue manager.	
MQIA_PUBSUB_MAXMSG_RETRY_COUNT	MQLONG	The number of attempts to reprocess a failed command message under sync point	
MQIA_PUBSUB_MODE	MQLONG	Whether the publish/subscribe engine and the queued publish/subscribe interface are running. Applications to publish or subscribe using the application programming interface require the publish/subscribe engine. Queues that are monitored by the queued publish/subscribe interface require the queued publish/subscribe interface to be running.	
MQIA_PUBSUB_NP_MSG	MQLONG	Whether to discard (or keep) an undelivered input message	
MQIA_PUBSUB_NP_RESP	MQLONG	Controls the behavior of undelivered response messages	

Table 270. MQINQ attribute selectors for the queue manager (continued)

Selector	Length of field	Description	Note
MQIA_PUBSUB_SYNC_PT	MQLONG	Whether only persistent (or all) messages are processed under sync point	
MQIA_QMGR_CFCONLOS	MQLONG	Specifies the action to be taken when the queue manager loses connectivity to the administration structure or any CF structures with CFCONLOS set to ASQMGR	z/OS
MQIA_RECEIVE_TIMEOUT	MQLONG	Approximately how long a TCP/IP channel waits to receive data, including heartbeats, from its partner, before returning to the inactive state. The value is numeric, qualified by MQIA_RECEIVE_TIMEOUT_TYPE.	z/OS
MQIA_RECEIVE_TIMEOUT_MIN	MQLONG	Minimum time that a TCP/IP channel waits to receive data, including heartbeats, from its partner, before returning to the inactive state	z/OS
MQIA_RECEIVE_TIMEOUT_TYPE	MQLONG	Approximately how long a TCP/IP channel waits to receive data, including heartbeats, from its partner, before returning to the inactive state. MQIA_RECEIVE_TIMEOUT_TYPE is the qualifier applied to MQIA_RECEIVE_TIMEOUT.	z/OS
MQIA_REMOTE_EVENT	MQLONG	Control attribute for remote events	Not z/OS
MQIA_SECURITY_CASE	MQLONG	Case of security profiles	z/OS
MQIA_SSL_EVENT	MQLONG	Control attribute for channel events	
MQIA_SSL_FIPS_REQUIRED	MQLONG	Use only FIPS-certified algorithms for cryptography	
MQIA_SSL_RESET_COUNT	MQLONG	TLS key reset count	
MQIA_START_STOP_EVENT	MQLONG	Control attribute for start stop events	Not z/OS
MQIA_STATISTICS_AUTO_CLUSSDR	MQLONG	Controls collection of statistics monitoring information for cluster sender channels	
MQIA_STATISTICS_CHANNEL	MQLONG	Controls collection of statistics data for channels	
MQIA_STATISTICS_INTERVAL	MQLONG	How often to write statistics monitoring data	Not z/OS
MQIA_STATISTICS_MQI	MQLONG	Controls collection of statistics monitoring information for queue manager	Not z/OS
MQIA_STATISTICS_Q	MQLONG	Controls collection of statistics data for queues	Not z/OS
MQIA_SYNCPOINT	MQLONG	sync point availability	
MQIA_TCP_CHANNELS	MQLONG	Maximum number of channels that can be current, or clients that can be connected, using the TCP/IP transmission protocol	z/OS

Table 270. MQINQ attribute selectors for the queue manager (continued)

Selector	Length of field	Description	Note
MQIA_TCP_KEEP_ALIVE	MQLONG	Whether to use the TCP KEEPALIVE facility to check that the other end of the connection is still available	z/OS
MQIA_TCP_STACK_TYPE	MQLONG	Whether the channel initiator can use only the TCP/IP address space specified in TCPNAME, or can optionally bind to any selected TCP/IP address	z/OS
MQIA_TRACE_ROUTE_RECORDING	MQLONG	Controls recording of trace-route information	z/OS
MQIA_TREE_LIFE_TIME	MQLONG	Lifetime of unused non-administrative topics	
MQIA_TRIGGER_INTERVAL	MQLONG	Trigger interval	

IntAttrCount
Type: MQLONG - input

This is the number of elements in the *IntAttrs* array. Zero is a valid value.

If IntAttrCount is at least the number of MQIA_* selectors in the **Selectors** parameter, all integer attributes requested are returned.

IntAttrs
Type: MQLONG x *IntAttrCount* - output

This is an array of *IntAttrCount* integer attribute values.

Integer attribute values are returned in the same order as the MQIA_* selectors in the **Selectors** parameter. If the array contains more elements than the number of MQIA_* selectors, the excess elements are unchanged.

If *Hobj* represents a queue, but an attribute selector does not apply to that type of queue, the specific value MQIAV_NOT_APPLICABLE is returned. It is returned for the corresponding element in the *IntAttrs* array.

If the **IntAttrCount** or **SelectorCount** parameter is zero, *IntAttrs* is not referred to. In this case, the parameter address passed by programs written in C or S/390 assembler might be null.

CharAttrLength
Type: MQLONG - input

This is the length in bytes of the **CharAttrs** parameter.

CharAttrLength must be at least the sum of the lengths of the requested character attributes (see Selectors). Zero is a valid value.

CharAttrs
Type: MQCHAR x *CharAttrLength* - output

This is the buffer in which the character attributes are returned, concatenated together. The length of the buffer is given by the **CharAttrLength** parameter.

Character attributes are returned in the same order as the MQCA_* selectors in the **Selectors** parameter. The length of each attribute string is fixed for each attribute (see Selectors), and the value in it is padded to the right with blanks if necessary. You can provider a buffer larger than needed to contain all the requested character attributes and padding. The bytes beyond the last attribute value returned are unchanged.

If *Hobj* represents a queue, but an attribute selector does not apply to that type of queue, a character string consisting entirely of asterisks (*) is returned. The asterisk is returned as the value of that attribute in *CharAttrs*.

If the *CharAttrLength* or **SelectorCount** parameter is zero, *CharAttrs* is not referred to. In this case, the parameter address passed by programs written in C or S/390 assembler might be null.

CompCode

Type: MQLONG - output

The completion code:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_CHAR_ATTRS_TOO_SHORT
(2008, X'7D8') Not enough space allowed for character attributes.

MQRC_INT_ATTR_COUNT_TOO_SMALL
(2022, X'7E6') Not enough space allowed for integer attributes.

MQRC_SELECTOR_NOT_FOR_TYPE
(2068, X'814') Selector not applicable to queue type.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CHAR_ATTR_LENGTH_ERROR
(2006, X'7D6') Length of character attributes not valid.

MQRC_CHAR_ATTRS_ERROR
(2007, X'7D7') Character attributes string not valid.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_INT_ATTR_COUNT_ERROR
(2021, X'7E5') Count of integer attributes not valid.

MQRC_INT_ATTRS_ARRAY_ERROR
(2023, X'7E7') Integer attributes array not valid.

MQRC_NOT_OPEN_FOR_INQUIRE
(2038, X'7F6') Queue not open for inquire.

MQRC_OBJECT_CHANGED
(2041, X'7F9') Object definition changed since opened.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_Q_DELETED
(2052, X'804') Queue deleted.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SELECTOR_COUNT_ERROR
(2065, X'811') Count of selectors not valid.

MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.

MQRC_SELECTOR_LIMIT_EXCEEDED
(2066, X'812') Count of selectors too large.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes; see Reason codes

Usage notes

1. The values returned are a snapshot of the selected attributes. There is no guarantee that the attributes remain the same before the application can act upon the returned values.
2. When you open a model queue, a dynamic local queue is created. A dynamic local queue is created even if you open the model queue to inquire about its attributes.

   The attributes of the dynamic queue are largely the same as the attributes of the model queue at the time that the dynamic queue is created. If you then use the MQINQ call on this queue, the queue manager returns the attributes of the dynamic queue, and not the attributes of the model queue. See Table 273 on page 2835 for details of which attributes of the model queue are inherited by the dynamic queue.
3. If the object being inquired is an alias queue, the attribute values returned by the MQINQ call are the attributes of the alias queue. The are not the attributes of the base queue or topic to which the alias resolves.
4. If the object being inquired is a cluster queue, the attributes that can be inquired depend on how the queue is opened:
   - You can open a cluster queue for inquire plus one or more of the input, browse, or set operations. To do so, there must be a local instance of the cluster queue for the open to succeed. In this case, the attributes that can be inquired are the attributes that are valid for local queues.
   - If the cluster queue is opened for inquire alone, or inquire and output, only the attributes listed can be inquired. The **QType** attribute has the value MQQT_CLUSTER in this case:
     - MQCA_Q_DESC
     - MQCA_Q_NAME
     - MQIA_DEF_BIND
     - MQIA_DEF_PERSISTENCE
     - MQIA_DEF_PRIORITY
     - MQIA_INHIBIT_PUT
     - MQIA_Q_TYPE

     You can open the cluster queue with no fixed binding. You can open it with MQOO_BIND_NOT_FIXED specified on the MQOPEN call. Alternatively, specify MQOO_BIND_AS_Q_DEF, and set the **DefBind** attribute of the queue to MQBND_BIND_NOT_FIXED. If you open a cluster queue with no fixed binding, successive MQINQ calls for the queue might inquire different instances of the cluster queue. However, it is typical for all the instances have the same attribute values.
   - An alias queue object can be defined for a cluster. Because TARGTYPE and TARGET are not cluster attributes, the process performing an MQOPEN process on the alias queue is not aware of the object to which the alias resolves.

     During the initial MQOPEN, the alias queue resolves to a queue manager and a queue in the cluster. Name resolution takes place again at the remote queue manager and it is here that the TARGTPYE of the alias queue is resolved.

     If the alias queue resolves to a topic alias, then publication of messages put to the alias queue takes place at this remote queue manager.

   See Cluster queues

5. You might want to inquire a number of attributes, and then set some of them using the MQSET call. To program inquire and set efficiently, position the attributes to be set at the beginning of the selector arrays. If you do so, the same arrays with reduced counts can be used for MQSET.
6. If more than one of the warning situations arise (see the **CompCode** parameter), the reason code returned is the first one in the following list that applies:
   a. MQRC_SELECTOR_NOT_FOR_TYPE
   b. MQRC_INT_ATTR_COUNT_TOO_SMALL
   c. MQRC_CHAR_ATTRS_TOO_SHORT
7. The following topic have information about object attributes:
   - "Attributes for queues" on page 2833
   - "Attributes for namelists" on page 2869
   - "Attributes for process definitions" on page 2872
   - "Attributes for the queue manager" on page 2792

C invocation

```
MQINQ (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
      CharAttrLength, CharAttrs, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;           /* Connection handle */
MQHOBJ   Hobj;            /* Object handle */
MQLONG   SelectorCount;   /* Count of selectors */
MQLONG   Selectors[n];    /* Array of attribute selectors */
MQLONG   IntAttrCount;    /* Count of integer attributes */
MQLONG   IntAttrs[n];     /* Array of integer attributes */
MQLONG   CharAttrLength;  /* Length of character attributes buffer */
MQCHAR   CharAttrs[n];    /* Character attributes */
MQLONG   CompCode;        /* Completion code */
MQLONG   Reason;          /* Reason code qualifying CompCode */
```

COBOL invocation

```
     CALL 'MQINQ' USING HCONN, HOBJ, SELECTORCOUNT, SELECTORS-TABLE,
                        INTATTRCOUNT, INTATTRS-TABLE, CHARATTRLENGTH,
                        CHARATTRS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN            PIC S9(9) BINARY.
**   Object handle
 01  HOBJ             PIC S9(9) BINARY.
**   Count of selectors
 01  SELECTORCOUNT    PIC S9(9) BINARY.
**   Array of attribute selectors
 01  SELECTORS-TABLE.
  02  SELECTORS        PIC S9(9) BINARY OCCURS n TIMES.
**   Count of integer attributes
 01  INTATTRCOUNT     PIC S9(9) BINARY.
**   Array of integer attributes
 01  INTATTRS-TABLE.
  02  INTATTRS         PIC S9(9) BINARY OCCURS n TIMES.
**   Length of character attributes buffer
 01  CHARATTRLENGTH   PIC S9(9) BINARY.
**   Character attributes
 01  CHARATTRS        PIC X(n).
**   Completion code
 01  COMPCODE         PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON           PIC S9(9) BINARY.
```

PL/I invocation

```
call MQINQ (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount,
           IntAttrs, CharAttrLength, CharAttrs, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn           fixed bin(31);  /* Connection handle */
dcl Hobj            fixed bin(31);  /* Object handle */
dcl SelectorCount   fixed bin(31);  /* Count of selectors */
dcl Selectors(n)    fixed bin(31);  /* Array of attribute selectors */
dcl IntAttrCount    fixed bin(31);  /* Count of integer attributes */
dcl IntAttrs(n)     fixed bin(31);  /* Array of integer attributes */
dcl CharAttrLength  fixed bin(31);  /* Length of character attributes
                                       buffer */
dcl CharAttrs       char(n);        /* Character attributes */
dcl CompCode        fixed bin(31);  /* Completion code */
dcl Reason          fixed bin(31);  /* Reason code qualifying
                                       CompCode */
```

High Level Assembler invocation

```
        CALL MQINQ,(HCONN,HOBJ,SELECTORCOUNT,SELECTORS,INTATTRCOUNT,  X
              INTATTRS,CHARATTRLENGTH,CHARATTRS,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN           DS  F      Connection handle
HOBJ            DS  F      Object handle
SELECTORCOUNT   DS  F      Count of selectors
SELECTORS       DS  (n)F   Array of attribute selectors
INTATTRCOUNT    DS  F      Count of integer attributes
INTATTRS        DS  (n)F   Array of integer attributes
CHARATTRLENGTH  DS  F      Length of character attributes buffer
CHARATTRS       DS  CL(n)  Character attributes
COMPCODE        DS  F      Completion code
REASON          DS  F      Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQINQ Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
      CharAttrLength, CharAttrs, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn          As Long   'Connection handle'
Dim Hobj           As Long   'Object handle'
Dim SelectorCount  As Long   'Count of selectors'
Dim Selectors      As Long   'Array of attribute selectors'
Dim IntAttrCount   As Long   'Count of integer attributes'
Dim IntAttrs       As Long   'Array of integer attributes'
Dim CharAttrLength As Long   'Length of character attributes buffer'
Dim CharAttrs      As String 'Character attributes'
Dim CompCode       As Long   'Completion code'
Dim Reason         As Long   'Reason code qualifying CompCode'
```

MQINQMP - Inquire message property:

The MQINQMP call returns the value of a property of a message.

Syntax

MQINQMP (*Hconn*, *Hmsg*, *InqPropOpts*, *Name*, *PropDesc*, *Type*, *ValueLength*, *Value*, *DataLength*, *CompCode*, *Reason*)

Parameters

Hconn
Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* must match the connection handle that was used to create the message handle specified in the **Hmsg** parameter.

If the message handle was created using MQHC_UNASSOCIATED_HCONN then a valid connection must be established on the thread inquiring a property of the message handle otherwise the call fails with MQRC_CONNECTION_BROKEN.

Hmsg
Type: MQHMSG - input

This is the message handle to be inquired. The value was returned by a previous **MQCRTMH** call.

InqPropOpts
Type: MQIMPO - input/output

See the MQIMPO data type for details.

Name
Type: MQCHARV - input/output

The name of the property to inquire.

If no property with this name can be found, the call fails with reason MQRC_PROPERTY_NOT_AVAILABLE.

You can use the wildcard character percent sign (%) at the end of the property name. The wildcard matches zero or more characters, including the period (.) character. This allows an application to inquire the value of many properties. Call MQINQMP with option MQIMPO_INQ_FIRST to get the first matching property and again with the option MQIMPO_INQ_NEXT to get the next matching property. When no more matching properties are available, the call fails with MQRC_PROPERTY_NOT_AVAILABLE. If the *ReturnedName* field of the InqPropOpts structure is initialized with an address or offset for the returned name of the property, this is completed on return from MQINQMP with the name of the property that has been matched. If the *VSBufSize* field of the *ReturnedName* in the InqPropOpts structure is less than the length of the returned property name the completion code is set MQCC_FAILED with reason MQRC_PROPERTY_NAME_TOO_BIG.

Properties that have known synonyms are returned as follows:

1. Properties with the prefix "mqps." are returned as the IBM MQ property name. For example, "MQTopicString" is the returned name rather than "mqps.Top"
2. Properties with the prefix "jms." or "mcd." are returned as the JMS header field name, for example, "JMSExpiration" is the returned name rather than "jms.Exp".
3. Properties with the prefix "usr." are returned without that prefix, for example, "Color" is returned rather than "usr.Color".

Properties with synonyms are only returned once.

In the C programming language, the following macro variables are defined for inquiring on all properties and then all properties that begin "usr.":

MQPROP_INQUIRE_ALL
Inquire on all properties of the message.

MQPROP_INQUIRE_ALL can be used in the following way:

```
MQCHARV Name = {MQPROP_INQUIRE_ALL};
```

MQPROP_INQUIRE_ALL_USR
Inquire on all properties of the message that start "usr.". The returned name is returned without the "usr." prefix.

If MQIMP_INQ_NEXT is specified but Name has changed since the previous call or this is the first call, then MQIMPO_INQ_FIRST is implied.

See Property names and Property name restrictions for further information about the use of property names.

PropDesc
Type: MQPD - output

This structure is used to define the attributes of a property, including what happens if the property is not supported, what message context the property belongs to, and what messages the property should be copied into. See MQPD for details of this structure.

Type
Type: MQLONG - input/output

On return from the MQINQMP call this parameter is set to the data type of *Value*. The data type can be any of the following:

MQTYPE_BOOLEAN
A boolean.

MQTYPE_BYTE_STRING
a byte string.

MQTYPE_INT8
An 8-bit signed integer.

MQTYPE_INT16
A 16-bit signed integer.

MQTYPE_INT32
A 32-bit signed integer.

MQTYPE_INT64
A 64-bit signed integer.

MQTYPE_FLOAT32
A 32-bit floating-point number.

MQTYPE_FLOAT64
A 64-bit floating-point number.

MQTYPE_STRING
A character string.

MQTYPE_NULL
The property exists but has a null value.

If the data type of the property value is not recognized then MQTYPE_STRING is returned and a string representation of the value is placed into the *Value* area. A string representation of the data type can be found in the *TypeString* field of the *InqPropOpts* parameter. A warning completion code is returned with reason MQRC_PROP_TYPE_NOT_SUPPORTED.

Additionally, if the option MQIMPO_CONVERT_TYPE is specified, conversion of the property value is requested. Use *Type* as an input to specify the data type that you want the property to be returned as. See the description of the MQIMPO_CONVERT_TYPE option of the MQIMPO structure for details of data type conversion.

If you do not request type conversion, you can use the following value on input:

MQTYPE_AS_SET
The value of the property is returned without converting its data type.

ValueLength
Type: MQLONG - input

The length in bytes of the Value area. Specify zero for properties that you do not require the value returned for. These could be properties which are designed by an application to have a null value or an empty string. Also specify zero if the MQIMPO_QUERY_LENGTH option has been specified; in this case no value is returned.

Value
Type: MQBYTEx *ValueLength* - output

This is the area to contain the inquired property value. The buffer should be aligned on a boundary appropriate for the value being returned. Failure to do so can result in an error when the value is later accessed.

If *ValueLength* is less than the length of the property value, as much of the property value as possible is moved into *Value* and the call fails with completion code MQCC_FAILED and reason MQRC_PROPERTY_VALUE_TOO_BIG.

The character set of the data in *Value* is given by the ReturnedCCSID field in the InqPropOpts parameter. The encoding of the data in *Value* is given by the ReturnedEncoding field in the InqPropOpts parameter.

In the C programming language, the parameter is declared as a pointer-to-void; the address of any type of data can be specified as the parameter.

If the *ValueLength* parameter is zero, *Value* is not referred to and its value passed by programs written in C or System/390 assembler can be null.

DataLength
Type: MQLONG - output

This is the length in bytes of the actual property value as returned in the *Value* area.

If *DataLength* is less than the property value length, *DataLength* is still filled in on return from the MQINQMP call. This allows the application to determine the size of the buffer required to accommodate the property value, and then reissue the call with a buffer of the appropriate size.

The following values can also be returned.

If the *Type* parameter is set to MQTYPE_STRING or MQTYPE_BYTE_STRING:

MQVL_EMPTY_STRING
The property exists but contains no characters or bytes.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_PROP_NAME_NOT_CONVERTED
(2492, X'09BC') Returned property name not converted.

MQRC_PROP_VALUE_NOT_CONVERTED
(2466, X'09A2') Property value not converted.

MQRC_PROP_TYPE_NOT_SUPPORTED
(2467, X'09A3') Property data type is not supported.

MQRC_RFH_FORMAT_ERROR
(2421, X'0975') An MQRFH2 folder containing properties could not be parsed.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'089C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'0852') Unable to load adapter service module.

MQRC_ASID_MISMATCH
(2157, X'086D') Primary and home ASIDs differ.

MQRC_BUFFER_ERROR
(2004, X'07D4') Value parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'07D5') Value length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'08AB') MQI call entered before previous call completed.

MQRC_CONNECTION_BROKEN
(2009, X'07D9') Connection to queue manager lost.

MQRC_DATA_LENGTH_ERROR
(2010, X'07DA') Data length parameter not valid.

MQRC_IMPO_ERROR
(2464, X'09A0') Inquire message property options structure not valid.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle not valid.

MQRC_MSG_HANDLE_IN_USE
(2499, X'09C3') Message handle already in use.

MQRC_OPTIONS_ERROR
(2046, X'07F8') Options not valid or not consistent.

MQRC_PD_ERROR
(2482, X'09B2') Property descriptor structure not valid.

MQRC_PROP_CONV_NOT_SUPPORTED
(2470, X'09A6') Conversion from the actual to requested data type not supported.

MQRC_PROPERTY_NAME_ERROR
(2442, X'098A') Invalid property name.

MQRC_PROPERTY_NAME_TOO_BIG
(2465, X'09A1') Property name too large for returned name buffer.

MQRC_PROPERTY_NOT_AVAILABLE
(2471, X'09A7) Property not available.

MQRC_PROPERTY_VALUE_TOO_BIG
(2469, X'09A5') Property value too large for the Value area.

MQRC_PROP_NUMBER_FORMAT_ERROR
(2472, X'09A8') Number format error encountered in value data.

MQRC_PROPERTY_TYPE_ERROR
(2473, X'09A9') Invalid requested property type.

MQRC_SOURCE_CCSID_ERROR
(2111, X'083F') Property name coded character set identifier not valid.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'0871') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'0893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

C invocation

```
MQINQMP (Hconn, Hmsg, &InqPropOpts, &Name, &PropDesc, &Type,
ValueLength, Value, &DataLength, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;       /* Connection handle */
MQHMSG  Hmsg;        /* Message handle */
MQIMPO  InqPropOpts; /* Options that control the action of MQINQMP */
MQCHARV Name;        /* Property name */
MQPD    PropDesc;    /* Property descriptor */
MQLONG  Type;        /* Property data type */
MQLONG  ValueLength; /* Length in bytes of the Value area */
MQBYTE  Value[n];    /* Area to contain the property value */
MQLONG  DataLength;  /* Length of the property value */
MQLONG  CompCode;    /* Completion code */
MQLONG  Reason;      /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQINQMP' USING HCONN, HMSG, INQMSGOPTS, NAME, PROPDESC, TYPE,
VALUELENGTH, VALUE, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN       PIC S9(9) BINARY.
**   Message handle
 01  HMSG      PIC S9(18) BINARY.
**   Options that control the action of MQINQMP
 01  INQMSGOPTS.
     COPY CMQIMPOV.
**   Property name
 01  NAME.
     COPY CMQCHRVV.
```

```
**   Property descriptor
 01  PROPDESC.
     COPY CMQPDV.
**   Property data type
 01  TYPE        PIC S9(9) BINARY.
**   Length in bytes of the VALUE area
 01  VALUELENGTH PIC S9(9) BINARY.
**   Area to contain the property value
 01  VALUE       PIC X(n).
**   Length of the property value
 01  DATALENGTH  PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE    PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON      PIC S9(9) BINARY.
```

PL/I invocation

```
call MQINQMP (Hconn, Hmsg, InqPropOpts, Name, PropDesc, Type,
ValueLength, Value, DataLength, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn       fixed bin(31);  /* Connection handle */
dcl Hmsg        fixed bin(63);  /* Message handle */
dcl InqPropOpts like MQIMPO;    /* Options that control the action of MQINQMP */
dcl Name        like MQCHARV;   /* Property name */
dcl PropDesc    like MQPD;      /* Property descriptor */
dcl Type        fixed bin (31); /* Property data type */
dcl ValueLength fixed bin (31); /* Length in bytes of the Value area */
dcl Value       char (n);       /* Area to contain the property value */
dcl DataLength  fixed bin (31); /* Length of the property value */
dcl CompCode    fixed bin (31); /* Completion code */
dcl Reason      fixed bin (31); /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQINQMP,(HCONN,HMSG,INQMSGOPTS,NAME,PROPDESC,TYPE,
VALUELENGTH,VALUE,DATALENGTH,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN        DS         F     Connection handle
HMSG         DS         D     Message handle
INQMSGOPTS   CMQIMPOA   ,     Options that control the action of MQINQMP
NAME         CMQCHRVA   ,     Property name
PROPDESC     CMQPDA     ,     Property descriptor
TYPE         DS         F     Property data type
VALUELENGTH  DS         F     Length in bytes of the VALUE area
VALUE        DS         CL(n) Area to contain the property value
DATALENGTH   DS         F     Length of the property value
COMPCODE     DS         F     Completion code
REASON       DS         F     Reason code qualifying COMPCODE
```

MQMHBUF - Convert message handle into buffer:

The MQMHBUF call converts a message handle into a buffer and is the inverse of the MQBUFMH call.

Syntax

MQMHBUF (*Hconn*, *Hmsg*, *MsgHBufOpts*, *Name*, *MsgDesc*, *BufferLength*, *Buffer*, *DataLength*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* must match the connection handle that was used to create the message handle specified in the **Hmsg** parameter.

If the message handle was created using MQHC_UNASSOCIATED_HCONN, a valid connection must be established on the thread deleting the message handle. If a valid connection is not established, the call fails with MQRC_CONNECTION_BROKEN.

Hmsg

Type: MQHMSG - input

This is the message handle for which a buffer is required. The value was returned by a previous MQCRTMH call.

MsgHBufOpts

Type: MQMHBO - input

The MQMHBO structure allows applications to specify options that control how buffers are produced from message handles.

See “MQMHBO - Message handle to buffer options” on page 2450 for details.

Name

Type: MQCHARV - input

The name of the property or properties to put into the buffer.

If no property matching the name can be found, the call fails with MQRC_PROPERTY_NOT_AVAILABLE.

You can use a wildcard to put more than one property into the buffer. To do this, use the wildcard character '%' at the end of the property name. This wildcard matches zero or more characters, including the '.' character.

In the C programming language, the following macro variables are defined for inquiring on all properties and all properties that begin 'usr':

MQPROP_INQUIRE_ALL
Put all properties of the message into the buffer

MQPROP_INQUIRE_ALL_USR
Put all properties of the message that start with the characters 'usr.' into the buffer.

See Property names and Property name restrictions for further information about the use of property names.

MsgDesc

Type: MQMD - input/output

The *MsgDesc* structure describes the contents of the buffer area.

On output, the *Encoding*, *CodedCharSetId* and *Format* fields are set to correctly describe the encoding, character set identifier, and format of the data in the buffer area as written by the call.

Data in this structure is in the character set and encoding of the application.

BufferLength
Type: MQLONG - input

BufferLength is the length of the Buffer area, in bytes.

Buffer
Type: MQBYTExBufferLength - output

Buffer defines the area to contain the message properties. You must align the buffer on a 4-byte boundary.

If *BufferLength* is less than the length required to store the properties in *Buffer*, MQMHBUF fails with MQRC_PROPERTY_VALUE_TOO_BIG.

The contents of the buffer can change even if the call fails.

DataLength
Type: MQLONG - output

DataLength is the length, in bytes, of the returned properties in the buffer. If the value is zero, no properties matched the value given in *Name* and the call fails with reason code MQRC_PROPERTY_NOT_AVAILABLE.

If *BufferLength* is less than the length required to store the properties in the buffer, the MQMHBUF call fails with MQRC_PROPERTY_VALUE_TOO_BIG, but a value is still entered into *DataLength*. This allows the application to determine the size of the buffer required to accommodate the properties, and then reissue the call with the required *BufferLength*.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'089C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_MHBO_ERROR
(2501, X'095C') Message handle to buffer options structure not valid.

MQRC_BUFFER_ERROR
(2004, X'07D4') Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'07D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'08AB') MQI call entered before previous call completed.

MQRC_CONNECTION_BROKEN
(2009, X'07D9') Connection to queue manager lost.

MQRC_DATA_LENGTH_ERROR
(2010, X'07DA') Data length parameter not valid.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle not valid.

MQRC_MD_ERROR
(2026, X'07EA') Message descriptor not valid.

MQRC_MSG_HANDLE_IN_USE
(2499, X'09C3') Message handle already in use.

MQRC_OPTIONS_ERROR
(2046, X'07FE') Options not valid or not consistent.

MQRC_PROPERTY_NAME_ERROR
(2442, X'098A') Property name is not valid.

MQRC_PROPERTY_NOT_AVAILABLE
(2471, X'09A7') Property not available.

MQRC_PROPERTY_VALUE_TOO_BIG
(2469, X'09A5') BufferLength value is too small to contain specified properties.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

C invocation

```
MQMHBUF (Hconn, Hmsg, &MsgHBufOpts, &Name, &MsgDesc, BufferLength, Buffer,
         &DataLength, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;        /* Connection handle */
MQHMSG  Hmsg;         /* Message handle */
MQMHBO  MsgHBufOpts;  /* Options that control the action of MQMHBUF */
MQCHARV Name;         /* Property name */
MQMD    MsgDesc;      /* Message descriptor */
MQLONG  BufferLength; /* Length in bytes of the Buffer area */
MQBYTE  Buffer[n];    /* Area to contain the properties */
MQLONG  DataLength;   /* Length of the properties */
MQLONG  CompCode;     /* Completion code */
MQLONG  Reason;       /* Reason code qualifying CompCode */
```

Usage notes

MQMHBUF converts a message handle into a buffer.

You can use it with an MQGET API exit to access certain properties, using the message property APIs, and then pass these in a buffer back to an application designed to use MQRFH2 headers rather than message handles.

This call is the inverse of the MQBUFMH call, which you can use to parse message properties from a buffer into a message handle.

COBOL invocation

```
     CALL 'MQMHBUF' USING HCONN, HMSG, MSGHBUFOPTS, NAME, MSGDESC,
                          BUFFERLENGTH, BUFFER, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN         PIC S9(9) BINARY.
**   Message handle
 01  HMSG          PIC S9(18) BINARY.
**   Options that control the action of MQMHBUF
 01  MSGHBUFOPTS.
     COPY CMQMHBOV.
**   Property name
 01  NAME
     COPY CMQCHRVV.
**   Message descriptor
 01  MSGDESC
     COPY CMQMDV.
**   Length in bytes of the Buffer area */
 01  BUFFERLENGTH  PIC S9(9) BINARY.
**   Area to contain the properties
 01  BUFFER        PIC X(n).
**   Length of the properties
 01  DATALENGTH    PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE      PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON        PIC S9(9) BINARY.
```

PL/I invocation

```
call MQMHBUF (Hconn, Hmsg, MsgHBufOpts, Name, MsgDesc, BufferLength, Buffer,
DataLength, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn        fixed bin(31); /* Connection handle */
dcl Hmsg         fixed bin(63); /* Message handle */
dcl MsgHBufOpts  like MQMHBO;   /* Options that control the action of MQMHBUF */
dcl Name         like MQCHARV;  /* Property name */
dcl MsgDesc      like MQMD;     /* Message descriptor */
dcl BufferLength fixed bin(31); /* Length in bytes of the Buffer area */
dcl Buffer       char(n);       /* Area to contain the properties */
dcl DataLength   fixed bin(31); /* Length of the properties */
dcl CompCode     fixed bin(31); /* Completion code */
dcl Reason       fixed bin(31); /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQMHBUF,(HCONN,HMSG,MSGHBUFOPTS,NAME,MSGDESC,BUFFERLENGTH,
               BUFFER,DATALENGTH,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN        DS       F      Connection handle
HMSG         DS       D      Message handle
MSGHBUFOPTS  CMQMHBOA ,      Options that control the action of MQMHBUF
NAME         CMQCHRVA ,      Property name
MSGDESC      CMQMDA   ,      Message descriptor
BUFFERLENGTH DS       F      Length in bytes of the BUFFER area
BUFFER       DS       CL(n)  Area to contain the properties
```

```
DATALENGTH   DS       F      Length of the properties
COMPCODE     DS       F      Completion code
REASON       DS       F      Reason code qualifying COMPCODE
```

MQOPEN - Open object:

The MQOPEN call establishes access to an object.

The following types of object are valid:

- Queue (including distribution lists)
- Namelist
- Process definition
- Queue manager
- Topic

Syntax

MQOPEN (*Hconn*, *ObjDesc*, *Options*, *Hobj*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

ObjDesc

Type: MQOD - input/output

This is a structure that identifies the object to be opened; see "MQOD - Object descriptor" on page 2453 for details.

If the *ObjectName* field in the **ObjDesc** parameter is the name of a model queue, a dynamic local queue is created with the attributes of the model queue; this happens whatever options you specify on the **Options** parameter. Subsequent operations using the *Hobj* returned by the MQOPEN call are performed on the new dynamic queue, and not on the model queue. This is true even for the MQINQ and MQSET calls. The name of the model queue in the **ObjDesc** parameter is replaced with the name of the dynamic queue created. The type of the dynamic queue is determined by the value of the **DefinitionType** attribute of the model queue (see "Attributes for queues" on page 2833). For information about the close options applicable to dynamic queues, see the description of the MQCLOSE call.

Options

Type: MQLONG - input

You must specify at least one of the following options:

- MQOO_BROWSE
- MQOO_INPUT_* (only one of these)
- MQOO_INQUIRE
- MQOO_OUTPUT
- MQOO_SET

- MQOO_BIND_* (only one of these)

See the following table for details of these options; other options can be specified as required. To specify more than one option, either add the values together (do not add the same constant more than once), or combine the values using the bitwise OR operation (if the programming language supports bit operations). Combinations that are not valid are noted; all other combinations are valid. Only options that are applicable to the type of object specified by *ObjDesc* are allowed. The following table shows valid MQOPEN options for queries and topics.

Option	Alias [1]	Local and Model	Remote	Nonlocal Cluster	Distribution list	Topic
MQOO_INPUT_AS_Q_DEF	Yes	Yes	No	No	No	No
MQOO_INPUT_SHARED	Yes	Yes	No	No	No	No
MQOO_INPUT_EXCLUSIVE	Yes	Yes	No	No	No	No
MQOO_OUTPUT	Yes	Yes	Yes	Yes	Yes	Yes
MQOO_BROWSE	Yes	Yes	No	No	No	No
MQOO_CO_OP	Yes	Yes	No	No	No	No
MQOO_INQUIRE	Yes	Yes	[2]	Yes	No	No
MQOO_SET	Yes	Yes	[2]	No	No	No
MQOO_BIND_ON_OPEN [3]	Yes	Yes	Yes	Yes	Yes	No
MQOO_BIND_NOT_FIXED [3]	Yes	Yes	Yes	Yes	Yes	No
MQOO_BIND_ON_GROUP [3]	Yes	Yes	Yes	Yes	Yes	No
MQOO_BIND_AS_Q_DEF [3]	Yes	Yes	Yes	Yes	Yes	No
MQOO_SAVE_ALL_CONTEXT	Yes	Yes	No	No	No	No
MQOO_PASS_IDENTITY_CONTEXT	Yes	Yes	Yes	Yes	Yes	[4]
MQOO_PASS_ALL_CONTEXT	Yes	Yes	Yes	Yes	Yes	Yes
MQOO_SET_IDENTITY_CONTEXT	Yes	Yes	Yes	Yes	Yes	[4]
MQOO_SET_ALL_CONTEXT	Yes	Yes	Yes	Yes	Yes	Yes
MQOO_NO_READ_AHEAD	Yes	Yes	No	No	No	No
MQOO_READ_AHEAD	Yes	Yes	No	No	No	No
MQOO_READ_AHEAD_AS_Q_DEF	Yes	Yes	No	No	No	No
MQOO_ALTERNATE_USER_AUTHORITY	Yes	Yes	Yes	Yes	Yes	Yes
MQOO_FAIL_IF_QUIESCING	Yes	Yes	Yes	Yes	Yes	Yes
MQOO_RESOLVE_LOCAL_Q	Yes	Yes	Yes	Yes	No	No
MQOO_RESOLVE_LOCAL_TOPIC	No	No	No	No	No	Yes
MQOO_NO_MULTICAST	No	No	No	No	No	Yes

Notes:

1. The validity of options for aliases depends on the validity of the option for the queue to which the alias resolves.
2. This option is valid only for the local definition of a remote queue.
3. This option can be specified for any queue type, but is ignored if the queue is not a cluster queue. However, the **DefBind** queue attribute overrides the base queue even when the alias queue is not in a cluster.
4. These attributes can be used with a topic, but affect only the context set for the retained message, not the context fields sent to any subscriber.

Access options: The following options control the type of operations that can be performed on the object:

MQOO_INPUT_AS_Q_DEF
Open queue to get messages using queue-defined default.

The queue is opened for use with subsequent MQGET calls. The type of access is either shared or exclusive, depending on the value of the **DefInputOpenOption** queue attribute; see "Attributes for queues" on page 2833 for details.

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues.

MQOO_INPUT_SHARED
Open queue to get messages with shared access.

The queue is opened for use with subsequent MQGET calls. The call can succeed if the queue is currently open by this or another application with MQOO_INPUT_SHARED, but fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open with MQOO_INPUT_EXCLUSIVE.

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues.

MQOO_INPUT_EXCLUSIVE
Open queue to get messages with exclusive access.

The queue is opened for use with subsequent MQGET calls. The call fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open by this or another application for input of any type (MQOO_INPUT_SHARED or MQOO_INPUT_EXCLUSIVE).

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues.

MQOO_OUTPUT

Open queue to put messages, or a topic or topic string to publish messages.

The queue or topic is opened for use with subsequent MQPUT calls.

An MQOPEN call with this option can succeed even if the **InhibitPut** queue attribute is set to MQQA_PUT_INHIBITED (although subsequent MQPUT calls fail while the attribute is set to this value).

This option is valid for all types of queue, including distribution lists, and topics.

The following notes apply to these options:

- Only one of these options can be specified.
- An MQOPEN call with one of these options can succeed even if the **InhibitGet** queue attribute is set to MQQA_GET_INHIBITED (although subsequent MQGET calls fail while the attribute is set to this value).
- If the queue is defined as not being shareable (that is, the **Shareability** queue attribute has the value MQQA_NOT_SHAREABLE), attempts to open the queue for shared access are treated as attempts to open the queue with exclusive access.
- If an alias queue is opened with one of these options, the test for exclusive use (or for whether another application has exclusive use) is against the base queue to which the alias resolves.
- These options are not valid if **ObjectQMgrName** is the name of a queue manager alias; this is true even if the value of the **RemoteQMgrName** attribute in the local definition of a remote queue used for queue manager aliasing is the name of the local queue manager.

MQOO_BROWSE
Open queue to browse messages.

The queue is opened for use with subsequent MQGET calls with one of the following options:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_NEXT
- MQGMO_BROWSE_MSG_UNDER_CURSOR

This is allowed even if the queue is currently open for MQOO_INPUT_EXCLUSIVE. An MQOPEN call with the MQOO_BROWSE option establishes a browse cursor, and positions it logically before the first message on the queue; see MQGMO - Options field for further information.

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues. It is also not valid if *ObjectQMgrName* is the name of a queue manager alias; this is true even if the value of the **RemoteQMgrName** attribute in the local definition of a remote queue used for queue manager aliasing is the name of the local queue manager.

MQOO_CO_OP
Open as a cooperating member of the set of handles.

This option is valid only with the MQOO_BROWSE option. If it is specified without MQOO_BROWSE, MQOPEN returns with MQRC_OPTIONS_ERROR.

The handle returned is considered to be a member of a cooperating set of handles for subsequent MQGET calls with one of the following options:

- MQGMO_MARK_BROWSE_CO_OP
- MQGMO_UNMARKED_BROWSE_MSG
- MQGMO_UNMARK_BROWSE_CO_OP

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues.

MQOO_INQUIRE
Open object to inquire attributes.

The queue, namelist, process definition, or queue manager is opened for use with subsequent MQINQ calls.

This option is valid for all types of object other than distribution lists. It is not valid if *ObjectQMgrName* is the name of a queue manager alias; this is true even if the value of the **RemoteQMgrName** attribute in the local definition of a remote queue used for queue manager aliasing is the name of the local queue manager.

MQOO_SET
Open queue to set attributes.

The queue is opened for use with subsequent MQSET calls.

This option is valid for all types of queue other than distribution lists. It is not valid if *ObjectQMgrName* is the name of a local definition of a remote queue; this is true even if the value of the **RemoteQMgrName** attribute in the local definition of a remote queue used for queue manager aliasing is the name of the local queue manager.

Binding options: The following options apply when the object being opened is a cluster queue; these options control the binding of the queue handle to an instance of the cluster queue:

MQOO_BIND_ON_OPEN
The local queue manager binds the queue handle to an instance of the destination queue when the queue is opened. As a result, all messages put using this handle are sent to the same instance of the destination queue, and by the same route.

This option is valid only for queues, and affects only cluster queues. If specified for a queue that is not a cluster queue, the option is ignored.

MQOO_BIND_NOT_FIXED
This stops the local queue manager binding the queue handle to an instance of the destination queue. As a result, successive MQPUT calls using this handle send the messages to *different* instances of the destination queue, or to the same instance but by different routes. It also allows the instance selected to be changed later by the local queue manager, by a remote queue manager, or by a message channel agent (MCA), according to network conditions.

Note: Client and server applications that need to exchange a *series* of messages to complete a transaction must not use MQOO_BIND_NOT_FIXED (or MQOO_BIND_AS_Q_DEF when *DefBind* has the value MQBND_BIND_NOT_FIXED), because successive messages in the series might be sent to different instances of the server application.

If MQOO_BROWSE or one of the MQOO_INPUT_* options is specified for a cluster queue, the queue manager is forced to select the local instance of the cluster queue. As a result, the binding of the queue handle is fixed, even if MQOO_BIND_NOT_FIXED is specified.

If MQOO_INQUIRE is specified with MQOO_BIND_NOT_FIXED, successive MQINQ calls using that handle might inquire different instances of the cluster queue, although typically all the instances have the same attribute values.

MQOO_BIND_NOT_FIXED is valid only for queues, and affects only cluster queues. If specified for a queue that is not a cluster queue, the option is ignored.

MQOO_BIND_ON_GROUP
Allows an application to request that a group of messages are all allocated to the same destination instance.

This option is valid only for queues, and affects only cluster queues. If specified for a queue that is not a cluster queue, the option is ignored.

MQOO_BIND_AS_Q_DEF
The local queue manager binds the queue handle in the way defined by the **DefBind** queue attribute. The value of this attribute is either MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXED, or MQBND_BIND_ON_GROUP.

MQOO_BIND_AS_Q_DEF is the default when MQOO_BIND_ON_OPEN, MQOO_BIND_NOT_FIXED, or MQOO_BIND_ON_GROUP is not specified.

MQOO_BIND_AS_Q_DEF aids program documentation. It is not intended that this option is used with either of the other two bind options, but because its value is zero such use cannot be detected.

Context options: The following options control the processing of message context:

MQOO_SAVE_ALL_CONTEXT
Context information is associated with this queue handle. This information is set from the context of any message retrieved using this handle. For more information about message context, see Message context and Controlling context information.

This context information can be passed to a message that is then put on a queue using the MQPUT or MQPUT1 calls. See the MQPMO_PASS_IDENTITY_CONTEXT and MQPMO_PASS_ALL_CONTEXT options described in "MQPMO - Put-message options" on page 2477.

Until a message has been successfully retrieved, context cannot be passed to a message being put on a queue.

A message retrieved using one of the MQGMO_BROWSE_* browse options does not have its context information saved (although the context fields in the **MsgDesc** parameter are set after a browse).

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues. One of the MQOO_INPUT_* options must be specified.

MQOO_PASS_IDENTITY_CONTEXT
This allows the MQPMO_PASS_IDENTITY_CONTEXT option to be specified in the **PutMsgOpts** parameter when a message is put on a queue; this gives the message the identity context information from an input queue that was opened with the MQOO_SAVE_ALL_CONTEXT option. For more information about message context, see Message context and Controlling context information.

The MQOO_OUTPUT option must be specified.

This option is valid for all types of queue, including distribution lists.

MQOO_PASS_ALL_CONTEXT
This allows the MQPMO_PASS_ALL_CONTEXT option to be specified in the **PutMsgOpts** parameter when a message is put on a queue; this gives the message the identity and origin context information from an input queue that was opened with the MQOO_SAVE_ALL_CONTEXT option. For more information about message context, see Message context and Controlling context information.

This option implies MQOO_PASS_IDENTITY_CONTEXT, which need not therefore be specified. The MQOO_OUTPUT option must be specified.

This option is valid for all types of queue, including distribution lists.

MQOO_SET_IDENTITY_CONTEXT
This allows the MQPMO_SET_IDENTITY_CONTEXT option to be specified in the **PutMsgOpts** parameter when a message is put on a queue; this gives the message the identity context information contained in the **MsgDesc** parameter specified on the MQPUT or MQPUT1 call. For more information about message context, see Message context and Controlling context information.

This option implies MQOO_PASS_IDENTITY_CONTEXT, which need not therefore be specified. The MQOO_OUTPUT option must be specified.

This option is valid for all types of queue, including distribution lists.

MQOO_SET_ALL_CONTEXT
This allows the MQPMO_SET_ALL_CONTEXT option to be specified in the **PutMsgOpts** parameter when a message is put on a queue; this gives the message the identity and origin context information contained in the **MsgDesc** parameter specified on the MQPUT or MQPUT1 call. For more information about message context, see Message context and Controlling context information.

This option implies the following options, which need not therefore be specified:

- MQOO_PASS_IDENTITY_CONTEXT
- MQOO_PASS_ALL_CONTEXT
- MQOO_SET_IDENTITY_CONTEXT

The MQOO_OUTPUT option must be specified.

This option is valid for all types of queue, including distribution lists.

Read ahead options:

When you call MQOPEN with MQOO_READ_AHEAD, the IBM MQ client only enables read-ahead if certain conditions are met. These conditions include:

- Both the client and remote queue manager must be at WebSphere MQ Version 7 or later.
- The client application must be compiled and linked against the threaded IBM MQ MQI client libraries.

- The client channel must be using TCP/IP protocol
- The channel must have a non-zero SharingConversations (SHARECNV) setting in both the client and server channel definitions.

The following options control whether non-persistent messages are sent to the client before an application requests them. The following notes apply to the read ahead options:

- Only one of these options can be specified.
- These options are valid only for local, alias, and model queues. They are not valid for remote queues, distribution lists, topics or queue managers.
- These options are only applicable when one of MQOO_BROWSE, MQOO_INPUT_SHARED and MQOO_INPUT_EXCLUSIVE are also specified although it is not an error to specify these options with MQOO_INQUIRE or MQOO_SET.
- If the application is not running as an IBM MQ client, these options are ignored.

MQOO_NO_READ_AHEAD
Non-persistent messages are not sent the client before an application requests them.

MQOO_READ_AHEAD
Non-persistent messages are sent to the client before an application requests them.

MQOO_READ_AHEAD_AS_Q_DEF
Read ahead behavior is determined by the default read ahead attribute of the queue being opened. This is the default value.

Other options: The following options control authorization checking, what happens when the queue manager is quiescing, whether to resolve the local queue name, and multicast:

MQOO_ALTERNATE_USER_AUTHORITY
The *AlternateUserId* field in the **ObjDesc** parameter contains a user identifier to use to validate this MQOPEN call. The call can succeed only if this *AlternateUserId* is authorized to open the object with the specified access options, regardless of whether the user identifier under which the application is running is authorized to do so. This does not apply to any context options specified, however, which are always checked against the user identifier under which the application is running.

This option is valid for all types of object.

MQOO_FAIL_IF_QUIESCING
The MQOPEN call fails if the queue manager is in quiescing state.

On z/OS, for a CICS or IMS application, this option also forces the MQOPEN call to fail if the connection is in quiescing state.

This option is valid for all types of object.

For information about client channels see Overview of IBM MQ MQI clients.

MQOO_RESOLVE_LOCAL_Q
Fill the ResolvedQName in the MQOD structure with the name of the local queue that was opened. Similarly, the ResolvedQMgrName is filled with the name of the local queue manager hosting the local queue. If the MQOD structure is less than Version 3, MQOO_RESOLVE_LOCAL_Q is ignored with no error being returned.

The local queue is always returned when either a local, alias, or model queue is opened, but this is not the case when, for example, a remote queue or a non-local cluster queue is opened without the MQOO_RESOLVE_LOCAL_Q option; the ResolvedQName and ResolvedQMgrName are filled with the RemoteQName and RemoteQMgrName found in the remote queue definition, or similarly with the chosen remote cluster queue.

If you specify MQOO_RESOLVE_LOCAL_Q when opening, for example, a remote queue, ResolvedQName is the transmission queue to which messages are put. The ResolvedQMgrName is filled with the name of the local queue manager hosting the transmission queue.

If you are authorized for browse, input, or output on a queue, you have the required authority to specify this flag on the MQOPEN call. No special authority is needed.

This option is valid only for queues and queue managers.

MQOO_RESOLVE_LOCAL_TOPIC
Fill the ResolvedQName in the MQOD structure with the name of the administrative topic opened.

MQOO_NO_MULTICAST
Publication messages are not sent using multicast.

This option is valid only with the MQOO_OUTPUT option. If it is specified without MQOO_OUTPUT, MQOPEN returns with MQRC_OPTIONS_ERROR.

This option is valid only for a topic.

Hobj
Type: MQHOBJ - output

This handle represents the access that has been established to the object. It must be specified on subsequent IBM MQ calls that operate on the object. It ceases to be valid when the MQCLOSE call is issued, or when the unit of processing that defines the scope of the handle terminates.

The scope of the object handle returned is the same as the scope of the connection handle specified on the call. See MQCONN - Hconn parameter for information about handle scope.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_MULTIPLE_REASONS
(2136, X'858') Multiple reason codes returned.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ALIAS_BASE_Q_TYPE_ERROR
(2001, X'7D1') Alias base queue not a valid type.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_NOT_AVAILABLE
(2345, X'929') Coupling facility not available.

MQRC_CF_STRUC_AUTH_FAILED
(2348, X'92C') Coupling-facility structure authorization check failed.

MQRC_CF_STRUC_ERROR
(2349, X'92D') Coupling-facility structure not valid.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CF_STRUC_LIST_HDR_IN_USE
(2347, X'92B') Coupling-facility structure list-header in use.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CLUSTER_EXIT_ERROR
(2266, X'8DA') Cluster workload exit failed.

MQRC_CLUSTER_PUT_INHIBITED
(2268, X'8DC') Put calls inhibited for all queues in cluster.

MQRC_CLUSTER_RESOLUTION_ERROR
(2189, X'88D') Cluster name resolution failed.

MQRC_CLUSTER_RESOURCE_ERROR
(2269, X'8DD') Cluster resource error.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_DB2_NOT_AVAILABLE
(2342, X'926') Db2 subsystem not available.

MQRC_DEF_XMIT_Q_TYPE_ERROR
(2198, X'896') Default transmission queue not local.

MQRC_DEF_XMIT_Q_USAGE_ERROR
(2199, X'897') Default transmission queue usage error.

MQRC_DYNAMIC_Q_NAME_ERROR
(2011, X'7DB') Name of dynamic queue not valid.

MQRC_HANDLE_NOT_AVAILABLE
(2017, X'7E1') No more handles available.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_MULTIPLE_REASONS
(2136, X'858') Multiple reason codes returned.

MQRC_NAME_IN_USE
(2201, X'899') Name in use.

MQRC_NAME_NOT_VALID_FOR_TYPE
(2194, X'892') Object name not valid for object type.

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_OBJECT_ALREADY_EXISTS
(2100, X'834') Object exists.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OBJECT_IN_USE
(2042, X'7FA') Object already open with conflicting options.

MQRC_OBJECT_LEVEL_INCOMPATIBLE
(2360, X'938') Object level not compatible.

MQRC_OBJECT_NAME_ERROR
(2152, X'868') Object name not valid.

MQRC_OBJECT_NOT_UNIQUE
(2343, X'927') Object not unique.

MQRC_OBJECT_Q_MGR_NAME_ERROR
(2153, X'869') Object queue manager name not valid.

MQRC_OBJECT_RECORDS_ERROR
(2155, X'86B') Object records not valid.

MQRC_OBJECT_STRING_ERROR
(2441, X'0989') Objectstring field not valid

MQRC_OBJECT_TYPE_ERROR
(2043, X'7FB') Object type not valid.

MQRC_OD_ERROR
(2044, X'7FC') Object descriptor structure not valid.

MQRC_OPTION_NOT_VALID_FOR_TYPE
(2045, X'7FD') Option not valid for object type.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_PAGESET_FULL
(2192, X'890') External storage medium is full.

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_Q_TYPE_ERROR
(2057, X'809') Queue type not valid.

MQRC_RECS_PRESENT_ERROR
(2154, X'86A') Number of records present not valid.

MQRC_REMOTE_Q_NAME_ERROR
(2184, X'888') Remote queue name not valid.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_RESPONSE_RECORDS_ERROR
(2156, X'86C') Response records not valid.

MQRC_SECURITY_ERROR
(2063, X'80F') Security error occurred.

MQRC_SELECTOR_SYNTAX_ERROR
2459 (X'099B') An MQOPEN, MQPUT1 or MQSUB call was issued but a selection string was specified which contained a syntax error.

MQRC_STOPPED_BY_CLUSTER_EXIT
(2188, X'88C') Call rejected by cluster workload exit.

MQRC_STORAGE_MEDIUM_FULL
(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

MQRC_UNKNOWN_ALIAS_BASE_Q
(2082, X'822') Unknown alias base queue.

MQRC_UNKNOWN_DEF_XMIT_Q
(2197, X'895') Unknown default transmission queue.

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRC_UNKNOWN_OBJECT_Q_MGR
(2086, X'826') Unknown object queue manager.

MQRC_UNKNOWN_REMOTE_Q_MGR
(2087, X'827') Unknown remote queue manager.

MQRC_UNKNOWN_XMIT_Q
(2196, X'894') Unknown transmission queue.

MQRC_WRONG_CF_LEVEL
(2366, X'93E') Coupling-facility structure is wrong level.

MQRC_XMIT_Q_TYPE_ERROR
(2091, X'82B') Transmission queue not local.

MQRC_XMIT_Q_USAGE_ERROR
(2092, X'82C') Transmission queue with wrong usage.

For detailed information about these codes, see:

- z/OS IBM MQ for z/OS messages, completion, and reason codes for IBM MQ for z/OS.
- Reason codes for all other IBM MQ platforms except z/OS.

General usage notes

1. The object opened is one of the following:
   - A queue to:
     - Get or browse messages (using the MQGET call)
     - Put messages (using the MQPUT call)
     - Inquire about the attributes of the queue (using the MQINQ call)
     - Set the attributes of the queue (using the MQSET call)

     If the queue named is a model queue, a dynamic local queue is created. See the **ObjDesc** parameter described in "MQOPEN - Open object" on page 2725.

     A distribution list is a special type of queue object that contains a list of queues. It can be opened to put messages, but not to get or browse messages, or to inquire or set attributes. See usage note 8 for further details.

     A queue that has `QSGDISP(GROUP)` is a special type of queue definition that cannot be used with the MQOPEN or MQPUT1 calls.
   - A namelist to inquire about the names of the queues in the list (using the MQINQ call).
   - A process definition to inquire about the process attributes (using the MQINQ call).
   - The queue manager to inquire about the attributes of the local queue manager (using the MQINQ call).
   - A topic to publish a message (using the MQPUT call)
2. An application can open the same object more than once. A different object handle is returned for each open. Each handle that is returned can be used for the functions for which the corresponding open was performed.
3. If the object being opened is a queue other than a cluster queue, all name resolution within the local queue manager takes place at the time of the MQOPEN call. This can include:
   - Resolution of the name of a local definition of a remote queue to the name of the remote queue manager, and the name by which the queue is known at the remote queue manager
   - Resolution of the remote queue manager name to the name of a local transmission queue
   - (z/OS only) Resolution of the remote queue manager name to the name of the shared transmission queue used by the IGQ agent (applies only if the local and remote queue managers belong to the same queue-sharing group)

- Alias resolution to the name of a base queue or a topic object.

However, be aware that subsequent MQINQ or MQSET calls for the handle relate solely to the name that has been opened, and not to the object resulting after name resolution has occurred. For example, if the object opened is an alias, the attributes returned by the MQINQ call are the attributes of the alias, not the attributes of the base queue or a topic object to which the alias resolves.

If the object being opened is a cluster queue, name resolution can occur at the time of the MQOPEN call, or be deferred until later. The point at which resolution occurs is controlled by the MQOO_BIND_* options specified on the MQOPEN call:

- MQOO_BIND_ON_OPEN
- MQOO_BIND_NOT_FIXED
- MQOO_BIND_AS_Q_DEF
- MQOO_BIND_ON_GROUP

See Name resolution for more information about name resolution for cluster queues.

4. An MQOPEN call with the MQOO_BROWSE option establishes a browse cursor, for use with MQGET calls that specify the object handle and one of the browse options. This allows the queue to be scanned without altering its contents. A message that has been found by browsing can be removed from the queue by using the MQGMO_MSG_UNDER_CURSOR option.

   Multiple browse cursors can be active for a single application by issuing several MQOPEN requests for the same queue.
5. Applications started by a trigger monitor are passed the name of the queue that is associated with the application when the application is started. This queue name can be specified in the **ObjDesc** parameter to open the queue. See “MQTMC2 - Trigger message 2 (character format)” on page 2596 for further details.

Read ahead options

When you call MQOPEN with MQOO_READ_AHEAD, the IBM MQ client only enables read-ahead if certain conditions are met. These conditions include:

- Both the client and remote queue manager must be at WebSphere MQ Version 7 or later.
- The client application must be compiled and linked against the threaded IBM MQ MQI client libraries.
- The client channel must be using TCP/IP protocol
- The channel must have a non-zero SharingConversations (SHARECNV) setting in both the client and server channel definitions.

The following notes apply to the use of read ahead options.

1. The read ahead options are applicable only when one, and only one, of the MQOO_BROWSE, MQOO_INPUT_SHARED and MQOO_INPUT_EXCLUSIVE options are also specified. An error is not thrown if a read ahead options are specified with the MQOO_ INQUIRE or MQOO_SET options.
2. Read ahead is not enabled when requested if the options used on the first MQGET call are not supported for use with read ahead. Also, read ahead is disabled when the client is connecting to a queue manager that does not support read ahead.
3. If the application is not running as an IBM MQ client, read ahead options are ignored.

Cluster queues

The following notes apply to the use of cluster queues.

1. When a cluster queue is opened for the first time, and the local queue manager is not a full repository queue manager, the local queue manager obtains information about the cluster queue from a full repository queue manager. When the network is busy, it can take several seconds for the local queue manager to receive the needed information from the repository queue manager. As a result, the application issuing the MQOPEN call might have to wait for up to 10 seconds before control returns

from the MQOPEN call. If the local queue manager does not receive the needed information about the cluster queue within this time, the call fails with reason code MQRC_CLUSTER_RESOLUTION_ERROR.

2. When a cluster queue is opened and there are multiple instances of the queue in the cluster, the instance opened depends on the options specified on the MQOPEN call:
   - If the options specified include any of the following:
     – MQOO_BROWSE
     – MQOO_INPUT_AS_Q_DEF
     – MQOO_INPUT_EXCLUSIVE
     – MQOO_INPUT_SHARED
     – MQOO_SET

     the instance of the cluster queue opened must be the local instance. If there is no local instance of the queue, the MQOPEN call fails.
   - If the options specified include none of the options described previously, but include one or both of the following:
     – MQOO_INQUIRE
     – MQOO_OUTPUT

     the instance opened is the local instance if there is one, and a remote instance otherwise (if using the CLWLUSEQ defaults). The instance chosen by the queue manager can, however, be altered by a cluster workload exit (if there is one).
3. If there is a subscription for the queue, but it is not acknowledged by a full repository, the object is not present in the cluster and the call fails with reason code MQRC_OBJECT_NAME.

For more information about cluster queues, see Cluster queues.

Distribution lists

The following notes apply to the use of distribution lists.

Distribution lists are supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ MQI clients connected to these systems.

1. Fields in the MQOD structure must be set as follows when opening a distribution list:
   - *Version* must be MQOD_VERSION_2 or greater.
   - *ObjectType* must be MQOT_Q.
   - *ObjectName* must be blank or the null string.
   - *ObjectQMgrName* must be blank or the null string.
   - *RecsPresent* must be greater than zero.
   - One of *ObjectRecOffset* and *ObjectRecPtr* must be zero and the other nonzero.
   - No more than one of *ResponseRecOffset* and *ResponseRecPtr* can be nonzero.
   - There must be *RecsPresent* object records, addressed by either *ObjectRecOffset* or *ObjectRecPtr*. The object records must be set to the names of the destination queues to be opened.
   - If one of *ResponseRecOffset* and *ResponseRecPtr* is nonzero, there must be *RecsPresent* response records present. They are set by the queue manager if the call completes with reason code MQRC_MULTIPLE_REASONS.

   A version-2 MQOD can also be used to open a single queue that is not in a distribution list, by ensuring that *RecsPresent* is zero.
2. Only the following open options are valid in the **Options** parameter:
   - MQOO_OUTPUT
   - MQOO_PASS_*_CONTEXT

- MQOO_SET_*_CONTEXT
- MQOO_ALTERNATE_USER_AUTHORITY
- MQOO_FAIL_IF_QUIESCING

3. The destination queues in the distribution list can be local, alias, or remote queues, but they cannot be model queues. If a model queue is specified, that queue fails to open, with reason code MQRC_Q_TYPE_ERROR. However, this does not prevent other queues in the list being opened successfully.
4. The completion code and reason code parameters are set as follows:
   - If the open operations for the queues in the distribution list all succeed or fail in the same way, the completion code and reason code parameters are set to describe the common result. The MQRR response records (if provided by the application) are not set in this case.

     For example, if every open succeeds, the completion code is set to MQCC_OK and the reason code is set to MQRC_NONE; if every open fails because none of the queues exists, the parameters are set to MQCC_FAILED and MQRC_UNKNOWN_OBJECT_NAME.
   - If the open operations for the queues in the distribution list do not all succeed or fail in the same way:
     – The completion code parameter is set to MQCC_WARNING if at least one open succeeded, and to MQCC_FAILED if all failed.
     – The reason code parameter is set to MQRC_MULTIPLE_REASONS.
     – The response records (if provided by the application) are set to the individual completion codes and reason codes for the queues in the distribution list.
5. When a distribution list has been opened successfully, the handle *Hobj* returned by the call can be used on subsequent MQPUT calls to put messages to queues in the distribution list, and on an MQCLOSE call to relinquish access to the distribution list. The only valid close option for a distribution list is MQCO_NONE.

   The MQPUT1 call can also be used to put a message to a distribution list; the MQOD structure defining the queues in the list is specified as a parameter on that call.
6. Each successfully opened destination in the distribution list counts as a separate handle when checking whether the application has exceeded the permitted maximum number of handles (see the **MaxHandles** queue manager attribute). This is true even when two or more of the destinations in the distribution list resolve to the same physical queue. If the MQOPEN or MQPUT1 call for a distribution list would cause the number of handles in use by the application to exceed *MaxHandles*, the call fails with reason code MQRC_HANDLE_NOT_AVAILABLE.
7. Each destination that is opened successfully has the value of its **OpenOutputCount** attribute incremented by one. If two or more of the destinations in the distribution list resolve to the same physical queue, that queue has its **OpenOutputCount** attribute incremented by the number of destinations in the distribution list that resolve to that queue.
8. Any change to the queue definitions that would have caused a handle to become invalid had the queues been opened individually (for example, a change in the resolution path), does not cause the distribution-list handle to become invalid. However, it does result in a failure for that particular queue when the distribution-list handle is used on a subsequent MQPUT call.
9. A distribution list can contain only one destination.

Remote queues

The following notes apply to the use of remote queues.

A remote queue can be specified in one of two ways in the **ObjDesc** parameter of this call.

- By specifying for *ObjectName* the name of a local definition of the remote queue. In this case, *ObjectQMgrName* refers to the local queue manager, and can be specified as blanks or (in the C programming language) a null string.

The security validation performed by the local queue manager verifies that the user is authorized to open the local definition of the remote queue.

- By specifying for *ObjectName* the name of the remote queue as known to the remote queue manager. In this case, *ObjectQMgrName* is the name of the remote queue manager.

  The security validation performed by the local queue manager verifies that the user is authorized to send messages to the transmission queue resulting from the name resolution process.

In either case:

- No messages are sent by the local queue manager to the remote queue manager to check that the user is authorized to put messages on the queue.
- When a message arrives at the remote queue manager, the remote queue manager might reject it because the user originating the message is not authorized.

See the *ObjectName* and *ObjectQMgrName* fields described in "MQOD - Object descriptor" on page 2453 for more information.

Objects

Security

The following notes relate to the security aspects of using MQOPEN.

The queue manager performs security checks when an MQOPEN call is issued, to verify that the user identifier under which the application is running has the appropriate level of authority before access is permitted. The authority check is made on the name of the object being opened, and not on the name, or names, resulting after a name has been resolved.

If the object being opened is an alias queue which points at a topic object, the queue manager performs a security check on the alias queue name, before performing a security check for the topic as if the topic object had been used directly.

If the object being opened is a topic object, whether with *ObjectName* alone or by using the *ObjectString* (with or without a basing *ObjectName*), the queue manager performs the security check by using the resultant topic string, taken from within the topic object specified in *ObjectName*, and if required concatenating it with that provided in *ObjectString*, and then finding the closest topic object at or above that point in the topic tree to perform the security check against. This might not be the same topic object that was specified in *ObjectName*.

If the object being opened is a model queue, the queue manager performs a full security check against both the name of the model queue and the name of the dynamic queue that is created. If the resulting dynamic queue is then opened explicitly, a further resource security check is performed against the name of the dynamic queue.

On z/OS, the queue manager performs security checks only if security is enabled. For more information about security checking, see Setting up security on z/OS .

Attributes

The following notes relate to attributes.

The attributes of an object can change while an application has the object open. In many cases, the application does not notice this, but for certain attributes the queue manager marks the handle as no longer valid. These attributes are:

- Any attribute that affects the name resolution of the object. This applies regardless of the open options used, and includes the following:

- A change to the **BaseQName** attribute of an alias queue that is open.
- A change to the **TargetType** attribute of an alias queue that is open.
- A change to the **RemoteQName** or **RemoteQMgrName** queue attributes, for any handle that is open for this queue, or for a queue that resolves through this definition as a queue manager alias.
- Any change that causes a currently open handle for a remote queue to resolve to a different transmission queue, or to fail to resolve to one at all. For example, this can include:
  - A change to the **XmitQName** attribute of the local definition of a remote queue, whether the definition is being used for a queue, or for a queue manager alias.
  - (z/OS only) A change to the value of the **IntraGroupqueuing** queue manager attribute, or a change in the definition of the shared transmission queue (SYSTEM.QSG.TRANSMIT.QUEUE) used by the IGQ agent.

  There is one exception to this: the creation of a new transmission queue. A handle that would have resolved to this queue had it been present when the handle was opened, but instead resolved to the default transmission queue, is not made invalid.
- A change to the **DefXmitQName** queue manager attribute. In this case all open handles that resolved to the previously named queue (that resolved to it only because it was the default transmission queue) are marked as invalid. Handles that resolved to this queue for other reasons are not affected.

- The **Shareability** queue attribute, if there are two or more handles that are currently providing MQOO_INPUT_SHARED access for this queue, or for a queue that resolves to this queue. If so, *all* handles that are open for this queue, or for a queue that resolves to this queue, are marked as invalid, regardless of the open options.

  On z/OS, the handles previously described are marked as invalid if one or more handles is currently providing MQOO_INPUT_SHARED or MQOO_INPUT_EXCLUSIVE access to the queue.
- The **Usage** queue attribute, for all handles that are open for this queue, or for a queue that resolves to this queue, regardless of the open options.

When a handle is marked as invalid, all subsequent calls (other than MQCLOSE) using this handle fail with reason code MQRC_OBJECT_CHANGED. The application must issue an MQCLOSE call (using the original handle) and then reopen the queue. Any uncommitted updates against the old handle from previous successful calls can still be committed or backed out, as required by the application logic.

If changing an attribute causes this to happen, use a special force version of the call.

C invocation

```
MQOPEN (Hconn, &ObjDesc, Options, &Hobj, &CompCode,
       &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;     /* Connection handle */
MQOD     ObjDesc;   /* Object descriptor */
MQLONG   Options;   /* Options that control the action of MQOPEN */
MQHOBJ   Hobj;      /* Object handle */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQOPEN' USING HCONN, OBJDESC, OPTIONS, HOBJ, COMPCODE, REASON
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN     PIC S9(9) BINARY.
**   Object descriptor
 01  OBJDESC.
     COPY CMQODV.
**   Options that control the action of MQOPEN
```

```
 01  OPTIONS   PIC S9(9) BINARY.
**   Object handle
 01  HOBJ      PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE  PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON    PIC S9(9) BINARY.
```

PL/I invocation

```
call MQOPEN (Hconn, ObjDesc, Options, Hobj, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn     fixed bin(31);  /* Connection handle */
dcl ObjDesc   like MQOD;      /* Object descriptor */
dcl Options   fixed bin(31);  /* Options that control the action of
                                 MQOPEN */
dcl Hobj      fixed bin(31);  /* Object handle */
dcl CompCode  fixed bin(31);  /* Completion code */
dcl Reason    fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQOPEN,(HCONN,OBJDESC,OPTIONS,HOBJ,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN     DS      F  Connection handle
OBJDESC   CMQODA  ,  Object descriptor
OPTIONS   DS      F  Options that control the action of MQOPEN
HOBJ      DS      F  Object handle
COMPCODE  DS      F  Completion code
REASON    DS      F  Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQOPEN Hconn, ObjDesc, Options, Hobj, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn    As Long 'Connection handle'
Dim ObjDesc  As MQOD 'Object descriptor'
Dim Options  As Long 'Options that control the action of MQOPEN'
Dim Hobj     As Long 'Object handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

MQPUT - Put message:

The MQPUT call puts a message on a queue or distribution list, or to a topic. The queue, distribution list, or topic must already be open.

Syntax

MQPUT (*Hconn*, *Hobj*, *MsgDesc*, *PutMsgOpts*, *BufferLength*, *Buffer*, *CompCode*, *Reason*)

Parameters

Hconn
Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hobj
Type: MQHOBJ - input

This handle represents the queue to which the message is added, or the topic to which the message is published. The value of *Hobj* was returned by a previous MQOPEN call that specified the MQOO_OUTPUT option.

MsgDesc
Type: MQMD - input/output

This structure describes the attributes of the message being sent, and receives information about the message after the put request is complete. See "MQMD - Message descriptor" on page 2387 for details.

If the application provides a version-1 MQMD, the message data can be prefixed with an MQMDE structure to specify values for the fields that exist in the version-2 MQMD but not the version-1. The *Format* field in the MQMD must be set to MQFMT_MD_EXTENSION to indicate that an MQMDE is present. See "MQMDE - Message descriptor extension" on page 2442 for more details.

The application does not need to provide an MQMD structure if a valid message handle is supplied in the *OriginalMsgHandle* or *NewMsgHandle* fields of the MQPMO structure. If nothing is provided in one of these fields, the descriptor of the message is taken from the descriptor associated with the message handles.

If you use, or plan to use, API exits then we recommend that you explicitly supply an MQMD structure and do not use the message descriptors associated with the message handles. This is because the API Exit associated with MQPUT or MQPUT1 call is unable to ascertain which MQMD values are used by the queue manager to complete the MQPUT or MQPUT1 request.

PutMsgOpts
Type: MQPMO - input/output

See "MQPMO - Put-message options" on page 2477 for details.

BufferLength
Type: MQLONG - input

The length of the message in *Buffer*. Zero is valid, and indicates that the message contains no application data. The upper limit for *BufferLength* depends on various factors:

- If the destination is a local queue or resolves to a local queue, the upper limit depends on whether:

– The local queue manager supports segmentation.
– The sending application specifies the flag that allows the queue manager to segment the message. This flag is MQMF_SEGMENTATION_ALLOWED, and can be specified either in a version-2 MQMD, or in an MQMDE used with a version-1 MQMD.

If both of these conditions are satisfied, *BufferLength* cannot exceed 999 999 999 minus the value of the *Offset* field in MQMD. The longest logical message that can be put is therefore 999 999 999 bytes (when *Offset* is zero). However, resource constraints imposed by the operating system or environment in which the application is running might result in a lower limit.

If one or both of the previous conditions is not satisfied, *BufferLength* cannot exceed the smaller of the queue's **MaxMsgLength** attribute and queue manager's **MaxMsgLength** attribute.

- If the destination is a remote queue or resolves to a remote queue, the conditions for local queues apply, *but at each queue manager through which the message must pass in order to reach the destination queue* ; in particular:
  1. The local transmission queue used to store the message temporarily at the local queue manager
  2. Intermediate transmission queues (if any) used to store the message at queue managers on the route between the local and destination queue managers
  3. The destination queue at the destination queue manager

  The longest message that can be put is therefore governed by the most restrictive of these queues and queue managers.

  When a message is on a transmission queue, additional information resides with the message data, and this reduces the amount of application data that can be carried. In this situation, subtract MQ_MSG_HEADER_LENGTH bytes from the *MaxMsgLength* values of the transmission queues when determining the limit for *BufferLength*.

  Note: Only failure to comply with condition 1 can be diagnosed synchronously (with reason code MQRC_MSG_TOO_BIG_FOR_Q or MQRC_MSG_TOO_BIG_FOR_Q_MGR) when the message is put. If conditions 2 or 3 are not satisfied, the message is redirected to a dead-letter (undelivered-message) queue, either at an intermediate queue manager or at the destination queue manager. If this happens, a report message is generated if one was requested by the sender.

Buffer

Type: MQBYTExBufferLength - input

This is a buffer containing the application data to be sent. The buffer must be aligned on a boundary appropriate to the nature of the data in the message. 4-byte alignment is suitable for most messages (including messages containing IBM MQ header structures), but some messages might require more stringent alignment. For example, a message containing a 64-bit binary integer might require 8-byte alignment.

If *Buffer* contains character or numeric data, set the *CodedCharSetId* and *Encoding* fields in the **MsgDesc** parameter to the values appropriate to the data; this enables the receiver of the message to convert the data (if necessary) to the character set and encoding used by the receiver.

Note: All the other parameters on the MQPUT call must be in the character set and encoding of the local queue manager (given by the **CodedCharSetId** queue manager attribute and MQENC_NATIVE).

In the C programming language, the parameter is declared as a pointer-to-void; the address of any type of data can be specified as the parameter.

If the **BufferLength** parameter is zero, *Buffer* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler can be null.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_INCOMPLETE_GROUP
(2241, X'8C1') Message group not complete.

MQRC_INCOMPLETE_MSG
(2242, X'8C2') Logical message not complete.

MQRC_INCONSISTENT_PERSISTENCE
(2185, X'889') Inconsistent persistence specification.

MQRC_INCONSISTENT_UOW
(2245, X'8C5') Inconsistent unit-of-work specification.

MQRC_MULTIPLE_REASONS
(2136, X'858') Multiple reason codes returned.

MQRC_PRIORITY_EXCEEDS_MAXIMUM
(2049, X'801') Message Priority exceeds maximum value supported.

MQRC_UNKNOWN_REPORT_OPTION
(2104, X'838') Report option(s) in message descriptor not recognized.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ALIAS_TARGTYPE_CHANGED
(2480, X'09B0') Subscription target type has changed from queue to topic object.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BACKED_OUT
(2003, X'7D3') Unit of work backed out.

MQRC_BUFFER_ERROR
(2004, X'7D4') Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CALL_INTERRUPTED
(2549, X'9F5') MQPUT or MQCMIT was interrupted and reconnection processing cannot reestablish a definite outcome.

MQRC_CF_NOT_AVAILABLE
(2345, X'929') Coupling facility not available.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CFGR_ERROR
(2416, X'970') PCF group parameter structure MQCFGR in the message data is not valid.

MQRC_CFH_ERROR
(2235, X'8BB') PCF header structure not valid.

MQRC_CFIF_ERROR
(2414, X'96E') PCF integer filter parameter structure in the message data is not valid.

MQRC_CFIL_ERROR
(2236, X'8BC') PCF integer list parameter structure or PCIF*64 integer list parameter structure not valid.

MQRC_CFIN_ERROR
(2237, X'8BD') PCF integer parameter structure or PCIF*64 integer parameter structure not valid.

MQRC_CFSF_ERROR
(2415, X'96F') PCF string filter parameter structure in the message data is not valid.

MQRC_CFSL_ERROR
(2238, X'8BE') PCF string list parameter structure not valid.

MQRC_CFST_ERROR
(2239, X'8BF') PCF string parameter structure not valid.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CLUSTER_EXIT_ERROR
(2266, X'8DA') Cluster workload exit failed.

MQRC_CLUSTER_RESOLUTION_ERROR
(2189, X'88D') Cluster name resolution failed.

MQRC_CLUSTER_RESOURCE_ERROR
(2269, X'8DD') Cluster resource error.

MQRC_COD_NOT_VALID_FOR_XCF_Q
(2106, X'83A') COD report option not valid for XCF queue.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_CONTENT_ERROR
2554 (X'09FA') Message content could not be parsed to determine whether the message should be delivered to a subscriber with an extended message selector.

MQRC_CONTEXT_HANDLE_ERROR
(2097, X'831') Queue handle referred to does not save context.

MQRC_CONTEXT_NOT_AVAILABLE
(2098, X'832') Context not available for queue handle referred to.

MQRC_DATA_LENGTH_ERROR
(2010, X'7DA') Data length parameter not valid.

MQRC_DH_ERROR
(2135, X'857') Distribution header structure not valid.

MQRC_DLH_ERROR
(2141, X'85D') Dead letter header structure not valid.

MQRC_EPH_ERROR
(2420, X'974') Embedded PCF structure not valid.

MQRC_EXPIRY_ERROR
(2013, X'7DD') Expiry time not valid.

MQRC_FEEDBACK_ERROR
(2014, X'7DE') Feedback code not valid.

MQRC_GLOBAL_UOW_CONFLICT
(2351, X'92F') Global units of work conflict.

MQRC_GROUP_ID_ERROR
(2258, X'8D2') Group identifier not valid.

MQRC_HANDLE_IN_USE_FOR_UOW
(2353, X'931') Handle in use for global unit of work.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HEADER_ERROR
(2142, X'85E') MQ header structure not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_IIH_ERROR
(2148, X'864') IMS information header structure not valid.

MQRC_INCOMPLETE_GROUP
(2241, X'8C1') Message group not complete.

MQRC_INCOMPLETE_MSG
(2242, X'8C2') Logical message not complete.

MQRC_INCONSISTENT_PERSISTENCE
(2185, X'889') Inconsistent persistence specification.

MQRC_INCONSISTENT_UOW
(2245, X'8C5') Inconsistent unit-of-work specification.

MQRC_LOCAL_UOW_CONFLICT
(2352, X'930') Global unit of work conflicts with local unit of work.

MQRC_MD_ERROR
(2026, X'7EA') Message descriptor not valid.

MQRC_MDE_ERROR
(2248, X'8C8') Message descriptor extension not valid.

MQRC_MISSING_REPLY_TO_Q
(2027, X'7EB') Missing reply-to queue or MQPMO_SUPPRESS_REPLYTO was used

MQRC_MISSING_WIH
(2332, X'91C') Message data does not begin with MQWIH.

MQRC_MSG_FLAGS_ERROR
(2249, X'8C9') Message flags not valid.

MQRC_MSG_SEQ_NUMBER_ERROR
(2250, X'8CA') Message sequence number not valid.

MQRC_MSG_TOO_BIG_FOR_Q
(2030, X'7EE') Message length greater than maximum for queue.

MQRC_MSG_TOO_BIG_FOR_Q_MGR
(2031, X'7EF') Message length greater than maximum for queue manager.

MQRC_MSG_TYPE_ERROR
(2029, X'7ED') Message type in message descriptor not valid.

MQRC_MULTIPLE_REASONS
(2136, X'858') Multiple reason codes returned.

MQRC_NO_DESTINATIONS_AVAILABLE
(2270, X'8DE') No destination queues available.

MQRC_NOT_OPEN_FOR_OUTPUT
(2039, X'7F7') Queue not open for output.

MQRC_NOT_OPEN_FOR_PASS_ALL
(2093, X'82D') Queue not open for pass all context.

MQRC_NOT_OPEN_FOR_PASS_IDENT
(2094, X'82E') Queue not open for pass identity context.

MQRC_NOT_OPEN_FOR_SET_ALL
(2095, X'82F') Queue not open for set all context.

MQRC_NOT_OPEN_FOR_SET_IDENT
(2096, X'830') Queue not open for set identity context.

MQRC_OBJECT_CHANGED
(2041, X'7F9') Object definition changed since opened.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OFFSET_ERROR
(2251, X'8CB') Message segment offset not valid.

MQRC_OPEN_FAILED
(2137, X'859') Object not opened successfully.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_ORIGINAL_LENGTH_ERROR
(2252, X'8CC') Original length not valid.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_PAGESET_FULL
(2192, X'890') External storage medium is full.

MQRC_PCF_ERROR
(2149, X'865') PCF structures not valid.

MQRC_PERSISTENCE_ERROR
(2047, X'7FF') Persistence not valid.

MQRC_PERSISTENT_NOT_ALLOWED
(2048, X'800') Queue does not support persistent messages.

MQRC_PMO_ERROR
(2173, X'87D') Put-message options structure not valid.

MQRC_PMO_RECORD_FLAGS_ERROR
(2158, X'86E') Put message record flags not valid.

MQRC_PRIORITY_ERROR
(2050, X'802') Message priority not valid.

MQRC_PUBLICATION_FAILURE
(2502, X'9C6') The publication has not been delivered to any of the subscribers.

MQRC_PUT_INHIBITED
(2051, X'803') Put calls inhibited for the queue, for the queue to which this queue resolves, or the topic.

MQRC_PUT_MSG_RECORDS_ERROR
(2159, X'86F') Put message records not valid.

MQRC_PUT_NOT_RETAINED
(2479, X'09AF') Publication could not be retained

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_FULL
(2053, X'805') Queue already contains maximum number of messages.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_Q_SPACE_NOT_AVAILABLE
(2056, X'808') No space available on disk for queue.

MQRC_RECONNECT_FAILED
(2548, X'9F4') After reconnecting, an error occurred reinstating the handles for a reconnectable connection.

MQRC_RECS_PRESENT_ERROR
(2154, X'86A') Number of records present not valid.

MQRC_REPORT_OPTIONS_ERROR
(2061, X'80D') Report options in message descriptor not valid.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_RESPONSE_RECORDS_ERROR
(2156, X'86C') Response records not valid.

MQRC_RFH_ERROR
(2334, X'91E') MQRFH or MQRFH2 structure not valid.

MQRC_RMH_ERROR
(2220, X'8AC') Reference message header structure not valid.

MQRC_SEGMENT_LENGTH_ZERO
(2253, X'8CD') Length of data in message segment is zero.

MQRC_SEGMENTS_NOT_SUPPORTED
(2365, X'93D') Segments not supported.

MQRC_SELECTION_NOT_AVAILABLE
2551 (X'09F7') A possible subscriber for the publication exists, but the queue manager cannot check whether to send the publication to the subscriber.

MQRC_STOPPED_BY_CLUSTER_EXIT
(2188, X'88C') Call rejected by cluster workload exit.

MQRC_STORAGE_CLASS_ERROR
(2105, X'839') Storage class error.

MQRC_STORAGE_MEDIUM_FULL
(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_SYNCPOINT_LIMIT_REACHED
(2024, X'7E8') No more messages can be handled within current unit of work.

MQRC_SYNCPOINT_NOT_AVAILABLE
(2072, X'818') Syncpoint support not available.

MQRC_TM_ERROR
(2265, X'8D9') Trigger message structure not valid.

MQRC_TMC_ERROR
(2191, X'88F') Character trigger message structure not valid.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

MQRC_UOW_ENLISTMENT_ERROR
(2354, X'932') Enlistment in global unit of work failed.

MQRC_UOW_MIX_NOT_SUPPORTED
(2355, X'933') Mixture of unit-of-work calls not supported.

MQRC_UOW_NOT_AVAILABLE
(2255, X'8CF') Unit of work not available for the queue manager to use.

MQRC_WIH_ERROR
(2333, X'91D') MQWIH structure not valid.

MQRC_WRONG_MD_VERSION
(2257, X'8D1') Wrong version of MQMD supplied.

MQRC_XQH_ERROR
(2260, X'8D4') Transmission queue header structure not valid.

For detailed information about these codes, see Reason codes.

Topic usage notes

1. The following notes apply to the use of topics:
   a. When using MQPUT to publish messages on a topic, where one or more subscribers to that topic cannot be given the publication due to a problem with their subscriber queue (for example it is full), the Reason code returned to the MQPUT call and the delivery behavior is dependent on the setting of the PMSGDLV or NPMSGDLV attributes on the TOPIC. Note delivery of a publication to the dead letter queue when MQRO_DEAD_LETTER_Q is specified, or discarding the message when MQRO_DISCARD_MSG is specified, is considered as a successful delivery of the message. If none of the publications are delivered, the MQPUT returns with MQRC_PUBLICATION_FAILURE. This can happen in the following cases:
      - A message is published to a TOPIC with PMSGDLV or NPMSGDLV (depending on the persistence of the message) set to ALL and any subscription (durable or not) has a queue which cannot receive the publication.
      - A message is published to a TOPIC with PMSGDLV or NPMSGDLV (depending on the persistence of the message) set to ALLDUR and a durable subscription has a queue which cannot receive the publication.

      The MQPUT can return with MQRC_NONE even though publications could not be delivered to some subscribers in the following cases:
      - A message is published to a TOPIC with PMSGDLV or NPMSGDLV (depending on the persistence of the message) set to ALLAVAIL and any subscription, durable or not, has a queue which cannot receive the publication.
      - A message is published to a TOPIC with PMSGDLV or NPMSGDLV (depending on the persistence of the message) set to ALLDUR and a non-durable subscription has a queue which cannot receive the publication.

      You can use the USEDLQ topic attribute to determine whether the dead-letter queue is used when publication messages cannot be delivered to their correct subscriber queue. For more information about the use of USEDLQ, see DEFINE TOPIC.
   b. If there are no subscribers to the topic being used, the message published is not sent to any queue and is discarded. It does not matter whether the message is persistent or non-persistent, or whether it has unlimited expiry or has an expiry time, it is still discarded if there are no subscribers. The exception to this is if the message is to be retained, in which case, although it is not sent to any subscribers' queues, it is stored against the topic to be delivered to any new subscriptions or to any subscribers that ask for retained publications using MQSUBRQ.

MQPUT and MQPUT1

You can use both the MQPUT and MQPUT1 calls to put messages on a queue; which call to use depends on the circumstances

- Use the MQPUT call to place multiple messages on the *same* queue.

  An MQOPEN call specifying the MQOO_OUTPUT option is issued first, followed by one or more MQPUT requests to add messages to the queue; finally the queue is closed with an MQCLOSE call. This gives better performance than repeated use of the MQPUT1 call.
- Use the MQPUT1 call to put only *one* message on a queue.

This call encapsulates the MQOPEN, MQPUT, and MQCLOSE calls into a single call, minimizing the number of calls that must be issued.

Destination Queues

The following notes apply to the use of destination queues:

1. If an application puts a sequence of messages on the same queue without using message groups, the order of those messages is preserved if the conditions detailed are satisfied. Some conditions apply to both local and remote destination queues; other conditions apply only to remote destination queues.

   Conditions that apply to local and remote destination queues

   - All the MQPUT calls are within the same unit of work, or none of them is within a unit of work.

     Be aware that when messages are put onto a particular queue within a single unit of work, messages from other applications might be interspersed with the sequence of messages on the queue.
   - All the MQPUT calls are made using the same object handle *Hobj*.

     In some environments, message sequence is also preserved when different object handles are used, if the calls are made from the same application. The meaning of *same application* is determined by the environment:
     - On z/OS, the application is:
       - For CICS, the CICS task
       - For IMS, the task
       - For z/OS batch, the task
     - On IBM i, the application is the job.
     - On Windows and UNIX, the application is the thread.
   - The messages all have the same priority.
   - The messages are not put to a cluster queue with MQOO_BIND_NOT_FIXED specified (or with MQOO_BIND_AS_Q_DEF in effect when the DefBind queue attribute has the value MQBND_BIND_NOT_FIXED).

   Additional conditions that apply to remote destination queues

   - There is only one path from the sending queue manager to the destination queue manager.

     If some messages in the sequence might go on a different path (for example, because of reconfiguration, traffic balancing, or path selection based on message size), the order of the messages at the destination queue manager cannot be guaranteed.
   - Messages are not placed temporarily on dead-letter queues at the sending, intermediate, or destination queue managers.

     If one or more of the messages is put temporarily on a dead-letter queue (for example, because a transmission queue or the destination queue is temporarily full), the messages can arrive on the destination queue out of sequence.
   - The messages are either all persistent or all nonpersistent.

     If a channel on the route between the sending and destination queue managers has its **NonPersistentMsgSpeed** attribute set to MQNPMS_FAST, nonpersistent messages can jump ahead of persistent messages, resulting in the order of persistent messages relative to nonpersistent messages not being preserved. However, the order of persistent messages relative to each other, and of nonpersistent messages relative to each other, is preserved.

   If these conditions are not satisfied, you can use message groups to preserve message order, but this requires both the sending and receiving applications to use the message-grouping support. For more information about message groups, see:

   - MQMD - MsgFlags field
   - MQPMO_LOGICAL_ORDER
   - MQGMO_LOGICAL_ORDER

Distribution Lists

The following notes apply to the use of distribution lists.

Distribution lists are supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ MQI clients connected to these systems.

1. You can put messages to a distribution list using either a version-1 or a version-2 MQPMO. If you use a version-1 MQPMO (or a version-2 MQPMO with *RecsPresent* equal to zero), the application can provide no put message records or response records. You cannot identify the queues that encounter errors if the message is sent successfully to some queues in the distribution list and not others.

   If the application provides put message records or response records, set the *Version* field to MQPMO_VERSION_2.

   You can also use a version-2 MQPMO to send messages to a single queue that is not in a distribution list, by ensuring that *RecsPresent* is zero.
2. The completion code and reason code parameters are set as follows:
   - If the puts to the queues in the distribution list all succeed or fail in the same way, the completion code and reason code parameters are set to describe the common result. The MQRR response records (if provided by the application) are not set in this case.

     For example, if every put succeeds, the completion code and reason code are set to MQCC_OK and MQRC_NONE; if every put fails because all the queues are inhibited for puts, the parameters are set to MQCC_FAILED and MQRC_PUT_INHIBITED.
   - If the puts to the queues in the distribution list do not all succeed or fail in the same way:
     – The completion code parameter is set to MQCC_WARNING if at least one put succeeded, and to MQCC_FAILED if all failed.
     – The reason code parameter is set to MQRC_MULTIPLE_REASONS.
     – The response records (if provided by the application) are set to the individual completion codes and reason codes for the queues in the distribution list.

     If the put to a destination fails because the open for that destination failed, the fields in the response record are set to MQCC_FAILED and MQRC_OPEN_FAILED; that destination is included in *InvalidDestCount*.
3. If a destination in the distribution list resolves to a local queue, the message is placed on that queue in normal form (that is, not as a distribution-list message). If more than one destination resolves to the same local queue, one message is placed on the queue for each such destination.

   If a destination in the distribution list resolves to a remote queue, a message is placed on the appropriate transmission queue. Where several destinations resolve to the same transmission queue, a single distribution-list message containing those destinations can be placed on the transmission queue, even if those destinations were not adjacent in the list of destinations provided by the application. However, this can be done only if the transmission queue supports distribution-list messages (see DistLists).

   If the transmission queue does not support distribution lists, one copy of the message in normal form is placed on the transmission queue for each destination that uses that transmission queue.

   If a distribution list with the application message data is too large for a transmission queue, the distribution list message is split into smaller distribution-list messages, each containing fewer destinations. If the application message data only just fits on the queue, distribution-list messages cannot be used at all, and the queue manager generates one copy of the message in normal form for each destination that uses that transmission queue.

   If different destinations have different message priority or message persistence (this can occur when the application specifies MQPRI_PRIORITY_AS_Q_DEF or MQPER_PERSISTENCE_AS_Q_DEF), the messages are not held in the same distribution-list message. Instead, the queue manager generates as many distribution-list messages as are necessary to accommodate the differing priority and persistence values.
4. A put to a distribution list can result in:

- A single distribution-list message, or
- A number of smaller distribution-list messages, or
- A mixture of distribution list messages and normal messages, or
- Normal messages only.

Which of the above occurs depends on whether:

- The destinations in the list are local, remote, or a mixture.
- The destinations have the same message priority and message persistence.
- The transmission queues can hold distribution-list messages.
- The transmission queues' maximum message lengths are large enough to accommodate the message in distribution-list form.

However, regardless of which of the above occurs, each *physical* message resulting (that is, each normal message or distribution-list message resulting from the put) counts as only *one* message when:

- Checking whether the application has exceeded the permitted maximum number of messages in a unit of work (see the **MaxUncommittedMsgs** queue manager attribute).
- Checking whether the triggering conditions are satisfied.
- Incrementing queue depths and checking whether the queues' maximum queue depth would be exceeded.

5. Any change to the queue definitions that would have caused a handle to become invalid had the queues been opened individually (for example, a change in the resolution path), does not cause the distribution-list handle to become invalid. However, it does result in a failure for that particular queue when the distribution-list handle is used on a subsequent MQPUT call.

Headers

If a message is put with one or more IBM MQ header structures at the beginning of the application message data, the queue manager performs certain checks on the header structures to verify that they are valid. If the queue manager detects an error, the call fails with an appropriate reason code. The checks performed vary according to the particular structures that are present:

- Checks are performed only if a version-2 or later MQMD is used on the MQPUT or MQPUT1 call. Checks are not performed if a version-1 MQMD is used, even if an MQMDE is present at the start of the message data.
- Structures that are not supported by the local queue manager, and structures following the first MQDLH in the message, are not validated.
- The MQDH and MQMDE structures are validated completely by the queue manager.
- Other structures are validated partially by the queue manager (not all fields are checked).

General checks performed by the queue manager include the following:

- The *StrucId* field must be valid.
- The *Version* field must be valid.
- The *StrucLength* field must specify a value that is large enough to include the structure plus any variable-length data that forms part of the structure.
- The *CodedCharSetId* field must not be zero, or a negative value that is not valid (MQCCSI_DEFAULT, MQCCSI_EMBEDDED, MQCCSI_Q_MGR, and MQCCSI_UNDEFINED are not valid in most IBM MQ header structures).
- The **BufferLength** parameter of the call must specify a value that is large enough to include the structure (the structure must not extend beyond the end of the message).

In addition to general checks on structures, the following conditions must be satisfied:

- The sum of the lengths of the structures in a PCF message must equal the length specified by the **BufferLength** parameter on the MQPUT or MQPUT1 call. A PCF message is a message that has a format name of MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF.
- An IBM MQ structure must not be truncated, except in the following situations where truncated structures are permitted:
  - Messages that are report messages.
  - PCF messages.
  - Messages containing an MQDLH structure. (Structures *following* the first MQDLH can be truncated; structures preceding the MQDLH cannot.)
- An IBM MQ structure must not be split over two or more segments; the structure must be contained entirely within one segment.

Buffer

For the Visual Basic programming language, the following points apply:

- If the size of the **Buffer** parameter is less than the length specified by the **BufferLength** parameter, the call fails with reason code MQRC_BUFFER_LENGTH_ERROR.
- The **Buffer** parameter is declared as being of type `String`. If the data to be placed on the queue is not of type `String`, use the MQPUTAny call in place of MQPUT.

  The MQPUTAny call has the same parameters as the MQPUT call, except that the **Buffer** parameter is declared as being of type `Any`, allowing any type of data to be placed on the queue. However, this means that *Buffer* cannot be checked to ensure that it is at least *BufferLength* bytes in size.

C invocation

```
MQPUT (Hconn, Hobj, &MsgDesc, &PutMsgOpts, BufferLength, Buffer,
      &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;         /* Connection handle */
MQHOBJ   Hobj;          /* Object handle */
MQMD     MsgDesc;       /* Message descriptor */
MQPMO    PutMsgOpts;    /* Options that control the action of MQPUT */
MQLONG   BufferLength;  /* Length of the message in Buffer */
MQBYTE   Buffer[n];     /* Message data */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

COBOL invocation

```
     CALL 'MQPUT' USING HCONN, HOBJ, MSGDESC, PUTMSGOPTS, BUFFERLENGTH,
                        BUFFER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN         PIC S9(9) BINARY.
**   Object handle
 01  HOBJ          PIC S9(9) BINARY.
**   Message descriptor
 01  MSGDESC.
     COPY CMQMDV.
**   Options that control the action of MQPUT
 01  PUTMSGOPTS.
     COPY CMQPMOV.
**   Length of the message in BUFFER
 01  BUFFERLENGTH  PIC S9(9) BINARY.
**   Message data
 01  BUFFER        PIC X(n).
```

```
**   Completion code
 01  COMPCODE      PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON        PIC S9(9) BINARY.
```

PL/I invocation

```
call MQPUT (Hconn, Hobj, MsgDesc, PutMsgOpts, BufferLength, Buffer,
           CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn         fixed bin(31);  /* Connection handle */
dcl Hobj          fixed bin(31);  /* Object handle */
dcl MsgDesc       like MQMD;      /* Message descriptor */
dcl PutMsgOpts    like MQPMO;     /* Options that control the action of
                                     MQPUT */
dcl BufferLength  fixed bin(31);  /* Length of the message in Buffer */
dcl Buffer        char(n);        /* Message data */
dcl CompCode      fixed bin(31);  /* Completion code */
dcl Reason        fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQPUT,(HCONN,HOBJ,MSGDESC,PUTMSGOPTS,BUFFERLENGTH, X
               BUFFER,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN         DS       F      Connection handle
HOBJ          DS       F      Object handle
MSGDESC       CMQMDA   ,      Message descriptor
PUTMSGOPTS    CMQPMOA  ,      Options that control the action of MQPUT
BUFFERLENGTH  DS       F      Length of the message in BUFFER
BUFFER        DS       CL(n)  Message data
COMPCODE      DS       F      Completion code
REASON        DS       F      Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQPUT Hconn, Hobj, MsgDesc, PutMsgOpts, BufferLength, Buffer, CompCode,
      Reason
```

Declare the parameters as follows:

```
Dim Hconn        As Long   'Connection handle'
Dim Hobj         As Long   'Object handle'
Dim MsgDesc      As MQMD   'Message descriptor'
Dim PutMsgOpts   As MQPMO  'Options that control the action of MQPUT'
Dim BufferLength As Long   'Length of the message in Buffer'
Dim Buffer       As String 'Message data'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

MQPUT1 - Put one message:

The MQPUT1 call puts one message on a queue, or distribution list, or to a topic.

The queue, distribution list, or topic does not need to be open.

Syntax

MQPUT1 (*Hconn*, *ObjDesc*, *MsgDesc*, *PutMsgOpts*, *BufferLength*, *Buffer*, *CompCode*, *Reason*)

Parameters

Hconn
Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

ObjDesc
Type: MQOD - input/output

This is a structure that identifies the queue to which the message is added, or the topic to which the message is published. See "MQOD - Object descriptor" on page 2453 for details.

If the structure is a queue, the user must be authorized to open the queue for output. The queue must **not** be a model queue.

MsgDesc
Type: MQMD - input/output

This structure describes the attributes of the message being sent, and receives feedback information after the put request is complete. See "MQMD - Message descriptor" on page 2387 for details.

If the application provides a version-1 MQMD, the message data can be prefixed with an MQMDE structure to specify values for the fields that exist in the version-2 MQMD but not the version-1. Set the *Format* field in the MQMD to MQFMT_MD_EXTENSION to indicate that an MQMDE is present. See "MQMDE - Message descriptor extension" on page 2442 for more details.

The application does not need to provide an MQMD structure if a valid message handle is supplied in the *MsgHandle* field of the MQGMO structure or in the *OriginalMsgHandle* or *NewMsgHandle* fields of the MQPMO structure. If nothing is provided in one of these fields, the descriptor of the message is taken from the descriptor associated with the message handles.

PutMsgOpts
Type: MQPMO - input/output

See "MQPMO - Put-message options" on page 2477 for details.

BufferLength
Type: MQLONG - input

The length of the message in *Buffer*. Zero is valid, and indicates that the message contains no application data. The upper limit depends on various factors; see "MQPUT - Put message" on page 2743 for the description of the **BufferLength** parameter.

Buffer
Type: MQBYTExBufferLength - input

This is a buffer containing the application message data to be sent. Align the buffer on a boundary appropriate to the nature of the data in the message. 4-byte alignment is suitable for most messages (including messages containing IBM MQ header structures), but some messages might require more stringent alignment. For example, a message containing a 64-bit binary integer might require 8-byte alignment.

If *Buffer* contains character or numeric data, set the *CodedCharSetId* and *Encoding* fields in the **MsgDesc** parameter to the values appropriate to the data; this enables the receiver of the message to convert the data (if necessary) to the character set and encoding used by the receiver.

Note: All the other parameters on the MQPUT1 call must be in the character set and encoding of the local queue manager (given by the **CodedCharSetId** queue manager attribute and MQENC_NATIVE).

In the C programming language, the parameter is declared as a pointer-to-void; the address of any type of data can be specified as the parameter.

If the **BufferLength** parameter is zero, *Buffer* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler can be null.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_MULTIPLE_REASONS
(2136, X'858') Multiple reason codes returned.

MQRC_INCOMPLETE_GROUP
(2241, X'8C1') Message group not complete.

MQRC_INCOMPLETE_MSG
(2242, X'8C2') Logical message not complete.

MQRC_PRIORITY_EXCEEDS_MAXIMUM
(2049, X'801') Message Priority exceeds maximum value supported.

MQRC_UNKNOWN_REPORT_OPTION
(2104, X'838') Report options in message descriptor not recognized.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ALIAS_BASE_Q_TYPE_ERROR
(2001, X'7D1') Alias base queue not a valid type.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BACKED_OUT
(2003, X'7D3') Unit of work backed out.

MQRC_BUFFER_ERROR
(2004, X'7D4') Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_NOT_AVAILABLE
(2345, X'929') coupling facility not available.

MQRC_CF_STRUC_AUTH_FAILED
(2348, X'92C') Coupling-facility structure authorization check failed.

MQRC_CF_STRUC_ERROR
(2349, X'92D') Coupling-facility structure not valid.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CF_STRUC_LIST_HDR_IN_USE
(2347, X'92B') Coupling-facility structure list-header in use.

MQRC_CFGR_ERROR
(2416, X'970') PCF group parameter structure MQCFGR in the message data is not valid.

MQRC_CFH_ERROR
(2235, X'8BB') PCF header structure not valid.

MQRC_CFIF_ERROR
(2414, X'96E') PCF integer filter parameter structure in the message data is not valid.

MQRC_CFIL_ERROR
(2236, X'8BC') PCF integer list parameter structure or PCIF*64 integer list parameter structure not valid.

MQRC_CFIN_ERROR
(2237, X'8BD') PCF integer parameter structure or PCIF*64 integer parameter structure not valid.

MQRC_CFSF_ERROR
(2415, X'96F') PCF string filter parameter structure in the message data is not valid.

MQRC_CFSL_ERROR
(2238, X'8BE') PCF string list parameter structure not valid.

MQRC_CFST_ERROR
(2239, X'8BF') PCF string parameter structure not valid.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CLUSTER_EXIT_ERROR
(2266, X'8DA') Cluster workload exit failed.

MQRC_CLUSTER_RESOLUTION_ERROR
(2189, X'88D') Cluster name resolution failed.

MQRC_CLUSTER_RESOURCE_ERROR
(2269, X'8DD') Cluster resource error.

MQRC_COD_NOT_VALID_FOR_XCF_Q
(2106, X'83A') COD report option not valid for XCF queue.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_CONTENT_ERROR
2554 (X'09FA') Message content could not be parsed to determine whether the message can be delivered to a subscriber with an extended message selector.

MQRC_CONTEXT_HANDLE_ERROR
(2097, X'831') Queue handle referred to does not save context.

MQRC_CONTEXT_NOT_AVAILABLE
(2098, X'832') Context not available for queue handle referred to.

MQRC_DATA_LENGTH_ERROR
(2010, X'7DA') Data length parameter not valid.

MQRC_DB2_NOT_AVAILABLE
(2342, X'926') Db2 subsystem not available.

MQRC_DEF_XMIT_Q_TYPE_ERROR
(2198, X'896') Default transmission queue not local.

MQRC_DEF_XMIT_Q_USAGE_ERROR
(2199, X'897') Default transmission queue usage error.

MQRC_DH_ERROR
(2135, X'857') Distribution header structure not valid.

MQRC_DLH_ERROR
(2141, X'85D') Dead letter header structure not valid.

MQRC_EPH_ERROR
(2420, X'974') Embedded PCF structure not valid.

MQRC_EXPIRY_ERROR
(2013, X'7DD') Expiry time not valid.

MQRC_FEEDBACK_ERROR
(2014, X'7DE') Feedback code not valid.

MQRC_GLOBAL_UOW_CONFLICT
(2351, X'92F') Global units of work conflict.

MQRC_GROUP_ID_ERROR
(2258, X'8D2') Group identifier not valid.

MQRC_HANDLE_IN_USE_FOR_UOW
(2353, X'931') Handle in use for global unit of work.

MQRC_HANDLE_NOT_AVAILABLE
(2017, X'7E1') No more handles available.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HEADER_ERROR
(2142, X'85E') IBM MQ header structure not valid.

MQRC_IIH_ERROR
(2148, X'864') IMS information header structure not valid.

MQRC_LOCAL_UOW_CONFLICT
(2352, X'930') Global unit of work conflicts with local unit of work.

MQRC_MD_ERROR
(2026, X'7EA') Message descriptor not valid.

MQRC_MDE_ERROR
(2248, X'8C8') Message descriptor extension not valid.

MQRC_MISSING_REPLY_TO_Q
(2027, X'7EB') Missing reply-to queue.

MQRC_MISSING_WIH
(2332, X'91C') Message data does not begin with MQWIH.

MQRC_MSG_FLAGS_ERROR
(2249, X'8C9') Message flags not valid.

MQRC_MSG_SEQ_NUMBER_ERROR
(2250, X'8CA') Message sequence number not valid.

MQRC_MSG_TOO_BIG_FOR_Q
(2030, X'7EE') Message length greater than maximum for queue.

MQRC_MSG_TOO_BIG_FOR_Q_MGR
(2031, X'7EF') Message length greater than maximum for queue manager.

MQRC_MSG_TYPE_ERROR
(2029, X'7ED') Message type in message descriptor not valid.

MQRC_MULTIPLE_REASONS
(2136, X'858') Multiple reason codes returned.

MQRC_NO_DESTINATIONS_AVAILABLE
(2270, X'8DE') No destination queues available.

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OBJECT_IN_USE
(2042, X'7FA') Object already open with conflicting options.

MQRC_OBJECT_LEVEL_INCOMPATIBLE
(2360, X'938') Object level not compatible.

MQRC_OBJECT_NAME_ERROR
(2152, X'868') Object name not valid.

MQRC_OBJECT_NOT_UNIQUE
(2343, X'927') Object not unique.

MQRC_OBJECT_Q_MGR_NAME_ERROR
(2153, X'869') Object queue manager name not valid.

MQRC_OBJECT_RECORDS_ERROR
(2155, X'86B') Object records not valid.

MQRC_OBJECT_TYPE_ERROR
(2043, X'7FB') Object type not valid.

MQRC_OD_ERROR
(2044, X'7FC') Object descriptor structure not valid.

MQRC_OFFSET_ERROR
(2251, X'8CB') Message segment offset not valid.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_ORIGINAL_LENGTH_ERROR
(2252, X'8CC') Original length not valid.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_PAGESET_FULL
(2192, X'890') External storage medium is full.

MQRC_PCF_ERROR
(2149, X'865') PCF structures not valid.

MQRC_PERSISTENCE_ERROR
(2047, X'7FF') Persistence not valid.

MQRC_PERSISTENT_NOT_ALLOWED
(2048, X'800') Queue does not support persistent messages.

MQRC_PMO_ERROR
(2173, X'87D') Put-message options structure not valid.

MQRC_PMO_RECORD_FLAGS_ERROR
(2158, X'86E') Put message record flags not valid.

MQRC_PRIORITY_ERROR
(2050, X'802') Message priority not valid.

MQRC_PUBLICATION_FAILURE
(2502, X'9C6') The publication has not been delivered to any of the subscribers.

MQRC_PUT_INHIBITED
(2051, X'803') Put calls inhibited for the queue.

MQRC_PUT_MSG_RECORDS_ERROR
(2159, X'86F') Put message records not valid.

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_FULL
(2053, X'805') Queue already contains maximum number of messages.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_Q_SPACE_NOT_AVAILABLE
(2056, X'808') No space available on disk for queue.

MQRC_Q_TYPE_ERROR
(2057, X'809') Queue type not valid.

MQRC_RECS_PRESENT_ERROR
(2154, X'86A') Number of records present not valid.

MQRC_REMOTE_Q_NAME_ERROR
(2184, X'888') Remote queue name not valid.

MQRC_REPORT_OPTIONS_ERROR
(2061, X'80D') Report options in message descriptor not valid.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_RESPONSE_RECORDS_ERROR
(2156, X'86C') Response records not valid.

MQRC_RFH_ERROR
(2334, X'91E') MQRFH or MQRFH2 structure not valid.

MQRC_RMH_ERROR
(2220, X'8AC') Reference message header structure not valid.

MQRC_SECURITY_ERROR
(2063, X'80F') Security error occurred.

MQRC_SEGMENT_LENGTH_ZERO
(2253, X'8CD') Length of data in message segment is zero.

MQRC_SELECTION_NOT_AVAILABLE
2551 (X'09F7') A possible subscriber for the publication exists, but the queue manager cannot check whether to send the publication to the subscriber.

MQRC_STOPPED_BY_CLUSTER_EXIT
(2188, X'88C') Call rejected by cluster workload exit.

MQRC_STORAGE_CLASS_ERROR
(2105, X'839') Storage class error.

MQRC_STORAGE_MEDIUM_FULL
(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_SYNCPOINT_LIMIT_REACHED
(2024, X'7E8') No more messages can be handled within current unit of work.

MQRC_SYNCPOINT_NOT_AVAILABLE
(2072, X'818') Syncpoint support not available.

MQRC_TM_ERROR
(2265, X'8D9') Trigger message structure not valid.

MQRC_TMC_ERROR
(2191, X'88F') Character trigger message structure not valid.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

MQRC_UNKNOWN_ALIAS_BASE_Q
(2082, X'822') Unknown alias base queue.

MQRC_UNKNOWN_DEF_XMIT_Q
(2197, X'895') Unknown default transmission queue.

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRC_UNKNOWN_OBJECT_Q_MGR
(2086, X'826') Unknown object queue manager.

MQRC_UNKNOWN_REMOTE_Q_MGR
(2087, X'827') Unknown remote queue manager.

MQRC_UNKNOWN_XMIT_Q
(2196, X'894') Unknown transmission queue.

MQRC_UOW_ENLISTMENT_ERROR
(2354, X'932') Enlistment in global unit of work failed.

MQRC_UOW_MIX_NOT_SUPPORTED
(2355, X'933') Mixture of unit-of-work calls not supported.

MQRC_UOW_NOT_AVAILABLE
(2255, X'8CF') Unit of work not available for the queue manager to use.

MQRC_WIH_ERROR
(2333, X'91D') MQWIH structure not valid.

MQRC_WRONG_CF_LEVEL
(2366, X'93E') Coupling-facility structure is wrong level.

MQRC_WRONG_MD_VERSION
(2257, X'8D1') Wrong version of MQMD supplied.

MQRC_XMIT_Q_TYPE_ERROR
(2091, X'82B') Transmission queue not local.

MQRC_XMIT_Q_USAGE_ERROR
(2092, X'82C') Transmission queue with wrong usage.

MQRC_XQH_ERROR
(2260, X'8D4') Transmission queue header structure not valid.

For detailed information about these codes, see Reason codes.

Usage notes

1. Both the MQPUT and MQPUT1 calls can be used to put messages on a queue; which call to use depends on the circumstances:
   - Use the MQPUT call to place multiple messages on the *same* queue.

     An MQOPEN call specifying the MQOO_OUTPUT option is issued first, followed by one or more MQPUT requests to add messages to the queue; finally the queue is closed with an MQCLOSE call. This gives better performance than repeated use of the MQPUT1 call.
   - Use the MQPUT1 call to put only *one* message on a queue.

     This call encapsulates the MQOPEN, MQPUT, and MQCLOSE calls into a single call, minimizing the number of calls that must be issued.
2. If an application puts a sequence of messages on the same queue without using message groups, the order of those messages is preserved if certain conditions are satisfied. However, in most environments the MQPUT1 call does not satisfy these conditions, and so does not preserve message order. The MQPUT call must be used instead in these environments. See MQPUT usage notes for details.
3. The MQPUT1 call can be used to put messages to distribution lists. For general information about this, see the usage notes for the MQOPEN and MQPUT calls.

   Distribution lists are supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ clients connected to these systems.

   The following differences apply when using the MQPUT1 call:

   a. If the application provides MQRR response records, they must be provided using the MQOD structure; they cannot be provided using the MQPMO structure.
   b. The reason code MQRC_OPEN_FAILED is never returned by MQPUT1 in the response records; if a queue fails to open, the response record for that queue contains the reason code resulting from the open operation.

      If an open operation for a queue succeeds with a completion code of MQCC_WARNING, the completion code and reason code in the response record for that queue are replaced by the completion and reason codes resulting from the put operation.

      As with the MQOPEN and MQPUT calls, the queue manager sets the response records (if provided) only when the outcome of the call is not the same for all queues in the distribution list; this is indicated by the call completing with reason code MQRC_MULTIPLE_REASONS.
4. If the MQPUT1 call is used to put a message on a cluster queue, the call behaves as though MQOO_BIND_NOT_FIXED had been specified on the MQOPEN call.
5. If a message is put with one or more IBM MQ header structures at the beginning of the application message data, the queue manager performs certain checks on the header structures to verify that they are valid. For more information about this, see the usage notes for the MQPUT call.
6. If more than one of the warning situations arise (see the **CompCode** parameter), the reason code returned is the first one in the following list that applies:
   a. MQRC_MULTIPLE_REASONS
   b. MQRC_INCOMPLETE_MSG
   c. MQRC_INCOMPLETE_GROUP
   d. MQRC_PRIORITY_EXCEEDS_MAXIMUM or MQRC_UNKNOWN_REPORT_OPTION
7. For the Visual Basic programming language, the following points apply:
   - If the size of the **Buffer** parameter is less than the length specified by the **BufferLength** parameter, the call fails with reason code MQRC_BUFFER_LENGTH_ERROR.
   - The **Buffer** parameter is declared as being of type `String`. If the data to be placed on the queue is not of type `String`, use the MQPUT1Any call in place of MQPUT1.

The MQPUT1Any call has the same parameters as the MQPUT1 call, except that the **Buffer** parameter is declared as being of type Any, allowing any type of data to be placed on the queue. However, this means that *Buffer* cannot be checked to ensure that it is at least *BufferLength* bytes in size.

8. When an MQPUT1 call is issued with MQPMO_SYNCPOINT, the default behavior changes, so that the put operation is completed asynchronously. This might cause a change in the behavior of some applications that rely on certain fields in the MQOD and MQMD structures being returned, but which now contain undefined values. An application can specify MQPMO_SYNC_RESPONSE to ensure that the put operation is performed synchronously and that all the appropriate field values are completed.

C invocation

```
MQPUT1 (Hconn, &ObjDesc, &MsgDesc, &PutMsgOpts,
       BufferLength, Buffer, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;         /* Connection handle */
MQOD     ObjDesc;       /* Object descriptor */
MQMD     MsgDesc;       /* Message descriptor */
MQPMO    PutMsgOpts;    /* Options that control the action of MQPUT1 */
MQLONG   BufferLength;  /* Length of the message in Buffer */
MQBYTE   Buffer[n];     /* Message data */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

COBOL invocation

```
     CALL 'MQPUT1' USING HCONN, OBJDESC, MSGDESC, PUTMSGOPTS,
                         BUFFERLENGTH, BUFFER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN         PIC S9(9) BINARY.
**   Object descriptor
 01  OBJDESC.
     COPY CMQODV.
**   Message descriptor
 01  MSGDESC.
     COPY CMQMDV.
**   Options that control the action of MQPUT1
 01  PUTMSGOPTS.
     COPY CMQPMOV.
**   Length of the message in BUFFER
 01  BUFFERLENGTH  PIC S9(9) BINARY.
**   Message data
 01  BUFFER        PIC X(n).
**   Completion code
 01  COMPCODE      PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON        PIC S9(9) BINARY.
```

PL/I invocation

```
call MQPUT1 (Hconn, ObjDesc, MsgDesc, PutMsgOpts, BufferLength, Buffer,
            CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn         fixed bin(31);  /* Connection handle */
dcl ObjDesc       like MQOD;      /* Object descriptor */
dcl MsgDesc       like MQMD;      /* Message descriptor */
dcl PutMsgOpts    like MQPMO;     /* Options that control the action of
                                     MQPUT1 */
dcl BufferLength  fixed bin(31);  /* Length of the message in Buffer */
```

```
dcl Buffer        char(n);        /* Message data */
dcl CompCode      fixed bin(31);  /* Completion code */
dcl Reason        fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
        CALL MQPUT1,(HCONN,OBJDESC,MSGDESC,PUTMSGOPTS,BUFFERLENGTH,   X
              BUFFER,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN         DS       F      Connection handle
OBJDESC       CMQODA   ,      Object descriptor
MSGDESC       CMQMDA   ,      Message descriptor
PUTMSGOPTS    CMQPMOA  ,      Options that control the action of MQPUT1
BUFFERLENGTH  DS       F      Length of the message in BUFFER
BUFFER        DS       CL(n)  Message data
COMPCODE      DS       F      Completion code
REASON        DS       F      Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQPUT1 Hconn, ObjDesc, MsgDesc, PutMsgOpts, BufferLength, Buffer,
       CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn        As Long   'Connection handle'
Dim ObjDesc      As MQOD   'Object descriptor'
Dim MsgDesc      As MQMD   'Message descriptor'
Dim PutMsgOpts   As MQPMO  'Options that control the action of MQPUT1'
Dim BufferLength As Long   'Length of the message in Buffer'
Dim Buffer       As String 'Message data'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

MQSET - Set object attributes:

Use the MQSET call to change the attributes of an object represented by a handle. The object must be a queue.

Syntax

MQSET (*Hconn*, *Hobj*, *SelectorCount*, *Selectors*, *IntAttrCount*, *IntAttrs*, *CharAttrLength*, *CharAttrs*, *Compcode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hobj

Type: MQHOBJ - input

This handle represents the queue object with attributes that are to be set. The handle was returned by a previous MQOPEN call that specified the MQOO_SET option.

SelectorCount
Type: MQLONG - input

This is the count of selectors that are supplied in the *Selectors* array. It is the number of attributes that are to be set. Zero is a valid value. The maximum number allowed is 256.

Selectors
Type: MQLONGxSelectorCount - input

This is an array of **SelectorCount** attribute selectors; each selector identifies an attribute (integer or character) with a value that is to be set.

Each selector must be valid for the type of queue that *Hobj* represents. Only certain MQIA_* and MQCA_* values are allowed; as listed later.

Selectors can be specified in any order. Attribute values that correspond to integer attribute selectors (MQIA_* selectors) must be specified in *IntAttrs* in the same order in which these selectors occur in *Selectors*. Attribute values that correspond to character attribute selectors (MQCA_* selectors) must be specified in *CharAttrs* in the same order in which those selectors occur. MQIA_* selectors can be interleaved with the MQCA_* selectors; only the relative order within each type is important.

You can specify the same selector more than once; if you do, the last value specified for a particular selector is the one that takes effect.

Note:

1. The integer and character attribute selectors are allocated within two different ranges; the MQIA_* selectors reside within the range MQIA_FIRST through MQIA_LAST, and the MQCA_* selectors within the range MQCA_FIRST through MQCA_LAST.

   For each range, the constants MQIA_LAST_USED and MQCA_LAST_USED define the highest value that the queue manager accepts.
2. If all the MQIA_* selectors occur first, the same element numbers can be used to address corresponding elements in the *Selectors* and *IntAttrs* arrays.
3. If the **SelectorCount** parameter is zero, *Selectors* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler might be null.

The attributes that can be set are listed in the following table. No other attributes can be set using this call. For the MQCA_* attribute selectors, the constant that defines the length in bytes of the string that is required in *CharAttrs* is supplied in parentheses.

Table 271. MQSET attribute selectors for queues

Selector	Description	Note
MQCA_TRIGGER_DATA	Trigger data (MQ_TRIGGER_DATA_LENGTH).	
MQIA_DIST_LISTS	Distribution list support.	1
MQIA_INHIBIT_GET	Whether get operations are allowed.	
MQIA_INHIBIT_PUT	Whether put operations are allowed.	
MQIA_TRIGGER_CONTROL	Trigger control.	
MQIA_TRIGGER_DEPTH	Trigger depth.	
MQIA_TRIGGER_MSG_PRIORITY	Threshold message priority for triggers.	
MQIA_TRIGGER_TYPE	Trigger type.	

Note:

1. Supported only on AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM MQ MQI clients connected to these systems.

IntAttrCount
Type: MQLONG - input

This is the number of elements in the *IntAttrs* array, and must be at least the number of MQIA_* selectors in the **Selectors** parameter. Zero is a valid value if there are none.

IntAttrs
Type: MQLONGxIntAttrCount - input

This is an array of *IntAttrCount* integer attribute values. These attribute values must be in the same order as the MQIA_* selectors in the *Selectors* array.

If the **IntAttrCount** or **SelectorCount** parameter is zero, *IntAttrs* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler might be null.

CharAttrLength
Type: MQLONG - input

This is the length in bytes of the **CharAttrs** parameter, and must be at least the sum of the lengths of the character attributes specified in the *Selectors* array. Zero is a valid value if there are no MQCA_* selectors in *Selectors*.

CharAttrs
Type: MQCHAR x CharAttrLength - input

This is the buffer containing the character attribute values, concatenated together. The length of the buffer is given by the **CharAttrLength** parameter.

The characters attributes must be specified in the same order as the MQCA_* selectors in the *Selectors* array. The length of each character attribute is fixed (see Selectors). If the value to be set for an attribute contains fewer nonblank characters than the defined length of the attribute, pad the value in *CharAttrs* to the right with blanks to make the attribute value match the defined length of the attribute.

If the **CharAttrLength** or **SelectorCount** parameter is zero, *CharAttrs* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler might be null.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_NOT_AVAILABLE
(2345, X'929') Coupling facility not available.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CF_STRUC_LIST_HDR_IN_USE
(2347, X'92B') Coupling-facility structure list-header in use.

MQRC_CHAR_ATTR_LENGTH_ERROR
(2006, X'7D6') Length of character attributes not valid.

MQRC_CHAR_ATTRS_ERROR
(2007, X'7D7') Character attributes string not valid.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_DB2_NOT_AVAILABLE
(2342, X'926') Db2 subsystem not available.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_INHIBIT_VALUE_ERROR
(2020, X'7E4') Value for inhibit-get or inhibit-put queue attribute not valid.

MQRC_INT_ATTR_COUNT_ERROR
(2021, X'7E5') Count of integer attributes not valid.

MQRC_INT_ATTRS_ARRAY_ERROR
(2023, X'7E7') Integer attributes array not valid.

MQRC_NOT_OPEN_FOR_SET
(2040, X'7F8') Queue not open for set.

MQRC_OBJECT_CHANGED
(2041, X'7F9') Object definition changed since opened.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SELECTOR_COUNT_ERROR
(2065, X'811') Count of selectors not valid.

MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.

MQRC_SELECTOR_LIMIT_EXCEEDED
(2066, X'812') Count of selectors too large.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_TRIGGER_CONTROL_ERROR
(2075, X'81B') Value for trigger-control attribute not valid.

MQRC_TRIGGER_DEPTH_ERROR
(2076, X'81C') Value for trigger-depth attribute not valid.

MQRC_TRIGGER_MSG_PRIORITY_ERR
(2077, X'81D') Value for trigger-message-priority attribute not valid.

MQRC_TRIGGER_TYPE_ERROR
(2078, X'81E') Value for trigger-type attribute not valid.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

Usage notes

1. Using this call, the application can specify an array of integer attributes, or a collection of character attribute strings, or both. If no errors occur, the attributes specified are all set simultaneously. If an error occurs (for example, if a selector is not valid, or an attempt is made to set an attribute to a value that is not valid), the call fails and no attributes are set.
2. The values of attributes can be determined using the MQINQ call; see "MQINQ - Inquire object attributes" on page 2700 for details.

   Note: Not all attributes with values that can be inquired using the MQINQ call can have their values changed using the MQSET call. For example, no process-object or queue manager attributes can be set with this call.

3. Attribute changes are preserved across restarts of the queue manager (other than alterations to temporary dynamic queues, which do not survive restarts of the queue manager).
4. You cannot change the attributes of a model queue using the MQSET call. However, if you open a model queue using the MQOPEN call with the MQOO_SET option, you can use the MQSET call to set the attributes of the dynamic local queue that is created by the MQOPEN call.
5. If the object being set is a cluster queue, there must be a local instance of the cluster queue for the open to succeed.

For more information about object attributes, see:

- "Attributes for queues" on page 2833
- "Attributes for namelists" on page 2869
- "Attributes for process definitions" on page 2872
- "Attributes for the queue manager" on page 2792

C invocation

```
MQSET (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
      CharAttrLength, CharAttrs, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;           /* Connection handle */
MQHOBJ   Hobj;            /* Object handle */
MQLONG   SelectorCount;   /* Count of selectors */
MQLONG   Selectors[n];    /* Array of attribute selectors */
MQLONG   IntAttrCount;    /* Count of integer attributes */
MQLONG   IntAttrs[n];     /* Array of integer attributes */
MQLONG   CharAttrLength;  /* Length of character attributes buffer */
MQCHAR   CharAttrs[n];    /* Character attributes */
MQLONG   CompCode;        /* Completion code */
MQLONG   Reason;          /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQSET' USING HCONN, HOBJ, SELECTORCOUNT, SELECTORS-TABLE,
                   INTATTRCOUNT, INTATTRS-TABLE, CHARATTRLENGTH,
                   CHARATTRS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN            PIC S9(9) BINARY.
**   Object handle
 01  HOBJ             PIC S9(9) BINARY.
**   Count of selectors
 01  SELECTORCOUNT    PIC S9(9) BINARY.
**   Array of attribute selectors
 01  SELECTORS-TABLE.
  02  SELECTORS        PIC S9(9) BINARY OCCURS n TIMES.
**   Count of integer attributes
 01  INTATTRCOUNT     PIC S9(9) BINARY.
**   Array of integer attributes
 01  INTATTRS-TABLE.
  02  INTATTRS         PIC S9(9) BINARY OCCURS n TIMES.
**   Length of character attributes buffer
 01  CHARATTRLENGTH   PIC S9(9) BINARY.
**   Character attributes
 01  CHARATTRS        PIC X(n).
**   Completion code
 01  COMPCODE         PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON           PIC S9(9) BINARY.
```

PL/I invocation

```
call MQSET (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount,
           IntAttrs, CharAttrLength, CharAttrs, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn           fixed bin(31);  /* Connection handle */
dcl Hobj            fixed bin(31);  /* Object handle */
dcl SelectorCount   fixed bin(31);  /* Count of selectors */
dcl Selectors(n)    fixed bin(31);  /* Array of attribute selectors */
dcl IntAttrCount    fixed bin(31);  /* Count of integer attributes */
dcl IntAttrs(n)     fixed bin(31);  /* Array of integer attributes */
dcl CharAttrLength  fixed bin(31);  /* Length of character attributes
                                       buffer */
dcl CharAttrs       char(n);        /* Character attributes */
dcl CompCode        fixed bin(31);  /* Completion code */
dcl Reason          fixed bin(31);  /* Reason code qualifying
                                       CompCode */
```

High Level Assembler invocation

```
CALL MQSET,(HCONN,HOBJ,SELECTORCOUNT,SELECTORS,INTATTRCOUNT,  X
             INTATTRS,CHARATTRLENGTH,CHARATTRS,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN           DS  F      Connection handle
HOBJ            DS  F      Object handle
SELECTORCOUNT   DS  F      Count of selectors
SELECTORS       DS  (n)F   Array of attribute selectors
INTATTRCOUNT    DS  F      Count of integer attributes
INTATTRS        DS  (n)F   Array of integer attributes
CHARATTRLENGTH  DS  F      Length of character attributes buffer
CHARATTRS       DS  CL(n)  Character attributes
COMPCODE        DS  F      Completion code
REASON          DS  F      Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQSET Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
      CharAttrLength, CharAttrs, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn          As Long   'Connection handle'
Dim Hobj           As Long   'Object handle'
Dim SelectorCount  As Long   'Count of selectors'
Dim Selectors      As Long   'Array of attribute selectors'
Dim IntAttrCount   As Long   'Count of integer attributes'
Dim IntAttrs       As Long   'Array of integer attributes'
Dim CharAttrLength As Long   'Length of character attributes buffer'
Dim CharAttrs      As String 'Character attributes'
Dim CompCode       As Long   'Completion code'
Dim Reason         As Long   'Reason code qualifying CompCode'
```

MQSETMP - Set message property:

Use the MQSETMP call to set or modify a property of a message handle.

Syntax

MQSETMP (*Hconn*, *Hmsg*, *SetPropOpts*, *Name*, *PropDesc*, *Type*, *ValueLength*, *Value*, *Compcode*, *Reason*)

Parameters

Hconn
Type: MQHCONN - input

This handle represents the connection to the queue manager.

The value must match the connection handle that was used to create the message handle specified in the **Hmsg** parameter. If the message handle was created using MQHC_UNASSOCIATED_HCONN, a valid connection must be established on the thread setting a property of the message handle, otherwise the call fails with reason code MQRC_CONNECTION_BROKEN.

Hmsg
Type: MQHMSG - input

This is the message handle to be modified. The value was returned by a previous MQCRTMH call.

SetPropOpts
Type: MQSMPO - input

Control how message properties are set.

This structure allows applications to specify options that control how message properties are set. The structure is an input parameter on the MQSETMP call. See MQSMPO for further information.

Name
Type: MQCHARV- input

This is the name of the property to set.

See Property names and Property name restrictions for further information about the use of property names.

PropDesc
Type: MQPD - input/output

This structure is used to define the attributes of a property, including:

- what happens if the property is not supported
- what message context the property belongs to
- what messages the property is copied into as it flows

See MQPD for further information about this structure.

Type
Type: MQLONG - input

The data type of the property being set. It can be one of the following:

MQTYPE_BOOLEAN
A Boolean. *ValueLength* must be 4.

MQTYPE_BYTE_STRING
A byte string. *ValueLength* must be zero or greater.

MQTYPE_INT8
An 8-bit signed integer. *ValueLength* must be 1.

MQTYPE_INT16
A 16-bit signed integer. *ValueLength* must be 2.

MQTYPE_INT32
A 32-bit signed integer. *ValueLength* must be 4.

MQTYPE_INT64
A 64-bit signed integer. *ValueLength* must be 8.

MQTYPE_FLOAT32
A 32-bit floating-point number. *ValueLength* must be 4.

Note: this type is not supported with applications using IBM COBOL for z/OS.

MQTYPE_FLOAT64
A 64-bit floating-point number. *ValueLength* must be 8.

Note: this type is not supported with applications using IBM COBOL for z/OS.

MQTYPE_STRING
A character string. *ValueLength* must be zero or greater, or the special value MQVL_NULL_TERMINATED.

MQTYPE_NULL
The property exists but has a null value. *ValueLength* must be zero.

`ValueLength`
Type: MQLONG - input

The length in bytes of the property value in the *Value* parameter. Zero is valid only for null values or for strings or byte strings. Zero indicates that the property exists but that the value contains no characters or bytes.

The value must be greater than or equal to zero or the following special value if the *Type* parameter has MQTYPE_STRING set:

MQVL_NULL_TERMINATED
The value is delimited by the first null encountered in the string. The null is not included as part of the string. This value is invalid if MQTYPE_STRING is not also set.

Note: The null character used to terminate a string if MQVL_NULL_TERMINATED is set is a null from the character set of the Value.

`Value`
Type: MQBYTExValueLength - input

The value of the property to be set. The buffer must be aligned on a boundary appropriate to the nature of the data in the value.

In the C programming language, the parameter is declared as a pointer-to-void; the address of any type of data can be specified as the parameter.

If *ValueLength* is zero, *Value* is not referred to. In this case, the parameter address passed by programs written in C or System/390 assembler can be null.

`CompCode`
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_RFH_FORMAT_ERROR
(2421, X'0975') An MQRFH2 folder containing properties could not be parsed.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'089C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BUFFER_ERROR
(2004, X'07D4') Value parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'07D5') Value length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'08AB') MQI call entered before previous call completed.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle pointer not valid.

MQRC_MSG_HANDLE_IN_USE
(2499, X'09C3') Message handle already in use.

MQRC_OPTIONS_ERROR
(2046, X'07FE') Options not valid or not consistent.

MQRC_PD_ERROR
(2482, X'09B2') Property descriptor structure not valid.

MQRC_PROPERTY_NAME_ERROR
(2442, X'098A') Invalid property name.

MQRC_PROPERTY_TYPE_ERROR
(2473, X'09A9') Invalid property data type.

MQRC_PROP_NUMBER_FORMAT_ERROR
(2472, X'09A8') Number format error encountered in value data.

MQRC_SMPO_ERROR
(2463, X'099F') Set message property options structure not valid.

MQRC_SOURCE_CCSID_ERROR
(2111, X'083F') Property name coded character set identifier not valid.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

C invocation

```
MQSETMP (Hconn, Hmsg, &SetPropOpts, &Name, &PropDesc, Type,
ValueLength, &Value, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;       /* Connection handle */
MQHMSG   Hmsg;        /* Message handle */
MQSMPO   SetPropOpts; /* Options that control the action of MQSETMP */
MQCHARV  Name;        /* Property name */
MQPD     PropDesc;    /* Property descriptor */
MQLONG   Type;        /* Property data type */
MQLONG   ValueLength; /* Length of property value in Value */
MQBYTE   Value[n];    /* Property value */
MQLONG   CompCode;    /* Completion code */
MQLONG   Reason;      /* Reason code qualifying CompCode */
```

COBOL invocation

```
     CALL 'MQSETMP' USING HCONN, HMSG, SETMSGOPTS, NAME, PROPDESC, TYPE,
                          VALUELENGTH, VALUE, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN     PIC S9(9) BINARY.
**   Message handle
 01  HMSG    PIC S9(18) BINARY.
**   Options that control the action of MQSETMP
 01  SETMSGOPTS.
     COPY CMQSMPOV.
**   Property name
 01  NAME
    COPY CMQCHRVV.
**   Property descriptor
 01  PROPDESC.
     COPY CMQPDV.
**   Property data type
 01  TYPE          PIC S9(9) BINARY.
**   Length of property value in VALUE
 01  VALUELENGTH   PIC S9(9) BINARY.
**   Property value
 01  VALUE         PIC X(n).
**   Completion code
 01  COMPCODE  PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON    PIC S9(9) BINARY.
```

PL/I invocation

```
call MQSETMP (Hconn, Hmsg, SetPropOpts, Name, PropDesc, Type, ValueLength,
              Value, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn       fixed bin(31);  /* Connection handle */
dcl Hmsg        fixed bin(63);  /* Message handle */
dcl SetPropOpts like MQSMPO;    /* Options that control the action of MQSETMP */
dcl Name        like MQCHARV;   /* Property name */
dcl PropDesc    like MQPD;      /* Property descriptor */
dcl Type        fixed bin(31);  /* Property data type */
dcl ValueLength fixed bin(31);  /* Length of property value in Value */
dcl Value       char(n);        /* Property value */
dcl CompCode    fixed bin(31);  /* Completion code */
dcl Reason      fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQSETMP,(HCONN,HMSG,SETMSGHOPTS,NAME,PROPDESC,TYPE,VALUELENGTH,
      VALUE,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN        DS         F     Connection handle
HMSG         DS         D     Message handle
SETMSGOPTS   CMQSMPOA   ,     Options that control the action of MQSETMP
NAME         CMQCHRVA   ,     Property name
PROPDESC     CMQPDA     ,     Property descriptor
TYPE         DS         F     Property data type
VALUELENGTH  DS         F     Length of property value in VALUE
VALUE        DS         CL(n) Property value
COMPCODE     DS         F     Completion code
REASON       DS         F     Reason code qualifying COMPCODE
```

MQSTAT - Retrieve status information:

Use the MQSTAT call to retrieve status information. The type of status information returned is determined by the Type value specified on the call.

Syntax

MQSTAT (*Hconn*, *Type*, *Stat*, *Compcode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Type

Type: MQLONG - input

Type of status information being requested. The > valid values are:

MQSTAT_TYPE_ASYNC_ERROR
Return information about previous asynchronous put operations.

MQSTAT_TYPE_RECONNECTION

Return information about reconnection. If the connection is reconnecting or failed to reconnect, the information describes the failure which caused the connection to begin reconnecting.

This value is only valid for client connections. For other types of connection, the call fails with reason code **MQRC_ENVIRONMENT_ERROR**

MQSTAT_TYPE_RECONNECTION_ERROR

Return information about a previous failure related to reconnect. If the connection failed to reconnect, the information describes the failure which caused reconnection to fail.

This value is only valid for client connections. For other types of connection, the call fails with reason code **MQRC_ENVIRONMENT_ERROR**.

Stat

Type: MQSTS - input/output

Status information structure. See “MQSTS - Status reporting structure” on page 2578 for details.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_FUNCTION_NOT_SUPPORTED
(2298, X'8FA') The function requested is not available in the current environment.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_Q_MGR_STOPPING
(2162,X'872' - Queue manager stopping

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_STAT_TYPE_ERROR
(2430, X'97E' Error with MQSTAT type

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_STS_ERROR
(2426, X'97A') Error with MQSTS structure

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

Usage notes

1. A call to MQSTAT specifying a type of MQSTAT_TYPE_ASYNC_ERROR returns information about previous asynchronous MQPUT and MQPUT1 operations. The MQSTS structure passed back on return from the MQSTAT call contains the first recorded asynchronous warning or error information for that connection. If further errors or warnings follow the first, they do not normally alter these values. However, if an error occurs with a completion code of MQCC_WARNING, a subsequent failure with a completion code of MQCC_FAILED is returned instead.
2. If no errors have occurred since the connection was established or since the last call to MQSTAT then a CompCode of MQCC_OK and Reason of MQRC_NONE are returned in the MQSTS structure.
3. Counts of the number of asynchronous calls that have been processed under the connection handle are returned by way of three counter fields; PutSuccessCount, PutWarningCount and PutFailureCount. These counters are incremented by the queue manager each time an asynchronous operation is processed successfully, has a warning, or fails (note that for accounting purposes a put to a distribution list counts once per destination queue rather than once per distribution list). A counter is not incremented beyond the maximum positive value AMQ_LONG_MAX.
4. A successful call to MQSTAT results in any previous error information or counts being reset.
5. The behavior of MQSTAT depends on the value of the **MQSTAT Type** parameter you provide.
6.

   MQSTAT_TYPE_ASYNC_ERROR

   a. A call to MQSTAT specifying a type of MQSTAT_TYPE_ASYNC_ERROR returns information about previous asynchronous MQPUT and MQPUT1 operations. The MQSTS structure passed back on return from the MQSTAT call contains the first recorded asynchronous warning or error information for that connection. If further errors or warnings follow the first, they do not normally alter these values. However, if an error occurs with a completion code of MQCC_WARNING, a subsequent failure with a completion code of MQCC_FAILED is returned instead.
   b. If no errors have occurred since the connection was established or since the last call to MQSTAT then a CompCode of MQCC_OK and Reason of MQRC_NONE are returned in the MQSTS structure.
   c. Counts of the number of asynchronous calls that have been processed under the connection handle are returned by way of three counter fields; PutSuccessCount, PutWarningCount and PutFailureCount. These counters are incremented by the queue manager each time an asynchronous operation is processed successfully, has a warning, or fails (note that for accounting purposes a put to a distribution list counts once per destination queue rather than once per distribution list). A counter is not incremented beyond the maximum positive value AMQ_LONG_MAX.
   d. A successful call to MQSTAT results in any previous error information or counts being reset.

   MQSTAT_TYPE_RECONNECTION

   Suppose you call MQSTAT with Type set to MQSTAT_TYPE_RECONNECTION inside an event handler during reconnection. Consider these examples.

   The client is attempting reconnection or failed to reconnect.
   CompCode in the MQSTS structure is MQCC_FAILED and Reason might be either MQRC_CONNECTION_BROKEN or MQRC_Q_MGR_QUIESCING. ObjectType is MQOT_Q_MGR, ObjectName is the name of the queue manager, and ObjectQMgrName is blank.

   The client completed reconnection successfully or was never disconnected.
   CompCode in the MQSTS structure is MQCC_OK and the Reason is MQRC_NONE

   Subsequent calls to MQSTAT return the same results.

   MQSTAT_TYPE_RECONNECTION_ERROR

Suppose you call MQSTAT with Type set to MQSTAT_TYPE_RECONNECTION_ERROR in response to receiving MQRC_RECONNECT_FAILED to an MQI call. Consider these examples.

An authorization failure occurred when a queue was being reopened during reconnection to a different queue manager.

CompCode in the MQSTS structure is MQCC_FAILED and Reason is the reason that the reconnection failed, such as MQRC_NOT_AUTHORIZED. ObjectType is the type of object that caused the problem, such as MQOT_QUEUE, ObjectName is the name of the queue and ObjectQMgrName the name of the queue manager owning the queue.

A socket connection error occurred during reconnection.

CompCode in the MQSTS structure is MQCC_FAILED and Reason is the reason that the reconnection failed, such as MQRC_HOST_NOT_AVAILABLE. ObjectType is MQOT_Q_MGR, ObjectName is the name of the queue manager, and ObjectQMgrName is blank.

Subsequent calls to MQSTAT return the same results.

C invocation

```
MQSTAT (Hconn, StatType, &Stat, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;  /* Connection Handle */
MQLONG StatType; /* Status type */
MQSTS Stat;     /* Status information structure */
MQLONG CompCode; /* Completion code */
MQLONG Reason;  /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQSTAT' USING HCONN, STATTYPE, STAT, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
 01 HCONN      PIC S9(9)      BINARY.
** Status type
 01 STATTYPE   PIC S9(9)      BINARY.
** Status information
 01 STAT.
 COPY CMQSTSV.
** Completion code
 01 COMPCODE   PIC S9(9)      BINARY.
** Reason code qualifying COMPCODE
 01 REASON     PIC S9(9)      BINARY.
```

PL/I invocation

```
call MQSTAT (Hconn, StatType, Stat, Compcode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn       fixed bin(31); /* Connection handle */
dcl StatType    fixed bin(31); /* Status type */
dcl Stat        like MQSTS;    /* Status information structure */
dcl CompCode    fixed bin(31); /* Completion code */
dcl Reason      fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 Assembler invocation

```
CALL MQSTAT,(HCONN,STATTYPE,STAT,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN     DS     F    Connection handle
STATTYPE  DS     F    Status type
STAT      CMQSTSA,    Status information structure
COMPCODE  DS     F    Completion code
REASON    DS     F    Reason code qualifying COMPCODE
```

MQSUB - Register subscription:

Use the MQSUB call to register the applications subscription to a particular topic.

Syntax

MQSUB (*Hconn*, *SubDesc*, *Hobj*, *Hsub*, *Compcode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

SubDesc

Type: MQSD - input/output

This is a structure that identifies the object in use that is being registered by the application. See "MQSD - Subscription descriptor" on page 2549 for more information.

Hobj

Type: MQHOBJ - input/output

This handle represents the access that has been established to obtain the messages sent to this subscription. These messages can either be stored on a specific queue or the queue manager can manage their storage without using a specific queue.

To use a specific queue, you must associate it with the subscription when the subscription is created. You can do this in two ways:

- By using the DEFINE SUB MQSC command and provided that command with the name of a queue object.
- By providing this handle when calling MQSUB with the MQSO_CREATE

  If this handle is provided as an input parameter on the call, it must be a valid object handle returned from a previous MQOPEN call of a queue using at least one of the following options:
  - MQOO_INPUT_*
  - MQOO_BROWSE
  - MQOO_OUTPUT (if the queue is a remote queue)

  If this is not the case, the call fails with MQRC_HOBJ_ERROR. It cannot be an object handle to an alias queue that resolves to a topic object. If so, the call fails with MQRC_HOBJ_ERROR.

If the queue manager is to manage the storage of messages sent to this subscription, this should be set when you create the subscription, by using the MQSO_MANAGED option. The queue manager then returns this handle as an output parameter on the call. The handle that is returned is known as a managed handle. If MQHO_NONE is specified but MQSO_MANAGED is not specified, the call fails with MQRC_HOBJ_ERROR.

When a managed handle is returned to you by the queue manager, you can use it on an MQGET or MQCB call with or without browse options, on an MQINQ call, or on MQCLOSE. You cannot use it on MQPUT, MQSUB, MQSET; attempting to do so fails with MQRC_NOT_OPEN_FOR_OUTPUT, MQRC_HOBJ_ERROR, or MQRC_NOT_OPEN_FOR_SET.

If this subscription is being resumed using the MQSO_RESUME option in the MQSD structure, the handle can be returned to the application in this parameter by setting MQSO_MANAGED to MQHO_NONE. You can do this whether the subscription is using a managed handle or not and it can be useful to provide subscriptions created using DEFINE SUB with the handle to the subscription queue defined on that command. In the case where an administratively created subscription is being resumed, the queue opens with MQOO_INPUT_AS_Q_DEF and MQOO_BROWSE. If you need to specify other options, the application must open the subscription queue explicitly and provide the object handle on the call. If there is a problem opening the queue the call fails with MQRC_INVALID_DESTINATION. If the *Hobj* is provided, it must be equivalent to the *Hobj* in the original MQSUB call. This means if an object handle returned from an MQOPEN call is being provided, the handle must be to the same queue as previously used. If it is not the same queue, the call fails with MQRC_HOBJ_ERROR.

If this subscription is being altered using the MQSO_ALTER option in the MQSD structure, then a different *Hobj* can be provided. Any publications that have been delivered to the queue and were previously identified through this parameter stay on that queue and it is the responsibility of the application to retrieve those messages if the **Hobj** parameter now represents a different queue.

The table summarizes the use of this parameter with various subscription options:

Options	*Hobj*	**Description**
MQSO_CREATE + MQSO_MANAGED	Ignored on input	Creates a subscription with storage of messages managed by the queue manager
MQSO_CREATE	A valid object handle	Creates a subscription providing a specific queue as the destination for messages.
MQSO_RESUME	MQHO_NONE	Resumes a previously created subscription whether it was managed or not, and has the queue manager return the object handle for use by the application.
MQSO_RESUME	A valid, matching, object handle	Resumes a previously created subscription that uses a specific queue as the destination for messages and use an object handle with specific open options.
MQSO_ALTER + MQSO_MANAGED	MQHO_NONE	Alters an existing subscription that was previously using a specific queue, so it is now a managed subscription. The class of destination (managed or not) cannot be changed.
MQSO_ALTER	A valid object handle	Alters an existing subscription, whether it was managed or not, so that it now uses a specific queue. When the MQSO_MANAGED option is not used, the queue provided can be changed, but the class of destination (managed or not) cannot be changed.

Whether it was provided or returned, *Hobj* must be specified on subsequent MQGET or MQCB calls that want to receive the publication messages sent to this subscription.

The *Hobj* handle is no longer valid when the MQCLOSE call is issued on it, or when the unit of processing that defines the scope of the handle terminates (until the application disconnects). The scope of the object handle returned is the same as that of the connection handle specified on the call. See Hconn (MQHCONN) - output for information about handle scope. An MQCLOSE of the *Hobj* handle does not affect the *Hsub* handle.

Hsub

Type: MQHOBJ - output

This handle represents the subscription that has been made. It can be used for two further operations:

- It can be used on a subsequent MQSUBRQ call to request that publications be sent when the MQSO_PUBLICATIONS_ON_REQUEST option has been used when making the subscription.
- It can be used on a subsequent MQCLOSE call to remove the subscription that has been made. The *Hsub* handle ceases to be valid when the MQCLOSE call is issued, or when the unit of processing that defines the scope of the handle terminates. The scope of the object handle returned is the same as that of the connection handle specified on the call. An MQCLOSE of the *Hsub* handle does not affect the *Hobj* handle.

This handle cannot be passed to an MQGET or MQCB call. You must use the **Hobj** parameter. You cannot use this handle on any IBM MQ call other than MQCLOSE or MQSUBRQ. Passing this handle to any other IBM MQ call results in MQRC_HOBJ_ERROR.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion

MQCC_WARNING
Warning (partial completion)

MQCC_FAILED
Call failed

Reason

Type: MQLONG - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK, the reason code is as follows:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED, the reason code is one of the following:

MQRC_CLUSTER_RESOLUTION_ERROR
(2189, X'88D') Cluster name resolution failed.

MQRC_DURABILITY_NOT_ALLOWED
2436 (X'0984') An MQSUB call using the MQSO_DURABLE option failed.

MQRC_FUNCTION_NOT_SUPPORTED
2298 (X'08FA') The function requested is not available in the current environment.

MQRC_HOBJ_ERROR
2019 (X'07E3') Object handle Hobj not valid.

MQRC_IDENTITY_MISMATCH
2434 (X'0982') Subscription name matches existing subscription.

MQRC_NOT_AUTHORIZED
2035 (X'07F3') The user is not authorized to perform the operation.

MQRC_NO_SUBSCRIPTION
2428 (X'097C') The identified subscription name does not exist.

MQRC_OBJECT_STRING_ERROR
2441 (X'0989') Objectstring field not valid.

MQRC_OPTIONS_ERROR
2046 (X'07FE') Options parameter or field contains options that are not valid, or a combination of options that is not valid.

MQRC_Q_MGR_QUIESCING
2161 (X'0871') Queue manager quiescing.

MQRC_RECONNECT_Q_MGR_REQD
2555 (X'09FB'X) The MQCNO_RECONNECT_Q_MGR option is required.

MQRC_RETAINED_MSG_Q_ERROR
2525 (X'09DD') Retained publications which exist for the subscribed topic string, cannot be retrieved.

MQRC_RETAINED_NOT_DELIVERED
2526 (X'09DE') The retained publications which exist for the subscribed topic string, cannot be delivered to the subscription destination queue, and cannot be delivered to the dead-letter queue.

MQRC_SD_ERROR
2424 (X'0978') Subscription descriptor (MQSD) not valid.

MQRC_SELECTION_NOT_AVAILABLE
2551 (X'09F7') The selection string does not follow the IBM MQ selector syntax and no extended message selection provider was available.

MQRC_SELECTION_STRING_ERROR
2519 (X'09D7') The selection string must be specified as described in the MQCHARV structure documentation.

MQRC_SELECTOR_SYNTAX_ERROR
2459 (X'099B') An MQOPEN, MQPUT1, or MQSUB call was issued but a selection string was specified which contained a syntax error.

MQRC_SUB_USER_DATA_ERROR
2431 (X'097F') SubUserData field not valid.

MQRC_SUB_NAME_ERROR
2440 (X'0988') SubName field not valid.

MQRC_SUB_ALREADY_EXISTS
2432 (X'0980') Subscription already exists.

MQRC_SUB_USER_DATA_ERROR
2431 (X'097F') SubUserData field not valid.

MQRC_TOPIC_STRING_ERROR
2425 (X'0979') Topic string is not valid.

MQRC_UNKNOWN_OBJECT_NAME
2085 (X'0825') Object identified in the MQSD ObjectName field cannot be found.

MQRC_SUB_JOIN_NOT_ALTERABLE
29440 (X'7300') Subscription sharing mode is incompatible with existing subscription. This error could be returned when attempting to resume a JMS 2.0 shared subscription in a non-JMS application.

For detailed information about these codes, see Reason codes.

Usage notes

- The subscription is made to a topic, named either using the short name of a pre-defined topic object, the full name of the topic string, or it is formed by the concatenation of two parts. See the description of *ObjectName* and *ObjectString* in "MQSD - Subscription descriptor" on page 2549.
- The queue manager performs security checks when an MQSUB call is issued, to verify that the user identifier under which the application is running has the appropriate level of authority before access is permitted. The appropriate topic object is located in the topic hierarchy and an authority check is made on this topic object to ensure authority to subscribe is set. If the MQSO_MANAGED option is not used, an authority check is made on the destination queue to ensure that authority for output is set. If the MQSO_MANAGED option is used, no authority check is made on the managed queue for output or inquire access.
- If you do not provide an Hobj as input, the MQSUB call allocates two handles, an object handle (Hobj) and a subscription handle (Hsub).
- The Hobj returned on the MQSUB call when the MQSO_MANAGED option is used, can be inquired in order to find out attributes such as the Backout threshold and the Excessive backout requeue name. You can also inquire the name of the managed queue, but you must not attempt to directly open this queue.
- Subscriptions can be grouped allowing only a single publication to be delivered to the group of subscriptions even where more than one of the group matched the publication. Subscriptions are grouped using the MQSO_GROUP_SUB option and in order to group subscriptions they must be
  - using the same named queue (that is not using the MQSO_MANAGED option) on the same queue manager - represented by the Hobj parameter on the MQSUB call
  - share the same SubCorrelId
  - be of the same SubLevel

  These attributes define the set of subscriptions considered to be in the group, and are also the attributes that cannot be altered if a subscription is grouped. Alteration of SubLevel results in MQRC_SUBLEVEL_NOT_ALTERABLE, and alteration of any of the others (which can be changed if a subscription is not grouped) results in MQRC_GROUPING_NOT_ALTERABLE.
- Successful completion of the MQSUB call does not mean that the action completed. To check that this call has completed, see the DEFINE SUB step in Checking that async commands for distributed networks have finished.
- Fields in the MQSD are filled in on return from an MQSUB call which uses the MQSO_RESUME option. The MQSD returned can be passed directly into an MQSUB call which uses the MQSO_ALTER option with any changes you need to make to the subscription applied to the MQSD. Some fields have special considerations as noted in the table.

MQSD output from MQSUB

Field name in MQSD	Special considerations
Access or creation options	Some of the options can be reset on return from the MQSUB call. If you then reuse the MQSD in an MQSUB call, the option you require must be explicitly set.
Durability options, Destination options, Registration Options & Wildcard options	These options are set as appropriate
Publication options	These options are set as appropriate, except for MQSO_NEW_PUBLICATIONS_ONLY which is only applicable to MQSO_CREATE.

MQSD output from MQSUB

Field name in MQSD	Special considerations
Other options	These options are unchanged on return from an MQSUB call. They control how the API call is issued and are not stored with the subscription. They must be set as required on any subsequent MQSUB call reusing the MQSD.
ObjectName	This input only field is unchanged on return from an MQSUB call.
ObjectString	This input only field is unchanged on return from an MQSUB call. The Full topic name used is returned in the *ResObjectString* field, if a buffer is provided.
AlternateUserId and AlternateSecurityId	These input only fields are unchanged on return from an MQSUB call. They control how the API call is issued and are not stored with the subscription. They must set as required on any subsequent MQSUB call reusing the MQSD.
SubExpiry	On return from an MQSUB call using the MQSO_RESUME option, this field is set to the original expiry of the subscription and not the remaining expiry time. If you then reuse the MQSD in an MQSUB call using the MQSO_ALTER option you reset the expiry of the subscription to start counting down again.
SubName	This field is an input field on an MQSUB call and is not changed on output.
SubUserData and SelectionString	These variable length fields are returned on output from an MQSUB call using the MQSO_RESUME option, if a buffer is provided, and also a positive buffer length in *VSBufSize*. If no buffer is provided only the length is returned in the *VSLength* field of the MQCHARV. If the buffer provided is smaller than the space required to return the field, only *VSBufSize* bytes are returned in the provided buffer. If you then reuse the MQSD in an MQSUB call using the MQSO_ALTER option and a buffer is not provided but a non-zero *VSLength* is provided, if that length matches the existing length of the field, no alteration is made to the field.
SubCorrelId and PubAccountingToken	If you do not use MQSO_SET_CORREL_ID, then the *SubCorrelId* is generated by the queue manager. If you do not use MQSO_SET_IDENTITY_CONTEXT, then the *PubAccountingToken* is generated by the queue manager. These fields are returned in the MQSD from an MQSUB call using the MQSO_RESUME option. If they are generated by the queue manager, the generated value is returned on an MQSUB call using the MQSO_CREATE or MQSO_ALTER option.
PubPriority, SubLevel & PubApplIdentityData	These fields are returned in the MQSD.
ResObjectString	This output only field is returned in the MQSD if a buffer is provided.

C invocation

```
MQSUB (Hconn, &SubDesc, &Hobj, &Hsub, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHCONN Hconn;    /* Connection handle */
MQSD    SubDesc;  /* Subscription descriptor */
MQHOBJ  Hobj;     /* Object handle */
MQHOBJ  Hsub;     /* Subscription handle */
MQLONG  CompCode; /* Completion code */
MQLONG  Reason;   /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQSUB' USING HCONN, SUBDESC, HOBJ, HSUB, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN    PIC S9(9) BINARY.
**   Subscription descriptor
 01  SUBDESC.
     COPY CMQSDV.
**   Object handle
 01  HOBJ     PIC S9(9) BINARY.
**   Subscription handle
 01  HSUB     PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON   PIC S9(9) BINARY.
```

PL/I invocation

```
call MQSUB (Hconn, SubDesc, Hobj, Hsub, CompCode, Reason)
```

Declare the parameters as follows:

```
dcl Hconn    fixed bin(31); /* Connection handle */
dcl SubDesc  like MQSD;     /* Subscription descriptor */
dcl Hobj     fixed bin(31); /* Object handle */
dcl Hsub     fixed bin(31); /* Subscription handle */
dcl CompCode fixed bin(31); /* Completion code */
dcl Reason   fixed bin(31); /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQSUB,(HCONN,SUBDESC,HOBJ,HSUB,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN    DS      F  Connection handle
SUBDESC  CMQSDA  ,  Subscription descriptor
HOBJ     DS      F  Object handle
HSUB     DS      F  Subscription handle
COMPCODE DS      F  Completion code
REASON   DS      F  Reason code qualifying COMPCODE
```

MQSUBRQ - Subscription request:

Use the MQSUBRQ call to make a request for the retained publication, when the subscriber has been registered with MQSO_PUBLICATIONS_ON_REQUEST.

Syntax

MQSUBRQ (*Hconn*, *Hsub*, *Action*, *SubRqOpts*, *Compcode*, *Reason*)

Parameters

Hconn
Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hsub
Type: MQHOBJ - input

This handle represents the subscription for which an update is to be requested. The value of *Hsub* was returned from a previous MQSUB call.

Action
Type: MQLONG - input

This parameter controls the particular action that is being requested on the subscription. The following value must be specified:

MQSR_ACTION_PUBLICATION

This action requests that an update publication is sent for the specified topic. It can be used only if the subscriber specified the option MQSO_PUBLICATIONS_ON_REQUEST on the MQSUB call when it made the subscription. If the queue manager has a retained publication for the topic, this is sent to the subscriber. If not, the call fails. If an application is sent a publication which was retained, this is indicated by the MQIsRetained message property of that publication.

Since the topic in the existing subscription represented by the Hsub parameter can contain wildcards, the subscriber might receive multiple retained publications.

SubRqOpts
Type: MQSRO - input/output

These options control the action of MQSUBRQ, see "MQSRO - Subscription request options" on page 2575 for details.

If no options are required, programs written in C or S/390 assembler can specify a null parameter address instead of specifying the address of an MQSRO structure.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion

MQCC_WARNING
Warning (partial completion)

MQCC_FAILED
Call failed

Reason
Type: MQLONG - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_FUNCTION_NOT_SUPPORTED
2298 (X'08FA') The function requested is not available in the current environment.

MQRC_NO_RETAINED_MSG
2437 (X'0985') There are no retained publications currently stored for this topic.

MQRC_OPTIONS_ERROR
2046 (X'07FE') Options parameter or field contains options that are not valid, or a combination of options that is not valid.

MQRC_Q_MGR_QUIESCING
2161 (X'0871') Queue manager quiescing.

MQRC_SRO_ERROR
2438 (X'0986') On the MQSUBRQ call, the Subscription Request Options MQSRO is not valid.

MQRC_RETAINED_MSG_Q_ERROR
2525 (X'09DD') Retained publications which exist for the subscribed topic string, cannot be retrieved.

MQRC_RETAINED_NOT_DELIVERED
2526 (X'09DE') The retained publications which exist for the subscribed topic string, cannot be delivered to the subscription destination queue, and cannot be delivered to the dead-letter queue.

For detailed information about these codes, see Reason codes.

Usage notes

The following usage notes apply to the use of the Action code MQSR_ACTION_PUBLICATION:

1. If this verb completes successfully, the retained publications matching the subscription specified have been sent to the subscription and can be received by using MQGET or MQCB using the Hobj returned on the original MQSUB verb that created the subscription.
2. If the topic subscribed to by the original MQSUB verb that created the subscription contained a wildcard, more than one retained publication can be sent. The number of publications sent as a result of this call is recorded in the NumPubs field in the SubRqOpts structure.
3. If this verb completes with a reason code of MQRC_NO_RETAINED_MSG then there were no currently retained publications for the topic specified.#
4. If this verb completes with a reason code of MQRC_RETAINED_MSG_Q_ERROR or MQRC_RETAINED_NOT_DELIVERED then there are currently retained publications for the topic specified but an error has occurred that that meant they were unable to be delivered.

5. The application must have a current subscription to the topic before it can make this call. If the subscription was made in a previous instance of the application and a valid handle to the subscription is not available, the application must first call MQSUB with the MQSO_RESUME option to obtain a handle to it for use in this call.
6. The publications are sent to the destination that is registered for use with the current subscription of this application. If the publications must be sent somewhere else, the subscription must first be altered using the MQSUB call with the MQSO_ALTER option.

C invocation

```
MQSUB (Hconn, Hsub, Action, &SubRqOpts, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHCONN Hconn;     /* Connection handle */
MQHOBJ  Hsub;      /* Subscription handle */
MQLONG  Action;    /* Action requested by MQSUBRQ */
MQSRO   SubRqOpts; /* Options that control the action of MQSUBRQ */
MQLONG  CompCode;  /* Completion code */
MQLONG  Reason;    /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQSUBRQ' USING HCONN, HSUB, ACTION, SUBRQOPTS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN PIC S9(9) BINARY.
** Subscription handle
01 HSUB PIC S9(9) BINARY.
** Action requested by MQSUBRQ
01 ACTION PIC S9(9) BINARY.
** Options that control the action of MQSUBRQ
01 SUBRQOPTS.
COPY CMQSROV.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON PIC S9(9) BINARY.
```

PL/I invocation

```
call MQSUBRQ (Hconn, Hsub, Action, SubRqOpts, CompCode, Reason)
```

Declare the parameters as follows:

```
dcl Hconn fixed bin(31);    /* Connection handle */
dcl Hsub fixed bin(31);     /* Subscription handle */
dcl Action fixed bin(31);   /* Action requested by MQSUBRQ */
dcl SubRqOpts like MQSRO;   /* Options that control the action of MQSUBRQ */
dcl CompCode fixed bin(31); /* Completion code */
dcl Reason fixed bin(31);   /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQSUBRQ,(HCONN, HSUB, ACTION, SUBRQOPTS,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN DS F Connection handle
HSUB DS F Subscription handle
ACTION DS F Action requested by MQSUBRQ
SUBRQOPTS CMQSROA , Options that control the action of MQSUBRQ
COMPCODE DS F Completion code
REASON DS F Reason code qualifying COMPCODE
```

Attributes of objects

This collection of topics lists only those IBM MQ objects that can be the subject of an MQINQ function call, and gives details of the attributes that can be inquired on and the selectors to be used.

Attributes for the queue manager:

Some queue manager attributes are fixed for particular implementations; others can be changed by using the MQSC command `ALTER QMGR`.

The attributes can also be displayed by using the command `DISPLAY QMGR`. Most queue manager attributes can be inquired by opening a special MQOT_Q_MGR object, and using the MQINQ call with the handle returned.

The following table summarizes the attributes that are specific to the queue manager. The attributes are described in alphabetical order.

Note: The names of the attributes shown in this section are descriptive names used with the MQINQ call; the names are the same as for the PCF commands. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see Script (MQSC) Commands for more information.

Table 272. Attributes for the queue manager

Attribute	Description
AccountingConnOverride	Override accounting settings.
AccountingInterval	How often to write intermediate accounting records.
ActivityConnOverride	Override activity settings.
ActivityTrace	Controls the collection of IBM MQ MQI application activity trace.
AdoptNewMCACheck	Elements checked to determine whether to adopt new MCA.
AdoptNewMCAType	Whether to restart automatically an orphaned instance of an MCA of a particular channel type.
AlterationDate	Date when definition was last changed
AlterationTime	Time when definition was last changed
AuthorityEvent	Controls whether authorization (Not Authorized) events are generated
BridgeEvent	Control attribute for bridge events.
ChannelAutoDef	Controls whether automatic channel definition is permitted
ChannelAutoDefEvent	Controls whether channel automatic-definition events are generated
ChannelAutoDefExit	Name of user exit for automatic channel definition
ChannelEvent	Control attribute for channel events.
ChannelInitiatorControl	Control attribute for channel initiator
ChannelMonitoring	Online monitoring data for channels
ChannelStatistics	Controls collection of statistics data for channels.
ChinitAdapters	Number of adapter subtasks for processing IBM MQ calls.
ChinitDispatchers	Number of dispatchers to use for the channel initiator.
	Reserved for IBM use.
ChinitTraceAutoStart	Whether channel initiator trace should start automatically.
ChinitTraceTableSize	Size of channel initiator's trace data space.
ClusterSenderMonitoringDefault	Online monitoring data default for cluster sender channels
ClusterSenderStatistics	Controls collection of statistics monitoring information for cluster sender channels.
ClusterWorkloadData	User data for cluster workload exit
ClusterWorkloadExit	Name of user exit for cluster workload management
ClusterWorkloadLength	Maximum length of message data passed to cluster workload exit
CLWLMRUChannels	Number of most recently used channels for cluster workload balancing
CLWLUseQ	Cluster workload use remote queue.
CodedCharSetId	Coded character set identifier
CommandEvent	Control attribute for command events.

Table 272. Attributes for the queue manager (continued)

Attribute	Description
CommandInputQName attribute	Command input queue name
CommandLevel	Command level
CommandServerControl attribute	Control attribute for command server.
Configuration Event attribute	Control attribute for configuration events.
DeadLetterQName	Name of dead-letter queue
DEFCLXQ	Default cluster transmission queue type
DefXmitQName	Default transmission queue name
DistLists	Distribution list support
DNSGroup	Name of group for TCP listener when using Workload Manager Dynamic Domain Name Services support.
DNSWLM	Whether TCP listener registers with Workload Manager for Dynamic Domain Name Services.
ExpiryInterval	Interval between scans for expired messages
IGQPutAuthority	Intra-group queuing put authority
IGQUserId	Intra-group queuing user identifier
InhibitEvent	Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated
IPAddressVersion	Version of the Internet Protocol address
IntraGroupqueuing	Intra-group queuing support
ListenerTimer	Time interval between attempts to restart listener after APPC or TCP/IP failure.
LocalEvent	Controls whether local error events are generated
LoggerEvent	Controls whether logger events are generated
LUGroupName	Generic LU name for LU 6.2 listener that handles inbound transmissions for queue-sharing group.
LUName	Name of LU to use for outbound LU 6.2 transmissions.
LU62ARMSuffix	Suffix of SYS1.PARMLIB member APPCPMxx, that nominates LUADD for this channel initiator.
LU62Channels	Maximum number of current channels or connected clients that use LU 6.2.
MaxActiveChannels	Maximum number of channels that can be active at any time.
MaxChannels	Maximum number of current channels.
MaxHandles	Maximum number of handles
MaxMsgLength	Maximum message length in bytes
MaxPriority attribute	Maximum priority
MaxPropertiesLength	Maximum length of property data in bytes
MaxUncommittedMsgs	Maximum number of uncommitted messages within a unit of work
MQIAccounting	Controls collection of accounting information for MQI data.
MQIStatistics	Controls collection of statistics monitoring information for queue manager.
MsgMarkBrowseInterval	Interval after which the queue manager can remove the mark from browsed messages.
OutboundPortMin	With *OutboundPortMin*, defines range of port numbers to use when binding outgoing channels.
OutboundPortMin	With *OutboundPortMax*, defines range of port numbers to use when binding outgoing channels.
PerformanceEvent	Controls whether performance-related events are generated
Platform	Platform on which the queue manager is running
PubSubNPInputMsg	Whether to discard (or keep) an undelivered input message
PubSubNPResponse	Controls the behavior of undelivered
PubSubMaxMsgRetryCount	The number of retries when processing (under syncpoint) a failed command message
PubSubSyncPoint	Whether only persistent (or all) messages should be processed under syncpoint
PubSubMode	Whether the queued publish/subscribe interface is running
QMgrDesc	Queue manager description
QMgrIdentifier	Unique internally generated identifier of queue manager
QMgrName	Queue manager name
QSGName	Name of queue-sharing group
QueueAccounting	Controls collection of accounting information for queues.
QueueMonitoring	Online monitoring data for queues
QueueStatistics	Controls collection of statistics data for queues.
ReceiveTimeout	How long TCP/IP channel waits for data before returning to inactive state.

Table 272. Attributes for the queue manager (continued)

Attribute	Description
ReceiveTimeoutMin	Qualifier for *ReceiveTimeout*.
ReceiveTimeoutType	Minimum time that TCP/IP channel waits for data before returning to inactive state.
RemoteEvent	Controls whether remote error events are generated
RepositoryName	Name of cluster for which this queue manager provides repository services
RepositoryNamelist	Name of namelist object containing names of clusters for which this queue manager provides repository services
ScyCase	Case of security profiles
SharedQMgrName	Shared queue queue manager name
"SPLCAP" on page 2828	IBM MQ Advanced Message security protection for a queue manager turned on or off.
SSLCRLNamelist [1]	Name of namelist object containing names of authentication information objects.
SSLCryptoHardware [1]	Cryptographic hardware configuration string.
SSLEvent	Control attribute for TLS events.
SSLFIPSRequired	Use only FIPS-certified algorithms for cryptography.
SSLKeyRepository [1]	Location of TLS key repository.
SSLKeyResetCount	TLS key reset count.
SSLTasks [1]	Number of server subtasks for processing TLS calls.
StatisticsInterval	How often to write statistics monitoring data.
StartStopEvent	Controls whether start and stop events are generated
SyncPoint	Syncpoint availability
TCPChannels	Maximum number of current channels or connected clients that use TCP/IP.
TCPKeepAlive	Whether to use TCP KEEPALIVE to check other end of connection.
TCPName	Name of TCP/IP system that you are using.
TCPStackType	How channel initiator can use TCP/IP addresses.
TraceRouteRecording attribute	Controls recording of trace-route information.
TriggerInterval	Trigger-message interval
Version	Version
XrCapability	Specifies whether Telemetry commands are supported.

Notes:

1. This attribute cannot be inquired using the MQINQ call, and is not described in this section. See Change Queue Manager for details of this attribute.

Related information:

Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client

Federal Information Processing Standards (FIPS) for UNIX, Linux, and Windows

AccountingConnOverride (MQLONG):

This allows applications to override the setting of the ACCTMQI and ACCTQDATA values in the Qmgr attribute.

The value is one of the following:

MQMON_DISABLED
: Applications cannot override the setting of the ACCTMQI and ACCTQ Qmgr attributes using the Options field in the MQCNO structure on the MQCONNX call. This is the default value.

MQMON_ENABLED
: Applications can override the ACCTQ and ACCTMQI Qmgr attributes using the Options field in the MQCNO structure.

Changes to this value are only effective for connections to the queue manager after the change to the attribute.

This attribute is supported only on the following platforms:

-  IBM i
-  UNIX
-  Windows

To determine the value of this attribute, use the MQIA_ACCOUNTING_CONN_OVERRIDE selector with the MQINQ call.

AccountingInterval (MQLONG):

This specifies how long before intermediate accounting records are written (in seconds).

The value is an integer in the range 0 to 604800, with a default value of 1800 (30 minutes). Specify 0 to turn off intermediate records.

This attribute is supported only on the following platforms:

-  IBM i
-  UNIX
-  Linux
-  Windows

To determine the value of this attribute, use the MQIA_ACCOUNTING_INTERVAL selector with the MQINQ call.

ActivityConnOverride (MQLONG):

This allows applications to override the setting of the ACTVTRC value in the queue manager attribute.

The value is one of the following:

MQMON_DISABLED
: Applications cannot override the setting of the `ACTVTRC` queue manager attribute using the Options field in the MQCNO structure on the MQCONNX call. This is the default value.

MQMON_ENABLED
: Applications can override the `ACTVTRC` queue manager attribute using the Options field in the MQCNO structure.

Changes to this value are only effective for connections to the queue manager after the change to the attribute.

This attribute is supported only on the following platforms:

-  IBM i
-  UNIX
-  Windows

To determine the value of this attribute, use the MQIA_ACTIVITY_CONN_OVERRIDE selector with the `MQINQ` call.

ActivityTrace (MQLONG):

This controls the collection of IBM MQ MQI application activity trace.

The value is one of the following:

MQMON_ON
Collect IBM MQ MQI application activity trace.

MQMON_OFF
Do not collect IBM MQ MQI application activity trace. This is the default value.

If you set the queue manager attribute ACTVCONO to ENABLED, this value might be overridden for individual connections using the Options field in the MQCNO structure.

Changes to this value are only effective for connections to the queue manager after the change to the attribute.

This attribute is supported only on the following platforms:

- 

IBM i
- 

UNIX
- 

Windows

To determine the value of this attribute, use the MQIA_ACTIVITY_TRACE selector with the MQINQ call.

AdoptNewMCACheck (MQLONG):

This defines the elements to check to determine whether to adopt an MCA when a new inbound channel is detected that has the same name as an MCA that is already active

The value is one of the following:

MQADOPT_CHECK_Q_MGR_NAME
Check the queue manager name.

MQADOPT_CHECK_NET_ADDR
Check the network address.

MQADOPT_CHECK_ALL
Check the queue manager name and network address. If possible, perform this check to protect your channels from being shut down, inadvertently or maliciously. This is the default value.

MQADOPT_CHECK_NONE
Do not check any elements.

Changes to this attribute take effect the next time that a channel attempts to adopt a channel.

This attribute is supported only on z/OS.

To determine the value of this attribute, use the MQIA_ADOPTNEWMCA_CHECK selector with the MQINQ call.

AdoptNewMCAType (MQLONG):

This specifies whether to restart automatically an orphaned instance of an MCA of a particular channel type when a new inbound channel request matching the AdoptNewMCACheck attribute is detected

It is one of the following values:

MQADOPT_TYPE_NO
Adopting orphaned channel instances is not required. This is the default value.

MQADOPT_TYPE_ALL
Adopt all channel types.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_ADOPTNEWMCA_TYPE selector with the MQINQ call.

AlterationDate (MQCHAR12):

This is the date when the definition was last changed. The format of the date is `YYYY-MM-DD`, padded with two trailing blanks to make the length 12 bytes.

To determine the value of this attribute, use the MQCA_ALTERATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_DATE_LENGTH.

AlterationTime (MQCHAR8):

This is the time when the definition was last changed. The format of the time is `HH.MM.SS`.

To determine the value of this attribute, use the MQCA_ALTERATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_TIME_LENGTH.

AuthorityEvent (MQLONG):

This controls whether authorization (Not Authorized) events are generated. It is one of the following values:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_AUTHORITY_EVENT selector with the MQINQ call.

BridgeEvent (MQLONG):

This specifies whether IMS bridge events are generated.

The value is one of the following:

MQEVR_ENABLED
Generate IMS bridge events, as follows:
MQRC_BRIDGE_STARTED
MQRC_BRIDGE_STOPPED

MQEVR_DISABLED
Do not generate IMS bridge events; this is the default value.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_BRIDGE_EVENT selector with the MQINQ call.

ChannelAutoDef (MQLONG):

This attribute controls the automatic definition of channels of type MQCHT_RECEIVER and MQCHT_SVRCONN. Automatic definition of MQCHT_CLUSSDR channels is always enabled. The value is one of the following:

MQCHAD_DISABLED
Channel auto-definition disabled.

MQCHAD_ENABLED
Channel auto-definition enabled.

Multi This attribute is supported only on Multiplatforms.

To determine the value of this attribute, use the MQIA_CHANNEL_AUTO_DEF selector with the MQINQ call.

ChannelAutoDefEvent (MQLONG):

This controls whether channel automatic-definition events are generated. It applies to channels of type MQCHT_RECEIVER, MQCHT_SVRCONN, and MQCHT_CLUSSDR. The value is one of the following:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

Multi This attribute is supported only on Multiplatforms.

To determine the value of this attribute, use the MQIA_CHANNEL_AUTO_DEF_EVENT selector with the MQINQ call.

ChannelAutoDefExit (MQCHARn):

This is the name of the user exit for automatic channel definition. If this name is nonblank, and *ChannelAutoDef* has the value MQCHAD_ENABLED, the exit is called each time that the queue manager is about to create a channel definition. This applies to channels of type MQCHT_RECEIVER, MQCHT_SVRCONN, and MQCHT_CLUSSDR. The exit can then do one of the following:

- Create the channel definition without change.
- Modify the attributes of the channel definition that is created.
- Suppress creation of the channel entirely.

Note: Both the length and the value of this attribute are environment specific. See the introduction to the MQCD structure in "MQCD - Channel definition" on page 3559 for details of the value of this attribute in various environments.

z/OS On z/OS, this attribute applies only to cluster-sender and cluster-receiver channels.

To determine the value of this attribute, use the MQCA_CHANNEL_AUTO_DEF_EXIT selector with the MQINQ call. The length of this attribute is given by MQ_EXIT_NAME_LENGTH.

ChannelEvent (MQLONG):

This specifies whether channel events are generated.

It is one of the following values:

MQEVR_EXCEPTION
Only generate the following channel events:
- MQRC_CHANNEL_ACTIVATED
- MQRC_CHANNEL_CONV_ERROR
- MQRC_CHANNEL_NOT_ACTIVATED
- MQRC_CHANNEL_STOPPED with the following ReasonQualifiers:
  MQRQ_CHANNEL_STOPPED_ERROR
  MQRQ_CHANNEL_STOPPED_RETRY
  MQRQ_CHANNEL_STOPPED_DISABLED

  MQRC_CHANNEL_STOPPED_BY_USER

MQEVR_ENABLED
Generate all channel events. That is, in addition to those generated by EXCEPTION, generate the following channel events:
- MQRC_CHANNEL_STARTED
- MQRC_CHANNEL_STOPPED with the following ReasonQualifier:
  MQRQ_CHANNEL_STOPPED_OK

MQEVR_DISABLED
Do not generate channel events; this is the default value.

To determine the value of this attribute, use the MQIA_CHANNEL_EVENT selector with the MQINQ call.

ChannelInitiatorControl (MQLONG):

This specifies whether the channel initiator is to be started when the queue manager starts.

It is one of the following values:

MQSVC_CONTROL_MANUAL
The channel initiator is not to be started automatically.

MQSVC_CONTROL_Q_MGR
The channel initiator is to be started automatically when the queue manager starts.

To determine the value of this attribute, use the MQIA_CHINIT_CONTROL selector with the MQINQ call.

ChannelMonitoring (MQLONG) on Multiplatforms: 

This attribute specifies online monitoring data for channels.

The value is one of the following:

MQMON_NONE
Disable data collection for channel monitoring for all channels regardless of the setting of the MONCHL channel attribute. This is the default value.

MQMON_OFF
Turn monitoring data collection off for channels that specify QMGR in the MONCHL channel attribute.

MQMON_LOW
Turn monitoring data collection on with a low ratio of data collection for channels specifying QMGR in the MONCHL channel attribute.

MQMON_MEDIUM
Turn monitoring data collection on with a moderate ratio of data collection for channels specifying QMGR in the MONCHL channel attribute.

MQMON_HIGH
Turn monitoring data collection on with a high ratio of data collection for channels specifying QMGR in the MONCHL channel attribute.

z/OS On z/OS systems, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying `LOW`, `MEDIUM`, or `HIGH` makes no difference to your results.

To determine the value of this attribute, use the MQIA_MONITORING_CHANNEL selector with the MQINQ call.

ChannelStatistics (MQLONG) on Multiplatforms: Multi

This controls the collection of statistics data for channels.

The value is one of the following:

MQMON_NONE
Disable data collection for channel statistics for all channels regardless of the setting of the STATCHL channel attribute. This is the default value.

MQMON_OFF
Turn statistics data collection off for channels that specify QMGR in the STATCHL channel attribute.

MQMON_LOW
Turn statistics data collection on with a low ratio of data collection for channels specifying QMGR in the STATCHL channel attribute.

MQMON_MEDIUM
Turn statistics data collection on with a moderate ratio of data collection for channels specifying QMGR in the STATCHL channel attribute.

MQMON_HIGH
Turn statistics data collection on with a high ratio of data collection for channels specifying QMGR in the STATCHL channel attribute.

For most systems you are recommended to use MEDIUM. However, for a channel that processes a high volume of messages each second, you might want to reduce the sampling level by selecting LOW. Also, for a channel that processes only a few messages, and for which the most current information is important, you might want to select HIGH.

z/OS On z/OS systems, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying `LOW`, `MEDIUM`, or `HIGH` makes no difference to your results.

To determine the value of this attribute, use the MQIA_STATISTICS_CHANNEL selector with the MQINQ call.

ChinitAdapters (MQLONG):

This is the number of adapter subtasks to use to process IBM MQ calls. The value must be 0 - 9999, with a default value of 8.

The ratio of adapters to dispatchers (the ChinitDispatchers attribute) should be about 8 to 5. However, if you have only few channels, you do not have to decrease the value of this parameter from the default value. You can use the following values: for a test system, 8 (default); for a production system, 20. Ideally, you should have 20 adapters, which gives greater parallelism of IBM MQ calls. This is important for persistent messages. Fewer adapters might be better for nonpersistent messages.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_CHINIT_ADAPTERS selector with the MQINQ call.

ChinitDispatchers (MQLONG):

This is the number of dispatchers to use for the channel initiator. The value must be 0 - 9999, with a default value of 5.

As a guideline, allow one dispatcher for 50 current channels. However, if you have only few channels, you do not have to decrease the value of this attribute from the default value. If you are using TCP/IP, the greatest number of dispatchers that are used for TCP/IP channels is 100, even if you specify a larger value here. You can use the following settings: test systems, 5 (the default); production systems, 20 (you need 20 dispatchers to handle up to 1000 active channels).

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_CHINIT_DISPATCHERS selector with the MQINQ call.

ChinitTraceAutoStart (MQLONG):

This specifies whether to start channel initiator trace automatically.

The value is one of the following:

MQTRAXSTR_YES
Start channel initiator trace automatically. This is the default value.

MQTRAXSTR_NO
Do not start channel initiator trace automatically.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_CHINIT_TRACE_AUTO_START selector with the MQINQ call.

ChinitTraceTableSize (MQLONG):

This is the size of the channel initiator's trace data space (in MB).

The value must be in the range 0 through 2048, with a default value of 2.

Note: Whenever you use large z/OS data spaces, ensure that you have sufficient auxiliary storage on your system to support any related z/OS paging activity. You might also need to increase the size of your SYS1.DUMP data sets.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_CHINIT_TRACE_TABLE_SIZE selector with the MQINQ call.

ClusterSenderMonitoringDefault (MQLONG):

This specifies the value to be substituted for the ChannelMonitoring attribute of automatically-defined cluster sender channels.

The value is one of the following:

MQMON_Q_MGR
Collection of online monitoring data is inherited from the setting of the queue manager **ChannelMonitoring** attribute. This is the default value.

MQMON_OFF
Monitoring for the channel is disabled

MQMON_LOW
Unless *ChannelMonitoring* is MQMON_NONE, monitoring is enabled with a low rate of data collection with a minimal effect on system performance. The data collected is not likely to be the most current.

MQMON_MEDIUM
Unless *ChannelMonitoring* is MQMON_NONE, monitoring is enabled with a moderate rate of data collection with limited effect on system performance.

MQMON_HIGH
Unless *ChannelMonitoring* is MQMON_NONE, monitoring is enabled with a high rate of data collection with a likely effect on system performance. The data collected is the most current available.

To determine the value of this attribute, use the MQIA_MONITORING_AUTO_CLUSSDR selector with the MQINQ call.

ClusterSenderStatistics (MQLONG) on Multiplatforms: Multi

Because cluster sender channels can be automatically defined from the definition of CLUSRCVR in the repository, you cannot alter the setting of the STATCHL attribute for these auto-defined cluster sender channels using ALTER channel. For these channels the decision of whether to collect online monitoring data is based on the setting of this queue manager attribute.

The value is one of the following:

MQMON_Q_MGR
Statistics data collection for auto-defined cluster sender channels is based on the value of the queue manager attribute STATCHL. This is the default value.

MQMON_OFF
Switch off statistics data collection for auto-defined cluster sender channels.

MQMON_LOW
Enable statistics data collection for auto-defined cluster sender channels with a low ratio of data collection.

MQMON_MEDIUM
Enable statistics data collection for auto-defined cluster sender channels with a moderate ratio of data collection.

MQMON_HIGH
Enable statistics data collection for auto-defined cluster sender channels with a high ratio of data collection.

For most systems we recommend MEDIUM. However, for an auto-defined cluster sender channel that processes a high volume of messages each second, you might want to reduce the sampling level by selecting LOW. Also, for a channel that processes only a few messages, and for which the most current information is important, you might want to select HIGH.

z/OS On z/OS systems, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying `LOW`, `MEDIUM`, or `HIGH` makes no difference to your results.

To determine the value of this attribute, use the MQIA_STATISTICS_AUTO_CLUSSDR selector with the MQINQ call.

ClusterWorkloadData (MQCHAR32):

This is a user-defined 32-byte character string that is passed to the cluster workload exit when it is called. If there is no data to pass to the exit, the string is blank.

To determine the value of this attribute, use the MQCA_CLUSTER_WORKLOAD_DATA selector with the MQINQ call.

ClusterWorkloadExit (MQCHARn):

This is the name of the user exit for cluster workload management. If this name is not blank, the exit is called each time that a message is put to a cluster queue or moved from one cluster-sender queue to another. The exit can then either accept the queue instance selected by the queue manager as the destination for the message, or select another queue instance.

Note: Both the length and the value of this attribute are environment specific.

To determine the value of this attribute, use the MQCA_CLUSTER_WORKLOAD_EXIT selector with the MQINQ call. The length of this attribute is given by MQ_EXIT_NAME_LENGTH.

ClusterWorkloadLength (MQLONG):

This is the maximum length of message data that is passed to the cluster workload exit. The actual length of data passed to the exit is the minimum of the following:

- The length of the message.
- The queue manager's **`MaxMsgLength`** attribute.
- The **`ClusterWorkloadLength`** attribute.

To determine the value of this attribute, use the MQIA_CLUSTER_WORKLOAD_LENGTH selector with the MQINQ call.

CLWLMRUChannels (MQLONG):

This specifies the maximum number of most-recently-used cluster channels, to be considered for use by the cluster workload choice algorithm.

This is a value in the range 1 through 999999999.

To determine the value of this attribute, use the MQIA_CLWL_MRU_CHANNELS selector with the MQINQ call.

CLWLUseQ (MQLONG):

This specifies whether to use remote queues for the cluster workload.

The value is one of the following:

MQCLWL_USEQ_ANY
Use both local and remote queues.

MQCLWL_USEQ_LOCAL
Do not use remote queues. This is the default value.

To determine the value of this attribute, use the MQIA_CLWL_USEQ selector with the MQINQ call.

CodedCharSetId (MQLONG):

This defines the character set used by the queue manager for all character string fields defined in the MQI such as the names of objects, and queue creation date and time. The character set must be one that has single-byte characters for the characters that are valid in object names. It does not apply to application data carried in the message. The value depends on the environment:

- On z/OS, the value is set from the system parameters when the queue manager is started; the default value is 500.
- On Windows, the value is the primary CODEPAGE of the user creating the queue manager.
- On IBM i, the value is that which is set in the environment when the queue manager is first created.
- On UNIX, the value is the default CODESET for the locale of the user creating the queue manager.

To determine the value of this attribute, use the MQIA_CODED_CHAR_SET_ID selector with the MQINQ call.

CommandEvent (MQLONG):

This specifies whether command events are generated, as follows:

MQEVR_DISABLED
Do not generate command events. This is the default.

MQEVR_ENABLED
Generate command events.

MQEVR_NO_DISPLAY
Command events are generated for all successful commands other than MQINQ.

To determine the value of this attribute, use the MQIA_COMMAND_EVENT selector with the MQINQ call.

CommandInputQName (MQCHAR48):

This is the name of the command input queue defined on the local queue manager. This is a queue to which users can send commands, if authorized to do so. The name of the queue depends on the environment:

- On z/OS, the name of the queue is SYSTEM.COMMAND.INPUT; MQSC and PCF commands can be sent to it. See The MQSC commands for details of MQSC commands and Definitions of the Programmable Command Formats for details of PCF commands.
- In all other environments, the name of the queue is SYSTEM.ADMIN.COMMAND.QUEUE, and only PCF commands can be sent to it. However, an MQSC command can be sent to this queue if the MQSC command is enclosed within a PCF command of type MQCMD_ESCAPE. See Escape for information about the Escape command.

To determine the value of this attribute, use the MQCA_COMMAND_INPUT_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

CommandLevel (MQLONG):

This indicates the level of system control commands supported by the queue manager. This can be one of the following values:

MQCMDL_LEVEL_1
Level 1 of system control commands.

This value is returned by the following versions:

- MQSeries for AIX Version 2.2
- MQSeries for MVS/ESA
  - Version 1.1.1
  - Version 1.1.2
  - Version 1.1.3
- MQSeries for OS/400
  - Version 2.3
  - Version 3.1
  - Version 3.6
- MQSeries for Windows Version 2.0

MQCMDL_LEVEL_101
MQSeries for Windows Version 2.0.1.

MQCMDL_LEVEL_110
MQSeries for Windows Version 2.1.

MQCMDL_LEVEL_114
MQSeries for Version 1.1.4.

MQCMDL_LEVEL_120
MQSeries for Version 1.2.0.

MQCMDL_LEVEL_200
MQSeries for Windows NT Version 2.0.

MQCMDL_LEVEL_210
MQSeries for OS/390 Version 2.1.0.

MQCMDL_LEVEL_220
Level 220 of system control commands.

This value is returned by the following versions:

- MQSeries for AT&T GIS UNIX Version 2.2
- MQSeries for SINIX and DC/OSx Version 2.2
- MQSeries for SunOS Version 2.2
- MQSeries for Tandem NonStop Kernel Version 2.2

MQCMDL_LEVEL_221
Level 221 of system control commands.

This value is returned by the following versions:

- MQSeries for AIX Version 2.2.1

MQCMDL_LEVEL_320
Level 320 of system control commands.

This value is returned by the following versions of:

- MQSeries for OS/400
  - Version 3.2
  - Version 3.7

MQCMDL_LEVEL_420
Level 420 of system control commands.

This value is returned by the following versions:

- MQSeries for IBM i
  - Version 4.2.0
  - Version 4.2.1

MQCMDL_LEVEL_500
Level 500 of system control commands.

This value is returned by the following versions:

- MQSeries for AIX Version 5.0
- MQSeries for HP-UX Version 5.0
- MQSeries for Solaris Version 5.0
- MQSeries for Windows NT Version 5.0

MQCMDL_LEVEL_510
Level 510 of system control commands.

This value is returned by the following versions:

- MQSeries for AIX Version 5.1
- MQSeries for AS/400 Version 5.1
- MQSeries for HP-UX Version 5.1
- MQSeries for IBM WebSphere MQ Version 5.1
- MQSeries for Compaq Tru64 UNIX Version 5.1
- MQSeries for Solaris Version 5.1
- MQSeries for Windows NT Version 5.1

MQCMDL_LEVEL_520
Level 520 of system control commands.

This value is returned by the following versions:

- MQSeries for AIX Version 5.2
- MQSeries for AS/400 Version 5.2
- MQSeries for HP-UX Version 5.2
- MQSeries for Linux Version 5.2
- MQSeries for OS/390 Version 5.2
- MQSeries for Sun Solaris Version 5.2
- MQSeries for Windows NT Version 5.2

MQCMDL_LEVEL_530
Level 530 of system control commands.

This value is returned by the following versions:

- IBM WebSphere MQ for AIX Version 5.3
- IBM WebSphere MQ for HP-UX Version 5.3
- IBM WebSphere MQ for iSeries Version 5.3
- IBM WebSphere MQ for Linux for Intel Version 5.3

- IBM WebSphere MQ for Linux for zSeries Version 5.3
- IBM WebSphere MQ for Solaris Version 5.3
- IBM WebSphere MQ for Windows Version 5.3
- IBM WebSphere MQ for z/OS Version 5.3

MQCMDL_LEVEL_600
Level 600 of system control commands.

This value is returned by the following versions:
- IBM WebSphere MQ for AIX Version 6.0
- IBM WebSphere MQ for HP-UX Version 6.0
- IBM WebSphere MQ for iSeries Version 6.0
- IBM WebSphere MQ for Linux Version 6.0
- IBM WebSphere MQ for Solaris Version 6.0
- IBM WebSphere MQ for Windows Version 6.0
- IBM WebSphere MQ for z/OS Version 6.0

MQCMDL_LEVEL_700
Level 700 of system control commands.

This value is returned by the following versions:
- IBM WebSphere MQ for AIX Version 7.0
- IBM WebSphere MQ for HP-UX Version 7.0
- IBM WebSphere MQ for IBM i Version 7.0
- IBM WebSphere MQ for Linux Version 7.0
- IBM WebSphere MQ for Solaris Version 7.0
- IBM WebSphere MQ for Windows Version 7.0
- IBM WebSphere MQ for z/OS Version 7.0

MQCMDL_LEVEL_701
Level 701 of system control commands.

This value is returned by the following versions:
- IBM WebSphere MQ for AIX Version 7.0.1
- IBM WebSphere MQ for HP-UX Version 7.0.1
- IBM WebSphere MQ for IBM i Version 7.0.1
- IBM WebSphere MQ for Linux Version 7.0.1
- IBM WebSphere MQ for Solaris Version 7.0.1
- IBM WebSphere MQ for Windows Version 7.0.1
- IBM WebSphere MQ for z/OS Version 7.0.1

MQCMDL_LEVEL_710
Level 710 of system control commands.

This value is returned by the following versions:
- IBM WebSphere MQ for AIX Version 7.1
- IBM WebSphere MQ for HP-UX Version 7.1
- IBM WebSphere MQ for IBM i Version 7.1
- IBM WebSphere MQ for Linux Version 7.1
- IBM WebSphere MQ for Solaris Version 7.1
- IBM WebSphere MQ for Windows Version 7.1
- IBM WebSphere MQ for z/OS Version 7.1

MQCMDL_LEVEL_750
Level 750 of system control commands.

This value is returned by the following versions:

- IBM WebSphere MQ for AIX Version 7.5
- IBM WebSphere MQ for HP-UX Version 7.5
- IBM WebSphere MQ for IBM i Version 7.5
- IBM WebSphere MQ for Linux Version 7.5
- IBM WebSphere MQ for Solaris Version 7.5
- IBM WebSphere MQ for Windows Version 7.5

MQCMDL_LEVEL_800
Level 800 of system control commands.

This value is returned by the following versions:

- IBM WebSphere MQ for AIX Version 8.0
- IBM WebSphere MQ for HP-UX Version 8.0
- IBM WebSphere MQ for IBM i Version 8.0
- IBM WebSphere MQ for Linux Version 8.0
- IBM WebSphere MQ for Solaris Version 8.0
- IBM WebSphere MQ for Windows Version 8.0
- IBM WebSphere MQ for z/OS Version 8.0

MQCMDL_LEVEL_801
Level 801 of system control commands.

This value is returned by the following versions:

- IBM MQ for AIX Version 8.0.0.2
- IBM MQ for HP-UX Version 8.0.0.2
- IBM MQ for IBM i Version 8.0.0.2
- IBM MQ for Linux Version 8.0.0.2
- IBM MQ for Solaris Version 8.0.0.2

MQCMDL_LEVEL_802
Level 802 of system control commands.

This value is returned by the following versions:

- IBM MQ for AIX Version 8.0.0.3
- IBM MQ for HP-UX Version 8.0.0.3
- IBM MQ for IBM i Version 8.0.0.3
- IBM MQ for Linux Version 8.0.0.3
- IBM MQ for Solaris Version 8.0.0.3
- IBM MQ for Windows Version 8.0.0.3

V 9.0.0 **MQCMDL_LEVEL_900**
Level 900 of system control commands.

This value is returned by the following versions:

- IBM MQ for AIX Version 9.0
- IBM MQ for HP-UX Version 9.0
- IBM MQ for IBM i Version 9.0
- IBM MQ for Linux Version 9.0
- IBM MQ for Solaris Version 9.0

- IBM MQ for Windows Version 9.0
- IBM MQ for z/OS Version 9.0

MQCMDL_LEVEL_901
Level 901 of system control commands.

This value is returned by the following versions:
- IBM MQ for Linux Version 9.0.1
- IBM MQ for Windows Version 9.0.1
- IBM MQ for z/OS Version 9.0.1

V 9.0.2 **MQCMDL_LEVEL_902**
Level 902 of system control commands.

This value is returned by the following versions:
- IBM MQ for Linux Version 9.0.2
- IBM MQ for Windows Version 9.0.2
- IBM MQ for z/OS Version 9.0.2

MQCMDL_LEVEL_903
Level 903 of system control commands.

This value is returned by the following versions:
- IBM MQ for Linux Version 9.0.3
- IBM MQ for Windows Version 9.0.3
- IBM MQ for z/OS Version 9.0.3

V 9.0.4 **MQCMDL_LEVEL_904**
Level 904 of system control commands.

This value is returned by the following versions:
- IBM MQ for AIX Version 9.0.4
- IBM MQ for Linux Version 9.0.4
- IBM MQ for Windows Version 9.0.4
- IBM MQ for z/OS Version 9.0.4

The set of system control commands that corresponds to a particular value of the **CommandLevel** attribute varies according to the value of the **Platform** attribute; both must be used to decide which system control commands are supported.

To determine the value of this attribute, use the MQIA_COMMAND_LEVEL selector with the MQINQ call.

CommandServerControl (MQLONG):

Specifies whether the command server is to be started when the queue manager starts.

The value can be any of the following values:

MQSVC_CONTROL_MANUAL
The command server is not to be started automatically.

MQSVC_CONTROL_Q_MGR
The command server is to be started automatically when the queue manager starts.

This attribute is not supported on z/OS.

To determine the value of this attribute, use the MQIA_CMD_SERVER_CONTROL selector with the MQINQ call.

ConfigurationEvent (MQLONG):

Controls whether configuration events are generated.

To determine the value of this attribute, use the MQIA_CONFIGURATION_EVENT selector with the MQINQ call.

The value can be any of the following values:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

DeadLetterQName (MQCHAR48):

This is the name of a queue defined on the local queue manager as the dead-letter (undelivered-message) queue. Messages are sent to this queue if they cannot be routed to their correct destination.

For example, messages are put on this queue when:

- A message arrives at a queue manager, destined for a queue that is not yet defined on that queue manager
- A message arrives at a queue manager, but the queue for which it is destined cannot receive it because, possibly:
  - The queue is full
  - Put requests are inhibited
  - The sending node does not have authority to put messages on the queue

Applications can also put messages on the dead-letter queue.

Report messages are treated in the same way as ordinary messages; if the report message cannot be delivered to its destination queue (usually the queue specified by the *ReplyToQ* field in the message descriptor of the original message), the report message is placed on the dead-letter (undelivered-message) queue.

Note: Messages that have passed their expiry time (see MQMD - Expiry field) are **not** transferred to this queue when they are discarded. However, an expiration report message (MQRO_EXPIRATION) is still generated and sent to the *ReplyToQ* queue, if requested by the sending application.

Messages are not put on the dead-letter (undelivered-message) queue when the application that issued the put request has been notified synchronously of the problem by means of the reason code returned by the MQPUT or MQPUT1 call (for example, a message put on a local queue for which put requests are inhibited).

Messages on the dead-letter (undelivered-message) queue sometimes have their application message data prefixed with an MQDLH structure. This structure contains extra information that indicates why the message was placed on the dead-letter (undelivered-message) queue. See “MQDLH - Dead-letter header” on page 2309 for more details of this structure.

This queue must be a local queue, with a **Usage** attribute of MQUS_NORMAL.

If a queue manager does not support a dead-letter (undelivered-message) queue, or one has not been defined, the name is all blanks. All IBM MQ queue managers support a dead-letter (undelivered-message) queue, but by default it is not defined.

If the dead-letter (undelivered-message) queue is not defined, full, or unusable for some other reason, a message which would have been transferred to it by a message channel agent is retained instead on the transmission queue.

To determine the value of this attribute, use the MQCA_DEAD_LETTER_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

DefClusterXmitQueueType (MQLONG):

The DefClusterXmitQueueType attribute controls which transmission queue is selected by default by cluster-sender channels to get messages from, to send the messages to cluster-receiver channels.

The values of DefClusterXmitQueueType are MQCLXQ_SCTQ or MQCLXQ_CHANNEL.

MQCLXQ_SCTQ

All cluster-sender channels send messages from SYSTEM.CLUSTER.TRANSMIT.QUEUE. The correlID of messages placed on the transmission queue identifies which cluster-sender channel the message is destined for.

SCTQ is set when a queue manager is defined. This behavior is implicit in versions of IBM WebSphere MQ, earlier than Version 7.5. In earlier versions, the queue manager attribute DefClusterXmitQueueType was not present.

MQCLXQ_CHANNEL

Each cluster-sender channel sends messages from a different transmission queue. Each transmission queue is created as a permanent dynamic queue from the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE.

If the queue manager attribute, DefClusterXmitQueueType, is set to CHANNEL, the default configuration is changed to cluster-sender channels being associated with individual cluster transmission queues. The transmission queues are permanent-dynamic queues created from the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE. Each transmission queue is associated with one cluster-sender channel. As one cluster-sender channel services a cluster transmission queue, the transmission queue contains messages for only one queue manager in one cluster. You can configure clusters so that each queue manager in a cluster contains only one cluster queue. In this case, the message traffic from a queue manager to each cluster queue is transferred separately from messages to other queues.

To query the value, call MQINQ, or send an Inquire Queue Manager (MQCMD_INQUIRE_Q_MGR) PCF command, setting the MQIA_DEF_CLUSTER_XMIT_Q_TYPE selector. To change the value, send a Change Queue Manager (MQCMD_CHANGE_Q_MGR) PCF command, setting the MQIA_DEF_CLUSTER_XMIT_Q_TYPE selector.

Related reference:

"MQINQ - Inquire object attributes" on page 2700
The MQINQ call returns an array of integers and a set of character strings containing the attributes of an object.

Related information:

Change Queue Manager
The Change Queue Manager (MQCMD_CHANGE_Q_MGR) command changes the specified attributes of the queue manager.

Inquire Queue Manager
The Inquire Queue Manager (**MQCMD_INQUIRE_Q_MGR**) command inquires about the attributes of a queue manager.

DefXmitQName (MQCHAR48):

This is the name of the transmission queue that is used for the transmission of messages to remote queue managers, if there is no other indication of which transmission queue to use.

If there is no default transmission queue, the name is entirely blank. The initial value of this attribute is blank.

To determine the value of this attribute, use the MQCA_DEF_XMIT_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

DistLists (MQLONG):

This indicates whether the local queue manager supports distribution lists on the MQPUT and MQPUT1 calls. It is one of the following values:

MQDL_SUPPORTED
Distribution lists supported.

MQDL_NOT_SUPPORTED
Distribution lists not supported.

To determine the value of this attribute, use the MQIA_DIST_LISTS selector with the MQINQ call.

DNSGroup (MQCHAR18):

This parameter is no longer used. See What changed in IBM MQ Version 8.0: WLM/DNS no longer supported.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQCA_DNS_GROUP selector with the MQINQ call. The length of this attribute is given by MQ_DNS_GROUP_NAME_LENGTH.

DNSWLM (MQLONG):

This parameter is no longer used. See What changed in IBM MQ Version 8.0: WLM/DNS no longer supported.

The value is one of the following:

MQDNSWLM_YES
This value may be seen on a queue manager migrated from an earlier release. The value is ignored.

MQDNSWLM_NO
This is the only value supported by the queue manager.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_DNS_WLM selector with the MQINQ call.

ExpiryInterval (MQLONG):

This indicates the frequency with which the queue manager scans the queues looking for expired messages. It is either a time interval in seconds in the range 1 through 99 999 999, or the following special value:

MQEXPI_OFF
The queue manager does not scan the queues looking for expired messages.

To determine the value of this attribute, use the MQIA_EXPIRY_INTERVAL selector with the MQINQ call.

z/OS This attribute is supported only on z/OS.

IGQPutAuthority (MQLONG):

This attribute applies only if the local queue manager is a member of a queue-sharing group. It indicates the type of authority checking that is performed when the local intra-group queuing agent (IGQ agent) removes a message from the shared transmission queue and places the message on a local queue. The value is one of the following:

MQIGQPA_DEFAULT
The user identifier checked for authorization is the value of the *UserIdentifier* field in the *separate* MQMD that is associated with the message when the message is on the shared transmission queue. This is the user identifier of the program that placed the message on the shared transmission queue, and is usually the same as the user identifier under which the remote queue manager is running.

If the RESLEVEL profile indicates that more than one user identifier is to be checked, the user identifier of the local IGQ agent (*IGQUserId*) is also checked.

MQIGQPA_CONTEXT
The user identifier checked for authorization is the value of the *UserIdentifier* field in the *separate* MQMD that is associated with the message when the message is on the shared transmission queue. This is the user identifier of the program that placed the message on the shared transmission queue, and is usually the same as the user identifier under which the remote queue manager is running.

If the RESLEVEL profile indicates that more than one user identifier is to be checked, the user identifier of the local IGQ agent (*IGQUserId*) and the value of the *UserIdentifier* field in the *embedded* MQMD are also checked. The latter user identifier is usually the user identifier of the application that originated the message.

MQIGQPA_ONLY_IGQ
The user identifier checked for authorization is the user identifier of the local IGQ agent (*IGQUserId*).

If the RESLEVEL profile indicates that more than one user identifier is to be checked, this user identifier is used for all checks.

MQIGQPA_ALTERNATE_OR_IGQ
The user identifier checked for authorization is the user identifier of the local IGQ agent (*IGQUserId*).

If the RESLEVEL profile indicates that more than one user identifier is to be checked, the value of the *UserIdentifier* field in the *embedded* MQMD is also checked. This user identifier is usually the user identifier of the application that originated the message.

To determine the value of this attribute, use the MQIA_IGQ_PUT_AUTHORITY selector with the MQINQ call.

z/OS This attribute is supported only on z/OS.

IGQUserId (MQLONG):

This attribute is applicable only if the local queue manager is a member of a queue-sharing group. It specifies the user identifier that is associated with the local intra-group queuing agent (IGQ agent). This identifier is one of the user identifiers that can be checked for authorization when the IGQ agent puts messages on local queues. The actual user identifiers checked depend on the setting of the **IGQPutAuthority** attribute, and on external security options.

If *IGQUserId* is blank, no user identifier is associated with the IGQ agent and the corresponding authorization check is not performed (although other user identifiers might still be checked for authorization).

To determine the value of this attribute, use the MQCA_IGQ_USER_ID selector with the MQINQ call. The length of this attribute is given by MQ_USER_ID_LENGTH.

z/OS This attribute is supported only on z/OS.

InhibitEvent (MQLONG):

This controls whether inhibit (Inhibit Get and Inhibit Put) events are generated. The value is one of the following:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_INHIBIT_EVENT selector with the MQINQ call.

On z/OS, you cannot use the MQINQ call to determine the value of this attribute.

IntraGroupqueuing (MQLONG):

This attribute applies only if the local queue manager is a member of a queue-sharing group. It indicates whether intra-group queuing is enabled for the queue-sharing group. The value is one of the following:

MQIGQ_DISABLED
All messages destined for other queue managers in the queue-sharing group are transmitted using conventional channels..

MQIGQ_ENABLED
Messages destined for other queue managers in the queue-sharing group are transmitted using the shared transmission queue if the following condition is satisfied:

- The length of the message data plus transmission header does not exceed 63 KB (64 512 bytes).

It is recommended that somewhat more space than the size of MQXQH be allocated for the transmission header; the constant MQ_MSG_HEADER_LENGTH is provided for this purpose.

If this condition is not satisfied, the message is transmitted using conventional channels.

Note: When intra-group queuing is enabled, the order of messages transmitted using the shared transmission queue is not preserved relative to those transmitted using conventional channels.

To determine the value of this attribute, use the MQIA_INTRA_GROUP_queuing selector with the MQINQ call.

z/OS This attribute is supported only on z/OS.

IPAddressVersion (MQLONG):

Specifies which IP address version, either IPv4 or IPv6, is used.

This attribute is only relevant for systems that run both IPv4 and IPv6 and only affects channels defined as having a *TransportType* of MQXPY_TCP when one of the following conditions is true:

- The channel's *ConnectionName* is a host name that resolves to both an IPv4 and IPv6 address and its **LocalAddress** parameter is not specified.
- The channel's *ConnectionName* and *LocalAddress* are both host names that resolve to both IPv4 and IPv6 addresses.

The value can be any of the following values:

MQIPADDR_IPv4
IPv4 is used.

MQIPADDR_IPv6
IPv6 is used.

To determine the value of this attribute, use the MQIA_IP_ADDRESS_VERSION selector with the MQINQ call.

ListenerTimer (MQLONG):

This is the time interval (in seconds) between IBM MQ attempts to restart the listener if there has been an APPC or TCP/IP failure. The value must be between 5 and 9999, with a default value of 60.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_LISTENER_TIMER selector with the MQINQ call.

LocalEvent (MQLONG):

This controls whether local error events are generated. The value is one of the following:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_LOCAL_EVENT selector with the MQINQ call.

On z/OS, you cannot use the MQINQ call to determine the value of this attribute.

LoggerEvent (MQLONG):

This controls whether recovery log events are generated. The value is one of the following:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_LOGGER_EVENT selector with the MQINQ call.

This attribute is supported only on Multiplatforms.

LUGroupName (MQCHAR8):

This is the generic LU name for the LU 6.2 listener that handles inbound transmissions for the queue-sharing group. If you leave this name blank, you cannot use this listener.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQCA_LU_GROUP_NAME selector with the MQINQ call. The length of this attribute is given by MQ_LU_NAME_LENGTH.

LUName (MQCHAR8):

This is the name of the LU to use for outbound LU 6.2 transmissions. Set this to the same LU that the listener uses for inbound transmissions. If you leave this name blank, the APPC/MVS default LU is used; this is variable, so always set LUName if you are using LU6.2.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQCA_LU_NAME selector with the MQINQ call. The length of this attribute is given by MQ_LU_NAME_LENGTH.

LU62ARMSuffix (MQCHAR2):

This is the suffix of the SYS1.PARMLIB member APPCPMxx, that nominates the LUADD for this channel initiator. The z/OS command SET APPC=xx is issued when ARM restarts the channel initiator. If you leave this name is blank, no SET APPC=xx is issued.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQCA_LU62_ARM_SUFFIX selector with the MQINQ call. The length of this attribute is given by MQ_ARM_SUFFIX_LENGTH.

LU62Channels (MQLONG):

This is the maximum number of channels that can be current, or clients that can be connected, that use the LU 6.2 transmission protocol.

The value must be in the range 0 through 9999, with a default value of 200. If you set this to zero, the LU 6.2 transmission protocol is not used.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_LU62_CHANNELS selector with the MQINQ call.

MaxActiveChannels (MQLONG):

This attribute is the maximum number of channels that can be *active* at any time.

The default is the value specified for the `MaxChannels` attribute.

For z/OS, the value must be in the range 1 through 9 999.

For all other platforms, the value must be in the range 1 through 65 535.

The **MaxActiveChannels** parameter is a queue manager attribute on z/OS only. On the other platforms, **MaxActiveChannels** is an attribute in the `qm.ini` file. See Configuration file stanzas for distributed queuing for information on how you set the **MaxActiveChannels** attribute on other platforms.

To determine the value of this attribute, use the MQIA_ACTIVE_CHANNELS selector with the **MQINQ** call.

Related information:
Channel states

MaxChannels (MQLONG):

This attribute is the maximum number of channels that can be *current* (including server-connection channels with connected clients).

For z/OS, the value must be in the range 1 through 9 999, with a default value of 200.

For all other platforms, the value must be in the range 1 through 65 535, with a default value of 100.

A system that is busy serving connections from the network might need a higher number than the default setting. Determine the value that is correct for your environment, ideally by observing the behavior of your system during testing.

The **MaxChannels** parameter is a queue manager attribute on z/OS only. On the other platforms, **MaxChannels** is an attribute in the `qm.ini` file. See Configuration file stanzas for distributed queuing for information on how you set the **MaxChannels** attribute on other platforms.

To determine the value of this attribute, use the MQIA_MAX_CHANNELS selector with the **MQINQ** call.

Related information:
Channel states

MaxHandles (MQLONG):

This is the maximum number of open handles that any one task can use concurrently. Each successful MQOPEN call for a single queue (or for an object that is not a queue) uses one handle. That handle becomes available for reuse when the object is closed. However, when a distribution list is opened, each queue in the distribution list is allocated a separate handle, and so that MQOPEN call uses as many handles as there are queues in the distribution list. This must be taken into account when deciding on a suitable value for *MaxHandles*.

The MQPUT1 call performs an MQOPEN call as part of its processing; as a result, MQPUT1 uses as many handles as MQOPEN would, but the handles are used only for the duration of the MQPUT1 call itself.

On z/OS, *task* means a CICS task, an MVS task, or an IMS dependent region.

The value is in the range 1 through 999 999 999. The default value is determined by the environment:

- On z/OS, the default value is 100.
- In all other environments, the default value is 256.

To determine the value of this attribute, use the MQIA_MAX_HANDLES selector with the MQINQ call.

MaxMsgLength (MQLONG):

This is the length of the longest *physical* message that the queue manager can handle. However, because the **MaxMsgLength** queue manager attribute can be set independently of the **MaxMsgLength** queue attribute, the longest physical message that can be placed on a queue is the lesser of those two values.

If the queue manager supports segmentation, an application can put a logical message that is longer than the lesser of the two **MaxMsgLength** attributes, but only if the application specifies the MQMF_SEGMENTATION_ALLOWED flag in MQMD. If that flag is specified, the upper limit for the length of a logical message is 999 999 999 bytes, but usually resource constraints imposed by the operating system, or by the environment in which the application is running, result in a lower limit.

The lower limit for the **MaxMsgLength** attribute is 32 KB (32 768 bytes). The upper limit is 100 MB (104 857 600 bytes).

To determine the value of this attribute, use the MQIA_MAX_MSG_LENGTH selector with the MQINQ call.

MaxPriority (MQLONG):

This is the maximum message priority supported by the queue manager. Priorities range from zero (lowest) to *MaxPriority* (highest).

To determine the value of this attribute, use the MQIA_MAX_PRIORITY selector with the MQINQ call.

MaxPropertiesLength (MQLONG):

This is used to control the size of the properties that can flow with a message. This includes both the property name in bytes and the size of the property value also in bytes.

To determine the value of this attribute, use the MQIA_MAX_PROPERTIES_LENGTH selector with the MQINQ call.

MaxUncommittedMsgs (MQLONG):

This is the maximum number of uncommitted messages that can exist within a unit of work. The number of uncommitted messages is the sum of the following since the start of the current unit of work:

- Messages put by the application with the MQPMO_SYNCPOINT option
- Messages retrieved by the application with the MQGMO_SYNCPOINT option
- Trigger messages and COA report messages generated by the queue manager for messages put with the MQPMO_SYNCPOINT option
- COD report messages generated by the queue manager for messages retrieved with the MQGMO_SYNCPOINT option

The following messages are not counted as uncommitted:

- Messages put or retrieved by the application outside a unit of work
- Trigger messages or COA/COD report messages generated by the queue manager as a result of messages put or retrieved outside a unit of work
- Expiration report messages generated by the queue manager (even if the call causing the expiration report message specified MQGMO_SYNCPOINT)
- Event messages generated by the queue manager (even if the call causing the event message specified MQPMO_SYNCPOINT or MQGMO_SYNCPOINT)

Note:

1. Exception report messages are generated by the Message Channel Agent (MCA), or by the application, and are treated in the same way as ordinary messages put or retrieved by the application.
2. When a message or segment is put with the MQPMO_SYNCPOINT option, the number of uncommitted messages is incremented by one regardless of how many physical messages actually result from the put. (More than one physical message might result if the queue manager must subdivide the message or segment.)
3. When a distribution list is put with the MQPMO_SYNCPOINT option, the number of uncommitted messages is incremented by one *for each physical message that is generated*. This can be as small as one, or as great as the number of destinations in the distribution list.

The lower limit for this attribute is 1; the upper limit is 999 999 999. The default value is 10000.

To determine the value of this attribute, use the MQIA_MAX_UNCOMMITTED_MSGS selector with the MQINQ call.

MQIAccounting (MQLONG):

This controls the collection of accounting information for MQI data.

The value is one of the following:

MQMON_ON
Collect API accounting data.

MQMON_OFF
Do not collect API accounting data. This is the default value.

If you set the queue manager attribute ACCTCONO to ENABLED, this value might be overridden for individual connections using the Options field in the MQCNO structure. Changes to this value are only effective for connections to the queue manager that occur after the change to the attribute.

This attribute is supported only on the following platforms:

- IBM i

-  UNIX
-  Windows

To determine the value of this attribute, use the MQIA_ACCOUNTING_MQI selector with the MQINQ call.

MQIStatistics (MQLONG):

This controls the collection of statistics monitoring information for the queue manager.

The value is one of the following:

MQMON_ON
Collect MQI statistics.

MQMON_OFF
Do not collect MQI statistics. This is the default value.

This attribute is supported only on the following platforms:

-  IBM i
-  UNIX
-  Windows

To determine the value of this attribute, use the MQIA_STATISTICS_MQI selector with the MQINQ call.

MsgMarkBrowseInterval (MQLONG):

Time interval in milliseconds after which the queue manager can automatically remove the mark from browse messages.

This is a time interval (in milliseconds) after which the queue manager can automatically remove the mark from browse messages.

This attribute describes the time interval for which messages that have been marked as browsed by a call to MQGET, using the get message option MQGMO_MARK_BROWSE_CO_OP, are expected to remain marked as browsed.

The queue manager might automatically unmark browsed messages that have been marked as browsed for the cooperating set of handles when they have been marked for more than this approximate interval.

This does not affect the state of any message marked as browse, that was obtained by a call to MQGET, using the get message option MQGMO_MARK_BROWSE_HANDLE.

The value is not less than -1 and not greater than 999 999 999. The default value is 5000. A *MsgMarkBrowseInterval* of -1 represents an unlimited time interval. A *MsgMarkBrowseInterval* of 0 causes the queue manager to unmark the message immediately.

To determine the value of this attribute, use the MQIA_MSG_MARK_BROWSE_INTERVAL selector with the MQINQ call.

OutboundPortMax (MQLONG):

This is the highest port number in the range, defined by OutboundPortMin and OutboundPortMax, of port numbers to be used to bind outgoing channels.

The value is an integer in the range 0 through 65535, and must be equal to or greater than the OutboundPortMin value. The default value is 0.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_OUTBOUND_PORT_MAX selector with the MQINQ call.

OutboundPortMin (MQLONG):

This is the lowest port number in the range, defined by OutboundPortMin and OutboundPortMax, of port numbers to be used to bind outgoing channels.

The value is an integer in the range 0 through 65535, and must be equal to or less than the OutboundPortMax value. The default value is 0.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_OUTBOUND_PORT_MIN selector with the MQINQ call.

PerformanceEvent (MQLONG):

This controls whether performance-related events are generated. It is one of the following values:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_PERFORMANCE_EVENT selector with the MQINQ call.

Platform (MQLONG):

This indicates the operating system on which the queue manager is running:

MQPL_AIX
AIX (same value as MQPL_UNIX).

MQPL_APPLIANCE
IBM MQ Appliance

MQPL_MVS
z/OS (same value as MQPL_ZOS).

MQPL_NSK
HP Integrity NonStop Server.

MQPL_OS390
z/OS (same value as MQPL_ZOS).

MQPL_OS400
IBM i.

MQPL_UNIX
UNIX.

MQPL_WINDOWS_NT
Windows systems.

MQPL_ZOS
z/OS.

To determine the value of this attribute, use the MQIA_PLATFORM selector with the MQINQ call.

PubSubNPInputMsg (MQLONG):

Whether to discard or keep an undelivered input message.

The value is one of the following:

MQUNDELIVERED_DISCARD
Non-persistent input messages may be discarded if they cannot be processed.

This is the default value.

MQUNDELIVERED_KEEP
Non-persistent input messages will not be discarded if they cannot be processed. In this situation the queued publish/subscribe interface will continue to retry the process at appropriate intervals and does not continue processing subsequent messages.

To determine the value of this attribute, use the MQIA_PUBSUB_NP_MSG selector with the MQINQ call.

PubSubNPResponse (MQLONG):

Controls the behavior of undelivered response messages.

The value is one of the following:

MQUNDELIVERED_NORMAL
Non-persistent responses which cannot be placed on the reply queue are put on the dead letter queue, if they cannot be placed on the DLQ then they are discarded.

MQUNDELIVERED_SAFE
Non-persistent responses which cannot be placed on the reply queue are put on the dead letter queue. If the response cannot be set and cannot be placed on the DLQ then the queued publish/subscribe interface will roll back the current operation and then retry at appropriate intervals and does not continue processing subsequent messages.

MQUNDELIVERED_DISCARD
Non-persistent responses are not placed on the reply queue are discarded.

This is the default value for new queue managers.

MQUNDELIVERED_KEEP
Non-persistent responses are not placed on the dead letter queue or discarded. Instead, the queued publish/subscribe interface will back out the current operation and then retry it at appropriate intervals.

To determine the value of this attribute, use the MQIA_PUBSUB_NP_RESP selector with the MQINQ call.

Default value for migrated queue managers.

If the queue manager has been migrated from IBM MQ V6.0, the initial value of this attribute depends on the values of *DiscardNonPersistentResponse* and *DLQNonPersistentResponse* before migration, as shown in the following table.

		DLQNonPersistentResponse		
		Yes	**No**	**Not set**
DiscardNonPersistentResponse	**Yes**	MQUNDELIVERED_NORMAL	MQUNDELIVERED_DISCARD	MQUNDELIVERED_NORMAL
	No	MQUNDELIVERED_SAFE	MQUNDELIVERED_KEEP	MQUNDELIVERED_SAFE
	Not set	If SyncPointPersistent = No, MQUNDELIVERED_SAFE else MQUNDELIVERED_NORMAL	If SyncPointPersistent = No, MQUNDELIVERED_KEEP else MQUNDELIVERED_DISCARD	If SyncPointPersistent = No, MQUNDELIVERED_SAFE else MQUNDELIVERED_NORMAL

PubSubMaxMsgRetryCount (MQLONG):

The number of retries when processing a failed command message under syncpoint.

The value is one of the following:

0 - 999 999 999
The default value is 5.

To determine the value of this attribute, use the MQIA_PUBSUB_MAXMSG_RETRY_COUNT selector with the MQINQ call.

PubSubSyncPoint (MQLONG):

Whether only persistent messages or all messages are processed under syncpoint.

The value is one of the following:

MQSYNCPOINT_IFPER
This makes the queued publish/subscribe interface receive non-persistent messages outside syncpoint. If the daemon receives a publication outside syncpoint, the daemon forwards the publication to subscribers known to it outside syncpoint.

This is the default value.

MQSYNCPOINT_YES
This makes the queued publish/subscribe interface receive all messages under syncpoint.

To determine the value of this attribute, use the MQIA_PUBSUB_SYNC_PT selector with the MQINQ call.

PubSubMode (MQLONG):

Whether the publish/subscribe engine and the queued publish/subscribe interface are running, therefore allowing applications to publish/subscribe by using the application programming interface and the queues that are being monitored by the queued publish/subscribe interface.

The value is one of the following:

MQPSM_COMPAT
The publish/subscribe engine is running. It is therefore possible to publish/subscribe by using the application programming interface. The queued publish/subscribe interface is not running, therefore any message that is put to the queues that are monitored by the queued publish/subscribe interface is not acted on. This setting is used for compatibility with WebSphere Message Broker V6 or earlier versions using this queue manager, because it must read the same queues from which the queued publish/subscribe interface normally reads.

MQPSM_DISABLED
The publish/subscribe engine and the queued publish/subscribe interface are not running. It is therefore not possible to publish/subscribe by using the application programming interface. Any publish/subscribe messages that are put to the queues that are monitored by the queued publish/subscribe interface are not acted on.

MQPSM_ENABLED
The publish/subscribe engine and the queued publish/subscribe interface are running. It is therefore possible to publish/subscribe by using the application programming interface and the queues that are being monitored by the queued publish/subscribe interface. This is the queue manager's initial default value.

To determine the value of this attribute, use the MQIA_PUBSUB_MODE selector with the MQINQ call.

QMgrDesc (MQCHAR64):

Use this field for a commentary describing the queue manager. The content of the field is of no significance to the queue manager, but the queue manager might require that the field contain only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, this field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the queue manager's character set (as defined by the **CodedCharSetId** queue manager attribute), those characters might be translated incorrectly if this field is sent to another queue manager.

- On z/OS, the default value is the product name and version number.
- In all other environments, the default value is blanks.

To determine the value of this attribute, use the MQCA_Q_MGR_DESC selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_DESC_LENGTH.

QMgrIdentifier (MQCHAR48):

This is an internally-generated unique name for the queue manager.

To determine the value of this attribute, use the MQCA_Q_MGR_IDENTIFIER selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_IDENTIFIER_LENGTH.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Linux, Windows, plus IBM MQ clients connected to these systems.

QMgrName (MQCHAR48):

This is the name of the local queue manager, that is, the name of the queue manager to which the application is connected.

The first 12 characters of the name are used to construct a unique message identifier (see MQMD - MsgId field). Queue managers that can intercommunicate must therefore have names that differ in the first 12 characters, in order for message identifiers to be unique in the queue manager network.

On z/OS, the name is the same as the subsystem name, which is limited to 4 nonblank characters.

To determine the value of this attribute, use the MQCA_Q_MGR_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_NAME_LENGTH.

QSGName (MQCHAR4):

This is the name of the queue-sharing group to which the local queue manager belongs. If the local queue manager does not belong to a queue-sharing group, the name is blank.

To determine the value of this attribute, use the MQCA_QSG_NAME selector with the MQINQ call. The length of this attribute is given by MQ_QSG_NAME_LENGTH.

z/OS This attribute is supported only on z/OS.

QueueAccounting (MQLONG):

This controls the collection of accounting information for queues.

The value is one of the following:

MQMON_NONE
Do not collect accounting data for queues, regardless of the setting of the queue accounting attribute ACCTQ. This is the default value.

MQMON_OFF
Do not collect accounting data for queues that specify QMGR in the ACCTQ queue attribute.

MQMON_ON
Collect accounting data for queues that specify QMGR in the ACCTQ queue attribute.

Changes to this value are only effective for connections to the queue manager that occur after the change to the attribute.

To determine the value of this attribute, use the MQIA_ACCOUNTING_Q selector with the MQINQ call.

QueueMonitoring (MQLONG):

This specifies the default setting for online monitoring of queues.

If the **QueueMonitoring** queue attribute is set to MQMON_Q_MGR, this attribute specifies the value which is assumed by the channel. The value can be:

MQMON_OFF
Online monitoring data collection is turned off. This is the queue manager's initial default value.

MQMON_NONE
Online monitoring data collection is turned off for queues regardless of the setting of their **QueueMonitoring** attribute.

MQMON_LOW
Online monitoring data collection is turned on, with a low ratio of data collection.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate ratio of data collection.

MQMON_HIGH
Online monitoring data collection is turned on, with a high ratio of data collection.

To determine the value of this attribute, use the MQIA_MONITORING_Q selector with the MQINQ call.

QueueStatistics (MQLONG):

This controls the collection of statistics data for queues.

It is one of the following values:

MQMON_NONE
Do not collect queue statistics for queues, regardless of the setting of the **QueueStatistics** queue attribute. This is the default value.

MQMON_OFF
Do not collect statistics data for queues that specify Queue Manager in the **QueueStatistics** queue attribute.

MQMON_ON
Collect statistics data for queues that specify Queue Manager in the **QueueStatistics** queue attribute.

To determine the value of this attribute, use the MQIA_STATISTICS_Q selector with the MQINQ call.

ReceiveTimeout (MQLONG):

This specifies how long a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to the inactive state. It applies only to message channels and not to MQI channels.

The exact meaning of the ReceiveTimeout is altered by the value specified in ReceiveTimeoutType. ReceiveTimeoutType can be set to one of the following:

- MQRCVTIME_EQUAL - this value is the number in seconds for the channel to wait. Specify a value in the range 0 - 999999.
- MQRCVTIME_ADD - this value is the number in seconds to add to the negotiated HBINT, and it determines how long a channel waits. Specify a value in the range 1 - 999999.
- MQRCVTIME_MULTIPLY - this value is a multiplier to apply to the negotiated HBINT. Specify a value of 0 or a value in the range 2 - 99.

The default value is 0.

Set ReceiveTimeoutType to MQRCVTIME_MULTIPLY or MQRCVTIME_EQUAL, and ReceiveTimeout to 0, to stop a channel from timing out its wait to receive data from its partner.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_RECEIVE_TIMEOUT selector with the MQINQ call.

ReceiveTimeoutMin (MQLONG):

This is the minimum time, in seconds, that a TCP/IP channel waits to receive data, including heartbeats, from its partner, before returning to the inactive state.

It applies only to message channels, not to MQI channels. The value must be in the range 0 through 999999, with a default of 0.

If you use ReceiveTimeoutType to specify that the TCP/IP channel wait time is to be calculated relative to the negotiated value of HBINT, and the resultant value is less than the value of this parameter, this value is used instead.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_RECEIVE_TIMEOUT_MIN selector with the MQINQ call.

ReceiveTimeoutType (MQLONG):

This is the qualifier, applied to ReceiveTimeout to define how long a TCP/IP channel waits to receive data, including heartbeats, from its partner, before returning to the inactive state. It applies only to message channels, not to MQI channels.

The value is one of the following:

MQRCVTIME_MULTIPLY
ReceiveTimeout is a multiplier to apply to the negotiated HBINT value to determine how long a channel waits. This is the default value.

MQRCVTIME_ADD
ReceiveTimeout is a value, in seconds, to add to the negotiated HBINT value to determine how long a channel waits.

MQRCVTIME_EQUAL
ReceiveTimeout is a value, in seconds, that the channel waits.

To stop a channel timing out its wait to receive data from its partner, set ReceiveTimeoutType to MQRCVTIME_MULTIPLY or MQRCVTIME_EQUAL, and ReceiveTimeout to 0.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_RECEIVE_TIMEOUT_TYPE selector with the MQINQ call.

RemoteEvent (MQLONG):

This controls whether remote error events are generated. It is one of the following values:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_REMOTE_EVENT selector with the MQINQ call.

RepositoryName (MQCHAR48):

This is the name of a cluster for which this queue manager provides a repository-manager service. If the queue manager provides this service for more than one cluster, *RepositoryNamelist* specifies the name of a namelist object that identifies the clusters, and *RepositoryName* is blank. At least one of *RepositoryName* and *RepositoryNamelist* must be blank.

To determine the value of this attribute, use the MQCA_REPOSITORY_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_NAME_LENGTH.

RepositoryNamelist (MQCHAR48):

This is the name of a namelist object that contains the names of clusters for which this queue manager provides a repository-manager service. If the queue manager provides this service for only one cluster, the namelist object contains only one name. Alternatively, *RepositoryName* can be used to specify the name of the cluster, in which case *RepositoryNamelist* is blank. At least one of *RepositoryName* and *RepositoryNamelist* must be blank.

To determine the value of this attribute, use the MQCA_REPOSITORY_NAMELIST selector with the MQINQ call. The length of this attribute is given by MQ_NAMELIST_NAME_LENGTH.

ScyCase(MQCHAR8):

Specifies whether the queue manager supports security profile names in mixed case, or in uppercase only.

The value is one of the following:

MQSCYC_UPPER
Security profile names must be in uppercase.

MQSCYC_MIXED
Security profile names can be in uppercase or in mixed case.

Changes to this attribute take effect when a Refresh Security command is run with *SecurityType(MQSECTYPE_CLASSES)* specified.

z/OS This attribute is supported only on z/OS.

To determine the value of this attribute, use the MQIA_SECURITY_CASE selector with the MQINQ call.

SharedQMgrName (MQLONG):

This specifies whether the *ObjectQmgrName* should be used or treated as the local queue manager on an MQOPEN call, for a shared queue, when the *ObjectQmgrName* is that of another queue manager in the queue-sharing group.

The value can be any of the following values:

MQSQQM_USE
ObjectQmgrName is used and the appropriate transmission queue is opened.

MQSQQM_IGNORE
If the target queue is shared, and the *ObjectQmgrName* is that of a queue manager in the same queue-sharing group, the open is performed locally.

This attribute is valid only on z/OS.

To determine the value of this attribute, use the MQIA_SHARED_Q_Q_MGR_NAME selector with the MQINQ call.

SPLCAP:

Indicates whether security capabilities of Advanced Message Security are available for a queue manager.

MQCAP_SUPPORTED
This is the default value if the AMS component is installed for the installation that the queue manager is running under.

MQCAP_NOT_SUPPORTED

SSLEvent (MQLONG):

This specifies whether TLS events are generated.

It is one of the following values:

MQEVR_ENABLED
Generate TLS events, as follows:
MQRC_CHANNEL_SSL_ERROR

MQEVR_DISABLED
Do not generate TLS events; this is the default value.

To determine the value of this attribute, use the MQIA_SSL_EVENT selector with the MQINQ call.

SSLFIPSRequired (MQLONG):

This lets you specify that only FIPS-certified algorithms are to be used if the cryptography is executed in IBM MQ, rather than in cryptographic hardware. If cryptographic hardware is configured, the cryptography modules used are those modules provided by the hardware product; these modules might or might not be FIPS-certified to a particular level depending on the hardware product in use.

The value is one of the following values:

MQSSL_FIPS_NO
Use any CipherSpec supported on the platform in use. This value is the default value.

MQSSL_FIPS_YES
Use only FIPS-certified cryptographic algorithms in the CipherSpecs allowed on all TLS connections from and to this queue manager.

This parameter is valid only on UNIX, Linux, Windows, and z/OS platforms.

To determine the value of this attribute, use the MQIA_SSL_FIPS_REQUIRED selector with the MQINQ call.

Related information:

Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client

Federal Information Processing Standards (FIPS) for UNIX, Linux, and Windows

SSLKeyResetCount (MQLONG):

This specifies when TLS channel message channel agents (MCAs) that initiate communication reset the secret key used for encryption on the channel.

The value represents the total number of unencrypted bytes that are sent and received on the channel before the secret key is renegotiated. The number of bytes includes control information sent by the MCA.

The value is a number in the range 0 through 999 999 999, with a default value of 0. If you specify a TLS secret key reset count in the range 1 byte through 32 KB, TLS channels will use a secret key reset count of 32 KB. This is to avoid the processing cost of excessive key resets which would occur for small TLS secret key reset values.

The secret key is renegotiated when the total number of unencrypted bytes sent and received by the initiating channel MCA exceeds the specified value. If channel heartbeats are enabled, the secret key is renegotiated before data is sent or received following a channel heartbeat, or when the total number of unencrypted bytes exceeds the specified value, whichever comes first.

The count of bytes sent and received for renegotiation includes control information sent and received by the channel MCA and is reset whenever a renegotiation occurs.

Use a value of 0 to indicate that secret keys are never renegotiated.

To determine the value of this attribute, use the MQIA_SSL_RESET_COUNT selector with the MQINQ call.

StartStopEvent (MQLONG):

This controls whether start and stop events are generated. The value is one of the following:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_START_STOP_EVENT selector with the MQINQ call.

StatisticsInterval (MQLONG):

This specifies how often (in seconds) to write statistics monitoring data to the monitoring queue.

The value is an integer in the range 0 to 604800, with a default value of 1800 (30 minutes).

To determine the value of this attribute, use the MQIA_STATISTICS_INTERVAL selector with the MQINQ call.

SyncPoint (MQLONG):

This indicates whether the local queue manager supports units of work and syncpointing with the MQGET, MQPUT, and MQPUT1 calls.

MQSP_AVAILABLE
Units of work and syncpointing available.

MQSP_NOT_AVAILABLE
Units of work and syncpointing not available.

- On z/OS this value is never returned.

To determine the value of this attribute, use the MQIA_SYNCPOINT selector with the MQINQ call.

TCPChannels (MQLONG):

This is the maximum number of channels that can be current, or clients that can be connected, that use the TCP/IP transmission protocol.

The value must be in the range 0 through 9999, with a default value of 200. If you specify 0, TCP/IP is not used.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_TCP_CHANNELS selector with the MQINQ call.

TCPKeepAlive (MQLONG):

This specifies whether to use TCP KEEPALIVE to check that the other end of the connection is still available. If it is not available, the channel is closed.

The value is one of the following:

MQTCPKEEP_YES
Use TCP KEEPALIVE as specified in the TCP profile configuration data set. If you specify the channel attribute KeepAliveInterval (KAINT), the value to which it is set is used.

MQTCPKEEP_NO
Do not use TCP KEEPALIVE. This is the default value.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_TCP_KEEP_ALIVE selector with the MQINQ call.

TCPName (MQCHAR8):

This is the name of either the only or preferred TCP/IP stack that will be used, depending on the value of TCPStackType. This parameter is only applicable in CINET multiple stack environments. The default value is TCPIP.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQCA_TCP_NAME selector with the MQINQ call. The length of this attribute is given by MQ_TCP_NAME_LENGTH.

TCPStackType (MQLONG):

This specifies whether the channel initiator can use only the TCP/IP stack specified in TCPName, or can optionally bind to any selected TCP/IP stack. This parameter is only applicable in CINET multiple stack environments.

The value is one of the following:

MQTCPSTACK_SINGLE
The channel initiator can use only the TCP/IP address spaces named in TCPName. This is the default value.

MQTCPSTACK_MULTIPLE
The channel initiator can use any TCP/IP address space available to it. It defaults to the one specified in TCPName if no other is specified for a channel or listener.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_TCP_STACK_TYPE selector with the MQINQ call.

TraceRouteRecording (MQLONG):

This controls the recording of trace- route information.

The value is one of the following:

MQRECORDING_DISABLED
No appending to trace- route messages allowed.

MQRECORDING_Q
Put trace- route messages to fixed named queue.

MQRECORDING_MSG
Put trace- route messages to a queue determined using the message itself. This is the default value

To determine the value of this attribute, use the MQIA_TRACE_ROUTE_RECORDING selector with the MQINQ call.

TriggerInterval (MQLONG):

This is a time interval (in milliseconds) used to restrict the number of trigger messages. This is relevant only when the *TriggerType* is MQTT_FIRST. In this case trigger messages are usually generated only when a suitable message arrives on the queue, and the queue was previously empty. Under certain circumstances, however, an additional trigger message can be generated with MQTT_FIRST triggering even if the queue was not empty. These additional trigger messages are not generated more often than every *TriggerInterval* milliseconds.

For more information on triggering, see Triggering channels.

The value is not less than 0 and not greater than 999 999 999. The default value is 999 999 999.

To determine the value of this attribute, use the MQIA_TRIGGER_INTERVAL selector with the MQINQ call.

TriggerInterval (MQLONG):

This is a time interval (in milliseconds) used to restrict the number of trigger messages. This is relevant only when the *TriggerType* is MQTT_FIRST. In this case trigger messages are usually generated only when a suitable message arrives on the queue, and the queue was previously empty. Under certain circumstances, however, an additional trigger message can be generated with MQTT_FIRST triggering even if the queue was not empty. These additional trigger messages are not generated more often than every *TriggerInterval* milliseconds.

For more information on triggering, see Triggering channels.

The value is not less than 0 and not greater than 999 999 999. The default value is 999 999 999.

To determine the value of this attribute, use the MQIA_TRIGGER_INTERVAL selector with the MQINQ call.

Version (MQCFST):

This is the version of the IBM MQ code as VVRRMMFF, where:

VV - Version

RR - Release

MM - Maintenance level

FF - Fix level

XrCapability(MQLONG):

This controls whether MQ Telemetry commands are supported by the queue manager.

The value is one of the following:

MQCAP_SUPPORTED
MQ Telemetry component installed and Telemetry commands are supported.

MQCAP_NOT_SUPPORTED
MQ Telemetry component not installed.

This attribute is supported only on the following platforms:

-  IBM i
-  UNIX
-  Windows

To determine the value of this attribute, use the MQIA_XR_CAPABILITY selector with the MQINQ call.

Attributes for queues:

There are five types of queue definition. Some queue attributes apply to all types of queue; other queue attributes apply only to certain types of queue.

Types of queue

The queue manager supports the following types of queue definition:

Local queue
You can store messages on a local queue. On z/OS you can make it a shared or private queue.

A queue is known to a program as *local* if it is owned by the queue manager to which the program is connected. You can get messages from, and put messages on, local queues.

The queue definition object holds the definition information of the queue as well as the physical messages put on the queue.

Local queue manager queue
The queue exists on the local queue manager. The queue is known as a private queue on z/OS.

Shared queue (z/OS only)
The queue exists in a shared repository that is accessible to all the queue managers that belong to the queue-sharing group that owns the shared repository.

Applications connected to any queue manager in the queue-sharing group can place messages on and remove messages from queues of this type. Such queues are effectively the same as local queues. The value of the **QType** queue attribute is MQQT_LOCAL.

Applications connected to the local queue manager can place messages on and remove messages from queues of this type. The value of the **QType** queue attribute is MQQT_LOCAL.

Cluster queue

You can store messages on a cluster queue on the queue manager where it is defined. A cluster queue is a queue that is hosted by a cluster queue manager and made available to other queue managers in the cluster. The value of the **QType** queue attribute is MQQT_CLUSTER.

A cluster queue definition is advertised to other queue managers in the cluster. The other queue managers in the cluster can put messages to a cluster queue without needing a corresponding remote-queue definition. A cluster queue can be advertised in more than one cluster by using a cluster namelist.

When a queue is advertised, any queue manager in the cluster can put messages to it. To put a message, the queue manager must find out, from the full repositories, where the queue is hosted. Then it adds some routing information to the message and puts the message on a cluster transmission queue.

A queue manager can store messages for other queue managers in a cluster on multiple transmission queues. You can configure a queue manager to store messages on multiple cluster transmission queues in two different ways. If you set the queue manager attribute DEFCLXQ to CHANNEL, a different cluster transmission queue is created automatically from SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE for each cluster-sender channel. If you set the CLCHNAME transmission queue option to match one or more cluster-senders channel, the queue manager can store messages for the matching channels on that transmission queue.

A cluster queue can be a queue that is shared by members of a queue-sharing group in IBM MQ for z/OS.

Remote queue
A remote queue is not a physical queue; it is the local definition of a queue that exists on a remote queue manager. The local definition of the remote queue contains information that tells the local queue manager how to route messages to the remote queue manager.

Applications connected to the local queue manager can place messages on queues of this type; the messages are placed on the local transmission queue used to route messages to the remote queue manager. Applications cannot remove messages from remote queues. The value of the **QType** queue attribute is MQQT_REMOTE.

You can also use a remote queue definition for:

- Reply-queue aliasing

  In this case the name of the definition is the name of a reply-to queue. For more information, see Reply-to queue aliases and clusters.
- Queue manager aliasing

  In this case the name of the definition is an alias for a queue manager, and not the name of a queue. For more information, see Queue manager aliases and clusters.

Alias queue
This is not a physical queue; it is an alternative name for a local queue, a shared queue, a cluster queue, or a remote queue. The name of the queue to which the alias resolves is part of the definition of the alias queue.

Applications connected to the local queue manager can place messages on queues of this type; the messages are placed on the queue to which the alias resolves. Applications can remove messages from queues of this type if the alias resolves to a local queue, a shared queue, or a cluster queue that has a local instance. The value of the **QType** queue attribute is MQQT_ALIAS.

Model queue
This is not a physical queue; it is a set of queue attributes from which a local queue can be created.

Messages cannot be stored on queues of this type.

Queue attributes

Some queue attributes apply to all types of queue; other queue attributes apply only to certain types of queue. The types of queue to which an attribute applies are shown in Table 273 on page 2835 and subsequent tables.

Table 273 on page 2835 summarizes the attributes that are specific to queues. The attributes are described in alphabetical order.

Note: The names of the attributes shown in this section are descriptive names used with the MQINQ and MQSET calls; the names are the same as for the PCF commands. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see Script (MQSC) commands for details.

Table 273. Attributes for queues. The columns apply as follows:

- The column for local queues applies also to shared queues.
- The column for model queues indicates which attributes are inherited by the local queue created from the model queue.
- The column for cluster queues indicates the attributes that can be inquired when the cluster queue is opened for inquire alone, or for inquire and output. If the cluster queue is opened for inquire plus one or more of input, browse, or set, the column for local queues applies instead.

Attribute	Description	Local	Model	Alias	Remote	Cluster
AlterationDate	Date when definition was last changed	✓		✓	✓	
AlterationTime	Time when definition was last changed	✓		✓	✓	
BackoutRequeueQName	Excessive backout requeue queue name	✓	✓			
BackoutThreshold	Backout threshold	✓	✓			
BaseQName	Queue name to which alias resolves			✓		
CFStrucName	Coupling-facility structure name	✓	✓			
CLCHNAME	Cluster-sender channel names	✓	✓			
ClusterName	Name of cluster to which queue belongs	✓		✓	✓	✓
ClusterNamelist	Name of namelist object containing names of clusters to which queue belongs	✓		✓	✓	
CLWLQueuePriority	Cluster workload queue priority	✓		✓	✓	✓
CLWLQueueRank	Cluster workload queue rank	✓		✓	✓	✓
CLWLUseQ	Use remote queue	✓				
CreationDate	Date that the queue was created	✓				
CreationTime	Time that the queue was created	✓				
CurrentQDepth	Current queue depth	✓				
DefaultPutResponse	Default put response	✓	✓	✓	✓	
DefBind	Default binding	✓		✓	✓	✓
DefinitionType attribute	Queue definition type	✓	✓			
DefInputOpenOption	Default input open option	✓	✓			
DefPersistence	Default message persistence	✓	✓	✓	✓	✓
DefPriority	Default message priority	✓	✓	✓	✓	✓
DefReadAhead	Default read ahead	✓	✓	✓		

Table 273. Attributes for queues (continued). The columns apply as follows:

- The column for local queues applies also to shared queues.
- The column for model queues indicates which attributes are inherited by the local queue created from the model queue.
- The column for cluster queues indicates the attributes that can be inquired when the cluster queue is opened for inquire alone, or for inquire and output. If the cluster queue is opened for inquire plus one or more of input, browse, or set, the column for local queues applies instead.

Attribute	Description	Local	Model	Alias	Remote	Cluster
DistLists	Distribution list support	✓	✓			
HardenGetBackout	Whether to maintain an accurate backout count	✓	✓			
IndexType	Index type	✓	✓			
InhibitGet	Whether get operations for the queue are allowed	✓	✓	✓		
InhibitPut	Whether put operations for the queue are allowed	✓	✓	✓	✓	✓
InitiationQName	Name of initiation queue	✓	✓			
MaxMsgLength	Maximum message length in bytes	✓	✓			
MaxQDepth	Maximum queue depth	✓	✓			
MsgDeliverySequence attribute	Message delivery sequence	✓	✓			
NonPersistentMessage Class	Reliability goal for non-persistent messages	✓	✓			
OpenInputCount	Number of opens for input	✓				
OpenOutputCount	Number of opens for output	✓				
PropertyControl	Property control	✓	✓	✓		
ProcessName	Process name	✓	✓			
QDepthHighEvent attribute	Whether Queue Depth High events are generated	✓	✓			
QDepthHighLimit	High limit for queue depth	✓	✓			
QDepthLowEvent attribute	Whether Queue Depth Low events are generated	✓	✓			
QDepthLowLimit attribute	Low limit for queue depth	✓	✓			
QDepthMaxEvent	Whether Queue Full events are generated	✓	✓			
QDesc	Queue description	✓	✓	✓	✓	✓
QName	Queue name	✓		✓	✓	✓
QServiceInterval	Target for queue service interval	✓	✓			

Table 273. Attributes for queues (continued). The columns apply as follows:

- The column for local queues applies also to shared queues.
- The column for model queues indicates which attributes are inherited by the local queue created from the model queue.
- The column for cluster queues indicates the attributes that can be inquired when the cluster queue is opened for inquire alone, or for inquire and output. If the cluster queue is opened for inquire plus one or more of input, browse, or set, the column for local queues applies instead.

Attribute	Description	Local	Model	Alias	Remote	Cluster
QServiceIntervalEvent attribute	Whether Service Interval High or Service Interval OK events are generated	✓	✓			
QSGDisp attribute	Queue-sharing group disposition	✓		✓	✓	
QueueAccounting	Queue accounting data collection	✓	✓	✓	✓	✓
QueueMonitoring	Online monitoring data for queues	✓	✓			
QueueStatistics	Queue statistics data collection	✓	✓	✓	✓	✓
QType	Queue type	✓		✓	✓	✓
RemoteQMgrName	Name of remote queue manager				✓	
RemoteQName	Name of remote queue				✓	
RetentionInterval	Retention interval	✓	✓			
Scope	Whether an entry for the queue also exists in a cell directory	✓		✓	✓	
Shareability	Queue shareability	✓	✓			
StorageClass	Storage class for queue	✓	✓			
TriggerControl	Trigger control	✓	✓			
TriggerData	Trigger data	✓	✓			
TriggerDepth	Trigger depth	✓	✓			
TriggerMsgPriority	Threshold message priority for triggers	✓	✓			
TriggerType	Trigger type	✓	✓			
Usage attribute	Queue usage	✓	✓			
XmitQName	Transmission queue name				✓	

Related information:
Cluster queues
Local queues

AlterationDate (MQCHAR12):

Date when definition was last changed.

Local	Model	Alias	Remote	Cluster
X		X	X	

This is the date when the definition was last changed. The format of the date is `YYYY-MM-DD`, padded with two trailing blanks to make the length 12 bytes (for example, `1992-09-23¬¬`, where ¬¬ represents two blank characters).

The values of certain attributes (for example, *CurrentQDepth*) change as the queue manager operates. Changes to these attributes do not affect *AlterationDate*.

To determine the value of this attribute, use the MQCA_ALTERATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_DATE_LENGTH.

AlterationTime (MQCHAR8):

Time when definition was last changed.

Local	Model	Alias	Remote	Cluster
X		X	X	

This is the time when the definition was last changed. The format of the time is `HH.MM.SS` using the 24-hour clock, with a leading zero if the hour is less than 10 (for example `09.10.20`).

- On z/OS, the time is Greenwich Mean Time (GMT), subject to the system clock being set accurately to GMT.
- In other environments, the time is local time.

The values of certain attributes (for example, *CurrentQDepth*) change as the queue manager operates. Changes to these attributes do not affect *AlterationTime*.

To determine the value of this attribute, use the MQCA_ALTERATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_TIME_LENGTH.

BackoutRequeueQName (MQCHAR48):

This is the excessive backout requeue queue name. Apart from allowing its value to be queried, the queue manager takes no action based on the value of this attribute.

Local	Model	Alias	Remote	Cluster
X	X			

Applications running inside WebSphere Application Server and those that use the IBM MQ Application Server Facilities use this attribute to determine where messages that have been backed out should go. For all other applications, the queue manager takes no action based on the value of the attribute.

IBM MQ classes for JMS uses this attribute to determine where to transfer a message that has already been backed out the maximum number of times as specified by the *BackoutThreshold* attribute.

To determine the value of this attribute, use the MQCA_BACKOUT_REQ_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

BackoutThreshold (MQLONG):

This is the backout threshold. Apart from allowing its value to be queried, the queue manager takes no action based on the value of this attribute.

Local	Model	Alias	Remote	Cluster
X	X			

Applications running inside of WebSphere Application Server and those that use the IBM MQ Application Server Facilities will use this attribute to determine if a message should be backed out. For all other applications, the queue manager takes no action based on the value of the attribute.

IBM MQ classes for JMS uses this attribute to determine how many times to allow a message to be backed out before transferring the message to the queue specified by the *BackoutRequeueQName* attribute.

To determine the value of this attribute, use the MQIA_BACKOUT_THRESHOLD selector with the MQINQ call.

BaseQName (MQCHAR48):

This is the name of a queue that is defined to the local queue manager.

Local	Model	Alias	Remote	Cluster
		X		

(For more information on queue names, see MQOD - ObjectName field.) The queue is one of the following types:

MQQT_LOCAL
Local queue.

MQQT_REMOTE
Local definition of a remote queue.

MQQT_CLUSTER
Cluster queue.

To determine the value of this attribute, use the MQCA_BASE_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

BaseType (MQCFIN):

The type of object to which the alias resolves.

Local	Model	Alias	Remote	Cluster
		X		

It is one of the following values:

MQOT_Q
Base object type is a queue

MQOT_TOPIC
Base object type is a topic

CFStrucName (MQCHAR12):

This is the name of the coupling-facility structure where the messages on the queue are stored. The first character of the name is in the range A through Z, and the remaining characters are in the range A through Z, 0 through 9, or blank.

Local	Model	Alias	Remote	Cluster
X	X			

To get the full name of the structure in the coupling facility, suffix the value of the **QSGName** queue manager attribute with the value of the **CFStrucName** queue attribute.

This attribute applies only to shared queues; it is ignored if *QSGDisp* does not have the value MQQSGD_SHARED.

To determine the value of this attribute, use the MQCA_CF_STRUC_NAME selector with the MQINQ call. The length of this attribute is given by MQ_CF_STRUC_NAME_LENGTH.

z/OS This attribute is supported only on z/OS.

ClusterChannelName (MQCHAR20):

`ClusterChannelName` is the generic name of the cluster-sender channels that use this queue as a transmission queue. The attribute specifies which cluster-sender channels send messages to a cluster-receiver channel from this cluster transmission queue.

Local	Model	Alias	Remote	Cluster
✓	✓			

The default queue manager configuration is for all cluster-sender channels to send messages from a single transmission queue, `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. The default configuration can be changed by modified by changing the queue manager attribute, `DefClusterXmitQueueType`. The default value of the attribute is `SCTQ`. You can change the value to `CHANNEL`. If you set the `DefClusterXmitQueueType` attribute to `CHANNEL`, each cluster-sender channel defaults to using a specific cluster transmission queue, `SYSTEM.CLUSTER.TRANSMIT.`*`ChannelName`*.

You can also set the transmission queue attribute `ClusterChannelName` attribute to a cluster-sender channel manually. Messages that are destined for the queue manager connected by the cluster-sender

channel are stored in the transmission queue that identifies the cluster-sender channel. They are not stored in the default cluster transmission queue. If you set the ClusterChannelName attribute to blanks, the channel switches to the default cluster transmission queue when the channel restarts. The default queue is either SYSTEM.CLUSTER.TRANSMIT.*ChannelName* or SYSTEM.CLUSTER.TRANSMIT.QUEUE, depending on the value of the queue manager DefClusterXmitQueueType attribute.

By specifying asterisks, "*", in ClusterChannelName, you can associate a transmission queue with a set of cluster-sender channels. The asterisks can be at the beginning, end, or any number of places in the middle of the channel name string. ClusterChannelName is limited to a length of 20 characters: MQ_CHANNEL_NAME_LENGTH.

ClusterName (MQCHAR48):

This is the name of the cluster to which the queue belongs.

Local	Model	Alias	Remote	Cluster
X		X	X	X

If the queue belongs to more than one cluster, *ClusterNamelist* specifies the name of a namelist object that identifies the clusters, and *ClusterName* is blank. At least one of *ClusterName* and *ClusterNamelist* must be blank.

To determine the value of this attribute, use the MQCA_CLUSTER_NAME selector with the MQINQ call. The length of this attribute is given by MQ_CLUSTER_NAME_LENGTH.

ClusterNamelist (MQCHAR48):

This is the name of a namelist object that contains the names of clusters to which this queue belongs.

Local	Model	Alias	Remote	Cluster
X		X	X	

If the queue belongs to only one cluster, the namelist object contains only one name. Alternatively, *ClusterName* can be used to specify the name of the cluster, in which case *ClusterNamelist* is blank. At least one of *ClusterName* and *ClusterNamelist* must be blank.

To determine the value of this attribute, use the MQCA_CLUSTER_NAMELIST selector with the MQINQ call. The length of this attribute is given by MQ_NAMELIST_NAME_LENGTH.

CLWLQueuePriority (MQLONG):

This is the cluster workload queue priority, a value in the range 0 through 9 representing the priority of the queue.

Local	Model	Alias	Remote	Cluster
X		X	X	X

For more information, see Cluster queues.

To determine the value of this attribute, use the MQIA_CLWL_Q_PRIORITY selector with the MQINQ call.

CLWLQueueRank (MQLONG):

This is the cluster workload queue rank, a value in the range 0 through 9 representing the rank of the queue.

Local	Model	Alias	Remote	Cluster
X		X	X	X

For more information, see Cluster queues.

To determine the value of this attribute, use the MQIA_CLWL_Q_RANK selector with the MQINQ call.

CLWLUseQ (MQLONG):

This defines the behavior of an MQPUT when the target queue has both a local instance and at least one remote cluster instance. If the put originates from a cluster channel, this attribute does not apply.

Local	Model	Alias	Remote	Cluster
X				

The value is one of the following:

MQCLWL_USEQ_ANY
Use remote and local queues.

MQCLWL_USEQ_LOCAL
Do not use remote queues.

MQCLWL_USEQ_AS_Q_MGR
Inherit definition from queue manager's MQIA_CLWL_USEQ.

For more information, see Cluster queues.

To determine the value of this attribute, use the MQCA_CLWL_USEQ selector with the MQINQ call. The length of this attribute is given by MQ_CLWL_USEQ_LENGTH.

CreationDate (MQCHAR12):

This is the date when the queue was created.

Local	Model	Alias	Remote	Cluster
X				

The format of the date is `YYYY-MM-DD`, padded with two trailing blanks to make the length 12 bytes (for example, `2013-09-23¬¬`, where ¬¬ represents 2 blank characters).

- On IBM i, the creation date of a queue can differ from that of the underlying operating system entity (file or userspace) that represents the queue.

To determine the value of this attribute, use the MQCA_CREATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_CREATION_DATE_LENGTH.

CreationTime (MQCHAR8):

This is the time when the queue was created.

Local	Model	Alias	Remote	Cluster
X				

The format of the time is `HH.MM.SS` using the 24-hour clock, with a leading zero if the hour is less than 10 (for example `09.10.20`).

- On z/OS, the time is Greenwich Mean Time (GMT), subject to the system clock being set accurately to GMT.
- In other environments, the time is local time.
- On IBM i, the creation time of a queue can differ from that of the underlying operating system entity (file or userspace) that represents the queue.

To determine the value of this attribute, use the MQCA_CREATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_CREATION_TIME_LENGTH.

CurrentQDepth (MQLONG):

This is the number of messages currently on the queue.

Local	Model	Alias	Remote	Cluster
X				

It is incremented during an MQPUT call, and during backout of an MQGET call. It is decremented during a nonbrowse MQGET call, and during backout of an MQPUT call. The effect of this is that the count includes messages that have been put on the queue within a unit of work, but that have not yet been committed, even though they are not eligible to be retrieved by the MQGET call. Similarly, it excludes messages that have been retrieved within a unit of work using the MQGET call, but that have yet to be committed.

The count also includes messages that have passed their expiry time but have not yet been discarded, although these messages are not eligible to be retrieved. See MQMD - Expiry field for more information.

Unit-of-work processing and the segmentation of messages can both cause *CurrentQDepth* to exceed *MaxQDepth*. However, this does not affect the retrievability of the messages; *all* messages on the queue can be retrieved using the MQGET call in the normal way.

The value of this attribute fluctuates as the queue manager operates.

To determine the value of this attribute, use the MQIA_CURRENT_Q_DEPTH selector with the MQINQ call.

DefaultPutResponse (MQLONG):

Specifies the type of response to be used for put operations to the queue when an application specifies MQPMO_RESPONSE_AS_Q_DEF.

Local	Model	Alias	Remote	Cluster
X	X	X	X	

It is one of the following values:

MQPRT_SYNC_RESPONSE
The put operation is issued synchronously, returning a response.

MQPRT_ASYNC_RESPONSE
The put operation is issued asynchronously, returning a subset of MQMD fields.

DefBind (MQLONG):

This is the default binding that is used when MQOO_BIND_AS_Q_DEF is specified on the MQOPEN call and the queue is a cluster queue.

Local	Model	Alias	Remote	Cluster
X		X	X	X

The value is one of the following:

MQBND_BIND_ON_OPEN
Binding fixed by MQOPEN call.

MQBND_BIND_NOT_FIXED
Binding not fixed.

MQBND_BIND_ON_GROUP
Allows an application to request that a group of messages are all allocated to the same destination instance. Because this value is new in IBM WebSphere MQ Version 7.1, it must not be used if any of the applications opening this queue are connecting to IBM WebSphere MQ Version 7.0.1 or earlier queue managers.

To determine the value of this attribute, use the MQIA_DEF_BIND selector with the MQINQ call.

DefinitionType (MQLONG):

This indicates how the queue was defined.

Local	Model	Alias	Remote	Cluster
X	X			

The value is one of the following:

MQQDT_PREDEFINED
The queue is a permanent queue created by the system administrator; only the system administrator can delete it.

Predefined queues are created using the DEFINE MQSC command, and can be deleted only by using the DELETE MQSC command. Predefined queues cannot be created from model queues.

Commands can be issued either by an operator, or by an authorized user sending a command message to the command input queue (see CommandInputQName attribute for more information).

MQQDT_PERMANENT_DYNAMIC
The queue is a permanent queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value MQQDT_PERMANENT_DYNAMIC for the **DefinitionType** attribute.

This type of queue can be deleted using the MQCLOSE call. See “MQCLOSE - Close object” on page 2644 for more details.

The value of the **QSGDisp** attribute for a permanent dynamic queue is MQQSGD_Q_MGR.

MQQDT_TEMPORARY_DYNAMIC
The queue is a temporary queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value MQQDT_TEMPORARY_DYNAMIC for the **DefinitionType** attribute.

This type of queue is deleted automatically by the MQCLOSE call when it is closed by the application that created it.

The value of the **QSGDisp** attribute for a temporary dynamic queue is MQQSGD_Q_MGR.

MQQDT_SHARED_DYNAMIC
The queue is a shared permanent queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value MQQDT_SHARED_DYNAMIC for the **DefinitionType** attribute.

This type of queue can be deleted using the MQCLOSE call. See “MQCLOSE - Close object” on page 2644 for more details.

The value of the **QSGDisp** attribute for a shared dynamic queue is MQQSGD_SHARED.

This attribute in a model queue definition does not indicate how the model queue was defined, because model queues are always predefined. Instead, the value of this attribute in the model queue is used to determine the *DefinitionType* of each of the dynamic queues created from the model queue definition using the MQOPEN call.

To determine the value of this attribute, use the MQIA_DEFINITION_TYPE selector with the MQINQ call.

DefInputOpenOption (MQLONG):

This is the default way in which to open the queue for input.

Local	Model	Alias	Remote	Cluster
X	X			

It applies if the MQOO_INPUT_AS_Q_DEF option is specified on the MQOPEN call when the queue is opened. The value is one of the following:

MQOO_INPUT_EXCLUSIVE
Open queue to get messages with exclusive access.

The queue is opened for use with subsequent MQGET calls. The call fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open by this or another application for input of any type (MQOO_INPUT_SHARED or MQOO_INPUT_EXCLUSIVE).

MQOO_INPUT_SHARED
Open queue to get messages with shared access.

The queue is opened for use with subsequent MQGET calls. The call can succeed if the queue is currently open by this or another application with MQOO_INPUT_SHARED, but fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open with MQOO_INPUT_EXCLUSIVE.

To determine the value of this attribute, use the MQIA_DEF_INPUT_OPEN_OPTION selector with the MQINQ call.

DefPersistence (MQLONG):

This is the default persistence of messages on the queue. It applies if MQPER_PERSISTENCE_AS_Q_DEF is specified in the message descriptor when the message is put.

Local	Model	Alias	Remote	Cluster
X	X	X	X	X

If there is more than one definition in the queue-name resolution path, the default persistence is taken from the value of this attribute in the *first* definition in the path at the time of the MQPUT or MQPUT1 call. This could be:

- An alias queue
- A local queue
- A local definition of a remote queue
- A queue manager alias
- A transmission queue (for example, the *DefXmitQName* queue)

The value is one of the following:

MQPER_PERSISTENT
The message survives system failures and queue manager restarts. Persistent messages cannot be placed on:

- Temporary dynamic queues
- Shared queues that map to a CFSTRUCT object at CFLEVEL(2) or below, or where the CFSTRUCT object is defined as RECOVER(NO).

Persistent messages can be placed on permanent dynamic queues, and predefined queues.

MQPER_NOT_PERSISTENT
The message does not normally survive system failures or queue manager restarts. This applies even if an intact copy of the message is found on auxiliary storage during a queue manager restart.

In the case of shared queues, nonpersistent messages *do* survive restarts of queue managers in the queue-sharing group, but do not survive failures of the coupling facility used to store messages on the shared queues.

Both persistent and nonpersistent messages can exist on the same queue.

To determine the value of this attribute, use the MQIA_DEF_PERSISTENCE selector with the MQINQ call.

DefPriority (MQLONG):

This is the default priority for messages on the queue. This applies if MQPRI_PRIORITY_AS_Q_DEF is specified in the message descriptor when the message is put on the queue.

Local	Model	Alias	Remote	Cluster
X	X	X	X	X

If there is more than one definition in the queue-name resolution path, the default priority for the message is taken from the value of this attribute in the *first* definition in the path at the time of the put operation. This could be:

- An alias queue
- A local queue
- A local definition of a remote queue
- A queue manager alias
- A transmission queue (for example, the *DefXmitQName* queue)

The way in which a message is placed on a queue depends on the value of the queue's **MsgDeliverySequence** attribute:

- If the **MsgDeliverySequence** attribute is MQMDS_PRIORITY, the logical position at which a message is placed on the queue depends on the value of the *Priority* field in the message descriptor.
- If the **MsgDeliverySequence** attribute is MQMDS_FIFO, messages are placed on the queue as though they had a priority equal to the *DefPriority* of the resolved queue, regardless of the value of the *Priority* field in the message descriptor. However, the *Priority* field retains the value specified by the application that put the message. See MsgDeliverySequence attribute for more information.

Priorities are in the range zero (lowest) through *MaxPriority* (highest); see MaxPriority attribute.

To determine the value of this attribute, use the MQIA_DEF_PRIORITY selector with the MQINQ call.

DefReadAhead (MQLONG):

Specifies the default read ahead behavior for non-persistent messages delivered to the client.

Local	Model	Alias	Remote	Cluster
X	X	X		

DefReadAhead can be set to one of the following values::

MQREADA_NO
Non-persistent messages are not sent ahead to the client before an applications requests them. A maximum of one non-persistent message can be lost if the client ends abnormally.

MQREADA_YES
Non-persistent messages are sent ahead to the client before an application requests them. Non-persistent messages can be lost if the client ends abnormally or if the client does not consume all the messages it is sent.

MQREADA_DISABLED
Read ahead of non-persistent messages in not enabled for this queue. Messages are not sent ahead to the client regardless of whether read ahead is requested by the client application.

To determine the value of this attribute, use the MQIA_DEF_READ_AHEAD selector with the MQINQ call.

DefPResp (MQLONG):

The default put response type (DEFPRESP) attribute defines the value used by applications when the PutResponseType within MQPMO has been set to MQPMO_RESPONSE_AS_Q_DEF. This attribute is valid for all queue types.

Local	Model	Alias	Remote	Cluster
Yes	Yes	Yes	Yes	Yes

The value is one of the following:

SYNC The put operation is issued synchronously returning a response.

ASYNC
The put operation is issued asynchronously, returning a subset of MQMD fields.

To determine the value of this attribute, use the MQIA_DEF_PUT_RESPONSE_TYPE selector with the MQINQ call.

DistLists (MQLONG):

This indicates whether distribution-list messages can be placed on the queue.

Local	Model	Alias	Remote	Cluster
X	X			

A message channel agent (MCA) sets the attribute to inform the local queue manager whether the queue manager at the other end of the channel supports distribution lists. This latter queue manager (called the *partnering* queue manager) is the one that next receives the message, after it has been removed from the local transmission queue by a sending MCA.

The sending MCA sets the attribute whenever it establishes a connection to the receiving MCA on the partnering queue manager. In this way, the sending MCA can cause the local queue manager to place on the transmission queue only messages that the partnering queue manager can process correctly.

This attribute is primarily for use with transmission queues, but the processing described is performed regardless of the usage defined for the queue (see Usage attribute).

The value is one of the following:

MQDL_SUPPORTED
Distribution-list messages can be stored on the queue, and transmitted to the partnering queue manager in that form. This reduces the amount of processing required to send the message to multiple destinations.

MQDL_NOT_SUPPORTED
Distribution-list messages cannot be stored on the queue, because the partnering queue manager does not support distribution lists. If an application puts a distribution-list message, and that message is to be placed on this queue, the queue manager splits the distribution-list message and places the individual messages on the queue instead. This increases the amount of processing required to send the message to multiple destinations, but ensures that the messages are processed correctly by the partnering queue manager.

To determine the value of this attribute, use the MQIA_DIST_LISTS selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

This attribute is not supported on z/OS.

HardenGetBackout (MQLONG):

For each message, a count is kept of the number of times that the message is retrieved by an MQGET call within a unit of work, and that unit of work subsequently backed out.

Local	Model	Alias	Remote	Cluster
X	X			

This count is available in the *BackoutCount* field in the message descriptor after the MQGET call has completed.

The message backout count survives restarts of the queue manager. However, to ensure that the count is accurate, information has to be *hardened* (recorded on disk or other permanent storage device) each time that an MQGET call retrieves a message within a unit of work for this queue. If this is not done, the queue manager fails, and the MQGET call backs out, the count might or might not be incremented.

Hardening information for each MQGET call within a unit of work, however, imposes additional processing cost, so set the **HardenGetBackout** attribute to MQQA_BACKOUT_HARDENED only if it is essential that the count is accurate.

On IBM i, UNIX, and Windows, the message backout count is always hardened, regardless of the setting of this attribute.

The following values are possible:

MQQA_BACKOUT_HARDENED
Hardening is used to ensure that the backout count for messages on this queue is accurate.

MQQA_BACKOUT_NOT_HARDENED
Hardening is not used to ensure that the backout count for messages on this queue is accurate. The count might therefore be lower than it should be.

To determine the value of this attribute, use the MQIA_HARDEN_GET_BACKOUT selector with the MQINQ call.

IndexType (MQLONG):

This specifies the type of index that the queue manager maintains for messages on the queue.

Local	Model	Alias	Remote	Cluster
X	X			

The type of index required depends on how the application retrieves messages, and whether the queue is a shared queue or a nonshared queue (see QSGDisp attribute). The following values are possible for *IndexType*:

MQIT_NONE
No index is maintained by the queue manager for this queue. Use this value for queues that are typically processed sequentially, that is, without using any selection criteria on the MQGET call.

MQIT_MSG_ID
The queue manager maintains an index that uses the message identifiers of the messages on the queue. Use this value queues where the application typically retrieves messages using the message identifier as the selection criterion on the MQGET call.

MQIT_CORREL_ID
The queue manager maintains an index that uses the correlation identifiers of the messages on the queue. Use this value for queues where the application typically retrieves messages using the correlation identifier as the selection criterion on the MQGET call.

MQIT_MSG_TOKEN
The queue manager maintains an index that uses the message tokens of the messages on the queue for use with the workload manager (WLM) functions of z/OS.

You must specify this option for WLM-managed queues; do not specify it for any other type of queue. Also, do not use this value for a queue where an application is not using the z/OS workload manager functions, but is retrieving messages using the message token as a selection criterion on the MQGET call.

MQIT_GROUP_ID
The queue manager maintains an index that uses the group identifiers of the messages on the queue. This value must be used for queues where the application retrieves messages using the MQGMO_LOGICAL_ORDER option on the MQGET call.

A queue with this index type cannot be a transmission queue. A shared queue with this index type must be defined to map to a CFSTRUCT object at CFLEVEL(3) or higher.

Note:

1. The physical order of messages on a queue with index type MQIT_GROUP_ID is not defined, as the queue is optimized for efficient retrieval of messages using the MQGMO_LOGICAL_ORDER option on the MQGET call. This means that the physical order of the messages is not typically the order in which the messages arrived on the queue.
2. If an MQIT_GROUP_ID queue has a *MsgDeliverySequence* of MQMDS_PRIORITY, the queue manager uses message priorities 0 and 1 to optimize the retrieval of messages in logical order. As a result, the first message in a group must not have a priority of zero or one; if it does, the message is processed as though it had a priority of two. The *Priority* field in the MQMD structure is not changed.

For more information about message groups, see the description of the group and segment options in MQGMO - Options field.

The index type that should be used in various cases is shown in Table 274 and Table 275.

Table 274. Suggested or required values of queue index type when MQGMO_LOGICAL_ORDER not specified

Selection criteria on MQGET call	Index type for nonshared queue	Index type for shared queue
None	Any	Any
Selection using one identifier:		
Message identifier	MQIT_MSG_ID suggested	MQIT_NONE or MQIT_MSG_ID required; MQIT_MSG_ID suggested
Correlation identifier	MQIT_CORREL_ID suggested	MQIT_CORREL_ID required
Group identifier	MQIT_GROUP_ID suggested	MQIT_GROUP_ID required
Selection using two identifiers:		
Message identifier plus correlation identifier	MQIT_MSG_ID or MQIT_CORREL_ID suggested	MQIT_NONE or MQIT_MSG_ID or MQIT_CORREL_ID required (For efficiency, it is suggested that the index type is chosen to match the MQMD field which will have the most distinct keys)
Message identifier plus group identifier	MQIT_MSG_ID or MQIT_GROUP_ID suggested	Not supported
Correlation identifier plus group identifier	MQIT_CORREL_ID or MQIT_GROUP_ID suggested	Not supported
Selection using three identifiers:		
Message identifier plus correlation identifier plus group identifier	MQIT_MSG_ID or MQIT_CORREL_ID or MQIT_GROUP_ID suggested	Not supported
Selection using group-related criteria:		
Group identifier plus message sequence number	MQIT_GROUP_ID required	MQIT_GROUP_ID required
Message sequence number (must be 1)	MQIT_GROUP_ID required	MQIT_GROUP_ID required
Selection using message token:		
Message token for application use	Do not use MQIT_MSG_TOKEN	
Message token for WLM use	MQIT_MSG_TOKEN required	Not supported

Table 275. Suggested or required values of queue index type when MQGMO_LOGICAL_ORDER specified

Selection criteria on MQGET call	Index type for nonshared queue	Index type for shared queue
None	MQIT_GROUP_ID required	MQIT_GROUP_ID required
Selection using one identifier:		
Message identifier	MQIT_GROUP_ID required	Not supported
Correlation identifier	MQIT_GROUP_ID required	Not supported
Group identifier	MQIT_GROUP_ID required	MQIT_GROUP_ID required
Selection using two identifiers:		
Message identifier plus correlation identifier	MQIT_GROUP_ID required	Not supported
Message identifier plus group identifier	MQIT_GROUP_ID required	Not supported

Table 275. Suggested or required values of queue index type when MQGMO_LOGICAL_ORDER specified (continued)

Selection criteria on MQGET call	Index type for nonshared queue	Index type for shared queue
Correlation identifier plus group identifier	MQIT_GROUP_ID required	Not supported
Selection using three identifiers:		
Message identifier plus correlation identifier plus group identifier	MQIT_GROUP_ID required	Not supported

To determine the value of this attribute, use the MQIA_INDEX_TYPE selector with the MQINQ call.

z/OS This attribute is supported only on z/OS.

InhibitGet (MQLONG):

This controls whether get operations for this queue are allowed.

Local	Model	Alias	Remote	Cluster
X	X	X		

If the queue is an alias queue, get operations must be allowed for both the alias and the base queue at the time of the get operation, for the MQGET call to succeed. The value is one of the following:

MQQA_GET_INHIBITED
Get operations are inhibited.

MQGET calls fail with reason code MQRC_GET_INHIBITED. This includes MQGET calls that specify MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT.

Note: If an MQGET call operating within a unit of work completes successfully, changing the value of the **InhibitGet** attribute subsequently to MQQA_GET_INHIBITED does not prevent the unit of work being committed.

MQQA_GET_ALLOWED
Get operations are allowed.

To determine the value of this attribute, use the MQIA_INHIBIT_GET selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

InhibitPut (MQLONG):

This controls whether put operations for this queue are allowed.

Local	Model	Alias	Remote	Cluster
X	X	X	X	X

If there is more than one definition in the queue-name resolution path, put operations must be allowed for *every* definition in the path (including any queue manager alias definitions) at the time of the put operation, for the MQPUT or MQPUT1 call to succeed. The value is one of the following:

MQQA_PUT_INHIBITED
Put operations are inhibited.

MQPUT and MQPUT1 calls fail with reason code MQRC_PUT_INHIBITED.

Note: If an MQPUT call operating within a unit of work completes successfully, changing the value of the **InhibitPut** attribute subsequently to MQQA_PUT_INHIBITED does not prevent the unit of work being committed.

MQQA_PUT_ALLOWED
Put operations are allowed.

To determine the value of this attribute, use the MQIA_INHIBIT_PUT selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

InitiationQName (MQCHAR48):

This is the name of a queue defined on the local queue manager; the queue must be of type MQQT_LOCAL.

Local	Model	Alias	Remote	Cluster
X				

The queue manager sends a trigger message to the initiation queue when application start-up is required as a result of a message arriving on the queue to which this attribute belongs. The initiation queue must be monitored by a trigger monitor application that starts the appropriate application after receipt of the trigger message.

To determine the value of this attribute, use the MQCA_INITIATION_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

MaxMsgLength (MQLONG):

This is an upper limit for the length of the longest *physical* message that can be placed on the queue.

Local	Model	Alias	Remote	Cluster
X	X			

However, because the **MaxMsgLength** queue attribute can be set independently of the **MaxMsgLength** queue manager attribute, the actual upper limit for the length of the longest physical message that can be placed on the queue is the lesser of those two values.

If the queue manager supports segmentation, it is possible for an application to put a *logical* message that is longer than the lesser of the two **MaxMsgLength** attributes, but only if the application specifies the MQMF_SEGMENTATION_ALLOWED flag in MQMD. If that flag is specified, the upper limit for the length of a logical message is 999 999 999 bytes, but usually resource constraints imposed by the operating system, or by the environment in which the application is running, result in a lower limit.

An attempt to place on the queue a message that is too long fails with one of the following reason codes:

- MQRC_MSG_TOO_BIG_FOR_Q if the message is too big for the queue
- MQRC_MSG_TOO_BIG_FOR_Q_MGR if the message is too big for the queue manager, but not too big for the queue

The lower limit for the **MaxMsgLength** attribute is zero; the upper limit is 100 MB (104 857 600 bytes).

For more information, see MQPUT - BufferLength parameter.

To determine the value of this attribute, use the MQIA_MAX_MSG_LENGTH selector with the MQINQ call.

MaxQDepth (MQLONG):

This is the defined upper limit for the number of physical messages that can exist on the queue at any one time.

Local	Model	Alias	Remote	Cluster
X	X			

An attempt to put a message on a queue that already contains *MaxQDepth* messages fails with reason code MQRC_Q_FULL.

Unit-of-work processing and the segmentation of messages can both cause the actual number of physical messages on the queue to exceed *MaxQDepth*. However, this does not affect the retrievability of the messages; *all* messages on the queue can be retrieved using the MQGET call.

The value of this attribute is zero or greater. The upper limit is determined by the environment:

- On AIX, HP-UX, z/OS, Solaris, Linux, and Windows, the value cannot exceed 999 999 999.
- On IBM i, the value cannot exceed 640 000.

Note: The storage space available to the queue might be exhausted even if there are fewer than *MaxQDepth* messages on the queue.

To determine the value of this attribute, use the MQIA_MAX_Q_DEPTH selector with the MQINQ call.

MsgDeliverySequence (MQLONG):

Local	Model	Alias	Remote	Cluster
X	X			

This determines the order in which the MQGET call returns messages to the application :

MQMDS_FIFO
Messages are returned in FIFO order (first in, first out).

An MQGET call returns the *first* message that satisfies the selection criteria specified on the call, regardless of the priority of the message.

MQMDS_PRIORITY
Messages are returned in priority order.

An MQGET call returns the *highest-priority* message that satisfies the selection criteria specified on the call. Within each priority level, messages are returned in FIFO order (first in, first out).

- On z/OS, if the queue has an *IndexType* of MQIT_GROUP_ID, the **MsgDeliverySequence** attribute specifies the order in which message groups are returned to the application. The particular sequence in which the groups are returned is determined by the position or priority of the first message in each group. The physical order of messages on the queue is not defined, as the queue is optimized for efficient retrieval of messages using the MQGMO_LOGICAL_ORDER option on the MQGET call.
- On z/OS, if *IndexType* is MQIT_GROUP_ID and *MsgDeliverySequence* is MQMDS_PRIORITY, the queue manager uses message priorities zero and one to optimize the retrieval of messages in logical order. As a result, the first message in a group must not have a priority of zero or one; if it does, the message is processed as though it had a priority of two. The *Priority* field in the MQMD structure is not changed.

If the relevant attributes are changed while there are messages on the queue, the delivery sequence is as follows:

- The order in which messages are returned by the MQGET call is determined by the values of the **MsgDeliverySequence** and **DefPriority** attributes in force for the queue at the time that the message arrives on the queue:
  - If *MsgDeliverySequence* is MQMDS_FIFO when the message arrives, the message is placed on the queue as though its priority were *DefPriority*. This does not affect the value of the *Priority* field in the message descriptor of the message; that field retains the value it had when the message was first put.
  - If *MsgDeliverySequence* is MQMDS_PRIORITY when the message arrives, the message is placed on the queue at the place appropriate to the priority given by the *Priority* field in the message descriptor.

  If the value of the **MsgDeliverySequence** attribute is changed while there are messages on the queue, the order of the messages on the queue is not changed.

  If the value of the **DefPriority** attribute is changed while there are messages on the queue, the messages are not necessarily delivered in FIFO order, even though the **MsgDeliverySequence** attribute is set to MQMDS_FIFO; those that were placed on the queue at the higher priority are delivered first.

To determine the value of this attribute, use the MQIA_MSG_DELIVERY_SEQUENCE selector with the MQINQ call.

NonPersistentMessageClass (MQLONG):

The reliability goal for nonpersistent messages.

Local	Model	Alias	Remote	Cluster
X	X			

This specifies the circumstances under which nonpersistent messages put on this queue are discarded:

MQNPM_CLASS_NORMAL
Nonpersistent messages are limited to the lifetime of the queue manager session; the messages are discarded in the event of a queue manager restart. This is valid only for non-shared queues, and is the default value.

MQNPM_CLASS_HIGH
The queue manager attempts to retain nonpersistent messages for the lifetime of the queue. Nonpersistent messages might still be lost in the event of a failure. This value is enforced for shared queues.

To determine the value of this attribute, use the MQIA_NPM_CLASS selector with the MQINQ call.

OpenInputCount (MQLONG):

This is the number of handles that are currently valid for removing messages from the queue by means of the MQGET call.

Local	Model	Alias	Remote	Cluster
X				

It is the total number of such handles known to the *local* queue manager. If the queue is a shared queue, the count does not include opens for input that were performed for the queue at other queue managers in the queue-sharing group to which the local queue manager belongs.

The count includes handles where an alias queue that resolves to this queue was opened for input. The count does not include handles where the queue was opened for actions that did not include input (for example, a queue opened only for browse).

The value of this attribute fluctuates as the queue manager operates.

To determine the value of this attribute, use the MQIA_OPEN_INPUT_COUNT selector with the MQINQ call.

OpenOutputCount (MQLONG):

This is the number of handles that are currently valid for adding messages to the queue by means of the MQPUT call.

Local	Model	Alias	Remote	Cluster
X				

It is the total number of such handles known to the *local* queue manager; it does not include opens for output that were performed for this queue at remote queue managers. If the queue is a shared queue, the count does not include opens for output that were performed for the queue at other queue managers in the queue-sharing group to which the local queue manager belongs.

The count includes handles where an alias queue that resolves to this queue was opened for output. The count does not include handles where the queue was opened for actions that did not include output (for example, a queue opened only for inquire).

The value of this attribute fluctuates as the queue manager operates.

To determine the value of this attribute, use the MQIA_OPEN_OUTPUT_COUNT selector with the MQINQ call.

ProcessName (MQCHAR48):

This is the name of a process object that is defined on the local queue manager. The process object identifies a program that can service the queue.

Local	Model	Alias	Remote	Cluster
X	X			

To determine the value of this attribute, use the MQCA_PROCESS_NAME selector with the MQINQ call. The length of this attribute is given by MQ_PROCESS_NAME_LENGTH.

PropertyControl (MQLONG):

Specifies how message properties are handled for messages that are retrieved from queues using the MQGET call with the MQGMO_PROPERTIES_AS_Q_DEF option.

Local	Model	Alias	Remote	Cluster
X	X	X		

The value is one of the following:

MQPROP_ALL
All properties of the message are included with the message when it is delivered to the application. The properties, except those in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data. If a message handle is supplied then the behavior is to return the properties in the message handle.

MQPROP_COMPATIBILITY
If the message contains a property with a prefix of mcd., jms., usr. or mqext., all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except those contained in the message descriptor (or extension), are discarded and are no longer accessible to the application. This is the default value; it allows applications which expect JMS related properties to be in an MQRFH2 header in the message data to continue to work unmodified. If a message handle is supplied then the behavior is to return the properties in the message handle..

MQPROP_FORCE_MQRFH2
Properties are always returned in the message data in an MQRFH2 header regardless of whether the application specifies a message handle. A valid message handle supplied in the MsgHandle field of the MQGMO structure on the MQGET call is ignored. Properties of the message are not accessible via the message handle.

MQPROP_NONE
All properties of the message, except those in the message descriptor (or extension), are removed from the message before the message is delivered to the application. If a message handle is supplied then the behavior is to return the properties in the message handle.

This parameter is applicable to Local, Alias and Model queues. To determine its value, use the MQIA_PROPERTY_CONTROL selector with the MQINQ call.

QDepthHighEvent (MQLONG):

This controls whether Queue Depth High events are generated.

Local	Model	Alias	Remote	Cluster
X	X			

A Queue Depth High event indicates that an application has put a message on a queue, and this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold (see the **QDepthHighLimit** attribute).

Note: The value of this attribute can change dynamically.

The value is one of the following:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_Q_DEPTH_HIGH_EVENT selector with the MQINQ call.

This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value.

QDepthHighLimit (MQLONG):

This is the threshold against which the queue depth is compared to generate a Queue Depth High event.

Local	Model	Alias	Remote	Cluster
X	X			

This event indicates that an application has put a message on a queue, and that this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold. See QDepthHighEvent attribute.

The value is expressed as a percentage of the maximum queue depth (**MaxQDepth** attribute), and is greater than or equal to 0 and less than or equal to 100. The default value is 80.

To determine the value of this attribute, use the MQIA_Q_DEPTH_HIGH_LIMIT selector with the MQINQ call.

This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value.

QDepthLowEvent (MQLONG):

This controls whether Queue Depth Low events are generated.

Local	Model	Alias	Remote	Cluster
X	X			

A Queue Depth Low event indicates that an application has retrieved a message from a queue, and that this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold (see QDepthLowLimit attribute).

Note: The value of this attribute can change dynamically.

The value is one of the following:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_Q_DEPTH_LOW_EVENT selector with the MQINQ call.

This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value.

QDepthLowLimit (MQLONG):

This is the threshold against which the queue depth is compared to generate a Queue Depth Low event.

Local	Model	Alias	Remote	Cluster
X	X			

This event indicates that an application has retrieved a message from a queue, and that this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold. See QDepthLowEvent attribute.

The value is expressed as a percentage of the maximum queue depth (**MaxQDepth** attribute), and is greater than or equal to 0 and less than or equal to 100. The default value is 20.

To determine the value of this attribute, use the MQIA_Q_DEPTH_LOW_LIMIT selector with the MQINQ call.

This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value.

QDepthMaxEvent (MQLONG):

This controls whether Queue Full events are generated. A Queue Full event indicates that a put to a queue has been rejected because the queue is full, that is, the queue depth has already reached its maximum value.

Local	Model	Alias	Remote	Cluster
X	X			

Note: The value of this attribute can change dynamically.

The value is one of the following:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_Q_DEPTH_MAX_EVENT selector with the MQINQ call.

This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value.

QDesc (MQCHAR64):

Use this field for descriptive commentary.

Local	Model	Alias	Remote	Cluster
X	X	X	X	X

The content of the field is of no significance to the queue manager, but the queue manager might require that the field contain only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, the field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the queue manager's character set (as defined by the **CodedCharSetId** queue manager attribute), those characters might be translated incorrectly if this field is sent to another queue manager.

To determine the value of this attribute, use the MQCA_Q_DESC selector with the MQINQ call. The length of this attribute is given by MQ_Q_DESC_LENGTH.

QName (MQCHAR48):

This is the name of a queue defined on the local queue manager.

Local	Model	Alias	Remote	Cluster
X		X	X	X

All queues defined on a queue manager share the same queue namespace. Therefore, an MQQT_LOCAL queue and an MQQT_ALIAS queue cannot have the same name.

To determine the value of this attribute, use the MQCA_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

QServiceInterval (MQLONG):

This is the service interval used for comparison to generate Service Interval High and Service Interval OK events.

Local	Model	Alias	Remote	Cluster
X	X			

See QServiceIntervalEvent attribute.

The value is in units of milliseconds, and is greater than or equal to zero, and less than or equal to 999 999 999.

To determine the value of this attribute, use the MQIA_Q_SERVICE_INTERVAL selector with the MQINQ call.

This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value.

QServiceIntervalEvent (MQLONG):

This controls whether Service Interval High or Service Interval OK events are generated.

Local	Model	Alias	Remote	Cluster
X	X			

- A Service Interval High event is generated when a check indicates that no messages have been retrieved from the queue for at least the time indicated by the **QServiceInterval** attribute.
- A Service Interval OK event is generated when a check indicates that messages have been retrieved from the queue within the time indicated by the **QServiceInterval** attribute.

Note: The value of this attribute can change dynamically.

The value is one of the following:

MQQSIE_HIGH
Queue Service Interval High events enabled.
- Queue Service Interval High events are **enabled** and
- Queue Service Interval OK events are **disabled**.

MQQSIE_OK
Queue Service Interval OK events enabled.
- Queue Service Interval High events are **disabled** and
- Queue Service Interval OK events are **enabled**.

MQQSIE_NONE
No queue service interval events enabled.
- Queue Service Interval High events are **disabled** and
- Queue Service Interval OK events are also **disabled**.

For shared queues, the value of this attribute is ignored; the value MQQSIE_NONE is assumed.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_Q_SERVICE_INTERVAL_EVENT selector with the MQINQ call.

On z/OS, you cannot use the MQINQ call to determine the value of this attribute.

QSGDisp (MQLONG):

This specifies the disposition of the queue.

Local	Model	Alias	Remote	Cluster
X		X	X	

The value is one of the following:

MQQSGD_Q_MGR
The object has queue manager disposition. This means that the object definition is known only to the local queue manager; the definition is not known to other queue managers in the queue-sharing group.

Each queue manager in the queue-sharing group can have an object with the same name and type as the current object, but these are separate objects and there is no correlation between them. Their attributes are not constrained to be the same as each other.

MQQSGD_COPY
The object is a local copy of a master object definition that exists in the shared repository. Each queue manager in the queue-sharing group can have its own copy of the object. Initially, all copies have the same attributes, but by using MQSC commands, you can alter each copy so that its attributes differ from those of the other copies. The attributes of the copies are resynchronized when the master definition in the shared repository is altered.

MQQSGD_SHARED
The object has shared disposition. This means that there exists in the shared repository a single instance of the object that is known to all queue managers in the queue-sharing group. When a queue manager in the group accesses the object, it accesses the single shared instance of the object.

To determine the value of this attribute, use the MQIA_QSG_DISP selector with the MQINQ call.

z/OS This attribute is supported only on z/OS.

QueueAccounting (MQLONG):

Local	Model	Alias	Remote	Cluster
X	X	X	X	

This controls the collection of accounting data for the queue. For accounting data to be collected for this queue, accounting data for this connection must also be enabled, using either the QMGR attribute ACCTQ or the Options field in the MQCNO structure on the MQCONNX call.

This attribute has one of the following values:

MQMON_Q_MGR
Accounting data for this queue is collected based on the setting of the QMGR attribute ACCTQ. This is the default setting.

MQMON_OFF
Do not collect accounting data for this queue.

MQMON_ON
Collect accounting data for this queue.

To determine the value of this attribute, use the MQIA_ACCOUNTING_Q selector with the MQINQ call.

QueueMonitoring (MQLONG):

Controls the collection of online monitoring data for queues.

Local	Model	Alias	Remote	Cluster
X	X			

The value is one of the following:

MQMON_Q_MGR
Collect monitoring data according to the setting of the **QueueMonitoring** queue manager attribute. This is the default value.

MQMON_OFF
Online monitoring data collection is turned off for this queue.

MQMON_LOW
If the value of the **QueueMonitoring** queue manager attribute is not MQMON_NONE, online monitoring data collection is turned on, with a low rate of data collection for this queue.

MQMON_MEDIUM
If the value of the **QueueMonitoring** queue manager attribute is not MQMON_NONE, online monitoring data collection is turned on, with a moderate rate of data collection for this queue.

MQMON_HIGH
If the value of the **QueueMonitoring** queue manager attribute is not MQMON_NONE, online monitoring data collection is turned on, with a high rate of data collection for this queue.

To determine the value of this attribute, use the MQIA_MONITORING_Q selector with the MQINQ call.

QueueStatistics (MQCHAR12):

Local	Model	Alias	Remote	Cluster
X	X	X	X	

This controls the collection of statistics data for the queue.

This attribute has one of the following values:

MQMON_Q_MGR
Accounting data for this queue is collected based on the setting of the QMGR attribute STATQ. This is the default setting.

MQMON_OFF
Switch off statistics data collection for this queue.

MQMON_ON
Enable statistics data collection for this queue.

QType (MQLONG):

Local	Model	Alias	Remote	Cluster
X		X	X	X

This is the type of queue; it has one of the following values:

MQQT_ALIAS
Alias queue definition.

MQQT_CLUSTER
Cluster queue.

MQQT_LOCAL
Local queue.

MQQT_REMOTE
Local definition of a remote queue.

To determine the value of this attribute, use the MQIA_Q_TYPE selector with the MQINQ call.

RemoteQMgrName (MQCHAR48):

Local	Model	Alias	Remote	Cluster
			X	

This is the name of the remote queue manager on which the queue **RemoteQName** is defined. If the **RemoteQName** queue has a **QSGDisp** value of MQQSGD_COPY or MQQSGD_SHARED, **RemoteQMgrName** can be the name of the queue-sharing group that owns **RemoteQName**.

If an application opens the local definition of a remote queue, **RemoteQMgrName** must not be blank and must not be the name of the local queue manager. If **XmitQName** is blank, the local queue with the same name as **RemoteQMgrName** is used as the transmission queue. If there is no queue with the name **RemoteQMgrName**, the queue identified by the **DefXmitQName** queue manager attribute is used.

If this definition is used for a queue manager alias, **RemoteQMgrName** is the name of the queue manager that is being aliased. It can be the name of the local queue manager. Otherwise, if **XmitQName** is blank when the open occurs, there must be a local queue with a name that is the same as **RemoteQMgrName**; this queue is used as the transmission queue.

If this definition is used for a reply-to alias, this name is the name of the queue manager that is to be the **ReplyToQMgr**.

Note: No validation is performed on the value specified for this attribute when the queue definition is created or modified.

To determine the value of this attribute, use the MQCA_REMOTE_Q_MGR_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_NAME_LENGTH.

RemoteQName (MQCHAR48):

Local	Model	Alias	Remote	Cluster
			X	

This is the name of the queue as it is known on the remote queue manager *RemoteQMgrName*.

If an application opens the local definition of a remote queue, when the open occurs *RemoteQName* must not be blank.

If this definition is used for a queue manager alias definition, when the open occurs *RemoteQName* must be blank.

If the definition is used for a reply-to alias, this name is the name of the queue that is to be the *ReplyToQ*.

Note: No validation is performed on the value specified for this attribute when the queue definition is created or modified.

To determine the value of this attribute, use the MQCA_REMOTE_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

RetentionInterval (MQLONG):

This is the period of time for which to retain the queue. After this time has elapsed, the queue is eligible for deletion.

Local	Model	Alias	Remote	Cluster
X	X			

The time is measured in hours, counting from the date and time when the queue was created. The creation date and time of the queue are recorded in the **CreationDate** and **CreationTime** attributes.

This information is provided to enable a housekeeping application or the operator to identify and delete queues that are no longer required.

Note: The queue manager never takes any action to delete queues based on this attribute, or to prevent the deletion of queues with a retention interval that has not expired; it is the user's responsibility to take any required action.

Use a realistic retention interval to prevent the accumulation of permanent dynamic queues (see DefinitionType attribute). However, this attribute can also be used with predefined queues.

To determine the value of this attribute, use the MQIA_RETENTION_INTERVAL selector with the MQINQ call.

Scope (MQLONG):

This controls whether an entry for this queue also exists in a cell directory.

Local	Model	Alias	Remote	Cluster
X		X	X	

A cell directory is provided by an installable Name service. The value is one of the following:

MQSCO_Q_MGR
The queue definition has queue manager scope: the definition of the queue does not extend beyond the queue manager that owns it. To open the queue for output from some other queue manager, either the name of the owning queue manager must be specified, or the other queue manager must have a local definition of the queue.

MQSCO_CELL
The queue definition has cell scope: the queue definition is also placed in a cell directory available to all the queue managers in the cell. The queue can be opened for output from any of the queue managers in the cell by specifying the name of the queue; the name of the queue manager that owns the queue need not be specified. However, the queue definition is not available to any queue manager in the cell that also has a local definition of a queue with that name, as the local definition takes precedence.

A cell directory is provided by an installable Name service.

Model and dynamic queues cannot have cell scope.

This value is only valid if a name service supporting a cell directory has been configured.

To determine the value of this attribute, use the MQIA_SCOPE selector with the MQINQ call.

Support for this attribute is subject to the following restrictions:
- On IBM i, the attribute is supported, but only MQSCO_Q_MGR is valid.
- On z/OS, the attribute is not supported.

Shareability (MQLONG):

This indicates whether the queue can be opened for input multiple times concurrently.

Local	Model	Alias	Remote	Cluster
X	X			

The value is one of the following:

MQQA_SHAREABLE
Queue is shareable.

Multiple opens with the MQOO_INPUT_SHARED option are allowed.

MQQA_NOT_SHAREABLE
Queue is not shareable.

An MQOPEN call with the MQOO_INPUT_SHARED option is treated as MQOO_INPUT_EXCLUSIVE.

To determine the value of this attribute, use the MQIA_SHAREABILITY selector with the MQINQ call.

StorageClass (MQCHAR8):

This is a user-defined name that defines the physical storage used to hold the queue. In practice, a message is written to disk only if it needs to be paged out of its memory buffer.

Local	Model	Alias	Remote	Cluster
X	X			

To determine the value of this attribute, use the MQCA_STORAGE_CLASS selector with the MQINQ call. The length of this attribute is given by MQ_STORAGE_CLASS_LENGTH.

z/OS This attribute is supported only on z/OS.

TriggerControl (MQLONG):

This controls whether trigger messages are written to an initiation queue to start an application to service the queue.

Local	Model	Alias	Remote	Cluster
X	X			

This is one of the following:

MQTC_OFF
No trigger messages are to be written for this queue. The value of *TriggerType* is irrelevant in this case.

MQTC_ON
Trigger messages are to be written for this queue when the appropriate trigger events occur.

To determine the value of this attribute, use the MQIA_TRIGGER_CONTROL selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

TriggerData (MQCHAR64):

This is free-format data that the queue manager inserts into the trigger message when a message arriving on this queue causes a trigger message to be written to the initiation queue.

Local	Model	Alias	Remote	Cluster
X	X			

The content of this data is of no significance to the queue manager. It is meaningful either to the trigger-monitor application that processes the initiation queue, or to the application that the trigger monitor starts.

The character string must not contain any nulls. It is padded to the right with blanks if necessary.

To determine the value of this attribute, use the MQCA_TRIGGER_DATA selector with the MQINQ call. To change the value of this attribute, use the MQSET call. The length of this attribute is given by MQ_TRIGGER_DATA_LENGTH.

TriggerDepth (MQLONG):

Local	Model	Alias	Remote	Cluster
X	X			

This is the number of messages of priority *TriggerMsgPriority* or greater that must be on the queue before a trigger message is written. This applies when *TriggerType* is set to MQTT_DEPTH. The value of *TriggerDepth* is one or greater. This attribute is not used otherwise.

To determine the value of this attribute, use the MQIA_TRIGGER_DEPTH selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

TriggerMsgPriority (MQLONG):

This is the message priority below which messages do not contribute to the generation of trigger messages (that is, the queue manager ignores these messages when deciding whether to generate a trigger message).

Local	Model	Alias	Remote	Cluster
X	X			

TriggerMsgPriority can be in the range zero (lowest) through *MaxPriority* (highest; see MaxPriority attribute); a value of zero causes all messages to contribute to the generation of trigger messages.

To determine the value of this attribute, use the MQIA_TRIGGER_MSG_PRIORITY selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

TriggerType (MQLONG):

This controls the conditions under which trigger messages are written as a result of messages arriving on this queue.

Local	Model	Alias	Remote	Cluster
X	X			

It has one of the following values:

MQTT_NONE
No trigger messages are written as a result of messages on this queue. This has the same effect as setting *TriggerControl* to MQTC_OFF.

MQTT_FIRST
A trigger message is written whenever the number of messages of priority *TriggerMsgPriority* or greater on the queue changes from 0 to 1.

MQTT_EVERY
A trigger message is written whenever a message of priority *TriggerMsgPriority* or greater arrives on the queue.

MQTT_DEPTH
A trigger message is written whenever the number of messages of priority *TriggerMsgPriority* or greater on the queue equals or exceeds *TriggerDepth*. After the trigger message has been written, *TriggerControl* is set to MQTC_OFF to prevent further triggering until it is explicitly turned on again.

To determine the value of this attribute, use the MQIA_TRIGGER_TYPE selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

Usage (MQLONG):

This indicates what the queue is used for.

Local	Model	Alias	Remote	Cluster
X	X			

The value is one of the following:

MQUS_NORMAL
This is a queue that applications use when putting and getting messages; the queue is not a transmission queue.

MQUS_TRANSMISSION
This is a queue used to hold messages destined for remote queue managers. When an application sends a message to a remote queue, the local queue manager stores the message temporarily on the appropriate transmission queue in a special format. A message channel agent then reads the message from the transmission queue, and transports the message to the remote queue manager. For more information about transmission queues, see Defining a transmission queue.

Only privileged applications can open a transmission queue for MQOO_OUTPUT to put messages on it directly. Usually, only utility applications do this. Ensure that the message data format is correct (see "MQXQH - Transmission-queue header" on page 2612) or errors might occur during the transmission process. Context is not passed or set unless one of the MQPMO_*_CONTEXT context options is specified.

To determine the value of this attribute, use the MQIA_USAGE selector with the MQINQ call.

XmitQName (MQCHAR48):

This is the transmission queue name. If this attribute is nonblank when an open occurs, either for a remote queue or for a queue manager alias definition, it specifies the name of the local transmission queue to be used for forwarding the message.

Local	Model	Alias	Remote	Cluster
			X	

If **XmitQName** is blank, the local queue with a name that is the same as **RemoteQMgrName** is used as the transmission queue. If there is no queue with the name **RemoteQMgrName**, the queue identified by the **DefXmitQName** queue manager attribute is used.

This attribute is ignored if the definition is being used as a queue manager alias and **RemoteQMgrName** is the name of the local queue manager. It is also ignored if the definition is used as a reply-to queue alias definition.

To determine the value of this attribute, use the MQCA_XMIT_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

Attributes for namelists:

The following table summarizes the attributes that are specific to namelists. The attributes are described in alphabetical order.

Namelists are supported on all IBM MQ systems, plus IBM MQ MQI clients connected to these systems.

Note: The names of the attributes shown in this section are descriptive names used with the MQINQ and MQSET calls; the names are the same as for the PCF commands. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see Script (MQSC) Commands for more information.

Table 276. Attributes for namelists

Attribute	Description
AlterationDate	Date when definition was last changed
AlterationTime	Time when definition was last changed
NameCount	Number of names in namelist
NamelistDesc	Namelist description
NamelistName	Namelist name
Names	A list of *NameCount* names
NamelistType	Namelist type
QSGDisp	Queue-sharing group disposition

AlterationDate (MQCHAR12):

This is the date when the definition was last changed. The format of the date is `YYYY-MM-DD`, padded with two trailing blanks to make the length 12 bytes.

To determine the value of this attribute, use the MQCA_ALTERATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_DATE_LENGTH.

AlterationTime (MQCHAR8):

This is the time when the definition was last changed. The format of the time is `HH.MM.SS`.

To determine the value of this attribute, use the MQCA_ALTERATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_TIME_LENGTH.

NameCount (MQLONG):

This is the number of names in the namelist. It is greater than or equal to zero. The following value is defined:

MQNC_MAX_NAMELIST_NAME_COUNT
Maximum number of names in a namelist.

To determine the value of this attribute, use the MQIA_NAME_COUNT selector with the MQINQ call.

NamelistDesc (MQCHAR64):

Use this field for descriptive commentary; its value is established by the definition process. The content of the field is of no significance to the queue manager, but the queue manager might require that the field contain only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, this field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the queue manager's character set (as defined by the **CodedCharSetId** queue manager attribute), those characters might be translated incorrectly if this field is sent to another queue manager.

To determine the value of this attribute, use the MQCA_NAMELIST_DESC selector with the MQINQ call.

The length of this attribute is given by MQ_NAMELIST_DESC_LENGTH.

NamelistName (MQCHAR48):

This is the name of a namelist that is defined on the local queue manager. For more information about namelist names, see the Other object names section.

Each namelist has a name that is different from the names of other namelists belonging to the queue manager, but might duplicate the names of other queue manager objects of different types (for example, queues).

To determine the value of this attribute, use the MQCA_NAMELIST_NAME selector with the MQINQ call.

The length of this attribute is given by MQ_NAMELIST_NAME_LENGTH.

NamelistType (MQLONG):

This specifies the nature of the names in the namelist, and indicates how the namelist is used. It is one of the following values:

MQNT_NONE
Namelist with no assigned type.

MQNT_Q
Namelist containing the names of queues.

MQNT_CLUSTER
Namelist containing the names of clusters.

MQNT_AUTH_INFO
Namelist containing the names of authentication-information objects.

To determine the value of this attribute, use the MQIA_NAMELIST_TYPE selector with the MQINQ call.

z/OS This attribute is supported only on z/OS.

Names (MQCHAR48xNameCount):

This is a list of *NameCount* names, where each name is the name of an object that is defined to the local queue manager. For more information about object names, see Rules for naming IBM MQ objects.

To determine the value of this attribute, use the MQCA_NAMES selector with the MQINQ call.

The length of each name in the list is given by MQ_OBJECT_NAME_LENGTH.

QSGDisp (MQLONG):

This specifies the disposition of the namelist. The value is one of the following:

MQQSGD_Q_MGR
The object has queue manager disposition: the object definition is known only to the local queue manager; the definition is not known to other queue managers in the queue-sharing group.

Each queue manager in the queue-sharing group can have an object with the same name and type as the current object, but these are separate objects and there is no correlation between them. Their attributes are not constrained to be the same as each other.

MQQSGD_COPY
The object is a local copy of a master object definition that exists in the shared repository. Each queue manager in the queue-sharing group can have its own copy of the object. Initially, all

copies have the same attributes, but you can alter each copy, using MQSC commands, so that its attributes differ from those of the other copies. The attributes of the copies are resynchronized when the master definition in the shared repository is altered.

To determine the value of this attribute, use the MQIA_QSG_DISP selector with the MQINQ call.

z/OS This attribute is supported only on z/OS.

Attributes for process definitions:

The following table summarizes the attributes that are specific to process definitions. The attributes are described in alphabetical order.

Note: The names of the attributes in this section are descriptive names used with the MQINQ and MQSET calls; the names are the same as for the PCF commands. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see Script (MQSC) Commands for more information.

Table 277. Attributes for process definitions

Attribute	Description
AlterationDate	Date when definition was last changed
AlterationTime	Time when definition was last changed
ApplId	Application identifier
ApplType	Application type
EnvData	Environment data
ProcessDesc	Process description
ProcessName	Process name
QSGDisp	Queue-sharing group disposition
UserData	User data

AlterationDate (MQCHAR12):

This is the date when the definition was last changed. The format of the date is `YYYY-MM-DD`, padded with two trailing blanks to make the length 12 bytes.

To determine the value of this attribute, use the MQCA_ALTERATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_DATE_LENGTH.

AlterationTime (MQCHAR8):

This is the time when the definition was last changed. The format of the time is `HH.MM.SS`.

To determine the value of this attribute, use the MQCA_ALTERATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_TIME_LENGTH.

ApplId (MQCHAR256):

This is a character string that identifies the application to be started. This information is for use by a trigger-monitor application that processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message.

The meaning of *ApplId* is determined by the trigger-monitor application. The trigger monitor provided by IBM MQ requires *ApplId* to be the name of an executable program. The following notes apply to the environments indicated:

- On z/OS, *ApplId* must be:

  - A CICS transaction identifier, for applications started using the CICS trigger-monitor transaction CKTI
  - An IMS transaction identifier, for applications started using the IMS trigger monitor CSQQTRMN
- On Windows systems, the program name can be prefixed with a drive and directory path.
- On UNIX, the program name can be prefixed with a directory path.

The character string cannot contain any nulls. It is padded to the right with blanks if necessary.

To determine the value of this attribute, use the MQCA_APPL_ID selector with the MQINQ call. The length of this attribute is given by MQ_PROCESS_APPL_ID_LENGTH.

ApplType (MQLONG):

This identifies the nature of the program to be started in response to the receipt of a trigger message. This information is for use by a trigger-monitor application that processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message.

ApplType can have any value, but the following values are recommended for standard types; restrict user-defined application types to values in the range MQAT_USER_FIRST through MQAT_USER_LAST:

MQAT_AIX
AIX application (same value as MQAT_UNIX).

MQAT_BATCH
Batch application

MQAT_BROKER
Broker application

MQAT_CICS
CICS transaction.

MQAT_CICS_BRIDGE
CICS bridge application.

MQAT_CICS_VSE
CICS/VSE transaction.

MQAT_DOS
IBM MQ MQI client application on PC DOS.

MQAT_IMS
IMS application.

MQAT_IMS_BRIDGE
IMS bridge application.

MQAT_JAVA
Java application.

MQAT_MVS
MVS or TSO application (same value as MQAT_ZOS).

MQAT_NOTES_AGENT
Lotus Notes Agent application.

MQAT_NSK
HP Integrity NonStop Server application.

MQAT_OS390
OS/390 application (same value as MQAT_ZOS).

MQAT_OS400
IBM i application.

MQAT_RRS_BATCH
RRS batch application.

MQAT_UNIX
UNIX application.

MQAT_UNKNOWN
Application of unknown type.

MQAT_USER
User application.

MQAT_VOS
Stratus VOS application.

MQAT_WINDOWS
16-bit Windows application.

MQAT_WINDOWS_NT
32-bit Windows application.

MQAT_WLM
z/OS workload manager application.

MQAT_XCF
XCF.

MQAT_ZOS
z/OS application.

MQAT_USER_FIRST
Lowest value for user-defined application type.

MQAT_USER_LAST
Highest value for user-defined application type.

To determine the value of this attribute, use the MQIA_APPL_TYPE selector with the MQINQ call.

EnvData (MQCHAR128):

This is a character string that contains environment-related information pertaining to the application to be started. This information is for use by a trigger-monitor application that processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message.

The meaning of *EnvData* is determined by the trigger-monitor application. The trigger monitor provided by IBM MQ appends *EnvData* to the parameter list passed to the started application. The parameter list consists of the MQTMC2 structure, followed by one blank, followed by *EnvData* with trailing blanks removed. The following notes apply to the environments indicated:

- On z/OS:
  - *EnvData* is not used by the trigger-monitor applications provided by IBM MQ.
  - If ApplType is MQAT_WLM, you can supply default values in EnvData for the ServiceName and ServiceStep fields in the work information header (MQWIH).
- On UNIX, *EnvData* can be set to the & character to run the started application in the background.

The character string cannot contain any nulls. It is padded to the right with blanks if necessary.

To determine the value of this attribute, use the MQCA_ENV_DATA selector with the MQINQ call. The length of this attribute is given by MQ_PROCESS_ENV_DATA_LENGTH.

ProcessDesc (MQCHAR64):

Use this field for descriptive commentary. The content of the field is of no significance to the queue manager, but the queue manager might require that the field contain only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, the field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the queue manager's character set (as defined by the **CodedCharSetId** queue manager attribute), those characters might be translated incorrectly if this field is sent to another queue manager.

To determine the value of this attribute, use the MQCA_PROCESS_DESC selector with the MQINQ call.

The length of this attribute is given by MQ_PROCESS_DESC_LENGTH.

ProcessName (MQCHAR48):

This is the name of a process definition that is defined on the local queue manager.

Each process definition has a name that is different from the names of other process definitions belonging to the queue manager. But the name of the process definition might be the same as the names of other queue manager objects of different types (for example, queues).

To determine the value of this attribute, use the MQCA_PROCESS_NAME selector with the MQINQ call.

The length of this attribute is given by MQ_PROCESS_NAME_LENGTH.

QSGDisp (MQLONG):

This specifies the disposition of the process definition. The value is one of the following:

MQQSGD_Q_MGR
: The object has queue manager disposition: the object definition is known only to the local queue manager; the definition is not known to other queue managers in the queue-sharing group.

  Each queue manager in the queue-sharing group can have an object with the same name and type as the current object, but these are separate objects and there is no correlation between them. Their attributes are not constrained to be the same as each other.

MQQSGD_COPY
: The object is a local copy of a master object definition that exists in the shared repository. Each queue manager in the queue-sharing group can have its own copy of the object. Initially, all copies have the same attributes, but you can alter each copy, using MQSC commands, so that its attributes differ from those of the other copies. The attributes of the copies are resynchronized when the master definition in the shared repository is altered.

To determine the value of this attribute, use the MQIA_QSG_DISP selector with the MQINQ call.

z/OS This attribute is supported only on z/OS.

UserData (MQCHAR128):

`UserData` is a character string that contains user information pertaining to the application to be started. This information is for use by a trigger-monitor application that processes messages on the initiation queue, or the application that is started by the trigger monitor. The information is sent to the initiation queue as part of the trigger message.

The meaning of *UserData* is determined by the trigger-monitor application. The trigger monitor provided by IBM MQ passes *UserData* to the started application as part of the parameter list. The parameter list consists of the MQTMC2 structure (containing *UserData*), followed by one blank, followed by *EnvData* with trailing blanks removed.

The character string cannot contain any nulls. It is padded to the right with blanks if necessary. For Microsoft Windows, the character string must not contain double quotation marks if the process definition is going to be passed to **runmqtrm**.

To determine the value of this attribute, use the MQCA_USER_DATA selector with the MQINQ call. The length of this attribute is given by MQ_PROCESS_USER_DATA_LENGTH.

Return codes

For each IBM MQ Message Queue Interface (MQI) and IBM MQ Administration Interface (MQAI) call, a **completion** code and a **reason** code are returned by the queue manager or by an exit routine, to indicate the success or failure of the call.

Applications must not depend upon errors being checked for in a specific order, except where specifically noted. If more than one completion code or reason code could arise from a call, the particular error reported depends on the implementation.

Applications checking for successful completion following an IBM MQ API call must always check the completion code. Do not assume the completion code value, based on the value of the reason code.

Completion codes

The completion code parameter (*CompCode*) allows the caller to see quickly whether the call completed successfully, completed partially, or failed. The following is a list of completion codes, with more detail than is given in the call descriptions:

MQCC_OK
The call completed fully; all output parameters have been set. The **Reason** parameter always has the value MQRC_NONE in this case.

MQCC_WARNING
The call completed partially. Some output parameters might have been set in addition to the *CompCode* and *Reason* output parameters. The **Reason** parameter gives additional information about the partial completion.

MQCC_FAILED
The processing of the call did not complete. The state of the queue manager is unchanged, except where specifically noted. The *CompCode* and *Reason* output parameters have been set; other parameters are unchanged, except where noted.

The reason might be a fault in the application program, or it might be the result of some situation external to the program, for example the user's authority might have been revoked. The **Reason** parameter gives additional information about the error.

Reason codes

The reason code parameter (*Reason*) qualifies the completion code parameter (*CompCode*).

If there is no special reason to report, MQRC_NONE is returned. A successful call returns MQCC_OK and MQRC_NONE.

If the completion code is either MQCC_WARNING or MQCC_FAILED, the queue manager always reports a qualifying reason; details are given under each call description.

Where user exit routines set completion codes and reasons, they must adhere to these rules. In addition, any special reason values defined by user exits must be less than zero, to ensure that they do not conflict with values defined by the queue manager. Exits can set reasons already defined by the queue manager, where appropriate.

Reason codes also occur in:

- The *Reason* field of the MQDLH structure
- The *Feedback* field of the MQMD structure

For complete descriptions of reason codes, see Reason codes.

Rules for validating MQI options

This section lists the situations that produce an MQRC_OPTIONS_ERROR reason code from an MQOPEN, MQPUT, MQPUT1, MQGET, MQCLOSE, or MQSUB call.

MQOPEN call

For the options of the MQOPEN call:

- At least *one* of the following must be specified:
  - MQOO_BROWSE
  - MQOO_INPUT_EXCLUSIVE [1]
  - MQOO_INPUT_SHARED [1]
  - MQOO_INPUT_AS_Q_DEF [1]
  - MQOO_INQUIRE
  - MQOO_OUTPUT
  - MQOO_SET
  - MQOO_BIND_ON_OPEN [2]
  - MQOO_BIND_NOT_FIXED [2]
  - MQOO_BIND_ON_GROUP [2]
  - MQOO_BIND_AS_Q_DEF [2]
- Only *one* of the following is allowed:
  - MQOO_READ_AHEAD
  - MQOO_NO_READ_AHEAD
  - MQOO_READ_AHEAD_AS_Q_DEF

1. Only *one* of the following is allowed:
   - MQOO_INPUT_EXCLUSIVE
   - MQOO_INPUT_SHARED
   - MQOO_INPUT_AS_Q_DEF
2. Only *one* of the following is allowed:
   - MQOO_BIND_ON_OPEN
   - MQOO_BIND_NOT_FIXED
   - MQOO_BIND_ON_GROUP
   - MQOO_BIND_AS_Q_DEF

   Note: The options that are listed previously are mutually exclusive. However, as the value of MQOO_BIND_AS_Q_DEF is zero, specifying it with either of the other two bind options does not result in reason code MQRC_OPTIONS_ERROR. MQOO_BIND_AS_Q_DEF is provided to aid program documentation.

- If MQOO_SAVE_ALL_CONTEXT is specified, one of the MQOO_INPUT_* options must also be specified.
- If one of the MQOO_SET_*_CONTEXT or MQOO_PASS_*_CONTEXT options are specified, MQOO_OUTPUT must also be specified.
- If MQOO_CO_OP is specified, MQOO_BROWSE must also be specified
- If MQOO_NO_MULTICAST is specified, MQOO_OUTPUT must also be specified.

MQPUT call

For the put-message options:
- The combination of MQPMO_SYNCPOINT and MQPMO_NO_SYNCPOINT is not allowed.
- Only *one* of the following is allowed:
  - MQPMO_DEFAULT_CONTEXT
  - MQPMO_NO_CONTEXT
  - MQPMO_PASS_ALL_CONTEXT
  - MQPMO_PASS_IDENTITY_CONTEXT
  - MQPMO_SET_ALL_CONTEXT
  - MQPMO_SET_IDENTITY_CONTEXT
- Only *one* of the following is allowed:
  - MQPMO_ASYNC_RESPONSE
  - MQPMO_SYNC_RESPONSE
  - MQPMO_RESPONSE_AS_TOPIC_DEF
  - MQPMO_RESPONSE_AS_Q_DEF
- MQPMO_ALTERNATE_USER_AUTHORITY is not allowed (it is valid only on the MQPUT1 call).

MQPUT1 call

For the put-message options, the rules are the same as for the MQPUT call, except for the following:
- MQPMO_ALTERNATE_USER_AUTHORITY is allowed.
- MQPMO_LOGICAL_ORDER is not allowed.

MQGET call

For the get-message options:
- Only *one* of the following is allowed:
  - MQGMO_NO_SYNCPOINT
  - MQGMO_SYNCPOINT
  - MQGMO_SYNCPOINT_IF_PERSISTENT
- Only *one* of the following is allowed:
  - MQGMO_BROWSE_FIRST
  - MQGMO_BROWSE_MSG_UNDER_CURSOR
  - MQGMO_BROWSE_NEXT
  - MQGMO_MSG_UNDER_CURSOR
- MQGMO_SYNCPOINT is not allowed with any of the following:
  - MQGMO_BROWSE_FIRST
  - MQGMO_BROWSE_MSG_UNDER_CURSOR
  - MQGMO_BROWSE_NEXT
  - MQGMO_LOCK

  - MQGMO_UNLOCK
- MQGMO_SYNCPOINT_IF_PERSISTENT is not allowed with any of the following:
  - MQGMO_BROWSE_FIRST
  - MQGMO_BROWSE_MSG_UNDER_CURSOR
  - MQGMO_BROWSE_NEXT
  - MQGMO_COMPLETE_MSG
  - MQGMO_UNLOCK
- MQGMO_MARK_SKIP_BACKOUT requires MQGMO_SYNCPOINT to be specified.
- The combination of MQGMO_WAIT and MQGMO_SET_SIGNAL is not allowed.
- If MQGMO_LOCK is specified, one of the following must also be specified:
  - MQGMO_BROWSE_FIRST
  - MQGMO_BROWSE_MSG_UNDER_CURSOR
  - MQGMO_BROWSE_NEXT
- If MQGMO_UNLOCK is specified, only the following values are allowed:
  - MQGMO_NO_SYNCPOINT
  - MQGMO_NO_WAIT

MQCLOSE call

For the options of the MQCLOSE call:

- The combination of MQCO_DELETE and MQCO_DELETE_PURGE is not allowed.
- Only one of the following is allowed:
  - MQCO_KEEP_SUB
  - MQCO_REMOVE_SUB

MQSUB call

For the options of the MQSUB call:

- At least one of the following must be specified:
  - MQSO_ALTER
  - MQSO_RESUME
  - MQSO_CREATE
- Only one of the following is allowed:
  - MQSO_DURABLE
  - MQSO_NON_DURABLE

  Note: The options that are listed previously are mutually exclusive. However, as the value of MQSO_NON_DURABLE is zero, specifying it with MQSO_DURABLE does not result in reason code MQRC_OPTIONS_ERROR. MQSO_NON_DURABLE is provided to aid program documentation.
- The combination of MQSO_GROUP_SUB and MQSO_MANAGED is not allowed.
- MQSO_GROUP_SUB requires MQSO_SET_CORREL_ID to be specified.
- Only one of the following is allowed:
  - MQSO_ANY_USERID
  - MQSO_FIXED_USERID
- MQSO_NEW_PUBLICATIONS_ONLY is allowed in combination with:
  - MQSO_CREATE
  - MQSO_ALTER, if MQSO_NEW_PUBLICATIONS_ONLY was set on the original subscription

- The combination of MQSO_PUBLICATIONS_ON_REQUEST and SubLevel greater than 1 is not allowed.
- Only one of the following is allowed:
  - MQSO_WILDCARD_CHAR
  - MQSO_WILDCARD_TOPIC
- MQSO_NO_MULTICAST requires MQSO_MANAGED to be specified.

Queued publish/subscribe command messages

An application can use MQRFH2 command messages to control a queued publish/subscribe application.

An application that is using MQRFH2 for publish/subscribe can send the following command messages to the SYSTEM.BROKER.CONTROL.QUEUE:

- "Delete Publication message" on page 2881
- "Deregister Subscriber message" on page 2882
- "Publish message" on page 2885
- "Register Subscriber message" on page 2888
- "Request Update message" on page 2893

If you are writing queued publish/subscribe applications, you must understand these messages, the queue manager response message, and the message descriptor (MQMD); see the following information:

- "Queue Manager Response message" on page 2895
- "MQMD settings for publications forwarded by a queue manager" on page 2900
- "MQMD settings in queue manager response messages" on page 2901
- "Publish/subscribe reason codes" on page 2896

The commands are contained in a psc folder in the **NameValueData** field of the MQRFH2 header. The message that can be sent by a broker in response to a command message is contained in a pscr folder.

The descriptions of each command list the properties that can be contained in a folder. Unless otherwise specified, the properties are optional and can occur only once.

Names of properties are shown as <Command>.

Values must be in string format, for example: Publish.

A string constant representing the value of a property is shown in parentheses, for example: (MQPSC_PUBLISH).

String constants are defined in the header file cmqpsc.h which is supplied with the queue manager.

Delete Publication message:

The **Delete Publication** command message is sent to a queue manager from a publisher, or from another queue manager, to tell the queue manager to delete any retained publications for the specified topics.

This message is sent to a queue monitored by the queue manager's queued publish/subscribe interface.

The input queue should be the queue that the original publication was sent to.

If you have the authority for some, but not all, of the topics that are specified in the **Delete Publication** command message, only those topics are deleted. A **Broker Response** message indicates which topics are not deleted.

Similarly, if a **Publish** command contains more than one topic, a **Delete Publication** command matching some, but not all, of those topics deletes only the publications for the topics that are specified in the **Delete Publication** command.

See "MQMD settings for publications forwarded by a queue manager" on page 2900 for details of the message descriptor (MQMD) parameters that are needed when sending a command message to the queue manager.

Properties:

Command (*MQPSC_COMMAND*)
The value is `DeletePub` (*MQPSC_DELETE_PUBLICATION*).

This property must be specified.

Topic> (*MQPSC_TOPIC*)
The value is a string that contains a topic for which retained publications are to be deleted. Wildcard characters can be included in the string to delete publications on more than one topic.

This property must be specified; it can be repeated for as many topics as needed.

DelOpt (*MQPSC_DELETE_OPTION*)
The delete options property can take one of the following values:
Local (*MQPSC_LOCAL*)

All retained publications for the specified topics are deleted at the local queue manager (that is, the queue manager to which this message is sent), whether they were published with the `Local` option or not.

Publications at other queue managers are not affected.

None (*MQPSC_NONE*)

All options take their default values. This has the same effect as omitting the `DelOpt` property. If other options are specified at the same time, `None` is ignored.

The default if this property is omitted is that all retained publications for the specified topics are deleted at all queue managers in the network, regardless of whether they were published with the `Local` option.

Example:
Here is an example of `NameValueData` for a **Delete Publication** command message. This is used by the sample application to delete, at the local queue manager, the retained publication that contains the latest score in the match between Team1 and Team2.

```
<psc>
  <Command>DeletePub</Command>
  <Topic>Sport/Soccer/State/LatestScore/Team1 Team2</Topic>
  <DelOpt>Local</DelOpt>
</psc>
```

Deregister Subscriber message:

The **Deregister Subscriber** command message is sent to a queue manager by a subscriber, or by another application on behalf of a subscriber, to indicate that it no longer wants to receive messages matching the given parameters.

This message is sent to SYSTEM.BROKER.CONTROL.QUEUE, the queue manager's control queue. The user must have the necessary authority to put a message onto this queue.

See MQMD settings for publications forwarded by a queue manager for details of the message descriptor (MQMD) parameters that are needed when sending a command message to the queue manager.

An individual subscription can be deregistered by specifying the corresponding topic, subscription point and filter values of the original subscription. If any of the values were not specified (that is, they took the default values) in the original subscription, they should be omitted when the subscription is deregistered.

All subscriptions for a subscriber, or a group of subscribers, can be deregistered by using the `DeregAll` option. For example, if `DeregAll` is specified, together with a subscription point (but no topic or filter), then all subscriptions for the subscriber on the specified subscription point are deregistered, regardless of the topic and filter. Any combination of topic, filter and subscription point is allowed; if all three are specified only one subscription can match, and the `DeregAll` option is ignored.

The message must be sent by the subscriber that registered the subscription; this is confirmed by checking the subscriber's user ID.

Subscriptions can also be deregistered by a system administrator using MQSC or PCF commands. However, the subscriptions registered with a temporary dynamic queue are associated with the queue, not just the queue name. If the queue is deleted, either explicitly, or by the application disconnecting from the queue manager, it is no longer possible to use the **Deregister Subscriber** command to deregister the subscriptions for that queue. The subscriptions can be deregistered using the developer workbench, and they are removed automatically by the queue manager the next time that it matches a publication to the subscription, or the next time the queue manager restarts. Under normal circumstances, applications should deregister their subscriptions before deleting the queue, or disconnecting from the queue manager.

If a subscriber sends a message to deregister a subscription, and receives a response message to say that this was processed successfully, some publications might still reach the subscriber queue if they were being processed by the queue manager at the same time as the subscription was being deregistered. If the messages are not removed from the queue, there might be a buildup of unprocessed messages on the subscriber queue. If the application executes a loop that includes an MQGET call with the appropriate `CorrelId` after sleeping for a while, these messages are cleared off the queue.

Similarly, if the subscriber uses a permanent dynamic queue, and deregisters and closes the queue with the *MQCO_DELETE_PURGE* option on an MQCLOSE call, the queue might not be empty. If any publications from the queue manager are not yet committed when the queue is deleted, an MQRC_Q_NOT_EMPTY return code is issued by the MQCLOSE call. The application can avoid this problem by sleeping and reissuing the MQCLOSE call from time to time.

Properties:

Command (*MQPSC_COMMAND*)
The value is DeregSub (*MQPSC_DEREGISTER_SUBSCRIBER*).

This property must be specified.

Topic (*MQPSC_TOPIC*)
The value is a string that contains the topic to be deregistered.

This property can, optionally, be repeated if multiple topics are to be deregistered. It can be omitted if DeregAll is specified in <RegOpt>.

The topics that are specified can be a subset of those that are registered if the subscriber wants to retain subscriptions for other topics. Wildcard characters are allowed, but a topic string that contains wildcard characters must exactly match the corresponding string that was specified in the **Deregister Subscriber** command message.

SubPoint (*MQPSC_SUBSCRIPTION_POINT*)
The value is a string that specifies the subscription point from which the subscription is to be detached.

This property must not be repeated. It can be omitted if a <Topic> is specified, or if DeregAll is specified in <RegOpt>. If you omit this property, the following happens:
- If you do **not** specify DeregAll, subscriptions matching the <Topic> property (and the <Filter> property, if present) are deregistered from the default subscription point.
- If you specify DeregAll, all subscriptions (matching the <Topic> and <Filter> properties if present) are deregistered from all subscription points.

Note that you cannot specify the default subscription point explicitly. Therefore, there is no way of deregistering all subscriptions from this subscription point only; you must specify the topics.

SubIdentity (*MQPSC_SUBSCRIPTION_IDENTITY*)
This is a variable-length string with a maximum length of 64 characters. It is used to represent an application with an interest in a subscription. The queue manager maintains a set of subscriber identities for each subscription. Each subscription can allow its identity set to hold only a single identity, or an unlimited number of identities.

If the SubIdentity is in the identity set for the subscription then it is removed from the set. If the identity set becomes empty as a result of this, the subscription is removed from the queue manager, unless LeaveOnly is specified as a value of the RegOpt property. If the identity set still contains other identities then the subscription is not removed from the queue manager, and publication flow is not interrupted.

If SubIdentity is specified, but the SubIdentity is not in the identity set for the subscription, then the **Deregister Subscriber** command fails with the return code *MQRCCF_SUB_IDENTITY_ERROR*.

Filter (*MQPSC_FILTER*)
The value is a string specifying the filter to be deregistered. It must match exactly, including case and any spaces, a subscription filter that has been previously registered.

This property can, optionally, be repeated if more than one filter is to be deregistered. It can be omitted if a <Topic> is specified, or if DeregAll is specified in <RegOpt>.

The filters specified can be a subset of those registered if the subscriber wants to retain subscriptions for other filters.

RegOpt (*MQPSC_REGISTRATION_OPTION*)
The registration options property can take the following values:

DeregAll
(*MQPSC_DEREGISTER_ALL*)

All matching subscriptions registered for this subscriber are to be deregistered.

If you specify DeregAll:
- <Topic>, <SubPoint>, and <Filter> can be omitted.
- <Topic> and <Filter> can be repeated, if required.
- <SubPoint> must not be repeated.

If you do **not** specify DeregAll:
- <Topic> must be specified, and can be repeated if required.
- <SubPoint> and <Filter> can be omitted.
- <SubPoint> must not be repeated.
- <Filter> can be repeated, if required.

If topics and filters are both repeated, then all subscriptions matching all combinations of the two are removed. For example, a **Deregister Subscriber** command that specifies three topics and three filters will attempt to remove nine subscriptions.

CorrelAsId

(*MQPSC_CORREL_ID_AS_IDENTITY*)

The CorrelId in the message descriptor (MQMD), which must not be zero, is used to identify the subscriber. It must match the CorrelId used in the original subscription.

FullResp

(*MQPSC_FULL_RESPONSE*)

When FullResp is specified all attributes of the subscription are returned in the response message, if the command does not fail.

When FullResp is specified DeregAll is not permitted in the **Deregister Subscriber** command. It is also not possible to specify multiple topics. The command fails with return code *MQRCCF_REG_OPTIONS_ERROR*, in both cases.

LeaveOnly

(*MQPSC_LEAVE_ONLY*)

When you specify this with a SubIdentity which is in the identity set for the subscription the SubIdentity is removed from the identity set for the subscription. The subscription is not removed from the queue manager, even if the resulting identity set is empty. If the SubIdentity value is not in the identity set the command fails with return code *MQRCCF_SUB_IDENTITY_ERROR*.

If LeaveOnly is specified with no SubIdentity, the command fails with return code *MQRCCF_REG_OPTIONS_ERROR*.

If neither LeaveOnly nor a SubIdentity are specified, then the subscription is removed regardless of the contents of the identity set for the subscription.

None

(MQPSC_NONE)

All options take their default values. This has the same effect as omitting the registration options property. If other options are specified at the same time, None is ignored.

VariableUserId

(*MQPSC_VARIABLE_USER_ID*)

When specified the identity of the subscriber (queue, queue manager and correlid) is not restricted to a single userid. This differs from the existing behavior of the queue manager that associates the userid of the original registration message with the subscriber's identity and from then on prevents any other user using that identity. If a new subscriber tries to use the same identity, the return code *MQRCCF_DUPLICATE_SUBSCRIPTION* is returned.

Any user can modify or deregister the subscription when they have suitable authority, avoiding the existing check that the userid must match that of the original subscriber.

To add this option to an existing subscription the command must come from the same userid as the original subscription itself.

If the subscription to be deregistered has VariableUserId set this must be set at deregister time to indicate which subscription is being deregistered. Otherwise, the userid of the **Deregister Subscriber** command is used to identify the subscription. This is overridden, along with the other subscriber identifiers, if a subscription name is supplied.

The default, if this property is omitted, is that no registration options are set.

QMgrName (*MQPSC_Q_MGR_NAME*)
The value is the queue manager name for the subscriber queue. It must match the QMgrName used in the original subscription.

If this property is omitted, the default is the ReplyToQMgr name in the message descriptor (MQMD). If the resulting name is blank, it defaults to the name of the queue manager.

QName (*MQPSC_Q_NAME*)
The value is the name of the subscriber queue. It must match the QName used in the original subscription.

If this property is omitted, the default is the ReplyToQ name in the message descriptor (MQMD), which must not be blank.

SubName (*MQPSC_SUBSCRIPTION_NAME*)
If you specify SubName on a **Deregister Subscriber** command the SubName value takes precedence over all other identifier fields except the userid, unless VariableUserId is set on the subscription itself. If VariableUserId is not set, the **Deregister Subscriber** command succeeds only if the userid of the command message matches that of the subscription, if not the command fails with return code *MQRCCF_DUPLICATE_IDENTITY*.

If a subscription exists that matches the traditional identity of this command but has no SubName the **Deregister Subscriber** command fails with return code *MQRCCF_SUB_NAME_ERROR*. If an attempt is made to deregister a subscription that has a SubName using a command message that matches the traditional identity but with no SubName specified the command succeeds.

SubUserData (*MQPSC_SUBSCRIPTION_USER_DATA*)
This is a variable-length text string. The value is stored by the queue manager with the subscription but has no influence on the delivery of the publication to the subscriber. The value can be altered by re-registering to the same subscription with a new value. This attribute is for the use of the application.

SubUserData is returned in the Metatopic information (MQCACF_REG_SUB_USER_DATA) for a subscription, if SubUserData is present.

Example:
Here is an example of NameValueData for a **Deregister Subscriber** command message. In this example, the sample application is deregistering its subscription to the topics which contain the latest score for all matches. The subscriber's identity, including the CorrelId, is taken from the defaults in the MQMD.

```
<psc>
  <Command>DeregSub</Command>
  <RegOpt>CorrelAsId</RegOpt>
  <Topic>Sport/Soccer/State/LatestScore/#</Topic>
</psc>
```

Publish message:

The **Publish** command message is put to a queue, or from a queue manager to a subscriber, to publish information on a specified topic or topics.

Authority to put a message onto a queue and authority to publish information on a specified topic or topics is necessary.

If the user has authority to publish information on some, but not all, topics, only those topics are used to publish; a warning response indicates which topics are not used to publish.

If a subscriber has any matching subscriptions, the queue manager forwards the **Publish** message to the subscriber queues defined in the corresponding **Register Subscriber** command messages.

See Queue Manager Response message for details of the message descriptor (MQMD) parameters needed when sending a command message to the queue manager, and used when a queue manager forwards a publication to a subscriber.

The queue manager forwards the **Publish** message to other queue managers in the network that have matching subscriptions, unless it is a local publication.

Publication data, if any, is included in the body of the message. The data can be described in an <mcd> folder in the NameValueData field of the MQRFH2 header.

Properties

Command (*MQPSC_COMMAND*)
The value is Publish (*MQPSC_PUBLISH*).

This property must be specified.

Topic (*MQPSC_TOPIC*)
The value is a string that contains a topic that categorizes this publication. No wildcard characters are allowed.

You must add the topic to the namelist SYSTEM.QPUBSUB.QUEUE.NAMELIST, see Adding a stream for instructions on how to complete this task.

This property must be specified, and can optionally be repeated for as many topics as needed.

SubPoint (*MQPSC_SUBSCRIPTION_POINT*)
The subscription point on which the publication is published.

In WebSphere Event Broker Version 6.0, the value of the <SubPoint> property is the value of the Subscription Point attribute of the Publication node that is handling the publishing.

In IBM WebSphere MQ Version 7.0.1, the value of the <SubPoint> property must match the name of a subscription point. See Adding a subscription point.

PubOpt (*MQPSC_PUBLICATION_OPTION*)
The publication options property can take the following values:

RetainPub
(*MQPSC_RETAIN_PUB*)

The queue manager is to retain a copy of the publication. If this option is not set, the publication is deleted as soon as the queue manager has sent the publication to all its current subscribers.

IsRetainedPub
(*MQPSC_IS_RETAINED_PUB*)

(Can only be set by a queue manager.) This publication has been retained by the queue manager. The queue manager sets this option to notify a subscriber that this publication was published earlier and has been retained, provided that the subscription has been registered with the InformIfRetained option. It is set only in response to a Register Subscriber or Request Update command message. Retained publications that are sent directly to subscribers do not have this option set.

Local
(*MQPSC_LOCAL*)

This option tells the queue manager that this publication must not be sent to other queue managers. All subscribers that registered at this queue manager receive this publication if they have matching subscriptions.

OtherSubsOnly
(*MQPSC_OTHER_SUBS_ONLY*)

This option allows simpler processing of conference-type applications, where a publisher is also a subscriber to the same topic. It tells the queue manager not to send the publication to the publisher's subscriber queue even if it has a matching subscription. The publisher's subscriber queue consists of its `QMgrName`, `QName`, and optional `CorrelId`, as described in the following list.

CorrelAsId
(*MQPSC_CORREL_ID_AS_IDENTITY*)

The `CorrelId` in the MQMD (which must not be zero) is part of the publisher's subscriber queue, in applications where the publisher is also a subscriber.

None
(*MQPSC_NONE*)

All options take their default values. This has the same effect as omitting the publication options property. If other options are specified at the same time, `None` is ignored.

You can have more than one publication option by introducing additional `<PubOpt>` elements.

The default, if this property is omitted, is that no publication options are set.

PubTime (*MQPSC_PUBLISH_TIMESTAMP*)
The value is an optional publication timestamp set by the publisher. It is 16 characters long with format:

```
YYYYMMDDHHMMSSTH
```

using Universal Time. This information is not checked by the queue manager before being sent to the subscribers.

SeqNum (*MQPSC_SEQUENCE_NUMBER*)
The value is an optional sequence number set by the publisher.

It must be incremented by 1 with each publication. However, this is not checked by the queue manager, which merely transmits this information to subscribers.

If publications on the same topic are published to different interconnected queue managers, it is the responsibility of the publishers to ensure that sequence numbers, if used, are meaningful.

QMgrName (*MQPSC_Q_MGR_NAME*)
The value is a string containing the name of the queue manager for the publisher's subscriber queue, in applications where the publisher is also a subscriber (see `OtherSubsOnly`).

If this property is omitted, the default is the `ReplyToQMgr` name in the message descriptor (MQMD). If the resulting name is blank, it defaults to the name of the queue manager.

QName (*MQPSC_Q_NAME*)
The value is a string containing the name of the publisher's subscriber queue, in applications where the publisher is also a subscriber (see `OtherSubsOnly`).

If this property is omitted, the default is the `ReplyToQ` name in the message descriptor (MQMD), which must not be blank if `OtherSubsOnly` is set.

Example

Here are some examples of *NameValueData* for a **Publish** command message.

The first example is for a publication sent by the match simulator in the sample application to indicate that a match has started.

```
<psc>
  <Command>Publish</Command>
  <Topic>Sport/Soccer/Event/MatchStarted</Topic>
</psc>
```

The second example is for a retained publication. The latest score in the match between Team1 and Team2 is published.

```
<psc>
  <Command>Publish</Command>
  <PubOpt>RetainPub</PubOpt>
  <Topic>Sport/Soccer/State/LatestScore/Team1 Team2</Topic>
</psc>
```

Register Subscriber message:

The **Register Subscriber** command message is sent to a queue manager by a subscriber, or by another application on behalf of a subscriber, to indicate that it wants to subscribe to one or more topics at a subscription point. A message content filter can also be specified.

In publish/subscribe filter expressions, nesting parentheses causes performance to decrease exponentially. Avoid nesting parentheses to a depth greater than about 6.

The message is sent to SYSTEM.BROKER.CONTROL.QUEUE, which is the queue manager's control queue. Authority to put a message to this queue is required, in addition to access authority (set by the queue manager's system administrator) for the topic, or topics, in the subscription.

If the user has authority on some, but not all, topics, only those with authority are registered; a warning response indicates those that are not registered.

See "MQMD settings in command messages to the queue manager" on page 2899 for details of the message descriptor (MQMD) parameters that are needed when sending a command message to the queue manager.

If the reply to queue is a temporary dynamic queue, the subscription is deregistered automatically by the queue manager when the queue is closed.

Properties

Command (*MQPSC_COMMAND*)
The value is RegSub (*MQPSC_REGISTER_SUBSCRIBER*). This property must be specified.

Topic (*MQPSC_TOPIC*)
The topic for which the subscriber wants to receive publications. Wildcard characters can be specified as part of the topic.

If you use the MQSC command **display sub** to examine the subscription created in this way, the value of the <Topic> tag is shown as the TOPICSTR property of the subscription.

This property is required, and can optionally be repeated for as many topics as needed.

SubPoint (*MQPSC_SUBSCRIPTION_POINT*)
The value is the subscription point to which the subscription is attached.

If this property is omitted, the default subscription point is used.

In WebSphere Event Broker Version 6.0, the value of the <SubPoint> property must match the value of the Subscription Point attribute of the Publication nodes that are subscribed to.

In IBM WebSphere MQ Version 7.0.1, the value of the <SubPoint> property must match the name of a subscription point. See Adding a subscription point.

Filter (*MQPSC_FILTER*)
The value is an SQL expression that is used as a filter on the contents of publication messages. If a publication on the specified topic matches the filter, it is sent to the subscriber. This property corresponds to the Selection String that is used in MQSUB and MQOPEN calls. For more information, see Selecting on the content of a message

If this property is omitted, no content filtering takes place.

RegOpt (*MQPSC_REGISTRATION_OPTION*)
This Registration Options property can take the following values:

AddName
(*MQPSC_ADD_NAME*)

When specified for an existing subscription that matches the traditional identity of this Register Subscription command, but with no current SubName value, the SubName specified in this command is added to the subscription.

If AddName is specified the SubName field is mandatory, otherwise MQRCCF_REG_OPTIONS_ERROR is returned.

CorrelAsId
(*MQPSC_CORREL_ID_AS_IDENTITY*)

The CorrelId in the message descriptor (MQMD) is used when sending matching publications to the subscriber queue. The CorrelId must not be zero,

FullResp
(*MQPSC_FULL_RESPONSE*)

When specified all attributes of the subscription are returned in the response message, if the command does not fail.

FullResp is valid only when the command message refers to a single subscription. Therefore, only one topic is permitted in the command; otherwise the command fails with return code *MQRCCF_REG_OPTIONS_ERROR*.

InformIfRet
(*MQPSC_INFORM_IF_RETAINED*)

The queue manager informs the subscriber if a publication is retained when it sends a Publish message in response to a **Register Subscriber** or **Request Update** command message. The queue manager does this by including the IsRetainedPub publication option in the message.

JoinExcl
(*MQPSC_JOIN_EXCLUSIVE*)

This option indicates that the specified SubIdentity should be added as the exclusive member of the identity set for the subscription, and that no other identities can be added to the set.

If the identity has already joined 'shared' and is the sole entry in the set, the set is changed to an exclusive lock held by this identity. Otherwise, if the subscription currently has other identities in the identity set (with shared access) the command fails with return code *MQRCCF_SUBSCRIPTION_IN_USE*.

JoinShared
(*MQPSC_JOIN_SHARED*)

This option indicates that the specified SubIdentity should be added to the identity set for the subscription.

If the subscription is currently locked exclusively (using the `JoinExcl` option), the command fails with return code *MQRCCF_SUBSCRIPTION_LOCKED*, unless the identity that has the subscription locked is the same identity as that in this command message. In this case the lock is automatically modified to a shared lock.

Local
(*MQPSC_LOCAL*)

The subscription is local and is not distributed to other queue managers in the network. Publications made at other queue managers are not delivered to this subscriber, unless it also has a corresponding global subscription.

NewPubsOnly
(*MQPSC_NEW_PUBS_ONLY*)

Retained publications that exist at the time the subscription is registered are not sent to the subscriber; only new publications are sent.

If a subscriber re-registers and changes this option so that it is no longer set, a publication that has already been sent to it might be sent again.

NoAlter
(*MQPSC_NO_ALTER*)

The attributes of an existing matching subscription is not changed.

When a subscription is being created, this option is ignored. All other options specified apply to the new subscription.

If a `SubIdentity` also has one of the join options (`JoinExcl` or `JoinShared`) specified, the identity is added to the identity set regardless of whether `NoAlter` is specified.

None
(*MQPSC_NONE*)

All registration options take their default values.

If the subscriber is already registered, its options are reset to their default values (note that this does not have the same affect as omitting the registration options property), and the subscription expiry is updated from the MQMD of the **Register Subscriber** message.

If other registration options are specified at the same time, `None` is ignored.

NonPers
(*MQPSC_NON_PERSISTENT*)

Publications matching this subscription are delivered to the subscriber as non-persistent messages.

Pers
(*MQPSC_PERSISTENT*)

Publications matching this subscription are delivered to the subscriber as persistent messages.

PersAsPub
(*MQPSC_PERSISTENT_AS_PUBLISH*)

Publications matching this subscription are delivered to the subscriber with the persistence specified by the publisher. This is the default behavior.

PersAsQueue
(*MQPSC_PERSISTENT_AS_Q*)

Publications matching this subscription are delivered to the subscriber with the persistence specified on the subscriber queue.

PubOnReqOnly
(*MQPSC_PUB_ON_REQUEST_ONLY*)

The queue manager does not send publications to the subscriber, except in response to a **Request Update** command message.

VariableUserId
(*MQPSC_VARIABLE_USER_ID*)

When specified the identity of the subscriber (queue, queue manager and correlid) is not restricted to a single userid. This differs from the existing behavior of the queue manager that associates the userid of the original registration message with the subscriber's identity and from then on prevents any other user using that identity. If a new subscriber tries to use the same identity *MQRCCF_DUPLICATE_SUBSCRIPTION* is returned.

This allows any user to modify or deregister the subscription if the user has suitable authority. There is therefore no need to check that the userid matches that of the original subscriber.

To add this option to an existing subscription the command must come from the same userid as the original subscription itself.

If the subscription of the **Request Update** command has `VariableUserId` set, this must be set at request update time to indicate which subscription is referred to. Otherwise, the userid of the **Request Update** command is used to identify the subscription. This is overridden, along with the other subscriber identifiers, if a subscription name is supplied.

If a **Register Subscriber** command message without this option set refers to an existing subscription which has this option set, the option is removed from this subscription and the userid of the subscription is now fixed. If there already exists a subscriber which has the same identity (queue, queue manager and correlation identifier) but with a different user ID associated to it, the command fails with return code *MQRCCF_DUPLICATE_IDENTITY* because there can only be one userid associated with a subscriber identity.

If the registration options property is omitted and the subscriber is already registered, its registration options are not changed and the subscription expiry is updated from the MQMD of the **Register Subscriber** message.

If the subscriber is not already registered, a new subscription is created with all registration options taking their default values.

The default values are `PersAsPub` and no other options set.

QMgrName (*MQPSC_Q_MGR_NAME*)
The value is the name of the queue manager for the subscriber queue, to which matching publications are sent by the queue manager.

If this property is omitted, the default is the `ReplyToQMgr` name in the message descriptor (MQMD). If the resulting name is blank, it defaults to the queue manager's `QMgrName`.

QName (*MQPSC_Q_NAME*)
The value is the name of the subscriber queue, to which matching publications are sent by the queue manager.

If this property is omitted, the default is the `ReplyToQ` name in the message descriptor (MQMD), which must not be blank in this case.

If the queue is a temporary dynamic queue, nonpersistent delivery of publications (`NonPers`) must be specified in the `<RegOpt>` property.

If the queue is a temporary dynamic queue, the subscription is deregistered automatically by the queue manager when the queue is closed.

SubName (*MQPSC_SUBSCRIPTION_NAME*)
This is a name given to a particular subscription. You can use it instead of the queue manager, queue and optional correlId to refer to a subscription.

If a subscription already exists with this **SubName** , any other attributes of the subscription (Topic, QMgrName, QName, CorrelId, UserId, RegOpts, UserSubData, and Expiry) are overridden with the attributes, if specified, that are passed in the new `Register Subscriber` command message. However, if **SubName** is used with no QName field specified, and a ReplyToQ is specified in the MQMD header, the subscriber queue is changed to be the ReplyToQ.

If a subscription that matches the traditional identity of this command already exists, but has no **SubName** , the Registration command fails with return code *MQRCCF_DUPLICATE_SUBSCRIPTION*, unless the **AddName** option is specified.

If you try to alter an existing named subscription by using another `Register Subscriber` command that specifies the same **SubName** , and the values of Topic, QMgrName, QName, and CorrelId in the new command match a different existing subscription, with or without a SubName defined, the command fails with return code *MQRCCF_DUPLICATE_SUBSCRIPTION*. This prevents two subscription names referring to the same subscription.

SubIdentity (*MQPSC_SUBSCRIPTION_IDENTITY*)
This string is used to represent an application with an interest in a subscription. It is a variable-length character string with a maximum length of 64 characters, and is optional. The queue manager maintains a set of subscriber identities for each subscription. Each subscription can allow its identity set to contain only one identity, or an unlimited number of identities (see the **JoinShared** and **JoinExcl** options).

A subscribe command that specifies the **JoinShared** or **JoinExcl** option adds the **SubIdentity** to the subscription's identity set, if it is not already there and if the existing set of identities allows such an action; that is, no other subscriber has joined exclusively or the identity set is empty.

Any alteration of the subscription's attributes as the result of a `Register Subscriber` command in which a **SubIdentity** is specified, only succeeds if it would be the only member of the set of identities for this subscription. Otherwise the command fails with return code *MQRCCF_SUBSCRIPTION_IN_USE*. This prevents a subscription's attributes from changing without other interested subscribers being aware.

If you specify a character string that is longer than 64 characters, the command fails with return code *MQRCCF_SUB_IDENTITY_ERROR*.

SubUserData (*MQPSC_SUBSCRIPTION_USER_DATA*)
This is a variable-length text string. The value is stored by the queue manager with the subscription, but has no influence on publication delivery to the subscriber. The value can be altered by re-registering to the same subscription with a new value. This attribute is there for the use of the application.

The **SubUserData** is returned in the Metatopic information (*MQCACF_REG_SUB_USER_DATA*) for a subscription if present.

If you specify more than one of the registration option values `NonPers`, `PersAsPub`, `PersAsQueue`, and `Pers`, then only the last one is used. You cannot combine these options in an individual subscription.

Example

Here is an example of `NameValueData` for a **Register Subscriber** command message. In the sample application, the results service uses this message to register a subscription to the topics containing the latest scores in all matches, with the 'Persistent as publish' option set. The subscriber's identity, including the `CorrelId`, is taken from the defaults in the MQMD.

```
<psc>
  <Command>RegSub</Command>
  <RegOpt>PersAsPub</RegOpt>
  <RegOpt>CorrelAsId</RegOpt>
  <Topic>Sport/Soccer/State/LatestScore/#</Topic>
</psc>
```

Request Update message:

The **Request Update** command message is sent from a subscriber to a queue manager, to request the current retained publications for the specified topic and subscription point that match the given (optional) filter.

This message is sent to *SYSTEM.BROKER.CONTROL.QUEUE*, the queue manager's control queue. Authority to put a message to this queue is required, in addition to access authority for the topic in the request update; this is set by the queue manager's system administrator.

This command is normally used if the subscriber specified the option `PubOnReqOnly` when it registered. If the queue manager has any matching retained publications, they are sent to the subscriber. If the queue manager has no matching retained publications, the request fails with return code *MQRCCF_NO_RETAINED_MSG*. The requester must have previously registered a subscription with the same Topic, SubPoint, and Filter values.

Properties:

Command (*MQPSC_COMMAND*)
The value is `ReqUpdate` (*MQPSC_REQUEST_UPDATE*). This property must be specified.

Topic (*MQPSC_TOPIC*)
The value is the topic that the subscriber is requesting; wildcard characters are allowed.

This property must be specified, but only one occurrence is allowed in this message.

SubPoint (*MQPSC_SUBSCRIPTION_POINT*)
The value is the subscription point to which the subscription is attached.

If this property is omitted, the default subscription point is used.

Filter (*MQPSC_FILTER*)
The value is an ESQL expression that is used as a filter on the contents of publication messages. If a publication on the specified topic matches the filter, it is sent to the subscriber.

The `<Filter>` property should have the same value as that specified on the original subscription for which you are now requesting an update.

If this property is omitted, no content filtering takes place.

RegOpt (*MQPSC_REGISTRATION_OPTION*)
The registration options property can take the following value:

CorrelAsId
(*MQPSC_CORREL_ID_AS_IDENTITY*)

The `CorrelId` in the message descriptor (MQMD), which must not be zero, is used when sending matching publications to the subscriber queue.

None
(*MQPSC_NONE*)

All options take their default values. This has the same effect as omitting the `<RegOpt>` property. If other options are specified at the same time, `None` is ignored.

VariableUserId
(*MQPSC_VARIABLE_USER_ID*)

When specified the identity of the subscriber (queue, queue manager, and correlid) is not restricted to a single userid. This differs from the existing behavior of the queue manager that associates the userid of the original registration message with the subscriber's identity and from then on prevents any other user using that identity. If a new subscriber tries to use the same identity, the command fails with return code *MQRCCF_DUPLICATE_SUBSCRIPTION*.

This allows any user to modify or deregister the subscription when they have suitable authority. Therefore, there is no need to check that the userid matches that of the original subscriber.

To add this option to an existing subscription, the command must come from the same userid as the original subscription.

If the subscription of the **Request Update** command has `VariableUserId` set, this must be set at request update time to indicate which subscription is referred to. Otherwise, the userid of the **Request Update** command is used to identify the subscription. This is overridden, along with the other subscriber identifiers, if a subscription name is supplied.

The default, if this property is omitted, is that no registration options are set.

QMgrName (*MQPSC_Q_MGR_NAME*)
The value is the name of the queue manager for the subscriber queue, to which the matching retained publication is sent by the queue manager.

If this property is omitted, the default is the `ReplyToQMgr` name in the message descriptor (MQMD). If the resulting name is blank, it defaults to the queue manager's `QMgrName`.

QName (*MQPSC_Q_NAME*)
The value is the name of the subscriber queue, to which the matching retained publication is sent by the queue manager.

If this property is omitted, the default is the `ReplyToQ` name in the message descriptor (MQMD), which must not be blank in this case.

SubName (*MQPSC_SUBSCRIPTION_NAME*)
This is a name given to a particular subscription. If specified on a **Request Update** command the `SubName` value takes precedence over all other identifier fields except the userid, unless `VariableUserId` is set on the subscription itself. If `VariableUserId` is not set, the *Request Update* command succeeds only if the userid of the command message matches that of the subscription. If the userid of the command message does not match that of the subscription, the command fails with return code *MQRCCF_DUPLICATE_IDENTITY*.

If `VariableUserId` is set, and the userid differs from that of the subscription, the command succeeds if the userid of the new command message has authority to browse the stream queue and put to the subscriber queue of the subscription. Otherwise, the command fails with return code *MQRCCF_NOT_AUTHORIZED*.

If a subscription exists that matches the traditional identity of this command, but has no `SubName`, the **Request Update** command fails with return code *MQRCCF_SUB_NAME_ERROR*.

If an attempt is made to request an update for a subscription that has a `SubName` using a command message that matches the traditional identity, but with no `SubName` specified, the command succeeds.

Example:
Here is an example of `NameValueData` for a **Request Update** command message. In the sample application, the results service uses this message to request retained publications containing the latest scores for all teams. The subscriber's identity, including the `CorrelId`, is taken from the defaults in the MQMD.

```
<psc>
  <Command>ReqUpdate</Command>
  <RegOpt>CorrelAsId</RegOpt>
  <Topic>Sport/Soccer/State/LatestScore/#</Topic>
</psc>
```

Queue Manager Response message:

A **Queue Manager Response** message is sent from a queue manager to the ReplyToQ of a publisher or a subscriber, to indicate the success or failure of a command message received by the queue manager if the command message descriptor specified that a response is required.

The response message is contained within the NameValueData field of the MQRFH2 header, in a <pscr> folder.

In the case of a warning or error, the response message contains the <psc> folder from the command message as well as the <pscr> folder. The message data, if any, is not contained in the queue manager response message. In the case of an error, none of the message that caused an error has been processed; in the case of a warning, some of the message might have been processed successfully.

If there is a failure sending a response:

- For publication messages, the queue manager tries to send the response to the IBM MQ dead-letter queue if the MQPUT fails. This allows the publication to be sent to subscribers even if the response cannot be sent back to the publisher.
- For other messages, or if the publication response cannot be sent to the dead-letter queue, an error is logged and the command message is normally rolled back. Whether this happens depends on how the MQInput node has been configured.

Properties:

Completion (*MQPSCR_COMPLETION*)
The completion code, which can take one of three values:
ok Command completed successfully
warning
Command completed but with warning
error Command failed

Response (*MQPSCR_RESPONSE*)
The response to a command message, if that command produced a completion code of warning or error. It contains a <Reason> property, and might contain other properties that indicate the cause of the warning or error.

In the case of one or more errors, there is only one response folder, indicating the cause of the first error only. In the case of one or more warnings, there is a response folder for each warning.

Reason (*MQPSCR_REASON*)
The reason code qualifying the completion code, if the completion code is a warning or error. It is set to one of the error codes listed in the following example. The <Reason> property is contained within a <Response> folder. The reason code can be followed by any valid property from the <psc> folder (for example, a topic name), indicating the cause of the error or warning. If you get a reason code of ????, check the data for correctness, for example, matching angled brackets (< >).

Examples:
Here are some examples of NameValueData in a **Queue Manager Response** message. A successful response might be the following:

```
<pscr>
  <Completion>ok</Completion>
</pscr>
```

Here is an example of a failure response; the failure is a filter error. The first NameValueData string contains the response; the second contains the original command.

```
<pscr>
  <Completion>error</Completion>
  <Response>
```

```
    <Reason>3150</Reason>
  </Reponse>
</pscr>

<psc>
 ...
 command message (to which
 the queue manager is responding)
 ...
</psc>
```

Here is an example of a warning response (due to unauthorized topics). The first `NameValueData` string contains the response; the second `NameValueData` string contains the original command.

```
<pscr>
  <Completion>warning</Completion>
  <Response>
    <Reason>3081</Reason>
    <Topic>topic1</Topic>
  </Reponse>
  <Response>
    <Reason>3081</Reason>
    <Topic>topic2</Topic>
  </Reponse>
</pscr>

<psc>
 ...
 command message (to which
 the queue manager is responding)
 ...
</psc>
```

Publish/subscribe reason codes:

These reason codes might be returned in the `Reason` field of a publish/subscribe response `<pscr>` folder. Constants that can be used to represent these codes in the C or C++ programming languages are also listed.

The MQRC_ constants require the IBM MQ `cmqc.h` header file. The MQRCCF_ constants require the IBM MQ `cmqcfc.h` header file (apart from *MQRCCF_FILTER_ERROR* and *MQRCCF_WRONG_USER*, which require the `cmqpsc.h` header file).

Reason code and text	Explanation	Issued by
2336 MQRC_RFH_COMMAND_ERROR	Valid values for the `<Command>` field of a `<psc>` folder are: RegSub, DeregSub, Publish, DeletePub, and ReqUpdate. Any other values result in this error code being issued.	Any command
2337 MQRC_RFH_ PARM_ERROR	The `<psc>` and `<mcd>` folders both have a set of valid parameters that can be specified within them. Check the descriptions of these folders and ensure that you have not specified incorrect parameters.	Any command
2338 MQRC_RFH_DUPLICATE_PARM	Some parameters (for example, Topic) within a `<psc>` folder can be repeated, but others (for example, Command) cannot be repeated. Check that you have not duplicated a non-repeatable parameter.	Any command

Reason code and text	Explanation	Issued by
2339 MQRC_RFH_PARM_MISSING	Some parameters within <psc> or <mcd> folders are optional and can be omitted; some are mandatory and must not be omitted. Check that you have included all mandatory parameters within your <psc> and <mcd> folders.	Any command
2551 MQRC_SELECTION_NOT_AVAILABLE	No extended message selection provider was available to determine which subscribers with a filter specified should receive the publication.	Publish, Register Subscriber, and Request Update
	No extended message selection provider was available to handle the filter of the specified subscriber.	Register Subscriber and Request Update
2554 MQRC_CONTENT_ERROR	An extended message selection provider found an error in the current or retained publication.	Publish and Request Update
3008 MQRCCF_COMMAND_FAILED	An internal error occurred which prevented the command from executing correctly. The error might occur if the command is reissued. The system event log for the queue manager contains information which should be used when reporting the problem to IBM.	Any command
3072 MQRCCF_TOPIC_ERROR	One or more of the values you supplied for the Topic parameter are incorrect. Check that your values for Topic conform to the specified restrictions.	Any command
3073 MQRCCF_NOT_REGISTERED	The combination of SubPoint, Topic, and Filter that you specified on your DeregSub or ReqUpdate command was either not a combination with which you had previously registered or, for the DeregSub command if the DeregAll option was specified, one of the SubPoint, Topic, or Filter properties was not used to deregister any subscription.	Deregister Subscriber and Request Update commands
3074 MQRCCF_Q_MGR_NAME_ERROR	The specified queue manager was not valid, or the queue manager was not available or did not exist.	Deregister Subscriber, Publish, Register Subscriber, and Request Update commands
3076 MQRCCF_Q_NAME_ERROR	The specified queue name was not valid, or the queue did not exist on the specified queue manager.	Deregister Subscriber, Publish, Register Subscriber, and Request Update commands
3077 MQRCCF_NO_RETAINED_MSG	There were no retained messages for the topic you specified. This might or might not be an error, depending on the design of your application program.	Request Update command

Reason code and text	Explanation	Issued by
3079 MQRCCF_INCORRECT_Q	RegSub, DeregSub, and ReqUpdate commands are always sent to the SYSTEM.BROKER.CONTROL.QUEUE queue of the queue manager for which they are intended. Publish and Delete Publication commands are sent to the input queue for the particular publish/subscribe message flow for which they are intended; this is determined when the message flow is designed. This error code is returned if a command is sent to the wrong queue.	Any command
3080 MQRCCF_CORREL_ID_ERROR	You have specified CorrelAsId as one of your RegOpt parameters. However, the CorrelId field of the MQMD does not contain a valid correlation identifier (that is, it is set to MQCI_NONE).	Deregister Subscriber and Register Subscriber commands
3081 MQRCCF_NOT_AUTHORIZED	You are not authorized to perform the requested action. Authorization settings for the queue manager are handled by the system administrator using the Topics Hierarchy editor.	Publish and Register Subscriber commands
3083 MQRCCF_REG_OPTIONS_ERROR	You have specified an unrecognized RegOpt parameter in the <psc> folder that contains your RegSub or DeregSub command.	Deregister Subscriber and Register Subscriber commands
3084 MQRCCF_PUB_OPTIONS_ERROR	You have specified an unrecognized PubOpt parameter in the <psc> folder that contains your Publish command.	Publish command
3087 MQRCCF_DEL_OPTIONS_ERROR	You have specified an unrecognized DelOpt parameter in the <psc> folder that contains your DeletePub command.	Delete Publication command
3150 MQRCCF_FILTER_ERROR	The value specified for the Filter parameter is not valid. Check the section that describes the valid syntax for filter expressions and ensure that your expression conforms.	Deregister Subscriber, Register Subscriber, and Request Update commands
3151 MQRCCF_WRONG_USER	A subscription that matches the one specified already exists; however, it was registered by a different user. A subscription can only be changed or deregistered by the user who originally registered it.	Deregister Subscriber, Register Subscriber, and Request Update commands
3152 MQRCCF_DUPLICATE_SUBSCRIPTION	A matching subscription already exists with a different subscription name.	
3153 MQRCCF_SUB_NAME_ERROR	Either the format of the subscription name is not valid, or a matching subscription already exists with no subscription name.	

Reason code and text	Explanation	Issued by
3154 MQRCCF_SUB_IDENTITY_ERROR	The subscription identity parameter is in error. Either the supplied value exceeds the maximum length allowed, or the subscription identity is not currently a member of the subscription's identity set and a Join registration option was not specified.	
3155 MQRCCF_SUBSCRIPTION_IN_USE	An attempt to modify or deregister a subscription was attempted by a member of the identity set when it was not the only member of this set.	
3156 MQRCCF_SUBSCRIPTION_LOCKED	The subscription is currently exclusively locked by another identity.	
3157 MQRCCF_ALREADY_JOINED	A Join registration option was specified but the subscriber identity was already a member of the subscription's identity set.	

MQMD settings in command messages to the queue manager:

Applications that send command messages to the queue manager use the following settings of fields in the message descriptor (MQMD). Fields that are left as the default value, or can be set to any valid value in the usual way, are not listed here.

Report
See `MsgType` and `CorrelId`.

MsgType
`MsgType` should be set to either *MQMT_REQUEST* or *MQMT_DATAGRAM*. *MQRC_MSG_TYPE_ERROR* will be returned if `MsgType` is not set to one of these values.

`MsgType` should be set to *MQMT_REQUEST* for a command message if a response is always required. The MQRO_PAN and MQRO_NAN flags in the `Report` field are not significant in this case.

If `MsgType` is set to *MQMT_DATAGRAM*, responses depend on the setting of the MQRO_PAN and MQRO_NAN flags in the `Report` field:

- MQRO_PAN alone means that the queue manager sends a response only if the command succeeds.
- MQRO_NAN alone means that the queue manager sends a response only if the command fails.
- If a command completes with a warning, a response is sent if either MQRO_PAN or MQRO_NAN is set.
- MQRO_PAN + MQRO_NAN means that the queue manager sends a response whether the command succeeds or fails. This has the same effect from the queue manager's perspective as setting `MsgType` to MQMT_REQUEST.
- If neither MQRO_PAN nor MQRO_NAN is set, no response is ever sent.

Format
Set to MQFMT_RF_HEADER_2

MsgId
This field is normally set to MQMI_NONE, so that the queue manager generates a unique value.

CorrelId
This field can be set to any value. If the sender's identity includes a `CorrelId`, specify this value, together with MQRO_PASS_CORREL_ID in the `Report` field, to ensure that it is set in all response messages sent by the queue manager to the sender.

ReplyToQ
This field defines the queue to which responses, if any, are to be sent. This might be the sender's queue; this has the advantage that the QName parameter can be omitted from the message. If, however, responses are to be sent to a different queue, the QName parameter is needed.

ReplyToQMgr
This field defines the queue manager for responses. If you leave this field blank (the default value), the local queue manager puts its own name in this field.

MQMD settings for publications forwarded by a queue manager:

A queue manager uses these settings of fields in the message descriptor (MQMD) when it sends a publication to a subscriber. All other fields in the MQMD are set to their default values.

Report
Report is set to MQRO_NONE.

MsgType
MsgType is set to MQMT_DATAGRAM.

Expiry
Expiry is set to the value in the Publish message received from the publisher. In the case of a retained message, the time outstanding is reduced by the approximate time that the message has been at the queue manager.

Format
Format is set to MQFMT_RF_HEADER_2

MsgId
MsgId is set to a unique value.

CorrelId
If CorrelId is part of the subscriber's identity, this is the value specified by the subscriber when registering. Otherwise, it is a non-zero value chosen by the queue manager.

Priority
Priority takes the value set by the publisher, or as resolved if the publisher specified MQPRI_PRIORITY_AS_Q_DEF.

Persistence
Persistence takes the value set by the publisher, or as resolved if the publisher specified MQPER_PERSISTENCE_AS_Q_DEF, unless specified otherwise in the Register Subscriber message for the subscriber to which this publication is being sent.

ReplyToQ
ReplyToQ is set to blanks.

ReplyToQMgr
ReplyToQMgr is set to the name of the queue manager.

UserIdentifier
UserIdentifier is the subscriber's user identifier, as set when the subscriber registered.

AccountingToken
AccountingToken is the subscriber's accounting token, as set when the subscriber first registered.

ApplIdentityData
ApplIdentityData is the subscriber's application identity data, as set when the subscriber first registered.

PutApplType
PutApplType is set to MQAT_BROKER.

PutApplName
PutApplName is set to the first 28 characters of the name of the queue manager.

PutDate
PutDate is the date when the message was put.

PutTime
PutTime is the time when the message was put.

ApplOriginData
ApplOriginData is set to blanks.

MQMD settings in queue manager response messages:

A queue manager uses these settings of fields in the message descriptor (MQMD) when sending a reply to a publication message. All other fields in the MQMD are set to their default values.

Report
Report is set to all zeros.

MsgType
MsgType is set to MQMT_REPLY.

Format
Format is set to MQFMT_RF_HEADER_2

MsgId
The setting of MsgId depends on the Report options in the original command message. By default, it is set to MQMI_NONE, so that the queue manager generates a unique value.

CorrelId
The setting of CorrelId depends on the Report options in the original command message. By default, this means that the CorrelId is set to the same value as the MsgId of the command message. This can be used to correlate commands with their responses.

Priority
Priority is set to the same value as in the original command message.

Persistence
Persistence is set to the value set in the original command message.

Expiry
Expiry is set to the same value as in the original command message received by the queue manager.

PutApplType
PutApplType is set to MQAT_BROKER.

PutApplName
PutApplName is set to the first 28 characters of name of the queue manager.

Other context fields are set as if generated with MQPMO_PASS_IDENTITY_CONTEXT.

Machine encodings

This section describes the structure of the *Encoding* field in the message descriptor.

See "MQMD - Message descriptor" on page 2387 for a summary of the fields in the structure.

The *Encoding* field is a 32-bit integer that is divided into four separate subfields; these subfields identify:

- The encoding used for binary integers
- The encoding used for packed-decimal integers
- The encoding used for floating-point numbers
- Reserved bits

Each subfield is identified by a bit mask that has 1-bits in the positions corresponding to the subfield, and 0-bits elsewhere. The bits are numbered such that bit 0 is the most significant bit, and bit 31 the least significant bit. The following masks are defined:

MQENC_INTEGER_MASK
Mask for binary-integer encoding.

This subfield occupies bit positions 28 through 31 within the *Encoding* field.

MQENC_DECIMAL_MASK
Mask for packed-decimal-integer encoding.

This subfield occupies bit positions 24 through 27 within the *Encoding* field.

MQENC_FLOAT_MASK
Mask for floating-point encoding.

This subfield occupies bit positions 20 through 23 within the *Encoding* field.

MQENC_RESERVED_MASK
Mask for reserved bits.

This subfield occupies bit positions 0 through 19 within the *Encoding* field.

Binary-integer encoding:

The following values are valid for the binary-integer encoding:

MQENC_INTEGER_UNDEFINED
Binary integers are represented using an encoding that is undefined.

MQENC_INTEGER_NORMAL
Binary integers are represented in the conventional way:

- The least significant byte in the number has the highest address of any of the bytes in the number; the most significant byte has the lowest address
- The least significant bit in each byte is adjacent to the byte with the next higher address; the most significant bit in each byte is adjacent to the byte with the next lower address

MQENC_INTEGER_REVERSED
Binary integers are represented in the same way as MQENC_INTEGER_NORMAL, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as MQENC_INTEGER_NORMAL.

Packed-decimal-integer encoding:

The following values are valid for the packed-decimal-integer encoding:

MQENC_DECIMAL_UNDEFINED
Packed-decimal integers are represented using an encoding that is undefined.

MQENC_DECIMAL_NORMAL
Packed-decimal integers are represented in the conventional way:

- Each decimal digit in the printable form of the number is represented in packed decimal by a single hexadecimal digit in the range X'0' through X'9'. Each hexadecimal digit occupies four bits, and so each byte in the packed decimal number represents two decimal digits in the printable form of the number.
- The least significant byte in the packed-decimal number is the byte that contains the least significant decimal digit. Within that byte, the most significant four bits contain the least significant decimal digit, and the least significant four bits contain the sign. The sign is either X'C' (positive), X'D' (negative), or X'F' (unsigned).
- The least significant byte in the number has the highest address of any of the bytes in the number; the most significant byte has the lowest address.
- The least significant bit in each byte is adjacent to the byte with the next higher address; the most significant bit in each byte is adjacent to the byte with the next lower address.

MQENC_DECIMAL_REVERSED
Packed-decimal integers are represented in the same way as MQENC_DECIMAL_NORMAL, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as MQENC_DECIMAL_NORMAL.

Floating-point encoding:

The following values are valid for the floating-point encoding:

MQENC_FLOAT_UNDEFINED
Floating-point numbers are represented using an encoding that is undefined.

MQENC_FLOAT_IEEE_NORMAL
Floating-point numbers are represented using the standard IEEE[2] floating-point format, with the bytes arranged as follows:

- The least significant byte in the mantissa has the highest address of any of the bytes in the number; the byte containing the exponent has the lowest address
- The least significant bit in each byte is adjacent to the byte with the next higher address; the most significant bit in each byte is adjacent to the byte with the next lower address

Details of the IEEE float encoding can be found in IEEE Standard 754.

MQENC_FLOAT_IEEE_REVERSED
Floating-point numbers are represented in the same way as MQENC_FLOAT_IEEE_NORMAL, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as MQENC_FLOAT_IEEE_NORMAL.

MQENC_FLOAT_S390
Floating-point numbers are represented using the standard System/390 floating-point format; this is also used by System/370.

2. The Institute of Electrical and Electronics Engineers

Constructing encodings:

To construct a value for the *Encoding* field in MQMD, the relevant constants that describe the required encodings can be added together (do not add the same constant more than once), or combined using the bitwise OR operation (if the programming language supports bit operations).

Whichever method is used, combine only one of the MQENC_INTEGER_* encodings with one of the MQENC_DECIMAL_* encodings and one of the MQENC_FLOAT_* encodings.

Analyzing encodings:

The *Encoding* field contains subfields; because of this, applications that need to examine the integer, packed decimal, or float encoding must use one of the techniques described.

Using bit operations

If the programming language supports bit operations, perform the following steps:

1. Select one of the following values, according to the type of encoding required:
   - MQENC_INTEGER_MASK for the binary integer encoding
   - MQENC_DECIMAL_MASK for the packed decimal integer encoding
   - MQENC_FLOAT_MASK for the floating point encoding

   Call the value A.
2. Combine the *Encoding* field with A using the bitwise AND operation; call the result B.
3. B is the encoding required, and can be tested for equality with each of the values that is valid for that type of encoding.

Using arithmetic

If the programming language *does not* support bit operations, perform the following steps using integer arithmetic:

1. Select one of the following values, according to the type of encoding required:
   - 1 for the binary integer encoding
   - 16 for the packed decimal integer encoding
   - 256 for the floating point encoding

   Call the value A.
2. Divide the value of the *Encoding* field by A ; call the result B.
3. Divide B by 16; call the result C.
4. Multiply C by 16 and subtract from B ; call the result D.
5. Multiply D by A ; call the result E.
6. E is the encoding required, and can be tested for equality with each of the values that is valid for that type of encoding.

Summary of machine architecture encodings:

Encodings for machine architectures are shown in Table 278.

Table 278. Summary of encodings for machine architectures

Machine architecture	Binary integer encoding	Packed-decimal integer encoding	Floating-point encoding
IBM i	normal	normal	IEEE normal
Intel x86	reversed	reversed	IEEE reversed
PowerPC®	normal	normal	IEEE normal
System/390	normal	normal	System/390

Report options and message flags

This section describes the *Report* and *MsgFlags* fields that are part of the message descriptor MQMD specified on the MQGET, MQPUT, and MQPUT1 calls.

The topics in this section describe:

- The structure of the report field and how the queue manager processes it
- How an application analyzes the report field
- The structure of the message-flags field

For more information about the MQMD message descriptor, see "MQMD - Message descriptor" on page 2387.

Structure of the report field:

This information describes the structure of the report field.

The *Report* field is a 32-bit integer that is divided into three separate subfields. These subfields identify:

- Report options that are rejected if the local queue manager does not recognize them
- Report options that are always accepted, even if the local queue manager does not recognize them
- Report options that are accepted only if certain other conditions are satisfied

Each subfield is identified by a bit mask that has 1-bits in the positions corresponding to the subfield, and 0-bits elsewhere. The bits in a subfield are not necessarily adjacent. The bits are numbered such that bit 0 is the most significant bit, and bit 31 the least significant bit. The following masks are defined to identify the subfields:

MQRO_REJECT_UNSUP_MASK
This mask identifies the bit positions within the *Report* field where report options that are not supported by the local queue manager cause the MQPUT or MQPUT1 call to fail with completion code MQCC_FAILED and reason code MQRC_REPORT_OPTIONS_ERROR.

This subfield occupies bit positions 3, and 11 through 13.

MQRO_ACCEPT_UNSUP_MASK
This mask identifies the bit positions within the *Report* field where report options that are not supported by the local queue manager are nevertheless accepted on the MQPUT or MQPUT1 calls. Completion code MQCC_WARNING with reason code MQRC_UNKNOWN_REPORT_OPTION are returned in this case.

This subfield occupies bit positions 0 through 2, 4 through 10, and 24 through 31.

The following report options are included in this subfield:

- MQRO_ACTIVITY

- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_DEAD_LETTER_Q
- MQRO_DISCARD_MSG
- MQRO_EXCEPTION
- MQRO_EXCEPTION_WITH_DATA
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION
- MQRO_EXPIRATION_WITH_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_NAN
- MQRO_NEW_MSG_ID
- MQRO_NONE
- MQRO_PAN
- MQRO_PASS_CORREL_ID
- MQRO_PASS_MSG_ID

MQRO_ACCEPT_UNSUP_IF_XMIT_MASK

This mask identifies the bit positions within the *Report* field where report options that are not supported by the local queue manager are nevertheless accepted on the MQPUT or MQPUT1 calls *provided* that both of the following conditions are satisfied:

- The message is destined for a remote queue manager.
- The application is not putting the message directly on a local transmission queue (that is, the queue identified by the *ObjectQMgrName* and *ObjectName* fields in the object descriptor specified on the MQOPEN or MQPUT1 call is not a local transmission queue).

Completion code MQCC_WARNING with reason code MQRC_UNKNOWN_REPORT_OPTION are returned if these conditions are satisfied, and MQCC_FAILED with reason code MQRC_REPORT_OPTIONS_ERROR if not.

This subfield occupies bit positions 14 through 23.

The following report options are included in this subfield:

- MQRO_COA
- MQRO_COA_WITH_DATA
- MQRO_COA_WITH_FULL_DATA
- MQRO_COD
- MQRO_COD_WITH_DATA
- MQRO_COD_WITH_FULL_DATA

If any options are specified in the *Report* field that the queue manager does not recognize, the queue manager checks each subfield in turn by using the bitwise AND operation to combine the *Report* field with the mask for that subfield. If the result of that operation is not zero, the completion code and reason codes described previously are returned.

If MQCC_WARNING is returned, it is not defined which reason code is returned if other warning conditions exist.

The ability to specify and have accepted report options that are not recognized by the local queue manager is useful when sending a message with a report option that is recognized and processed by a *remote* queue manager.

Analyzing the report field:

The *Report* field contains subfields; because of this, applications that need to check whether the sender of the message requested a particular report must use one of the techniques described.

Using bit operations

If the programming language supports bit operations, perform the following steps:

1. Select one of the following values, according to the type of report to be checked:
   - MQRO_COA_WITH_FULL_DATA for COA report
   - MQRO_COD_WITH_FULL_DATA for COD report
   - MQRO_EXCEPTION_WITH_FULL_DATA for exception report
   - MQRO_EXPIRATION_WITH_FULL_DATA for expiration report

   Call the value A.

   On z/OS, use the MQRO_*_WITH_DATA values instead of the MQRO_*_WITH_FULL_DATA values.
2. Combine the *Report* field with A using the bitwise AND operation; call the result B.
3. Test B for equality with each value that is possible for that type of report.

   For example, if A is MQRO_EXCEPTION_WITH_FULL_DATA, test B for equality with each of the following to determine what was specified by the sender of the message:
   - MQRO_NONE
   - MQRO_EXCEPTION
   - MQRO_EXCEPTION_WITH_DATA
   - MQRO_EXCEPTION_WITH_FULL_DATA

   The tests can be performed in whatever order is most convenient for the application logic.

Use a similar method to test for the MQRO_PASS_MSG_ID or MQRO_PASS_CORREL_ID options; select as the value A whichever of these two constants is appropriate, and then proceed as described previously.

Using arithmetic

If the programming language *does not* support bit operations, perform the following steps using integer arithmetic:

1. Select one of the following values, according to the type of report to be checked:
   - MQRO_COA for COA report
   - MQRO_COD for COD report
   - MQRO_EXCEPTION for exception report
   - MQRO_EXPIRATION for expiration report

   Call the value A.
2. Divide the *Report* field by A ; call the result B.
3. Divide B by 8 ; call the result C.
4. Multiply C by 8 and subtract from B ; call the result D.
5. Multiply D by A ; call the result E.
6. Test E for equality with each value that is possible for that type of report.

   For example, if A is MQRO_EXCEPTION, test E for equality with each of the following to determine what was specified by the sender of the message:
   - MQRO_NONE
   - MQRO_EXCEPTION
   - MQRO_EXCEPTION_WITH_DATA

- MQRO_EXCEPTION_WITH_FULL_DATA

The tests can be performed in whatever order is most convenient for the application logic.

The following pseudocode illustrates this technique for exception report messages:

```
A = MQRO_EXCEPTION
B = Report/A
C = B/8
D = B - C*8
E = D*A
```

Use a similar method to test for the MQRO_PASS_MSG_ID or MQRO_PASS_CORREL_ID options; select as the value `A` whichever of these two constants is appropriate, and then proceed as described previously, but replacing the value `8` in the previous steps by the value `2`.

Structure of the message-flags field:

This information describes the structure of the message-flags field.

The *MsgFlags* field is a 32-bit integer that is divided into three separate subfields. These subfields identify:

- Message flags that are rejected if the local queue manager does not recognize them
- Message flags that are always accepted, even if the local queue manager does not recognize them
- Message flags that are accepted only if certain other conditions are satisfied

Note: All subfields in *MsgFlags* are reserved for use by the queue manager.

Each subfield is identified by a bit mask that has 1-bits in the positions corresponding to the subfield, and 0-bits elsewhere. The bits are numbered such that bit 0 is the most significant bit, and bit 31 the least significant bit. The following masks are defined to identify the subfields:

MQMF_REJECT_UNSUP_MASK
This mask identifies the bit positions within the *MsgFlags* field where message flags that are not supported by the local queue manager cause the MQPUT or MQPUT1 call to fail with completion code MQCC_FAILED and reason code MQRC_MSG_FLAGS_ERROR.

This subfield occupies bit positions 20 through 31.

The following message flags are included in this subfield:

- MQMF_LAST_MSG_IN_GROUP
- MQMF_LAST_SEGMENT
- MQMF_MSG_IN_GROUP
- MQMF_SEGMENT
- MQMF_SEGMENTATION_ALLOWED
- MQMF_SEGMENTATION_INHIBITED

MQMF_ACCEPT_UNSUP_MASK
This mask identifies the bit positions within the *MsgFlags* field where message flags that are not supported by the local queue manager are nevertheless accepted on the MQPUT or MQPUT1 calls. The completion code is MQCC_OK.

This subfield occupies bit positions 0 through 11.

MQMF_ACCEPT_UNSUP_IF_XMIT_MASK
This mask identifies the bit positions within the *MsgFlags* field where message flags that are not supported by the local queue manager are nevertheless accepted on the MQPUT or MQPUT1 calls *provided* that both of the following conditions are satisfied:

- The message is destined for a remote queue manager.

- The application is not putting the message directly on a local transmission queue (that is, the queue identified by the *ObjectQMgrName* and *ObjectName* fields in the object descriptor specified on the MQOPEN or MQPUT1 call is not a local transmission queue).

Completion code MQCC_OK is returned if these conditions are satisfied, and MQCC_FAILED with reason code MQRC_MSG_FLAGS_ERROR if not.

This subfield occupies bit positions 12 through 19.

If there are flags specified in the *MsgFlags* field that the queue manager does not recognize, the queue manager checks each subfield in turn by using the bitwise AND operation to combine the *MsgFlags* field with the mask for that subfield. If the result of that operation is not zero, the completion code and reason codes described previously are returned.

Data conversion

This collection of topics describes the interface to the data-conversion exit, and the processing performed by the queue manager when data conversion is required.

For more information about data conversion, see the document *Data Conversion under IBM MQ* at http://www.ibm.com/support/docview.wss?uid=swg27005729.

The data-conversion exit is invoked as part of the processing of the MQGET call in order to convert the application message data to the representation required by the receiving application. Conversion of the application message data is optional; it requires the MQGMO_CONVERT option to be specified on the MQGET call.

The following subjects are described:

- The processing performed by the queue manager in response to the MQGMO_CONVERT option; see "Conversion processing" on page 2910.
- Processing conventions used by the queue manager when processing a built-in format; these conventions are recommended for user-written exits too. See "Processing conventions" on page 2911.
- Special considerations for converting report messages; see "Conversion of report messages" on page 2915.
- The parameters passed to the data-conversion exit; see "MQ_DATA_CONV_EXIT - Data conversion exit" on page 2928.
- A call that can be used from the exit to convert character data between different representations; see "MQXCNVC - Convert characters" on page 2922.
- The data-structure parameter that is specific to the exit; see "MQDXP - Data-conversion exit parameter" on page 2916.

Conversion processing:

This information describes the processing performed by the queue manager in response to the MQGMO_CONVERT option.

The queue manager performs the following actions if the MQGMO_CONVERT option is specified on the MQGET call, and there is a message to be returned to the application:

1. If one or more of the following is true, no conversion is necessary:
   - The message data is already in the character set and encoding required by the application issuing the MQGET call. The application must set the *CodedCharSetId* and *Encoding* fields in the **MsgDesc** parameter of the MQGET call to the values required, before issuing the call.
   - The length of the message data is zero.
   - The length of the **Buffer** parameter of the MQGET call is zero.

   In these cases the message is returned without conversion to the application issuing the MQGET call; the *CodedCharSetId* and *Encoding* values in the **MsgDesc** parameter are set to the values in the control information in the message, and the call completes with one of the following combinations of completion code and reason code:

Completion code	Reason code
MQCC_OK	MQRC_NONE
MQCC_WARNING	MQRC_TRUNCATED_MSG_ACCEPTED
MQCC_WARNING	MQRC_TRUNCATED_MSG_FAILED

   The following steps are performed only if the character set or encoding of the message data differs from the corresponding value in the **MsgDesc** parameter, and there is data to be converted:
2. If the *Format* field in the control information in the message has the value MQFMT_NONE, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_FORMAT_ERROR.

   In all other cases conversion processing continues.
3. The message is removed from the queue and placed in a temporary buffer that is the same size as the **Buffer** parameter. For browse operations, the message is copied into the temporary buffer, instead of being removed from the queue.
4. If the message has to be truncated to fit in the buffer, the following is done:
   - If the MQGMO_ACCEPT_TRUNCATED_MSG option was not specified, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_TRUNCATED_MSG_FAILED.
   - If the MQGMO_ACCEPT_TRUNCATED_MSG option *was* specified, the completion code is set to MQCC_WARNING, the reason code is set to MQRC_TRUNCATED_MSG_ACCEPTED, and conversion processing continues.
5. If the message can be accommodated in the buffer without truncation, or the MQGMO_ACCEPT_TRUNCATED_MSG option was specified, the following is done:
   - If the format is a built-in format, the buffer is passed to the queue manager's data-conversion service.
   - If the format is not a built-in format, the buffer is passed to a user-written exit with the same name as the format. If the exit cannot be found, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_FORMAT_ERROR.

   If no error occurs, the output from the data-conversion service or from the user-written exit is the converted message, plus the completion code and reason code to be returned to the application issuing the MQGET call.
6. If the conversion is successful, the queue manager returns the converted message to the application. In this case, the completion code and reason code returned by the MQGET call are one of the following combinations:

Completion code	Reason code
MQCC_OK	MQRC_NONE
MQCC_WARNING	MQRC_TRUNCATED_MSG_ACCEPTED

However, if the conversion is performed by a user-written exit, other reason codes can be returned, even when the conversion is successful.

If the conversion fails, the queue manager returns the unconverted message to the application, with the *CodedCharSetId* and *Encoding* fields in the **MsgDesc** parameter set to the values in the control information in the message, and with completion code MQCC_WARNING.

Processing conventions:

When converting a built-in format, the queue manager follows the processing conventions described.

User-written exits should also follow these conventions, although this is not enforced by the queue manager. The built-in formats converted by the queue manager are:

- MQFMT_ADMIN
- MQFMT_CICS (z/OS only)
- MQFMT_COMMAND_1
- MQFMT_COMMAND_2
- MQFMT_DEAD_LETTER_HEADER
- MQFMT_DIST_HEADER
- MQFMT_EVENT version 1
- MQFMT_EVENT version 2
- MQFMT_IMS
- MQFMT_IMS_VAR_STRING
- MQFMT_MD_EXTENSION
- MQFMT_PCF
- MQFMT_REF_MSG_HEADER
- MQFMT_RF_HEADER
- MQFMT_RF_HEADER_2
- MQFMT_STRING
- MQFMT_TRIGGER
- MQFMT_WORK_INFO_HEADER (z/OS only)
- MQFMT_XMIT_Q_HEADER

1. If the message expands during conversion, and exceeds the size of the **Buffer** parameter, the following is done:
   - If the MQGMO_ACCEPT_TRUNCATED_MSG option was not specified, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_CONVERTED_MSG_TOO_BIG.
   - If the MQGMO_ACCEPT_TRUNCATED_MSG option *was* specified, the message is truncated, the completion code is set to MQCC_WARNING, the reason code is set to MQRC_TRUNCATED_MSG_ACCEPTED, and conversion processing continues.
2. If truncation occurs (either before or during conversion), the number of valid bytes returned in the **Buffer** parameter can be *less than* the length of the buffer.

   This can occur, for example, if a 4-byte integer or a DBCS character straddles the end of the buffer. The incomplete element of information is not converted, and those bytes in the returned message do not contain valid information. This can also occur if a message that was truncated before conversion shrinks during conversion.

If the number of valid bytes returned is less than the length of the buffer, the unused bytes at the end of the buffer are set to nulls.

3. If an array or string straddles the end of the buffer, as much of the data as possible is converted; only the particular array element or DBCS character which is incomplete is not converted; preceding array elements or characters are converted.
4. If truncation occurs (either before or during conversion), the length returned for the **DataLength** parameter is the length of the unconverted message before truncation.
5. When strings are converted between single-byte character sets (SBCS), double-byte character sets (DBCS), or multi-byte character sets (MBCS), the strings can expand or contract.
   - In the PCF formats MQFMT_ADMIN, MQFMT_EVENT, and MQFMT_PCF, the strings in the MQCFST and MQCFSL structures expand or contract as necessary to accommodate the string after conversion.

     For the string-list structure MQCFSL, the strings in the list might expand or contract by different amounts. If this happens, the queue manager pads the shorter strings with blanks to make them the same length as the longest string after conversion.
   - In the format MQFMT_REF_MSG_HEADER, the strings addressed by the *SrcEnvOffset*, *SrcNameOffset*, *DestEnvOffset*, and *DestNameOffset* fields expand or contract as necessary to accommodate the strings after conversion.
   - In the format MQFMT_RF_HEADER, the *NameValueString* field expands or contracts as necessary to accommodate the name-value pairs after conversion.
   - In structures with fixed field sizes, the queue manager allows strings to expand or contract within their fixed fields, provided that no significant information is lost. In this regard, trailing blanks and characters following the first null character in the field are treated as insignificant.
     - If the string expands, but only insignificant characters need to be discarded to accommodate the converted string in the field, the conversion succeeds and the call completes with MQCC_OK and reason code MQRC_NONE (assuming no other errors).
     - If the string expands, but the converted string requires significant characters to be discarded in order to fit in the field, the message is returned unconverted and the call completes with MQCC_WARNING and reason code MQRC_CONVERTED_STRING_TOO_BIG.

       Note: Reason code MQRC_CONVERTED_STRING_TOO_BIG results in this case whether or not the MQGMO_ACCEPT_TRUNCATED_MSG option was specified.
     - If the string contracts, the queue manager pads the string with blanks to the length of the field.
6. For messages consisting of one or more MQ header structures followed by user data, one or more of the header structures might be converted, while the remainder of the message is not. However, (with two exceptions) the *CodedCharSetId* and *Encoding* fields in each header structure always correctly indicate the character set and encoding of the data that follows the header structure.

   The two exceptions are the MQCIH and MQIIH structures, where the values in the *CodedCharSetId* and *Encoding* fields in those structures are not significant. For those structures, the data following the structure is in the same character set and encoding as the MQCIH or MQIIH structure itself.
7. If the *CodedCharSetId* or *Encoding* fields in the control information of the message being retrieved, or in the **MsgDesc** parameter, specify values that are undefined or not supported, the queue manager might ignore the error if the undefined or unsupported value does not need to be used in converting the message.

   For example, if the *Encoding* field in the message specifies an unsupported float encoding, but the message contains only integer data, or contains floating-point data that does not require conversion (because the source and target float encodings are identical), the error might not be diagnosed.

   If the error is diagnosed, the message is returned unconverted, with completion code MQCC_WARNING and one of the MQRC_SOURCE_*_ERROR or MQRC_TARGET_*_ERROR reason codes (as appropriate); the *CodedCharSetId* and *Encoding* fields in the **MsgDesc** parameter are set to the values in the control information in the message.

If the error is not diagnosed and the conversion completes successfully, the values returned in the *CodedCharSetId* and *Encoding* fields in the **MsgDesc** parameter are those specified by the application issuing the MQGET call.

8. In all cases, if the message is returned to the application unconverted the completion code is set to MQCC_WARNING, and the *CodedCharSetId* and *Encoding* fields in the **MsgDesc** parameter are set to the values appropriate to the unconverted data. This is done for MQFMT_NONE also.

   The **Reason** parameter is set to a code that indicates why the conversion could not be carried out, unless the message also had to be truncated; reason codes related to truncation take precedence over reason codes related to conversion. (To determine if a truncated message was converted, check the values returned in the *CodedCharSetId* and *Encoding* fields in the **MsgDesc** parameter.)

   When an error is diagnosed, either a specific reason code is returned, or the general reason code MQRC_NOT_CONVERTED. The reason code returned depends on the diagnostic capabilities of the underlying data-conversion service.

9. If completion code MQCC_WARNING is returned, and more than one reason code is relevant, the order of precedence is as follows:
   a. The following reasons take precedence over all others; only one of the reasons in this group can arise:
      - MQRC_SIGNAL_REQUEST_ACCEPTED
      - MQRC_TRUNCATED_MSG_ACCEPTED
   b. The order of precedence within the remaining reason codes is not defined.

10. On completion of the MQGET call:
    - The following reason code indicates that the message was converted successfully:
      – MQRC_NONE
    - The following reason codes indicate that the message *might* have been converted successfully (check the *CodedCharSetId* and *Encoding* fields in the **MsgDesc** parameter to find out):
      – MQRC_MSG_MARKED_BROWSE_CO_OP
      – MQRC_TRUNCATED_MSG_ACCEPTED
    - All other reason codes indicate that the message was not converted.

    The following processing is specific to the built-in formats; it does not apply to user-defined formats:

11. With the exception of the following formats:
    - MQFMT_ADMIN
    - MQFMT_COMMAND_1
    - MQFMT_COMMAND_2
    - MQFMT_EVENT
    - MQFMT_IMS_VAR_STRING
    - MQFMT_PCF
    - MQFMT_STRING

    none of the built-in formats can be converted from or to character sets that do not have SBCS characters for the characters that are valid in queue names. If an attempt is made to perform such a conversion, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_SOURCE_CCSID_ERROR or MQRC_TARGET_CCSID_ERROR, as appropriate.

    The Unicode character set 

UTF-16 is an example of a character set that does not have SBCS characters for the characters that are valid in queue names.

12. If the message data for a built-in format is truncated, fields within the message that contain lengths of strings, or counts of elements or structures, are not adjusted to reflect the length of the data actually returned to the application; the values returned for such fields within the message data are the values applicable to the message *before truncation*.

When processing messages such as a truncated MQFMT_ADMIN message, ensure that the application does not attempt to access data beyond the end of the data returned.

13. If the format name is MQFMT_DEAD_LETTER_HEADER, the message data begins with an MQDLH structure, possibly followed by zero or more bytes of application message data. The format, character set, and encoding of the application message data are defined by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQDLH structure at the start of the message. Because the MQDLH structure and application message data can have different character sets and encodings, one, other, or both of the MQDLH structure and application message data might require conversion.

    The queue manager converts the MQDLH structure first, as necessary. If conversion is successful, or the MQDLH structure does not require conversion, the queue manager checks the *CodedCharSetId* and *Encoding* fields in the MQDLH structure to see if conversion of the application message data is required. If conversion is required, the queue manager invokes the user-written exit with the name given by the *Format* field in the MQDLH structure, or performs the conversion itself (if *Format* is the name of a built-in format).

    If the MQGET call returns a completion code of MQCC_WARNING, and the reason code is one of those indicating that conversion was not successful, one of the following applies:

    - The MQDLH structure could not be converted. In this case the application message data will not have been converted either.
    - The MQDLH structure was converted, but the application message data was not.

    The application can examine the values returned in the *CodedCharSetId* and *Encoding* fields in the **MsgDesc** parameter, and those in the MQDLH structure, in order to determine which of the previously applies.

14. If the format name is MQFMT_XMIT_Q_HEADER, the message data begins with an MQXQH structure, possibly followed by zero or more bytes of additional data. This additional data is usually the application message data (which may be of zero length), but there can also be one or more further MQ header structures present, at the start of the additional data.

    The MQXQH structure must be in the character set and encoding of the queue manager. The format, character set, and encoding of the data following the MQXQH structure are given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQMD structure contained *within* the MQXQH. For each subsequent MQ header structure present, the *Format*, *CodedCharSetId*, and *Encoding* fields in the structure describe the data that follows that structure; that data is either another MQ header structure, or the application message data.

    If the MQGMO_CONVERT option is specified for an MQFMT_XMIT_Q_HEADER message, the application message data and certain of the MQ header structures are converted, *but the data in the MQXQH structure is not*. On return from the MQGET call, therefore:

    - The values of the *Format*, *CodedCharSetId*, and *Encoding* fields in the **MsgDesc** parameter describe the data in the MQXQH structure, and not the application message data; the values are therefore not the same as those specified by the application that issued the MQGET call.

      The effect of this is that an application that repeatedly gets messages from a transmission queue with the MQGMO_CONVERT option specified must reset the *CodedCharSetId* and *Encoding* fields in the **MsgDesc** parameter to the values required for the application message data, before each MQGET call.
    - The values of the *Format*, *CodedCharSetId*, and *Encoding* fields in the last MQ header structure present describe the application message data. If there are no other MQ header structures present, the application message data is described by these fields in the MQMD structure within the MQXQH structure. If conversion is successful, the values will be the same as those specified in the **MsgDesc** parameter by the application that issued the MQGET call.

    If the message is a distribution-list message, the MQXQH structure is followed by an MQDH structure (plus its arrays of MQOR and MQPMR records), which in turn might be followed by zero or more further MQ header structures and zero or more bytes of application message data. Like the

MQXQH structure, the MQDH structure must be in the character set and encoding of the queue manager, and it is not converted on the MQGET call, even if the MQGMO_CONVERT option is specified.

The processing of the MQXQH and MQDH structures described previously is primarily intended for use by message channel agents when they get messages from transmission queues.

Conversion of report messages:

In general a report message can contain varying amounts of application message data, according to the report options specified by the sender of the original message. However, an activity report can contain data but without the report option mentioning *_WITH_DATA in the constant.

In particular, a report message can contain either:

1. No application message data
2. Some of the application message data from the original message

   This occurs when the sender of the original message specifies MQRO_*_WITH_DATA and the message is longer than 100 bytes.
3. All the application message data from the original message

   This occurs when the sender of the original message specifies MQRO_*_WITH_FULL_DATA, or specifies MQRO_*_WITH_DATA and the message is 100 bytes or shorter.

When the queue manager or message channel agent generates a report message, it copies the format name from the original message into the *Format* field in the control information in the report message. The format name in the report message might therefore imply a length of data that is different from the length actually present in the report message (cases 1 and 2 previously).

If the MQGMO_CONVERT option is specified when the report message is retrieved:

- For case 1 previously, the data-conversion exit is not invoked (because the report message has no data).
- For case 3 previously, the format name correctly implies the length of the message data.
- But for case 2 previously, the data-conversion exit is invoked to convert a message that is *shorter* than the length implied by the format name.

  In addition, the reason code passed to the exit is usually MQRC_NONE (that is, the reason code does not indicate that the message has been truncated). This happens because the message data was truncated by the *sender* of the report message, and not by the receiver's queue manager in response to the MQGET call.

Because of these possibilities, the data-conversion exit must not use the format name to deduce the length of data passed to it; instead the exit must check the length of data provided, and be prepared to convert less data than the length implied by the format name. If the data can be converted successfully, completion code MQCC_OK and reason code MQRC_NONE must be returned by the exit. The length of the message data to be converted is passed to the exit as the **InBufferLength** parameter.

Product-sensitive programming interface

MQDXP - Data-conversion exit parameter:

The MQDXP structure is a parameter that the queue manager passes to the data-conversion exit when the exit is invoked to convert the message data as part of the processing of the MQGET call. See the description of the MQ_DATA_CONV_EXIT call for details of the data conversion exit.

Character data in MQDXP is in the character set of the local queue manager; this is given by the **CodedCharSetId** queue manager attribute. Numeric data in MQDXP is in the native machine encoding; this is given by MQENC_NATIVE.

Only the *DataLength*, *CompCode*, *Reason*, and *ExitResponse* fields in MQDXP can be changed by the exit; changes to other fields are ignored. However, the *DataLength* field cannot be changed if the message being converted is a segment that contains only part of a logical message.

When control returns to the queue manager from the exit, the queue manager checks the values returned in MQDXP. If the values returned are not valid, the queue manager continues processing as though the exit had returned MQXDR_CONVERSION_FAILED in *ExitResponse* ; however, the queue manager ignores the values of the *CompCode* and *Reason* fields returned by the exit in this case, and uses instead the values those fields had on *input* to the exit. The following values in MQDXP cause this processing to occur:

- *ExitResponse* field not MQXDR_OK and not MQXDR_CONVERSION_FAILED
- *CompCode* field not MQCC_OK and not MQCC_WARNING
- *DataLength* field less than zero, or *DataLength* field changed when the message being converted is a segment that contains only part of a logical message.

The following table summarizes the fields in the structure.

Table 279. Fields in MQDXP

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
AppOptions	Application options	AppOptions
Encoding	Numeric encoding required by application	Encoding
CodedCharSetId	Character set required by application	CodedCharSetId
DataLength	Length in bytes of message data	DataLength
CompCode	Completion code	CompCode
Reason	Reason code qualifying *CompCode*	Reason
ExitResponse	Response from exit	ExitResponse
Hconn	Connection handle	Hconn
pEntryPoints	Address of the MQIEP structure	pEntryPoints

Fields

The MQDXP structure contains the following fields; the fields are described in alphabetical order.

AppOptions
Type: MQLONG

This is a copy of the *Options* field of the MQGMO structure specified by the application issuing the MQGET call. The exit might need to examine these to ascertain whether the MQGMO_ACCEPT_TRUNCATED_MSG option was specified.

This is an input field to the exit.

CodedCharSetId

Type: MQLONG

This is the coded character-set identifier of the character set required by the application issuing the MQGET call; see the *CodedCharSetId* field in the MQMD structure for more details. If the application specifies the special value MQCCSI_Q_MGR on the MQGET call, the queue manager changes this to the actual character-set identifier of the character set used by the queue manager, before invoking the exit.

If the conversion is successful, the exit must copy this to the *CodedCharSetId* field in the message descriptor.

This is an input field to the exit.

CompCode

Type: MQLONG

When the exit is invoked, this contains the completion code that is returned to the application that issued the MQGET call, if the exit does nothing. It is always MQCC_WARNING, because either the message was truncated, or the message requires conversion and this has not yet been done.

On output from the exit, this field contains the completion code to be returned to the application in the **CompCode** parameter of the MQGET call; only MQCC_OK and MQCC_WARNING are valid. See the description of the *Reason* field for suggestions on how the exit can set this field on output.

This is an input/output field to the exit.

DataLength

Type: MQLONG

When the exit is invoked, this field contains the original length of the application message data. If the message was truncated to fit into the buffer provided by the application, the size of the message provided to the exit is *smaller* than the value of *DataLength*. The size of the message provided to the exit is always given by the **InBufferLength** parameter of the exit, irrespective of any truncation that has occurred.

Truncation is indicated by the *Reason* field having the value MQRC_TRUNCATED_MSG_ACCEPTED on input to the exit.

Most conversions do not need to change this length, but an exit can do so if necessary; the value set by the exit is returned to the application in the **DataLength** parameter of the MQGET call. However, this length cannot be changed if the message being converted is a segment that contains only part of a logical message. This is because changing the length would cause the offsets of later segments in the logical message to be incorrect.

Note that, if the exit wants to change the length of the data, be aware that the queue manager has already decided whether the message data fits into the application's buffer, based on the length of the *unconverted* data. This decision determines whether the message is removed from the queue (or the browse cursor moved, for a browse request), and is not affected by any change to the data length caused by the conversion. For this reason it is recommended that conversion exits do not cause a change in the length of the application message data.

If character conversion does imply a change of length, a string can be converted into another string with the same length in bytes, truncating trailing blanks, or padding with blanks as necessary.

The exit is not invoked if the message contains no application message data; hence *DataLength* is always greater then zero.

This is an input/output field to the exit.

Encoding
Type: MQLONG

Numeric encoding required by application.

This is the numeric encoding required by the application issuing the MQGET call; see the *Encoding* field in the MQMD structure for more details.

If the conversion is successful, the exit copies this to the *Encoding* field in the message descriptor.

This is an input field to the exit.

ExitOptions
Type: MQLONG

This is a reserved field; its value is `0`.

ExitResponse
Type: MQLONG

Response from exit. This is set by the exit to indicate the success or otherwise of the conversion. It must be one of the following:

MQXDR_OK
Conversion was successful.

If the exit specifies this value, the queue manager returns the following to the application that issued the MQGET call:

- The value of the *CompCode* field on output from the exit
- The value of the *Reason* field on output from the exit
- The value of the *DataLength* field on output from the exit
- The contents of the exit's output buffer *OutBuffer*. The number of bytes returned is the lesser of the exit's **OutBufferLength** parameter, and the value of the *DataLength* field on output from the exit.

  If the *Encoding* and *CodedCharSetId* fields in the exit's message descriptor parameter are *both* unchanged, the queue manager returns:
- The value of the *Encoding* and *CodedCharSetId* fields in the MQDXP structure on *input* to the exit.

  If one or both of the *Encoding* and *CodedCharSetId* fields in the exit's message descriptor parameter has been changed, the queue manager returns:
- The value of the *Encoding* and *CodedCharSetId* fields in the exit's message descriptor parameter on output from the exit

MQXDR_CONVERSION_FAILED
Conversion was unsuccessful.

If the exit specifies this value, the queue manager returns the following to the application that issued the MQGET call:

- The value of the *CompCode* field on output from the exit
- The value of the *Reason* field on output from the exit
- The value of the *DataLength* field on *input* to the exit
- The contents of the exit's input buffer *InBuffer*. The number of bytes returned is given by the **InBufferLength** parameter

If the exit has altered *InBuffer*, the results are undefined.

ExitResponse is an output field from the exit.

Hconn
Type: MQHCONN

This is a connection handle which can be used on the MQXCNVC call. This handle is not necessarily the same as the handle specified by the application which issued the MQGET call.

pEntryPoints
Type: PMQIEP

The address of an MQIEP structure through which MQI and DCI calls can be made.

Reason
Type: MQLONG

Reason code qualifying *CompCode*.

When the exit is invoked, this contains the reason code that is returned to the application that issued the MQGET call, if the exit chooses to do nothing. Among possible values are MQRC_TRUNCATED_MSG_ACCEPTED, indicating that the message was truncated in order fit into the buffer provided by the application, and MQRC_NOT_CONVERTED, indicating that the message requires conversion but that this has not yet been done.

On output from the exit, this field contains the reason to be returned to the application in the **Reason** parameter of the MQGET call; the following is recommended:

- If *Reason* had the value MQRC_TRUNCATED_MSG_ACCEPTED on input to the exit, the *Reason* and *CompCode* fields must not be altered, irrespective of whether the conversion succeeds or fails.

  (If the *CompCode* field is not MQCC_OK, the application which retrieves the message can identify a conversion failure by comparing the returned *Encoding* and *CodedCharSetId* values in the message descriptor with the values requested; in contrast, the application cannot distinguish a truncated message from a message that fitted the buffer. For this reason, MQRC_TRUNCATED_MSG_ACCEPTED must be returned in preference to any of the reasons that indicate conversion failure.)
- If *Reason* had any other value on input to the exit:
  - If the conversion succeeds, *CompCode* must be set to MQCC_OK and *Reason* set to MQRC_NONE.
  - If the conversion fails, or the message expands and has to be truncated to fit in the buffer, *CompCode* must be set to MQCC_WARNING (or left unchanged), and *Reason* set to one of the values listed, to indicate the nature of the failure.

    Note if the message after conversion is too large for the buffer, it must be truncated only if the application that issued the MQGET call specified the MQGMO_ACCEPT_TRUNCATED_MSG option:
    - If it did specify that option, reason MQRC_TRUNCATED_MSG_ACCEPTED is returned.
    - If it did not specify that option, the message is returned unconverted, with reason code MQRC_CONVERTED_MSG_TOO_BIG.

The reason codes listed are recommended for use by the exit to indicate the reason that conversion failed, but the exit can return other values from the set of MQRC_* codes if deemed appropriate. In addition, the range of values MQRC_APPL_FIRST through MQRC_APPL_LAST are allocated for use by the exit to indicate conditions that the exit wants to communicate to the application issuing the MQGET call.

Note: If the message cannot be converted successfully, the exit must return MQXDR_CONVERSION_FAILED in the *ExitResponse* field, in order to cause the queue manager to return the unconverted message. This is true regardless of the reason code returned in the *Reason* field.

MQRC_APPL_FIRST
(900, X'384') Lowest value for application-defined reason code.

MQRC_APPL_LAST
(999, X'3E7') Highest value for application-defined reason code.

MQRC_CONVERTED_MSG_TOO_BIG
(2120, X'848') Converted data too large for buffer.

MQRC_NOT_CONVERTED
(2119, X'847') Message data not converted.

MQRC_SOURCE_CCSID_ERROR
(2111, X'83F') Source coded character set identifier not valid.

MQRC_SOURCE_DECIMAL_ENC_ERROR
(2113, X'841') Packed-decimal encoding in message not recognized.

MQRC_SOURCE_FLOAT_ENC_ERROR
(2114, X'842') Floating-point encoding in message not recognized.

MQRC_SOURCE_INTEGER_ENC_ERROR
(2112, X'840') Source integer encoding not recognized.

MQRC_TARGET_CCSID_ERROR
(2115, X'843') Target coded character set identifier not valid.

MQRC_TARGET_DECIMAL_ENC_ERROR
(2117, X'845') Packed-decimal encoding specified by receiver not recognized.

MQRC_TARGET_FLOAT_ENC_ERROR
(2118, X'846') Floating-point encoding specified by receiver not recognized.

MQRC_TARGET_INTEGER_ENC_ERROR
(2116, X'844') Target integer encoding not recognized.

MQRC_TRUNCATED_MSG_ACCEPTED
(2079, X'81F') Truncated message returned (processing completed).

This is an input/output field to the exit.

StrucId
Type: MQCHAR4

Structure identifier. The value must be:

MQDXP_STRUC_ID
Identifier for data conversion exit parameter structure.

For the C programming language, the constant MQDXP_STRUC_ID_ARRAY is also defined; this has the same value as MQDXP_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

Version
Type: MQLONG

Structure version number. The value must be:

MQDXP_VERSION_1
Version number for data-conversion exit parameter structure.

The following constant specifies the version number of the current version:

MQDXP_CURRENT_VERSION
Current version of data-conversion exit parameter structure.

Note: When a new version of this structure is introduced, the layout of the existing part is not changed. The exit must therefore check that the *Version* field is equal to or greater than the lowest version which contains the fields that the exit needs to use.

This is an input field to the exit.

C declaration

```
typedef struct tagMQDXP MQDXP;
struct tagMQDXP {
  MQCHAR4  StrucId;         /* Structure identifier */
  MQLONG   Version;         /* Structure version number */
  MQLONG   ExitOptions;     /* Reserved */
  MQLONG   AppOptions;      /* Application options */
  MQLONG   Encoding;        /* Numeric encoding required by
                               application */
  MQLONG   CodedCharSetId;  /* Character set required by application */
  MQLONG   DataLength;      /* Length in bytes of message data */
  MQLONG   CompCode;        /* Completion code */
  MQLONG   Reason;          /* Reason code qualifying CompCode */
  MQLONG   ExitResponse;    /* Response from exit */
  MQHCONN  Hconn;           /* Connection handle */
  PMQIEP   pEntryPoints;    /* Address of the MQIEP structure */
};
```

COBOL declaration (IBM i only)

```
**   MQDXP structure
  10 MQDXP.
**    Structure identifier
   15 MQDXP-STRUCID        PIC X(4).
**    Structure version number
   15 MQDXP-VERSION        PIC S9(9) BINARY.
**    Reserved
   15 MQDXP-EXITOPTIONS    PIC S9(9) BINARY.
**    Application options
   15 MQDXP-APPOPTIONS     PIC S9(9) BINARY.
**    Numeric encoding required by application
   15 MQDXP-ENCODING       PIC S9(9) BINARY.
**    Character set required by application
   15 MQDXP-CODEDCHARSETID PIC S9(9) BINARY.
**    Length in bytes of message data
   15 MQDXP-DATALENGTH     PIC S9(9) BINARY.
**    Completion code
   15 MQDXP-COMPCODE       PIC S9(9) BINARY.
**    Reason code qualifying COMPCODE
   15 MQDXP-REASON         PIC S9(9) BINARY.
**    Response from exit
   15 MQDXP-EXITRESPONSE   PIC S9(9) BINARY.
**    Connection handle
   15 MQDXP-HCONN          PIC S9(9) BINARY.
```

System/390 assembler declaration

```
MQDXP                 DSECT
MQDXP_STRUCID         DS   CL4  Structure identifier
MQDXP_VERSION         DS   F    Structure version number
MQDXP_EXITOPTIONS     DS   F    Reserved
MQDXP_APPOPTIONS      DS   F    Application options
MQDXP_ENCODING        DS   F    Numeric encoding required by application
MQDXP_CODEDCHARSETID  DS   F    Character set required by application
MQDXP_DATALENGTH      DS   F    Length in bytes of message data
MQDXP_COMPCODE        DS   F    Completion code
MQDXP_REASON          DS   F    Reason code qualifying COMPCODE
MQDXP_EXITRESPONSE    DS   F    Response from exit
MQDXP_HCONN           DS   F    Connection handle
```

```
*
MQDXP_LENGTH          EQU  *-MQDXP
                      ORG  MQDXP
MQDXP_AREA            DS   CL(MQDXP_LENGTH)
```

MQXCNVC - Convert characters:

The MQXCNVC call converts characters from one character set to another using the C programming language.

This call is part of the IBM MQ Data Conversion Interface (DCI), which is one of the IBM MQ framework interfaces.

Note: The call can be used from both application, and data-conversion exit environments.

Syntax

MQXCNVC (*Hconn*, *Options*, *SourceCCSID*, *SourceLength*, *SourceBuffer*, *TargetCCSID*, *TargetLength*, *TargetBuffer*, *DataLength*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager.

In a data-conversion exit, *Hconn* is normally the handle that is passed to the data-conversion exit in the *Hconn* field of the MQDXP structure; this handle is not necessarily the same as the handle specified by the application which issued the MQGET call.

On IBM i, the following special value can be specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

If you run a CICS TS 3.2 or higher application, ensure that the character conversion exit program, which invokes the MQXCNVC call, is defined as OPENAPI. This definition prevents the 2018 MQRC_HCONN_ERROR error caused by from an incorrect connection, and allows the MQGET to complete.

Options

Type: MQLONG - input

Options that control the action of MQXCNVC.

Zero or more of the options described can be specified. To specify more than one option, either add the values together (do not add the same constant more than once), or combine the values using the bitwise OR operation (if the programming language supports bit operations).

Default-conversion option: The following option controls the use of default character conversion:

MQDCC_DEFAULT_CONVERSION
Default conversion.

This option specifies that default character conversion can be used if one or both of the character sets specified on the call is not supported. This allows the queue manager to use an installation-specified default character set that approximates the specified character set, when converting the string.

Note: The result of using an approximate character set to convert the string is that some characters can be converted incorrectly. This can be avoided by using in the string only characters which are common to both the specified character set and the default character set.

The default character sets are defined by a configuration option when the queue manager is installed or restarted.

If MQDCC_DEFAULT_CONVERSION is not specified, the queue manager uses only the specified character sets to convert the string, and the call fails if one or both of the character sets is not supported.

This option is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows.

Padding option: The following option allows the queue manager to pad the converted string with blanks or discard insignificant trailing characters, in order to make the converted string fit the target buffer:

MQDCC_FILL_TARGET_BUFFER
Fill target buffer.

This option requests that conversion take place in such a way that the target buffer is filled completely:

- If the string contracts when it is converted, trailing blanks are added in order to fill the target buffer.
- If the string expands when it is converted, trailing characters that are not significant are discarded to make the converted string fit the target buffer. If this can be done successfully, the call completes with MQCC_OK and reason code MQRC_NONE.

  If there are too few insignificant trailing characters, as much of the string as can fit is placed in the target buffer, and the call completes with MQCC_WARNING and reason code MQRC_CONVERTED_MSG_TOO_BIG.

  Insignificant characters are:

  – Trailing blanks
  – Characters following the first null character in the string (but excluding the first null character itself)
- If the string, *TargetCCSID*, and *TargetLength* are such that the target buffer cannot be set completely with valid characters, the call fails with MQCC_FAILED and reason code MQRC_TARGET_LENGTH_ERROR. This can occur when *TargetCCSID* is a pure DBCS character set (such as V 9.0.0 UTF-16), but *TargetLength* specifies a length that is an odd number of bytes.
- *TargetLength* can be less than or greater than *SourceLength*. On return from MQXCNVC, *DataLength* has the same value as *TargetLength*.

If this option is not specified:

- The string is allowed to contract or expand within the target buffer as required. Insignificant trailing characters are not added or discarded.

  If the converted string fits in the target buffer, the call completes with MQCC_OK and reason code MQRC_NONE.

  If the converted string is too large for the target buffer, as much of the string as fits is placed in the target buffer, and the call completes with MQCC_WARNING and reason code MQRC_CONVERTED_MSG_TOO_BIG. Note fewer than *TargetLength* bytes can be returned in this case.
- *TargetLength* can be less than or greater than *SourceLength*. On return from MQXCNVC, *DataLength* is less than or equal to *TargetLength*.

This option is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows.

Encoding options: The options described can be used to specify the integer encodings of the source and target strings. The relevant encoding is used only when the corresponding character set identifier

indicates that the representation of the character set in main storage is dependent on the encoding used for binary integers. This affects only certain multibyte character sets (for example, V 9.0.0 UTF-16 character sets).

The encoding is ignored if the character set is a single-byte character set (SBCS), or a multibyte character set with representation in main storage that is not dependent on the integer encoding.

Only one of the MQDCC_SOURCE_* values must be specified, combined with one of the MQDCC_TARGET_* values:

MQDCC_SOURCE_ENC_NATIVE
Source encoding is the default for the environment and programming language.

MQDCC_SOURCE_ENC_NORMAL
Source encoding is normal.

MQDCC_SOURCE_ENC_REVERSED
Source encoding is reversed.

MQDCC_SOURCE_ENC_UNDEFINED
Source encoding is undefined.

MQDCC_TARGET_ENC_NATIVE
Target encoding is the default for the environment and programming language.

MQDCC_TARGET_ENC_NORMAL
Target encoding is normal.

MQDCC_TARGET_ENC_REVERSED
Target encoding is reversed.

MQDCC_TARGET_ENC_UNDEFINED
Target encoding is undefined.

The encoding values defined previously can be added directly to the *Options* field. However, if the source or target encoding is obtained from the *Encoding* field in the MQMD or other structure, the following processing must be done:

1. The integer encoding must be extracted from the *Encoding* field by eliminating the float and packed-decimal encodings; see "Analyzing encodings" on page 2904 for details of how to do this.
2. The integer encoding resulting from step 1 must be multiplied by the appropriate factor before being added to the *Options* field. These factors are:
   - MQDCC_SOURCE_ENC_FACTOR for the source encoding
   - MQDCC_TARGET_ENC_FACTOR for the target encoding

The following example code illustrates how this might be coded in the C programming language:

```
Options = (MsgDesc.Encoding & MQENC_INTEGER_MASK)
           * MQDCC_SOURCE_ENC_FACTOR
        + (DataConvExitParms.Encoding & MQENC_INTEGER_MASK)
           * MQDCC_TARGET_ENC_FACTOR;
```

If not specified, the encoding options default to undefined (MQDCC_*_ENC_UNDEFINED). In most cases, this does not affect the successful completion of the MQXCNVC call. However, if the corresponding character set is a multibyte character set with representation that is dependent on the encoding (for example, a V 9.0.0 UTF-16 character set), the call fails with reason code MQRC_SOURCE_INTEGER_ENC_ERROR or MQRC_TARGET_INTEGER_ENC_ERROR as appropriate.

The encoding options are supported in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Linux, Windows.

Default option: If none of the options described previously is specified, the following option can be used:

MQDCC_NONE
No options specified.

MQDCC_NONE is defined to aid program documentation. It is not intended that this option is used with any other, but as its value is zero, such use cannot be detected.

SourceCCSID
Type: MQLONG - input

This is the coded character set identifier of the input string in *SourceBuffer*.

SourceLength
Type: MQLONG - input

This is the length in bytes of the input string in *SourceBuffer* ; it must be zero or greater.

SourceBuffer
Type: MQCHAR x SourceLength - input

This is the buffer containing the string to be converted from one character set to another.

TargetCCSID
Type: MQLONG - input

This is the coded character set identifier of the character set to which *SourceBuffer* is to be converted.

TargetLength
Type: MQLONG - input

This is the length in bytes of the output buffer *TargetBuffer* ; it must be zero or greater. It can be less than or greater than *SourceLength*.

TargetBuffer
Type: MQCHAR x TargetLength - output

This is the string after it has been converted to the character set defined by *TargetCCSID*. The converted string can be shorter or longer than the unconverted string. The **DataLength** parameter indicates the number of valid bytes returned.

DataLength
Type: MQLONG - output

This is the length of the string returned in the output buffer *TargetBuffer*. The converted string can be shorter or longer than the unconverted string.

CompCode
Type: MQLONG - output

It is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_CONVERTED_MSG_TOO_BIG
(2120, X'848') Converted data too large for buffer.

If *CompCode* is MQCC_FAILED:

MQRC_DATA_LENGTH_ERROR
(2010, X'7DA') Data length parameter not valid.

MQRC_DBCS_ERROR
(2150, X'866') DBCS string not valid.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SOURCE_BUFFER_ERROR
(2145, X'861') Source buffer parameter not valid.

MQRC_SOURCE_CCSID_ERROR
(2111, X'83F') Source coded character set identifier not valid.

MQRC_SOURCE_INTEGER_ENC_ERROR
(2112, X'840') Source integer encoding not recognized.

MQRC_SOURCE_LENGTH_ERROR
(2143, X'85F') Source length parameter not valid.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_TARGET_BUFFER_ERROR
(2146, X'862') Target buffer parameter not valid.

MQRC_TARGET_CCSID_ERROR
(2115, X'843') Target coded character set identifier not valid.

MQRC_TARGET_INTEGER_ENC_ERROR
(2116, X'844') Target integer encoding not recognized.

MQRC_TARGET_LENGTH_ERROR
(2144, X'860') Target length parameter not valid.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

C invocation

```
MQXCNVC (Hconn, Options, SourceCCSID, SourceLength, SourceBuffer,
        TargetCCSID, TargetLength, TargetBuffer, &DataLength,
        &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;            /* Connection handle */
MQLONG   Options;          /* Options that control the action of
                              MQXCNVC */
MQLONG   SourceCCSID;      /* Coded character set identifier of string
                              before conversion */
MQLONG   SourceLength;     /* Length of string before conversion */
```

```
MQCHAR   SourceBuffer[n];  /* String to be converted */
MQLONG   TargetCCSID;      /* Coded character set identifier of string
                              after conversion */
MQLONG   TargetLength;     /* Length of output buffer */
MQCHAR   TargetBuffer[n];  /* String after conversion */
MQLONG   DataLength;       /* Length of output string */
MQLONG   CompCode;         /* Completion code */
MQLONG   Reason;           /* Reason code qualifying CompCode */
```

COBOL declaration (IBM i only)

```
     CALL 'MQXCNVC' USING HCONN, OPTIONS, SOURCECCSID, SOURCELENGTH,
                          SOURCEBUFFER, TARGETCCSID, TARGETLENGTH,
                          TARGETBUFFER, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN         PIC S9(9) BINARY.
**   Options that control the action of MQXCNVC
 01  OPTIONS       PIC S9(9) BINARY.
**   Coded character set identifier of string before conversion
 01  SOURCECCSID   PIC S9(9) BINARY.
**   Length of string before conversion
 01  SOURCELENGTH  PIC S9(9) BINARY.
**   String to be converted
 01  SOURCEBUFFER  PIC X(n).
**   Coded character set identifier of string after conversion
 01  TARGETCCSID   PIC S9(9) BINARY.
**   Length of output buffer
 01  TARGETLENGTH  PIC S9(9) BINARY.
**   String after conversion
 01  TARGETBUFFER  PIC X(n).
**   Length of output string
 01  DATALENGTH    PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE      PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON        PIC S9(9) BINARY.
```

S/390 assembler declaration

```
         CALL MQXCNVC,(HCONN,OPTIONS,SOURCECCSID,SOURCELENGTH,         X
               SOURCEBUFFER,TARGETCCSID,TARGETLENGTH,TARGETBUFFER,     X
               DATALENGTH,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN         DS  F      Connection handle
OPTIONS       DS  F      Options that control the action of MQXCNVC
SOURCECCSID   DS  F      Coded character set identifier of string before
*                        conversion
SOURCELENGTH  DS  F      Length of string before conversion
SOURCEBUFFER  DS  CL(n)  String to be converted
TARGETCCSID   DS  F      Coded character set identifier of string after
*                        conversion
TARGETLENGTH  DS  F      Length of output buffer
TARGETBUFFER  DS  CL(n)  String after conversion
DATALENGTH    DS  F      Length of output string
COMPCODE      DS  F      Completion code
REASON        DS  F      Reason code qualifying COMPCODE
```

MQ_DATA_CONV_EXIT - Data conversion exit:

The MQ_DATA_CONV_EXIT call describes the parameters that are passed to the data-conversion exit.

No entry point called MQ_DATA_CONV_EXIT is provided by the queue manager (see usage note 11).

This definition is part of the IBM MQ Data Conversion Interface (DCI), which is one of the IBM MQ framework interfaces.

Syntax

MQ_DATA_CONV_EXIT (*DataConvExitParms*, *MsgDesc*, *InBufferLength*, *InBuffer*, *OutBufferLength*, *OutBuffer*)

Parameters

DataConvExitParms
Type: MQDXP - input/output

This structure contains information relating to the invocation of the exit. The exit sets information in this structure to indicate the outcome of the conversion. See "MQDXP - Data-conversion exit parameter" on page 2916 for details of the fields in this structure.

MsgDesc
Type: MQMD - input/output

On input to the exit, this is the message descriptor associated with the message data passed to the exit in the **InBuffer** parameter.

Note: The **MsgDesc** parameter passed to the exit is always the most recent version of MQMD supported by the queue manager which invokes the exit. If the exit is intended to be portable between different environments, the exit will check the *Version* field in *MsgDesc* to verify that the fields that the exit needs to access are present in the structure.

In the following environments, the exit is passed a version-2 MQMD: AIX, HP-UX, IBM i, Solaris, Linux, Windows. In all other environments that support the data conversion exit, the exit is passed a version-1 MQMD.

On output, the exit will change the *Encoding* and *CodedCharSetId* fields to the values requested by the application, if conversion was successful; these changes are reflected back to the application. Any other changes that the exit makes to the structure are ignored; they are not reflected back to the application.

If the exit returns MQXDR_OK in the *ExitResponse* field of the MQDXP structure, but does not change the *Encoding* or *CodedCharSetId* fields in the message descriptor, the queue manager returns for those fields the values that the corresponding fields in the MQDXP structure had on input to the exit.

InBufferLength
Type: MQLONG - input

Length in bytes of *InBuffer*.

This is the length of the input buffer *InBuffer*, and specifies the number of bytes to be processed by the exit. *InBufferLength* is the lesser of the length of the message data before conversion, and the length of the buffer provided by the application on the MQGET call.

The value is always greater than zero.

InBuffer
Type: MQBYTExInBufferLength - input

Buffer containing the unconverted message.

This contains the message data before conversion. If the exit is unable to convert the data, the queue manager returns the contents of this buffer to the application after the exit has completed.

Note: The exit should not alter *InBuffer* ; if this parameter is altered, the results are undefined.

In the C programming language, this parameter is defined as a pointer-to-void.

OutBufferLength
Type: MQLONG - input

Length in bytes of *OutBuffer*.

This is the length of the output buffer *OutBuffer*, and is the same as the length of the buffer provided by the application on the MQGET call.

The value is always greater than zero.

OutBuffer
Type: MQBYTExOutBufferLength - output

Buffer containing the converted message.

On output from the exit, if the conversion was successful (as indicated by the value MQXDR_OK in the *ExitResponse* field of the **DataConvExitParms** parameter), *OutBuffer* contains the message data to be delivered to the application, in the requested representation. If the conversion was unsuccessful, any changes that the exit has made to this buffer are ignored.

In the C programming language, this parameter is defined as a pointer-to-void.

Usage notes

1. A data-conversion exit is a user-written exit which receives control during the processing of an MQGET call. The function performed by the data-conversion exit is defined by the provider of the exit; however, the exit must conform to the rules described here, and in the associated parameter structure MQDXP.

   The programming languages that can be used for a data-conversion exit are determined by the environment.
2. The exit is invoked only if *all* of the following statements are true:
   - The MQGMO_CONVERT option is specified on the MQGET call
   - The *Format* field in the message descriptor is not MQFMT_NONE
   - The message is not already in the required representation; that is, one or both of the message's *CodedCharSetId* and *Encoding* is different from the value specified by the application in the message descriptor supplied on the MQGET call
   - The queue manager has not already done the conversion successfully
   - The length of the application's buffer is greater than zero
   - The length of the message data is greater than zero
   - The reason code so far during the MQGET operation is MQRC_NONE or MQRC_TRUNCATED_MSG_ACCEPTED
3. When an exit is being written, consider coding the exit in a way that allows it to convert messages that have been truncated. Truncated messages can arise in the following ways:
   - The receiving application provides a buffer that is smaller than the message, but specifies the MQGMO_ACCEPT_TRUNCATED_MSG option on the MQGET call.

     In this case, the *Reason* field in the **DataConvExitParms** parameter on input to the exit has the value MQRC_TRUNCATED_MSG_ACCEPTED.
   - The sender of the message truncated it before sending it. This can happen with report messages, for example (see "Conversion of report messages" on page 2915 for more details).

In this case, the *Reason* field in the **DataConvExitParms** parameter on input to the exit has the value MQRC_NONE (if the receiving application provided a buffer that was large enough for the message).

Thus the value of the *Reason* field on input to the exit cannot always be used to decide whether the message has been truncated.

The distinguishing characteristic of a truncated message is that the length provided to the exit in the **InBufferLength** parameter is *less than* the length implied by the format name contained in the *Format* field in the message descriptor. The exit should therefore check the value of *InBufferLength* before attempting to convert any of the data; the exit *should not* assume that the full amount of data implied by the format name has been provided.

If the exit has not been written to convert truncated messages, and *InBufferLength* is less than the value expected, the exit will return MQXDR_CONVERSION_FAILED in the *ExitResponse* field of the **DataConvExitParms** parameter, with the *CompCode* and *Reason* fields set to MQCC_WARNING and MQRC_FORMAT_ERROR.

If the exit *has* been written to convert truncated messages, the exit will convert as much of the data as possible (see next usage note), taking care not to attempt to examine or convert data beyond the end of *InBuffer*. If the conversion completes successfully, the exit will leave the *Reason* field in the **DataConvExitParms** parameter unchanged. This returns MQRC_TRUNCATED_MSG_ACCEPTED if the message was truncated by the receiver's queue manager, and MQRC_NONE if the message was truncated by the sender of the message.

It is also possible for a message to expand *during* conversion, to the point where it is bigger than *OutBuffer*. In this case the exit must decide whether to truncate the message; the *AppOptions* field in the **DataConvExitParms** parameter indicates whether the receiving application specified the MQGMO_ACCEPT_TRUNCATED_MSG option.

4. Generally, all the data in the message provided to the exit in *InBuffer* is converted, or that none of it is. An exception to this, however, occurs if the message is truncated, either before conversion or during conversion; in this case there can be an incomplete item at the end of the buffer (for example: 1 byte of a double-byte character, or 3 bytes of a 4-byte integer). In this situation, consider omitting the incomplete item and set the unused bytes in the *OutBuffer* to nulls. However, complete elements or characters within an array or string *should* be converted.
5. When an exit is needed for the first time, the queue manager attempts to load an object that has the same name as the format (apart from extensions). The object loaded must contain the exit that processes messages with that format name. Consider making the exit name, and the name of the object that contains the exit identical, although not all environments require this.
6. A new copy of the exit is loaded when an application attempts to retrieve the first message that uses that *Format* since the application connected to the queue manager. For CICS or IMS applications, this means when the CICS or IMS subsystem connected to the queue manager. A new copy can also be loaded at other times, if the queue manager has discarded a previously loaded copy. For this reason, an exit must not attempt to use static storage to communicate information from one invocation of the exit to the next - the exit can be unloaded between the two invocations.
7. If there is a user-supplied exit with the same name as one of the built-in formats supported by the queue manager, the user-supplied exit does not replace the built-in conversion routine. The only circumstances in which such an exit is invoked are:
   - If the built-in conversion routine cannot handle conversions to or from either the *CodedCharSetId* or *Encoding* involved, or
   - If the built-in conversion routine has failed to convert the data (for example, because there is a field or character which cannot be converted).
8. The scope of the exit is environment-dependent. *Format* names must be chosen to minimize the risk of clashes with other formats. Consider starting with characters that identify the application defining the format name.
9. The data-conversion exit runs in an environment like that of the program which issued the MQGET call; environment includes address space and user profile (where applicable). The program could be

a message channel agent sending messages to a destination queue manager that does not support message conversion. The exit cannot compromise the queue manager's integrity, since it does not run in the queue manager's environment.

10. The only MQI call which can be used by the exit is MQXCNVC; attempting to use other MQI calls fails with reason code MQRC_CALL_IN_PROGRESS, or other unpredictable errors.
11. No entry point called MQ_DATA_CONV_EXIT is provided by the queue manager. However, a `typedef` is provided for the name MQ_DATA_CONV_EXIT in the C programming language, and this can be used to declare the user-written exit, to ensure that the parameters are correct. The name of the exit must be the same as the format name (the name contained in the *Format* field in MQMD), although this is not required in all environments.

    The following example illustrates how the exit that processes the format `MYFORMAT` can be declared in the C programming language:

```
#include "cmqc.h"
#include "cmqxc.h"

MQ_DATA_CONV_EXIT MYFORMAT;

void MQENTRY MYFORMAT(
     PMQDXP   pDataConvExitParms, /* Data-conversion exit parameter
                                     block */
     PMQMD    pMsgDesc,           /* Message descriptor */
     MQLONG   InBufferLength,     /* Length in bytes of InBuffer */
     PMQVOID  pInBuffer,          /* Buffer containing the unconverted
                                     message */
     MQLONG   OutBufferLength,    /* Length in bytes of OutBuffer */
     PMQVOID  pOutBuffer)         /* Buffer containing the converted
                                     message */
{
  /* C language statements to convert message */
}
```

12. On z/OS, if an API-crossing exit is also in force, it is called after the data-conversion exit.

C invocation

```
exitname (&DataConvExitParms, &MsgDesc, InBufferLength,
         InBuffer, OutBufferLength, OutBuffer);
```

The parameters passed to the exit are declared as follows:

```
MQDXP   DataConvExitParms;  /* Data-conversion exit parameter block */
MQMD    MsgDesc;            /* Message descriptor */
MQLONG  InBufferLength;     /* Length in bytes of InBuffer */
MQBYTE  InBuffer[n];        /* Buffer containing the unconverted
                               message */
MQLONG  OutBufferLength;    /* Length in bytes of OutBuffer */
MQBYTE  OutBuffer[n];       /* Buffer containing the converted
                               message */
```

COBOL declaration (IBM i only)

```
     CALL 'exitname' USING DATACONVEXITPARMS, MSGDESC, INBUFFERLENGTH,
                           INBUFFER, OUTBUFFERLENGTH, OUTBUFFER.
```

The parameters passed to the exit are declared as follows:

```
**   Data-conversion exit parameter block
 01  DATACONVEXITPARMS.
     COPY CMQDXPV.
**   Message descriptor
 01  MSGDESC.
     COPY CMQMDV.
**   Length in bytes of INBUFFER
 01  INBUFFERLENGTH     PIC S9(9) BINARY.
**   Buffer containing the unconverted message
```

```
 01  INBUFFER           PIC X(n).
**   Length in bytes of OUTBUFFER
 01  OUTBUFFERLENGTH    PIC S9(9) BINARY.
**   Buffer containing the converted message
 01  OUTBUFFER          PIC X(n).
```

System/390 assembler declaration

```
        CALL EXITNAME,(DATACONVEXITPARMS,MSGDESC,INBUFFERLENGTH,      X
              INBUFFER,OUTBUFFERLENGTH,OUTBUFFER)
```

The parameters passed to the exit are declared as follows:

```
DATACONVEXITPARMS  CMQDXPA  ,      Data-conversion exit parameter block
MSGDESC            CMQMDA   ,      Message descriptor
INBUFFERLENGTH     DS       F      Length in bytes of INBUFFER
INBUFFER           DS       CL(n)  Buffer containing the unconverted
*                                  message
OUTBUFFERLENGTH    DS       F      Length in bytes of OUTBUFFER
OUTBUFFER          DS       CL(n)  Buffer containing the converted
*                                  message
```

Properties specified as MQRFH2 elements

Non-message descriptor properties can be specified as elements in MQRFH2 header folders. Overview of MQRFH2 elements being specified as properties.

This retains compatibility with the previous versions of the IBM MQ JMS and XMS clients. This section describes how to specify properties in MQRFH2 headers.

To use MQRFH2 elements as properties, specify the elements as described in Using IBM MQ classes for Java . This information supplements the information described in “MQRFH2 - Rules and formatting header 2” on page 2508.

Mapping property data types to MQRFH2 data types:

This topic provides information on message property types mapped to their corresponding MQRFH2 data types.

Table 280. Supported MQRFH2 data types

Message property type	MQRFH2 data type
MQBYTE[]	bin.hex
MQBOOL	boolean
MQINT8	i1
MQINT16	i2
MQINT32	i4
MQINT64	i8
MQFLOAT32	r4
MQFLOAT64	r8
MQCHAR[]	string

Any element without a data type is assumed to be of type "string".

An MQRFH2 data type of `int`, meaning an integer of unspecified size, is treated as if it were an `i8`.

A null value is indicated by the element attribute `xsi:nil='true'`. Do not use the attribute `xsi:nil='false'` for non-null values.

For example, the following property has a null value:

```
<NullProperty xsi:nil='true'></NullProperty>
```

A byte or character string property can have an empty value. This is represented by an MQRFH2 element with a zero length element value.

For example, the following property has an empty value:

```
<EmptyProperty></EmptyProperty>
```

Supported MQRFH2 folders:

Overview of the use of message descriptor fields as properties.

The folders <jms>, <mcd>, <mqext>, and <usr> are described in The MQRFH2 header and JMS. The <usr> folder is used to transport any JMS application-defined properties that are associated with a message. Groups are not allowed in the <usr> folder.

The MQRFH2 header and JMS supports the following additional folders:

- <mq>

  This folder is used and reserved for MQ-defined properties that are used by IBM MQ.
- <mq_usr>

  This folder can be used to transport any application-defined properties that are not exposed as JMS user-defined properties, as the properties might not meet the requirements of a JMS property. This folder can contain groups that the <usr> folder cannot.
- Any folder marked with the content='properties' attribute.

  Such a folder is equivalent to the <mq_usr> folder in content.
- <mqps>

  This folder is used for IBM MQ publish/subscribe properties.

IBM MQ also supports the following folders that are already in use by WAS/SIB:

- <sib>

  This folder is used and reserved for WAS/SIB system message properties that are not exposed as JMS properties, or are mapped to JMS_IBM_* properties, but are exposed to WAS/SIB applications; these include forward and reverse routing paths properties.

  At least some cannot be exposed as JMS properties, because they are byte arrays. If your application adds properties to this folder, the value is either ignored or removed.
- <sib_usr>

  This folder is used and reserved for WAS/SIB user message properties that cannot be exposed as JMS user properties because they are not of supported types; they are exposed to WAS/SIB applications.

  These are user properties, that you can get or set through the SIMessage interface, but the content of the byte array is mapped to the required property value.

  If your IBM MQ application writes an arbitrary bin.hex element to the folder, the application probably receives an IOException, as it is not of the format expected to restore. If you add anything other than a bin.hex element you receive a ClassCastException.

  Do not attempt to make properties available to WAS/SIB by using this folder; instead user the <usr> folder for that purpose.
- <sib_context>

  This folder is used for WAS/SIB system message properties that are not exposed to WAS/SIB user applications or as JMS properties. These include security and transactional properties that are used for web services and similar.

  Your application must not add properties to this folder.

- <mqema>
  This folder was used by WAS/SIB instead of the <mqext> folder.

MQRFH2 folder names are case-sensitive.

The following folders are reserved, in any mixture of lowercase or uppercase characters:
- Any folder prefixed by mq or wmq ; reserved for use by IBM MQ.
- Any folder prefixed by sib ; reserved for use by WAS/SIB.
- <Root> and <Body> folders; reserved but not used.

The following folders are not recognized as containing message properties:
- <psc>
  Used by IBM Integration Bus to convey publish/subscribe command messages to the broker.
- <pscr>
  Used by IBM Integration Bus to contain information from the broker, in response to publish/subscribe command messages.
- Any folder not defined by IBM, that is not marked with the content='properties' attribute.

Do not specify content='properties' on the <psc> or <pscr> folders. If you do so, these folders are treated as properties and IBM Integration Bus is likely to stop functioning as expected.

If your application is building messages with properties, in MQRFH2 headers to be recognized as an MQRFH2 header containing properties, the header must be in the list of headers that can be chained at the head of the message.

The MQRFH2 can be preceded by any number of MQH standard headers, or an MQCIH, an MQDLH, an MQIIH, an MQTM, an MQTMC2, or an MQXQH. A string or an MQCFH ends parsing because they cannot be chained.

It is possible for a message to contain multiple MQRFH2 headers all carrying message properties. Folders with the same name can coexist in different headers unless otherwise restricted, for example by WAS/SIB. The folders are treated as one logical folder, if they are all in significant headers.

While folders from the significant headers cannot be merged with those folders in nonsignificant headers, folders with the same name within the significant headers can be merged, removing any conflicting properties. Your applications must not depend on the layout of properties within their message.

MQRFH2 groups are parsed for properties in user-defined folders, that is, not the <wmq>, <jms>, <mcd>, <usr>, <mqext>, <sib>, <sib_usr>, <sib_context>, and <mqema> folders.

Groups in the IBM-defined properties folders, except for the <wmq> and <mq> folders, are parsed for properties.

An MQRFH2 folder cannot contain mixed content; a folder or group can contain either groups or properties, or a value, but not both.

A segment of a message, either the first or a subsequent segment, cannot contain IBM MQ -defined properties other than those properties in the message descriptor. Therefore putting a message containing such properties with either MQMF_SEGMENT or MQMF_SEGMENTATION_ALLOWED set causes the put to fail with MQRC_SEGMENTATION_NOT_ALLOWED.

However, message groups can contain IBM MQ -defined properties.

Generation of MQRFH2 headers:

If IBM MQ converts message properties to their MQRFH2 representation, it must add the MQRFH2 to the message. It adds the MQRFH2 either as a separate header, or merges it with an existing header.

Generation of new MQRFH2 headers by IBM MQ might disrupt existing headers in a message. Applications that parse a message buffer for headers must be aware that the number and position of headers in a buffer might change in some circumstances. IBM MQ attempts to minimize the impact of adding properties to a message by merging message properties into an existing MQRFH2 header, where it can. It also attempts to minimize the impact by inserting a generated MQRFH2 into a fixed position relative to other headers in the message buffer.

A generated MQRFH2 header is placed following the MQMD, and any number of MQXQH, MQRFH, and MQDLH headers, whatever order they are in. The generated MQRFH2 header is placed immediately before the first header that is not an MQMD, MQXQH, MQDLH, or MQRFH header.

Rules for merging generated MQRFH2

The following rules apply to merging a generated MQRFH2 with an existing MQRFH2. The generated MQRFH2 header is merged with an existing MQRFH2 header, if:

1. The existing MQRFH2 is in the same position IBM MQ would place a generated MQRFH2, or earlier in the header chain.
2. The CCSID of the generated properties is the same as the NameValueCCSID of the existing MQRFH2.

Otherwise, the generated header is placed separately in the buffer, in the position described before.

Rules for merging folders in an existing MQRFH2

If message properties are merged into an existing MQRFH2, then the existing MQRFH2 is scanned for folders that match the message properties, and merges them. If a matching folder does not exist a new folder is added to the end of the existing folders. If a matching folder does exist, the folder is searched. Any matching properties are overwritten. Any new ones are added at the end of the folder.

MQRFH2 folder restrictions:

Overview of folder restrictions in MQRFH2 headers

The MQRFH2 restrictions apply to the following folders:

- Element names in the <usr> folder must not begin with the prefix JMS ; such property names are reserved for use by JMS and are not valid for user-defined properties.

  Such an element name does not cause parsing of the MQRFH2 to fail, but is not accessible to the IBM MQ message property APIs.
- Element names in the <usr> folder must not be, in any mixture of lower or uppercase, NULL, TRUE, FALSE, NOT, AND, OR, BETWEEN, LIKE, IN, IS and ESCAPE. These names match SQL keywords and make parsing selectors harder, because <usr> is the default folder used when no folder is specified for a particular property in a selector.

  Such an element name does not cause parsing of the MQRFH2 to fail, but is not accessible to the IBM MQ message property APIs.
- Element names in any folder considered to contain message properties must not contain a period (.) (Unicode character U+002E), because this is used in property names to indicate the hierarchy.

  Such an element name does not cause parsing of the MQRFH2 to fail, but is not accessible to the IBM MQ message property APIs.

In general, MQRFH2 headers that contain valid XML-style data can be parsed by IBM MQ without failure, although certain elements of the MQRFH2 are not accessible through the IBM MQ message property APIs.

MQRFH2 element name conflicts:

Overview of conflicts within MQRFH2 element names.

Only one value can be attached to a message property. If an attempt to access a property leads to a conflict of values, one is chosen in preference over another.

The IBM MQ syntax for accessing MQRFH2 elements allows unique identification of an element, if a folder contains no elements with the same name. If a folder contains more than one element with the same name, the value of the property used is the one closest to the head of the message.

This applies if two or more folders of the same name are contained in different significant MQRFH2 headers within the same message.

A conflict can result when the MQGET call is processed after a non-message descriptor property has been set twice: both through an MQSETMP call and directly in the raw MQRFH2 header.

If this happens, the property associated with the message by an API call takes preference over one in the message data, that is, the one in the raw MQRFH2 header. If a conflict occurs, it is considered to come logically before the message data.

Mapping from property names to MQRFH2 folder and element names:

Overview of the differences between property names and element names in the MQRFH2 header.

When using any of the defined APIs that ultimately generate MQRFH2 headers, in order to specify message properties (for example, MQ JMS), the property name is not necessarily the element name in the MQRFH2 folder.

Therefore, a mapping occurs from the property name to the MQRFH2 element, and in the reverse way, taking into account both the folder name that contains the element, and the element name. Some examples from IBM MQ classes for JMS are already documented in Using IBM MQ classes for Java.

Property name	MQRFH2 folder name	MQRFH2 element name
`JMSDestination`	`jms`	`Dst`
`JMSType`	`mcd`	`Type, Set, Fmt`
`xxx` (user defined, where `xxx` does not begin with JMS)	`usr`	`xxx`

Therefore, when a JMS application accesses the `JMSDestination` property this maps to the `Dst` element in the `<jms>` folder.

When specifying properties as MQRFH2 elements, IBM MQ defines its elements as follows:

Property name	MQRFH2 folder name	MQRFH2 group name	MQRFH2 element name
<Property>	<usr>	n/a	<Property>
<folder>. <Property>	<folder>	n/a	<Property>
<folder>. <group>. <Property>	<folder>	<group>	<Property>

For example, when an IBM MQ application attempts to access the Property1 property, this maps to the Property1 element in the <usr> folder. The wmq.Property2 property maps to the Property2 property in the <wmq> folder.

If the property name contains more than one . character, the MQRFH2 element name used is the one following the final . character, and MQRFH2 groups are used to form a hierarchy; nested MQRFH2 groups are permitted.

The JMS header and provider-specific properties that are contained in an MQRFH2 in the <mcd>, <jms>, and <mqext> folders are accessed by an IBM MQ application using the short names defined in Using IBM MQ classes for Java .

JMS user-defined properties are accessed from the <usr> folder. An IBM MQ application can use the <usr> folder for its application properties if it is acceptable for the property to appear to JMS applications as one of its user-defined properties.

If it is not acceptable, choose another folder; the <wmq_usr> folder is provided as a standard location for such non-JMS properties.

Your applications can specify and use any MQRFH2 folder with a well-defined use, not documented in "Properties specified as MQRFH2 elements" on page 2932 if you note the following:

1. The folder might already be in use, or might be used in the future, by another application providing undefined access to properties contained inside it; see Property names for the suggested naming convention for property names.
2. The properties are not accessible to previous versions of the IBM MQ classes for JMS or XMS client that can only access the <usr> folder for user-defined properties
3. The folder must be marked with the attribute content with the value set to properties, for example, content='properties'.

   "MQSETMP - Set message property" on page 2774 automatically adds this attribute as required. This attribute must not be added to any of the IBM-defined folders, for example, <jms> and <usr>. Doing so, causes the message to be rejected by the IBM MQ classes for JMS client before Version 7.0. with a MessageFormatException.

Because the <usr> folder is the default location for properties of the <Property> syntax, an IBM MQ application and a JMS application to access the same user-defined property value using the same name.

Reserved folder names

There are several reserved folder names. You cannot use such names as your folder prefixes; for example, Root.Property1 does not access a valid property because Root is reserved. The following list contains reserved folder names:

- Root
- Body
- Properties
- Environment
- LocalEnvironment

- DestinationList
- ExceptionList
- InputBody
- InputRoot
- InputProperties
- InputLocalEnvironment
- InputDestinationList
- InputExceptionList
- OutputRoot
- OutputLocalEnvironment
- OutputDestinationList
- OutputExceptionList

Mapping property descriptor fields into MQRFH2 headers:

When a property is translated into an MQRFH2 element the following element attributes are used to specify the significant fields of the property descriptor: This describes how MQPD fields are translated to MQRFH2 element attributes.

Support

The Support property descriptor field is split into three element attributes
- The **sr** element attribute specifies values in the MQPD_REJECT_UNSUP_MASK bit mask.
- The **sa** element attribute specifies values in the MQPD_ACCEPT_UNSUP_MASK bit mask.
- The **sx** element attribute specifies values in the MQPD_ACCEPT_UNSUP_IF_XMIT_MASK bit mask.

These element attributes are only valid in the <mq> folder and are ignored if set on elements in the other folders containing properties.

Support value	MQRFH2 element attribute	MQRFH2 attribute value
MQPD_SUPPORT_OPTIONAL	sa	optional This is the default value.
MQPD_SUPPORT_REQUIRED	sr	required
MQPD_SUPPORT_REQUIRED_IF_LOCAL	sx	local

Context

Use the **context** element attribute to indicate the message context to which a property belongs. Use one value only. This element attribute is valid on a property in any folder containing properties.

Context value	MQRFH2 attribute value
MQPD_NO_CONTEXT	none This is the default value.
MQPD_USER _CONTEXT	user

CopyOptions

Use the **copy** element attribute to indicate messages into which a property should be copied. More than one value is acceptable; separate multiple values with a comma. For example **copy='reply'** and **copy='publish,report'** are both valid. This element attribute is valid on a property in any folder containing properties.

Note: In the attribute definition, single quotation marks or double quotation marks are valid use, for example **copy='reply'** or **copy="report"**

CopyOption value	MQRFH2 attribute value
MQPD_COPY_FORWARD	forward
MQPD_COPY_REPLY	reply
MQPD_COPY_REPORT	report
MQPD_COPY_PUBLISH	publish
MQPD_COPY_ALL	all Do not specify this with any other value. When used with another value, this takes precedence over any value except **none**.
MQPD_COPY_DEFAULT	default This is the default value. It is equivalent to specifying the three values MQCOPY_FORWARD, MQCOPY_REPORT and MQCOPY_PUBLISH. Do not specify this with any other value.
MQPD_COPY_NONE	none Do not specify this with any other value. When used with another value, this takes precedence.

Restrictions to the <mq> MQRFH2 folder

When a message is put on to a queue, it is searched for an <mq> folder so that the message can be processed according to its MQ-defined properties. To allow the efficient parsing of MQ-defined properties, the following restrictions apply to the folder:

- Only properties in the first significant <mq> folder in the message are acted upon by MQ; properties in any other <mq> folder in the message are ignored.
- If the folder is in UTF-8, only single-byte UTF-8 characters are allowed in the folder. A multi-byte character in the folder, can cause parsing to fail, and the message to be rejected.
- Do not include MQRFH2 groups in the <mq> folder. The presence of Unicode character U+003C in a property value will cause the message to be rejected.
- Do not use escape strings in the folder. An escape string is treated as the actual value of the element.
- Only Unicode character U+0020 is treated as white space within the folder. All other characters are treated as significant and can cause parsing of the folder to fail, and the message to be rejected.

If parsing of the <mq> folder fails, or if the folder does not observe these restrictions, the message is rejected with CompCode **MQCC_FAILED** and Reason **MQRC_RFH_RESTRICTED_FORMAT_ERR**.

MQRFH2 headers that are not valid:

At the time an MQPUT, MQPUT1, or MQGET call processes, a partial parsing of any MQRFH2 headers in the message can occur to check what folders are included, and to determine if the folders contain properties. Overview of MQRFH2 headers that are not valid.

If the partial parsing of the message cannot complete successfully because the structure is not valid, for example, the `StrucLength` field is too small, then:

- The MQPUT or MQPUT1 call fails with reason code MQRC_RFH_ERROR, if it can be determined that the application includes some IBM MQ Version 7 option, so that existing applications do not fail.
- The MQGET call returns successfully, and the MQRFH2 containing the error is returned in the buffer you provided.

If the partial parsing fails because it cannot be detected whether a particular folder contains properties or not, for example, the folder begins `<<jms`, so parsing fails before the folder name is determined, then:

- The MQPUT or MQPUT1 call fails with reason code MQRC_RFH_FORMAT_ERROR, if it can be determined that the application includes some IBM MQ Version 7 option, so that existing applications do not fail.
- The MQGET call returns successfully, and the MQRFH2 containing the error is returned in the buffer you provided.
- While internally within the queue manager, the message is not rejected due to the badly formatted folder, but the folder is always treated as if no properties were contained inside it.

A message can flow through the queue manager network with a folder containing such a syntax error, but never being parsed and detected, while one or more folders in the message are:

- Valid
- Successfully parsed
- Used in the processing of the message

Therefore, detection is not guaranteed.

If one of your applications uses “MQSETMP - Set message property” on page 2774, or MQINQMP to access a property, and in so doing this causes an MQRFH2 folder to be fully parsed, detecting an error such that parsing cannot complete, this is indicated by an appropriate return code to the API call. No properties in the folder are made available to the application.

If an attempt is made to fully parse an MQRFH2 folder and the parser finds unrecognized element attributes, or an unrecognized data type, parsing continues and complete successfully with no warnings being issued; this does not constitute a parsing error.

Code page conversion

This section describes codeset names and CCSIDs, national language, z/OS conversion, IBM i conversion, and Unicode conversion support.

Each national language section lists the following information:

- The native CCSIDs supported
- The code page conversions that are not supported

The following terms are used in the information:

HP-UX **-8**
Indicates for HP-UX that the CCSID is for the HP-UX defined codeset *roman8*.

AIX **AIX**
Indicates IBM MQ for AIX.

HP-UX **HP-UX**
Indicates IBM MQ for HP-UX.

Linux **Linux**
Indicates IBM MQ for Linux for Intel and IBM MQ for Linux for zSeries.

NSS Client **HP Integrity NonStop Server**
Indicates IBM MQ client for HP Integrity NonStop Server.

IBM i **OS/400**
Indicates IBM MQ for IBM i.

Solaris **Solaris**
Indicates IBM MQ for Solaris.

Windows **Windows**
Indicates IBM MQ for Windows.

z/OS **z/OS**
Indicates IBM MQ for z/OS.

The default for data conversion is for the conversion to be performed at the target (receiving) system.

If the source product supports the conversion a channel can be set up and data exchanged by setting the channel attribute `DataConversion` to YES at the source.

Note:

1. Conversion for IBM MQ MQI client information takes place in the server, so the server must support conversion from the client CCSID to the server CCSID.
2. The conversion might include support added by CSD/PTF to the latest version of IBM MQ. Check the content of the latest service level to see if you need to install a CSD/PTF to enable this conversion.
3. The IBM MQ queue manager CCSID must be Mixed or SBCS.
4. Some CCSIDs, for example 850 on AIX, that are not supported by the operating system can still be used by the application and also can be set as the IBM MQ queue manager CCSID. This is allowed for backward compatibility purpose only and the conversion will fail if the relevant conversion tables are not installed.

See Table 281 on page 2942 for a cross-reference between some of the CCSID numbers and some industry codeset names.

Related reference:

"National languages" on page 2943
This information contains languages supported by IBM MQ.

Codeset names and CCSIDs:

Codeset names and the corresponding CCSIDs for each codeset name.

z/OS IBM MQ for z/OS provides more conversion than is listed in the language specific tables. For a complete list of conversions, see Table 314 on page 2974.

Table 281. Codeset names and CCSIDs

Codeset names	CCSIDs
ISO 8859-1	819
ISO 8859-2	912
ISO 8859-3	913
ISO 8859-5	915
ISO 8859-6	1089
ISO 8859-7	813
ISO 8859-8	916
ISO 8859-9	920
ISO 8859-13	921
ISO 8859-15 (euro)	923
big5	950
eucJP	954 5050 33722
eucKR	970
eucTW	964
eucCN	1383
PCK	943
GBK	1386
koi8-r	878

National languages:

This information contains languages supported by IBM MQ.

The languages supported by IBM MQ are:

US English:

Details of CCSIDs and CCSID conversion for US English.

Table 282. Native CCSIDs for US English on supported platforms

Platform	Native CCSIDs
IBM i z/OS	37, 924, 1140
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	437, 850, 1252, 5348, 858
NSS Client Linux Solaris	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

IBM i

Code page:

37 Does not convert to code pages 923, 858

924 Does not convert to code pages 437, 858, 1051, 1140, 1252, 1275, 5348

1140 Does not convert to code pages 924, 1051, 1275

German:

Details of CCSIDs and CCSID conversion for German.

Table 283. Native CCSIDs for German on supported platforms

Platform	Native CCSIDs
IBM i z/OS	273, 924, 1141
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	437, 850, 858, 1252, 5348

Table 283. Native CCSIDs for German on supported platforms (continued)

Platform	Native CCSIDs
HP Integrity NonStop Server Linux Solaris	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

IBM i

Code page:

273 Does not convert to code pages 858, 923, 924, 1275

924 Does not convert to code pages 273, 437, 858, 1051, 1141, 1252, 1275, 5348

1141 Does not convert to code pages 924, 1051, 1275

Danish and Norwegian:

Details of CCSIDs and CCSID conversion for Danish and Norwegian.

Table 284. native CCSIDs for Danish and Norwegian on supported platforms

Platform	Native CCSIDs
IBM i z/OS	277, 924, 1142
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	850, 858, 865, 1252, 5348
HP Integrity NonStop Server Linux Solaris	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

IBM i

Code page:

277 Does not convert to code pages 858, 923, 924, 1275

924 Does not convert to code pages 277, 858, 865, 1051, 1142, 1252, 1275, 5348

1142 Does not convert to code pages 924, 865, 1051, 1275

AIX

Code page:

819 Does not convert to code page 865

HP-UX

Code page:

1051 Does not convert to code page 865

Windows

Code page:

865 Does not convert to code pages 1051, 1275

Finnish and Swedish:

Details of CCSIDs and CCSID conversion for Finnish and Swedish.

Table 285. Native CCSIDs for Finnish and Swedish on supported platforms

Platform	Native CCSIDs
 IBM i  z/OS	278, 924, 1143
 AIX	819, 923, 5348
 HP-UX	819, 923, 1051
 Windows	437, 850, 858, 865, 1252, 5348
 HP Integrity NonStop Server  Linux  Solaris	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

IBM i

Code page:

278 Does not convert to code pages 858, 923, 924, 1275

924 Does not convert to code pages 278, 437, 858, 865, 1051, 1143, 1252, 1275, 5348

1143 Does not convert to code pages 865, 924, 1051, 1275

AIX

AIX

Code page:

819 Does not convert to code page 865

850 Does not convert to code page 865

HP-UX

HP-UX

Code page:

1051 Does not convert to code page 865

Windows

Windows

Code page:

865 Does not convert to code pages 1051, 1275

Italian:

Details of CCSIDs and CCSID conversion for Italian.

Table 286. Native CCSIDs for Italian on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	280, 924, 1144
AIX AIX	819, 923, 5348
HP-UX HP-UX	819, 923, 1051
Windows Windows	437, 850, 858, 1252, 5348
NSS Client HP Integrity NonStop Server Linux Linux Solaris Solaris	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

IBM i

Code page:

280 Does not convert to code pages 858, 923, 924, 1275

924 Does not convert to code pages 280, 437, 858, 1051, 1144, 1252, 1275, 5348

1144 Does not convert to code pages 924, 1051, 1275

Spanish:

Details of CCSIDs and CCSID conversion for Spanish.

Table 287. Native CCSIDs for Spanish on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	284, 924, 1145
AIX AIX	819, 923, 5348
HP-UX HP-UX	819, 923, 1051
Windows Windows	437, 850, 858, 1252, 5348
NSS Client HP Integrity NonStop Server Linux Linux Solaris Solaris	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

IBM i

Code page:

284 Does not convert to code pages 858, 923, 924, 1275

924 Does not convert to code pages 284, 437, 858, 1051, 1145, 1252, 1275, 5348

1145 Does not convert to code pages 924, 1051, 1275

UK English /Gaelic:

Details of CCSIDs and CCSID conversion for UK English/Gaelic.

Table 288. Native CCSIDs for UK English / Gaelic on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	285, 924, 1146
AIX AIX	819, 923, 5348
HP-UX HP-UX	819, 923, 1051
Windows Windows	437, 850, 858, 1252, 5348
NSS Client HP Integrity NonStop Server Linux Linux Solaris Solaris	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i
IBM i

Code page:

285 Does not convert to code pages 858, 923, 924, 1275

924 Does not convert to code pages 285, 437, 858, 1051, 1146, 1252, 1275, 5348

1146 Does not convert to code pages 924, 1051, 1275

French:

Details of CCSIDs and CCSID conversion for French.

Table 289. Native CCSIDs for French on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	297, 924, 1147
AIX AIX	819, 923, 5348
HP-UX HP-UX	819, 923, 1051
Windows Windows	437, 850, 858, 1252, 5348

Table 289. Native CCSIDs for French on supported platforms (continued)

Platform	Native CCSIDs
NSS Client HP Integrity NonStop Server Linux Linux Solaris Solaris	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i
IBM i

Code page:

297 Does not convert to code pages 858, 923, 924, 1275, 5348

924 Does not convert to code pages 297, 437, 858, 1051, 1147, 1252, 1275, 5348

1147 Does not convert to code pages 924, 1051, 1275

Multilingual:

Details of CCSIDs and CCSID conversion for Multilingual.

Table 290. Native CCSIDs for multilingual conversion on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	500, 924, 1148
AIX AIX	819, 923, 5348
HP-UX HP-UX	819, 923, 1051
Windows Windows	437, 850, 858, 1252, 5348
NSS Client HP Integrity NonStop Server Linux Linux Solaris Solaris	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

IBM i

Code page:

500 Does not convert to code pages 858, 923

924 Does not convert to code pages 437, 858, 1051, 1148, 1252, 1275, 5348

1148 Does not convert to code pages 924, 1051, 1275

Portuguese:

Details of CCSIDs and CCSID conversion for Portuguese.

Table 291. Native CCSIDs for Portuguese on supported platforms

Platform	Native CCSIDs
IBM i	37, 500, 924, 1140
z/OS IBM i	500, 924, 1140
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	850, 858, 860, 1252, 5348
NSS Client HP Integrity NonStop Server Linux Linux Solaris Solaris	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

IBM i

Code page:

37 Does not convert to code pages 858, 923, 1275

500 Does not convert to code pages 858, 923, 1275

924 Does not convert to code pages 858, 860, 1051, 1140, 1252, 1275, 5348

1140 Does not convert to code pages 860, 924, 1051, 1275

HP-UX

HP-UX

Code page:

1051 Does not convert to code page 860

Windows

Windows

Code page:

860 Does not convert to code pages 1051, 1275

Icelandic:

Details of CCSIDs and CCSID conversion for Icelandic.

Table 292. Native CCSIDs for Icelandic on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	871, 924, 1149
AIX AIX	819, 923, 5348
HP-UX HP-UX	819, 923, 1051
Windows Windows	850, 858, 861, 1252, 5348
NSS Client HP Integrity NonStop Server Linux Linux Solaris Solaris	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i
IBM i

Code page:

871 Does not convert to code pages 858, 923, 924, 1275, 5348

924 Does not convert to code pages 858, 861, 871, 1051, 1149, 1252, 1275, 5348

1149 Does not convert to code pages 924, 1051, 1275

HP-UX
HP-UX

Code page:

1051 Does not convert to code page 861

Windows
Windows

Code page:

861 Does not convert to code pages 1051, 1275

Eastern European languages:

Details of CCSIDs and CCSID conversion for Eastern European Languages. The typical languages using these CCSIDs include Albanian, Croatian, Czech, Hungarian, Polish, Romanian, Serbian, Slovak, and Slovenian.

Table 293. Native CCSIDs for Eastern European languages on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	870, 1153
Windows Windows	852, 1250, 5346, 9044
AIX AIX HP-UX HP-UX NSS Client HP Integrity NonStop Server Linux Linux Solaris Solaris	912
Eastern European Apple client	1282
Romanian Apple client	1285
Croatian Apple client	1284

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

z/OS
z/OS

Code page:

870 Does not convert to code pages 1284, 1285

1153 Does not convert to code pages 1250, 1284, 1285

IBM i
IBM i

Code page:

870 Does not convert to code pages 1284, 1285, 5346, 9044

1153 Does not convert to code pages 1282, 1284, 1285, 5346, 9044

HP-UX Solaris Linux
HP-UX, Solaris Solaris, Linux

Code page:

912 Does not convert to code pages 1284, 1285

NSS Client

NSS Client HP Integrity NonStop Server

Code page:

912 Does not convert to code pages 1153, 1284, 1285, 9044

Windows

Windows

Code page:

852 Does not convert to code pages 1284, 1285

1250 Does not convert to code pages 1284, 1285

9044 Does not convert to code pages 912, 1282, 1284, 1285

Cyrillic:

Details of CCSIDs and CCSID conversion for Cyrillic. The typical languages using these CCSIDs include Belarussion, Bulgarian, Macedonian, Russian, and Serbian.

Table 294. Native CCSIDs for Cyrillic on supported platforms

Platform	Native CCSIDs
z/OS z/OS	1025
IBM i IBM i	880, 1025
Windows Windows	855, 866, 1131, 1251, 5347
Solaris Solaris	878, 915
AIX AIX NSS Client HP Integrity NonStop Server HP-UX HP-UX Linux Linux	915
Apple client	1283

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

IBM i

Code page:

880 Does not convert to code pages 855, 866, 878, 1131, 5347

1025 Does not convert to code pages 878, 5347

Windows

Windows

Code page:

855 Does not convert to code page 1131

866 Does not convert to code page 1131

1131 Does not convert to code pages 855, 866, 880, 1283

Estonian:

Details of CCSIDs and CCSID conversion for Estonian.

Table 295. Native CCSIDs for Estonian on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	1122, 1157
Windows Windows	902, 922, 1257, 5353, 9449
AIX AIX HP-UX HP-UX Linux Linux Solaris Solaris	902, 922
NSS Client HP Integrity NonStop Server	922

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS
z/OS

Code page:

1122 Does not convert to code pages 902, 1157, 9449

1157 Does not convert to code pages 922, 1122, 1257, 9449

IBM i
IBM i

Code page:

1122 Does not convert to code pages 902, 5353, 9449

1157 Does not convert to code pages 922, 5353, 9449

HP-UX Solaris Linux
HP-UX, Solaris, Linux

Code page:

902 Does not convert to code pages 922, 1122, 9449

922 Does not convert to code pages 902, 1157, 9449

Windows
Windows

Code page:

5353 Does not convert to code page 9449

9449 Does not convert to code pages 902, 922, 1122, 1157, 1257, 5353

902 Does not convert to code pages 922, 1122, 9449

NSS Client
HP Integrity NonStop Server

Code page:

922 Does not convert to code pages 902, 1157, 9449

Latvian and Lithuanian:

Details of CCSIDs and CCSID conversion for Latvian and Lithuanian.

Table 296. Native CCSIDs for Latvian and Lithuanian on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	1112, 1156
Windows Windows	901, 921, 1257, 5353, 9449
AIX AIX HP-UX HP-UX Linux Linux Solaris Solaris	901, 921
NSS Client HP Integrity NonStop Server,	921

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS
z/OS

Code page:

1112 Does not convert to code pages 901, 1156, 9449

1156 Does not convert to code pages 901, 1156, 9449

IBM i

IBM i

Code page:

1112 Does not convert to code page 5353

1153 Does not convert to code pages 921, 5353, 9449

HP-UX, Solaris, Linux

Code page:

902 Does not convert to code pages 921, 1112, 1257, 9449

921 Does not convert to code pages 901, 1156, 9449

Windows

Code page:

901 Does not convert to code pages 921, 1112, 1257, 9449

5355 Does not convert to code page 9449

9449 Does not convert to code pages 901, 921, 1112, 1156, 1257

HP Integrity NonStop Server

Code page:

921 Does not convert to code pages 901, 1156, 9449

Ukrainian:

Details of CCSIDs and CCSID conversion for Ukrainian.

Table 297. Native CCSIDs for Ukranian on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	1123
Windows Windows	1124, 1125, 1251, 5347
AIX AIX NSS Client HP Integrity NonStop Server HP-UX HP-UX Linux Linux Solaris Solaris	1124

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

IBM i
IBM i

Code page:

1123 Does not convert to code page 5347

HP-UX
HP-UX

Code page:

1124 Does not convert to code page 5347

Windows
Windows

Code page:

1125 Does not convert to code page 1123

Greek:

Details of CCSIDs and CCSID conversion for Greek.

Table 298. Native CCSIDs for Greek on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	875
HP-UX HP-UX	813 (see note)
Windows Windows	869, 1253, 5349
AIX AIX NSS Client HP Integrity NonStop Server Linux Linux NCR Solaris Solaris	813
Apple client	1280
DOS client	737

Note: HP-UX Only the ISO codeset is supported on HP-UX. The HP-UX proprietary greek8 codeset has no registered CCSID and is not supported.

All non-client platforms support conversion between their native CCSIDs, the native CCSIDs of the other platforms with the following exceptions.

IBM i
IBM i

Code page:

875 Does not convert to code page 5349

Windows
Windows

Code page:

1253 Does not convert to code page 737

5349 Does not convert to code page 737

Turkish:

Details of CCSIDs and CCSID conversion for Turkish.

Table 299. Native CCSIDs for Turkish on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	1026
HP-UX HP-UX	920 (see note)
Windows Windows	857, 1254, 5350
AIX AIX NSS Client HP Integrity NonStop Server Linux Linux Solaris Solaris	920
Apple client	1281

Note: HP-UX Only the ISO codeset is supported on HP-UX. The HP-UX proprietary turkish8 codeset has no registered CCSID and is not supported.

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i
IBM i

Code page:

1026 Does not convert to code page 5350

Hebrew:

Details of CCSIDs and CCSID conversion for Hebrew.

Table 300. Native CCSIDs for Hebrew on supported platforms

Platform	Native CCSIDs
z/OS z/OS	424, 803, 4899, 12712
IBM i IBM i	424
AIX AIX	916, 9048
HP-UX HP-UX	916 (see note)
Windows Windows	1255, 5351
NSS Client HP Integrity NonStop Server Linux Linux Solaris Solaris	916

Note: HP-UX Only the ISO codeset is supported on HP-UX. The HP-UX proprietary greek8 codeset has no registered CCSID and is not supported.

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS
z/OS

Code page:

424 Does not convert to code pages 867, 4899, 9048, 12712

803 Does not convert to code pages 867, 4899, 5351, 9048, 12712

4899 Does not convert to code pages 424, 803, 856, 862, 916, 1255

12712 Does not convert to code pages 424, 803, 856, 916, 1255

IBM i
IBM i

Code page:

424 Does not convert to code pages 803, 867, 4899, 5351, 9048, 12712

Code page 424 also converts to and from CCSID 4952, which is a variant of 856.

AIX
AIX

Code page:

916 Does not convert to code pages 867, 4899, 9048, 12712

9048 Does not convert to code pages 424, 803, 856, 862, 916, 1255

Windows

Windows

Code page:

1255 Does not convert to code pages 867, 4899, 9048, 12712

5351 Does not convert to code page 803

Arabic:

Details of CCSIDs and CCSID conversion for Arabic

Table 301. Native CCSIDs for Arabic on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	420
AIX AIX	1046, 1089
HP-UX HP-UX	1089 (see note)
Windows Windows	720, 864, 1256, 5352
NSS Client HP Integrity NonStop Server Linux Linux Solaris Solaris	1089

Note: HP-UX Only the ISO codeset is supported on HP-UX. The HP-UX proprietary arabic8 codeset has no registered CCSID and is not supported.

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

IBM i

IBM i

Code page:

420 Does not convert to code page 5352

HP-UX Solaris Linux NSS Client

HP-UX, Solaris, Linux, HP Integrity NonStop Server, Tru64

Code page:

1089 Does not convert to code page 720

Windows

Windows

Code page:

720 Does not convert to code pages 1089, 5352

5352 Does not convert to code page 720

Farsi:

Details of CCSIDs and CCSID conversion for Farsi.

Table 302. Native CCSIDs for Farsi on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	1097
AIX AIX NSS Client HP Integrity NonStop Server HP-UX HP-UX Linux Linux Solaris Solaris Windows Windows	1098 (see note)

Note: The native CCSID for these platforms has not been standardized and might change.

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms.

Urdu:

Details of CCSIDs and CCSID conversion for Urdu.

Table 303. Native CCSIDs for Urdu on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	918
Windows Windows	868
AIX AIX NSS Client HP Integrity NonStop Server HP-UX HP-UX Linux Linux Solaris Solaris	1006

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

IBM i

IBM i

Code page:

918 Does not convert to code page 1006

Thai:

Details of CCSIDs and CCSID conversion for Thai.

Table 304. Native CCSIDs for Thai on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	838
AIX AIX NSS Client HP Integrity NonStop Server HP-UX HP-UX Linux Linux Solaris Solaris Windows Windows	874 (see note)

Note: The native CCSID for these platforms has not been standardized and might change.

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms.

Lao:

Details of CCSIDs and CCSID conversion for Lao.

Table 305. Native CCSIDs for Lao on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	1132

Table 305. Native CCSIDs for Lao on supported platforms (continued)

Platform	Native CCSIDs
AIX AIX NSS Client HP Integrity NonStop Server HP-UX HP-UX Linux Linux Solaris Solaris Windows Windows	1133

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms.

Vietnamese:

Details of CCSIDs and CCSID conversion for Vietnamese.

Table 306. Native CCSIDs for Vietnamese on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	1130
Windows Windows	1258, 5354
AIX AIX NSS Client HP Integrity NonStop Server HP-UX HP-UX Linux Linux Solaris Solaris	1129

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

IBM i

IBM i

Code page:

1130 Does not convert to code pages 1129, 5354

Japanese Latin SBCS:

Details of CCSIDs and CCSID conversion for Japanese Latin SBCS.

Table 307. Native CCSIDs for Japanese Latin SBCS on supported platforms

Platform	Native CCSIDs
IBM i z/OS	1027
AIX	932, 5050, 33722 (see Note 1)
Windows	932, 943 (see Note 2)
NSS Client Linux Solaris	943, 5050
HP-UX	Not known

Note:

1. AIX 5050 and 33722 are CCSIDs related to base code page 954 on AIX. The CCSID reported by the operating system is 33722.
2. Windows Windows NT uses code page 932 but this is best represented by the CCSID of 943. However, not all platforms of IBM MQ support this CCSID.

   On IBM MQ for Windows CCSID 932 is used to represent code page 932, but a change to file `../conv/table/ccsid.tbl` can be made which changes the CCSID used to 943.

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS
z/OS

Code page:

1027 Does not convert to code pages 932, 942, 943, 954, 5050, 33722

IBM i
IBM i

Code page:

1027 Does not convert to code page 932

AIX
AIX

Code page:

932 Does not convert to code page 1027

5050 Does not convert to code page 1027

33722 Does not convert to code page 1027

Linux

Linux

Code page:

943 Does not convert to code page 1027

5050 Does not convert to code page 1027

Solaris

Solaris

Code page:

943 Does not convert to code page 1027

5050 Does not convert to code page 1027

NSS Client

HP Integrity NonStop Server

Code page:

943 Does not convert to code page 1027

5050 Does not convert to code page 1027

Japanese Katakana SBCS:

Details of CCSIDs and CCSID conversion for Japanese Katakana SBCS.

Table 308. Native CCSIDs for Japanese Katakana SBCS on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	290
HP-UX HP-UX	897
AIX AIX	932, 5050, 33722 (see Note 1)
Windows Windows	932, 943 (see Note 2)
NSS Client HP Integrity NonStop Server Linux Linux Solaris Solaris	943, 5050

Note:

1. AIX 5050 and 33722 are CCSIDs related to base code page 954 on AIX. The CCSID reported by the operating system is 33722.

2. Windows Windows NT uses code page 932 but this is best represented by the CCSID of 943. However, not all platforms of IBM MQ support this CCSID.

   On IBM MQ for Windows CCSID 932 is used to represent code page 932, but a change to file `../conv/table/ccsid.tbl` can be made which changes the CCSID used to 943.
3. In addition to the previous conversions, the IBM MQ products on AIX, HP-UX, Solaris, Linux and Tru64 support conversion from CCSID 897 to CCSIDs 37, 273, 277, 278, 280, 284, 285, 290, 297, 437, 500, 819, 850, 1027, and 1252.

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS
z/OS

Code page:

290 Does not convert to code pages 932, 943, 954, 5050, 33722

IBM i
IBM i

Code page:

290 Does not convert to code page 932

AIX
AIX

Code page:

932 Does not convert to code pages 290, 897

5050 Does not convert to code pages 290, 897

33722 Does not convert to code pages 290, 897

HP-UX
HP-UX

Code page:

897 Does not convert to code pages 932, 943, 954, 5050, 33722

Linux
Linux

Code page:

943 Does not convert to code pages 290, 897

5050 Does not convert to code pages 290, 897

Solaris
Solaris

Code page:

943 Does not convert to code pages 290, 897

5050 Does not convert to code pages 290, 897

NSS Client

HP Integrity NonStop Server

Code page:

943 Does not convert to code pages 290, 897

5050 Does not convert to code pages 290, 897

Japanese Kanji/ Latin Mixed:

Details of CCSIDs and CCSID conversion for Japanese Kanji/Latin Mixed.

Table 309. Native CCSIDs for Japanese Kanji/ Latin Mixed on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	1399, 5035 (see Note 1)
AIX AIX	932, 5050, 33722 (see Note 2)
HP-UX HP-UX	932, 954, 5039 (see Note 3)
Windows Windows	932, 943 (see Note 4)
NSS Client HP Integrity NonStop Server Linux Linux Solaris Solaris	943, 5050

Note:

1. IBM i z/OS 5035 is a CCSID related to code page 939
2. AIX 5050 and 33722 are CCSIDs related to base code page 954 on AIX. The CCSID reported by the operating system is 33722.
3. HP-UX Code sets `japan15` and SJIS on HP-UX are represented by CCSID 932. These have a few DBCS characters having different representations in SJIS so 932 may be converted incorrectly if the conversion is not performed on an HP-UX system. IBM MQ for HP-UX supports 5039, the correct CCSID for HP SJIS. A change to file `/var/mqm/conv/ccsid.tbl` can be made to change the CCSID used from 932 to 5039.
4. Windows Windows NT uses code page 932 but this is best represented by the CCSID of 943. However, not all platforms of IBM MQ support this CCSID.

   On IBM MQ for Windows CCSID 932 is used to represent code page 932, but a change to file `../conv/table/ccsid.tbl` can be made which changes the CCSID used to 943.

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

z/OS

Code page:

1399 Does not convert to code pages 954, 5035, 5050, 33722

5035 Does not convert to code pages 954, 1399, 5050, 33722

IBM i

IBM i

Code page:

1399 Does not convert to code page 5039

5035 Does not convert to code page 5039

HP-UX

HP-UX

Code page:

932 Does not convert to code pages 942, 943, 1399

954 Does not convert to code pages 942, 943, 1399

5039 Does not convert to code pages 942, 943, 1399

NSS Client

HP Integrity NonStop Server

Code page:

943 Does not convert to code page 1399

5050 Does not convert to code page 1399

Japanese Kanji/ Katakana Mixed:

Details of CCSIDs and CCSID conversion for Japanese Kanji/Katakana Mixed.

Table 310. Native CCSIDs for Japanese Kanji/ Katakana Mixed on supported platforms

Platform	Native CCSIDs
z/OS z/OS	1390, 5026 (see Note 1)
IBM i IBM i	5026 (see Note 1)
AIX AIX	932, 5050, 33722 (see Note 2)
HP-UX HP-UX	932, 954, 5039 (see Note 3)
Windows Windows	932, 943 (see Note 4)
NSS Client HP Integrity NonStop Server Linux Linux Solaris Solaris	943, 5050

Note:

1. z/OS IBM i CCSID 1390 does not accept lower case characters. 5026 is a CCSID related to code page 930. CCSID 5026 is the CCSID reported on IBM i when the Japanese Katakana (DBCS) feature is selected.
2. AIX 5050 and 33722 are CCSIDs related to base code page 954 on AIX. The CCSID reported by the operating system is 33722.
3. HP-UX Code sets `japan15` and SJIS on HP-UX are represented by CCSID 932. These have a few DBCS characters having different representations in SJIS so 932 may be converted incorrectly if the conversion is not performed on an HP-UX system. IBM MQ for HP-UX supports 5039, the correct CCSID for HP SJIS. A change to file `/var/mqm/conv/ccsid.tbl` can be made to change the CCSID used from 932 to 5039.
4. Windows Windows NT uses code page 932 but this is best represented by the CCSID of 943. However, not all platforms of IBM MQ support this CCSID.

   On IBM MQ for Windows, CCSID 932 is used to represent code page 932, but a change to file `../conv/table/ccsid.tbl` can be made that changes the CCSID used to 943.

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS
z/OS

Code page:

1390 Does not convert to code pages 954, 5026, 5050, 33722

Does not accept lower case characters.

5026 Does not convert to code pages 954, 1390, 5050, 33722

IBM i
IBM i

Code page:

5026 Does not convert to code pages 1390, 5039

HP-UX
HP-UX

Code page:

932 Does not convert to code pages 942, 943, 1390

954 Does not convert to code pages 942, 943, 1390

5039 Does not convert to code pages 942, 943, 1390

NSS Client
HP Integrity NonStop Server

Code page:

943 Does not convert to code page 1390

5050 Does not convert to code page 1390

Korean:

Details of CCSIDs and CCSID conversion for Korean.

Table 311. Native CCSIDs for Korean on supported platforms

Platform	Native CCSIDs
IBM i z/OS z/OS	933, 1364
AIX NSS Client HP Integrity NonStop Server HP-UX HP-UX Linux Solaris	970
Windows	949, 1363

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS
z/OS

Code page:

933 Does not convert to code page 970

1364 Does not convert to code page 970

HP-UX
HP-UX

Code page:

970 Does not convert to code pages 949, 1363, 1364

Simplified Chinese:

Details of CCSIDs and CCSID conversion for Simplified Chinese.

Table 312. Native CCSIDs for Simplified Chinese on supported platforms

Platform	Native CCSIDs
z/OS z/OS	935, 1388
IBM i IBM i	935, 1388
AIX AIX	1383, 1386
HP-UX HP-UX	1381 (see Note 1)
Windows Windows	1381, 1386(see Note 2)
NSS Client HP Integrity NonStop Server Linux Linux Solaris Solaris	1383

Note:

1. HP-UX Code sets `prc15` and `hp15CN` on HP-UX are represented by CCSID 1381.
2. Windows Windows uses code page 936 but this is best represented by the CCSID of 1386. However, not all platforms of IBM MQ support this CCSID.

   On IBM MQ for Windows CCSID 1381 is used to represent code page 936, but a change to file `../conv/table/ccsid.tbl` can be made which changes the CCSID used to 1386.
3. IBM MQ supports the Chinese GB18030 standard.

   z/OS Linux Windows Solaris On z/OS, Linux, Windows, and Solaris, conversion support is provided between Unicode (UTF-8 and UTF-16) and CCSID 1388 (EBCDIC with GB18030 extensions), Unicode (UTF-8 and UTF-16) and CCSID 5488 (GB18030), and between CCSID 1388 and CCSID 5488.

   Note:

   IBM i On IBM i, support is provided by the operating system for conversion between Unicode (UTF-8 and UTF-16) and CCSID 1388 (EBCDIC with GB18030 extensions).

   HP-UX On HP-UX there is currently no support available on the HP11 operating system for GB18030. On HP11i, patch PHCO_26456 provides conversion support between GB18030 (CCSID 5488) and Unicode. Support is not provided for the conversion between GB18030 and 1388 (EBCDIC).

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS
z/OS

Code page:

935 Does not convert to code page 1383

1388 Does not convert to code page 1383

HP-UX

HP-UX

Code page:

1381 Does not convert to code pages 1383, 1386, 1388

Traditional Chinese:

Details of CCSIDs and CCSID conversion for Traditional Chinese.

Table 313. Native CCSIDs for Traditional Chinese on supported platforms

Platform	Native CCSIDs
IBM i IBM i z/OS z/OS	937
HP-UX HP-UX	938, 950, 964 (see Note)
Windows Windows	950
AIX AIX NSS Client HP Integrity NonStop Server Linux Linux Solaris Solaris	950, 964

Note: HP-UX Code set `roc15` on HP-UX is represented by CCSID 938.

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

z/OS

Code page:

937 Does not convert to code page 964

1388 Does not convert to code page 1383

HP-UX

HP-UX

Code page:

938 Does not convert to code page 948

950 Does not convert to code page 948

964 Does not convert to code page 948

Linux Solaris

Linux and Solaris

Code page:

964 Does not convert to code page 938

z/OS conversion support: z/OS

A list of supported CCSID conversions.

Table 314. IBM MQ for z/OS CCSID conversion support

CCSID	Converts to and from CCSIDS
37	256, 273, 275, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 720, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-866, 869-871, 874-875, 880, 897, 903-905, 912, 914-916, 920-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1097, 1100, 1112, 1114-1115, 1122, 1124, 1126, 1130-1132, 1137, 1140-1149, 1200, 1208, 1250-1255, 1257-1258, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5210-5211, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 16804, 17248, 17584, 25473, 25479, 25480, 25617, 25619, 25664, 28709
256	37, 273, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 737, 775, 819, 833, 836, 838, 850, 852, 857, 860-866, 869-871, 875, 880, 905, 1025-1027, 1112, 1122, 1200, 1208, 1251-1252, 1275, 4386, 4929, 4932, 4934, 4946, 4948, 4953, 4960, 4971, 5123, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 13121, 13488, 16804, 17248, 17584, 28709
259	437, 808, 850-852, 855-858, 860-865, 867, 869, 872, 874, 899, 901-902, 915, 1098, 1161-1162, 1200, 1208, 1250-1258, 4946, 4948, 4951-4953, 4960, 4970, 5346, 5348, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584
273	37, 256, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1250, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951-4953, 4960, 4970-4971, 5012, 5123, 5346, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
274	500, 1047
275	37, 437, 500, 819, 850, 1047, 1200, 1208, 1252, 4946, 5348, 8229, 13488, 17584, 28709
277	37, 256, 273, 278, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
278	37, 256, 273, 277, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
280	37, 256, 273, 277-278, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709

Table 314. IBM MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
281	1047
282	500, 1047, 1200, 1208, 13488, 17584
284	37, 256, 273, 277-278, 280, 285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
285	37, 256, 273, 277-278, 280, 284, 290, 297, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
290	37, 256, 273, 277-278, 280, 284-285, 297, 367, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 860-865, 870-871, 895-897, 1009, 1025-1027, 1040-1043, 1047, 1088, 1112, 1122, 1139, 1200, 1208, 1252, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4960, 4992, 5123, 8229, 8482, 9025, 9044, 9049, 9056, 13121, 13488, 17248, 17584, 25473, 25617, 25619, 25664, 28709
293	1200, 1208, 13488, 17584
297	37, 256, 273, 277-278, 280, 284-285, 290, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
300	301, 941, 1200, 1208, 1351, 4396, 8492, 13488, 16684, 17584
301	300, 941, 1200, 1208, 1351, 4396, 8492, 13488, 16684, 17584
367	37, 256, 273, 277-278, 280, 284, 290, 297, 500, 819, 833, 836, 850, 871, 875, 1009, 1026-1027, 1041, 1088, 1115, 1126, 1200, 1208, 4386, 4929, 4932, 4946, 4971, 5123, 5211, 8229, 8482, 9025, 13121, 13488, 17584, 25617, 25664, 28709
420	37, 256, 424, 437, 500, 720, 737, 775, 819, 850, 852, 857, 860-865, 1008, 1046, 1089, 1098, 1112, 1122, 1127, 1200, 1208, 1252, 1256, 4946, 4948, 4953, 4960, 5104, 5142, 5352, 8229, 8612, 9044, 9049, 9056, 9238, 13488, 16804, 17248, 17584, 28709
423	37, 256, 273, 277-278, 280, 284-285, 297, 437, 500, 737, 775, 813, 819, 838, 850-852, 857, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1009, 1025-1027, 1041-1043, 1112, 1122, 1200, 1208, 1252-1253, 1280, 4909, 4934, 4946, 4948, 4953, 4960, 4970-4971, 5012, 5123, 8229, 9030, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 28709
424	37, 256, 420, 437, 500, 737, 775, 803, 819, 836, 850, 852, 856-857, 860-865, 916, 1112, 1122, 1200, 1208, 1252, 1255, 4932, 4946, 4948, 4952-4953, 4960, 5012, 5351, 8229, 8612, 9044, 9049, 9056, 13488, 16804, 17248, 17584, 28709
437	37, 256, 259, 273, 275, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-863, 865-866, 869-871, 874-875, 880, 897, 903, 905, 912, 914-916, 920-924, 1025-1027, 1040-1043, 1047, 1051, 1097, 1098, 1114-1115, 1126, 1140-1149, 1200, 1208, 1252, 1257, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5210-5211, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 28709

Table 314. IBM MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
500	37, 256, 273-275, 277-278, 280, 282, 284-285, 290, 297, 367, 420, 423-424, 437, 737, 775, 813, 819, 833, 836, 838, 850-852, 855-858, 860-866, 869-871, 874-875, 880, 891, 895, 897, 903-905, 912, 914-916, 920-924, 1004, 1009-1021, 1023, 1025-1027, 1040-1043, 1046-1047, 1051, 1088-1089, 1097, 1100-1107, 1112, 1114-1115, 1122, 1124-1126, 1129-1133, 1137, 1140-1149, 1200, 1208, 1250-1258, 1275, 1280-1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951-4953, 4960, 4970-4971, 5012, 5123, 5142, 5210-5211, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 9238, 13121, 13488, 16804, 17248, 17584, 25473, 25479, 25480, 25617, 25619, 25664, 28709
720	37, 420, 864, 1200, 1208, 1256, 4960, 8229, 8612, 9056, 13488, 16804, 17248, 17584, 28709
737	37, 256, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 833, 836, 838, 850, 869-871, 875, 880, 905, 1025-1027, 1097, 1200, 1208, 1252-1253, 1280, 4386, 4909, 4929, 4932, 4934, 4946, 4971, 5123, 8229, 8482, 8612, 9025, 9030, 9061, 13121, 13488, 16804, 17584, 28709
775	37, 256, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 833, 836, 838, 850, 870-871, 875, 880, 905, 1025-1027, 1097, 1112, 1122, 1200, 1208, 1252, 1257, 4386, 4929, 4932, 4934, 4946, 4971, 5123, 8229, 8482, 8612, 9025, 9030, 13121, 13488, 16804, 17584, 28709
803	424, 819, 850, 856, 862, 916, 1200, 1208, 1252, 1255, 4946, 4952, 5012, 13488, 17584
806	1200, 1208, 13488, 17584
808	259, 858-859, 872, 923-924, 1140, 1148, 1153-1154, 1200, 1208, 5347, 5348, 13488, 17584
813	37, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252-1253, 1280, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5349, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
819	37, 256, 273, 275, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 803, 813, 833, 836, 838, 850, 852, 855, 857-858, 860-861, 863-866, 869-871, 874-875, 880, 897, 903, 905, 912, 914-916, 920-924, 1004, 1025-1027, 1041-1043, 1047, 1051, 1088-1089, 1097, 1098, 1112, 1114, 1122-1123, 1126, 1130, 1132, 1137, 1140-1149, 1200, 1208, 1250-1255, 1257-1258, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5210, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 16804, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
833	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 437, 500, 737, 775, 819, 836, 850, 852, 855, 857, 860-865, 870-871, 891, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1126, 1200, 1208, 1252, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4960, 5123, 8229, 8482, 9025, 9044, 9049, 9056, 13121, 13488, 17248, 17584, 25617, 25619, 25664, 28709
834	926, 951, 1200, 1208, 1362, 4930, 9026, 13488, 17584
835	927, 947, 1200, 1208, 4931, 9027, 13488, 17584, 21427
836	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 424, 437, 500, 737, 775, 819, 833, 850, 852, 855, 857, 870-871, 875, 903, 1009, 1025-1027, 1040-1043, 1088, 1112, 1114-1115, 1122, 1200, 1208, 1252, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4971, 5123, 5210-5211, 8229, 8482, 9025, 9044, 9049, 13121, 13488, 17584, 25479, 25617, 25619, 25664, 28709
837	928, 1200, 1208, 1380, 1385, 4933, 13488, 17584

Table 314. IBM MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
838	37, 256, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 775, 813, 819, 850, 852, 857, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1112, 1122, 1200, 1208, 1252, 4909, 4934, 4946, 4948, 4953, 4960, 4970-4971, 5012, 5123, 8229, 9030, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 28709
848	924, 1148, 1158, 1200, 1208, 5347, 13488, 17584
849	924, 1148, 1154, 1200, 1208, 5347, 13488, 17584
850	37, 256, 259, 273, 275, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 737, 775, 803, 813, 819, 833, 836, 838, 852, 855-858, 860-866, 869-871, 874-875, 880, 897, 903, 905, 912, 914-916, 920-924, 1004, 1025-1027, 1040-1043, 1047, 1051, 1088-1089, 1097, 1098, 1100, 1112, 1114, 1122, 1126, 1130, 1132, 1140-1149, 1200, 1208, 1250-1257, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951-4953, 4960, 4970-4971, 5012, 5123, 5210, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 16804, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
851	259, 423, 500, 875, 1200, 1208, 4971, 13488, 17584
852	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 836, 838, 850, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1047, 1088, 1097, 1200, 1208, 1250, 1252, 1282, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5346, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 25664, 28709
855	37, 259, 273, 277-278, 280, 284-285, 290, 297, 437, 500, 819, 833, 836, 850, 852, 857, 866, 870-871, 878, 880, 912, 915, 1025-1027, 1040-1043, 1088, 1200, 1208, 1250-1252, 1283, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 5123, 5346, 5347, 8229, 8482, 9025, 9044, 9049, 13121, 13488, 17584, 25617, 25619, 25664, 28709
856	259, 273, 424, 500, 803, 850, 862, 916, 1200, 1208, 1255, 4946, 4952, 5012, 5351, 13488, 17584
857	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1088, 1097, 1200, 1208, 1252, 1254, 1281, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5350, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 25664, 28709
858	37, 259, 273, 277-278, 280, 284-285, 297, 437, 500, 808, 819, 850, 860-861, 865, 871-872, 901-902, 923-924, 1047, 1051, 1140-1149, 1153-1157, 1160-1162, 1164, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
859	808, 872, 901-902, 1153-1157, 1160-1162, 1164, 1200, 1208, 13488, 17584
860	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 838, 850, 852, 857-858, 861, 863, 865, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 923-924, 1025-1027, 1041-1043, 1097, 1140, 1145-1146, 1148, 1200, 1208, 1252, 4386, 4909, 4929, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 28709
861	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 838, 850, 852, 857-858, 860, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 923-924, 1025-1027, 1041-1043, 1097, 1148, 1149, 1200, 1208, 1252, 4386, 4909, 4929, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 28709

Table 314. IBM MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
862	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 803, 833, 838, 850, 856, 870-871, 875, 880, 905, 916, 1025-1027, 1097, 1200, 1208, 1252, 1255, 4386, 4929, 4934, 4946, 4952, 4971, 5012, 5123, 5351, 8229, 8482, 8612, 9025, 9030, 12712, 13121, 13488, 16804, 17584, 28709
863	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 838, 850, 852, 857, 860-861, 865, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1041-1043, 1051, 1097, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 28709
864	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 720, 819, 833, 838, 850, 870-871, 875, 880, 905, 918, 1008, 1025-1027, 1046, 1089, 1097, 1127, 1200, 1208, 1252, 1256, 4386, 4929, 4934, 4946, 4960, 4971, 5104, 5123, 5142, 5352, 8229, 8482, 8612, 9025, 9030, 9056, 9238, 13121, 13488, 16804, 17248, 17584, 28709
865	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 819, 833, 838, 850, 858, 860, 863, 870-871, 875, 880, 905, 923-924, 1025-1027, 1097, 1142-1143, 1148, 1200, 1208, 1252, 4386, 4929, 4934, 4946, 4971, 5123, 5348, 8229, 8482, 8612, 9025, 9030, 13121, 13488, 16804, 17584, 28709
866	37, 256, 437, 500, 819, 850, 855, 870, 878, 880, 915, 1025, 1200, 1208, 1251-1252, 1283, 4946, 4951, 5347, 8229, 13488, 17584, 28709
867	259, 1153-1155, 1160, 1200, 1208, 4899, 5351, 9048, 12712, 13488, 17584
868	918, 1006, 1200, 1208, 13488, 17584
869	37, 256, 259, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 813, 819, 838, 850, 852, 857, 860-861, 863, 870-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252-1254, 1280, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5349, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
870	37, 256, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-866, 869, 871, 874-875, 880, 897, 903, 912, 915-916, 920, 1009, 1025-1027, 1040-1043, 1047, 1088, 1112, 1122, 1200, 1208, 1250, 1252, 1282, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5346, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
871	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869, 870, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
872	259, 808, 858-859, 923-924, 1140-1149, 1153-1155, 1200, 1208, 5347, 5348, 13488, 17584
874	37, 259, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
875	37, 256, 273, 277-278, 280, 284-285, 297, 367, 423, 437, 500, 737, 775, 813, 819, 836, 838, 850-852, 857, 860-865, 869-871, 874, 880, 897, 903, 912, 916, 920, 1009, 1025-1027, 1041-1043, 1047, 1088, 1112, 1122, 1200, 1208, 1252-1253, 1280, 4909, 4932, 4934, 4946, 4948, 4953, 4960, 4970-4971, 5012, 5123, 5349, 8229, 9030, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
878	855, 866, 880, 915, 1025, 1131, 1200, 1208, 1251, 1283, 4951, 5347, 13488, 17584

Table 314. IBM MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
880	37, 256, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 775, 813, 819, 838, 850, 852, 855, 857, 860-866, 869-871, 874-875, 878, 897, 903, 912, 915-916, 920, 1009, 1025-1027, 1041-1043, 1112, 1122, 1200, 1208, 1251-1252, 1283, 4909, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5347, 8229, 9030, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 28709
891	500, 833, 1088, 1200, 1208, 4929, 9025, 13121, 13488, 17584, 25664
895	290, 500, 1027, 1041, 1200, 1208, 4386, 5123, 8482, 13488, 17584, 25617
896	290, 1027, 1041, 1200, 1208, 4386, 4992, 5123, 8482, 13488, 17584, 25617
897	37, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252, 4386, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 8229, 8482, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
899	259
901	259, 858-859, 902, 923-924, 1140, 1148, 1156-1157, 1200, 1208, 5348, 5353, 13488, 17584
902	259, 858-859, 901, 923-924, 1140, 1148, 1156-1157, 1200, 1208, 5348, 5353, 13488, 17584
903	37, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 836, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 912, 916, 920, 1025-1027, 1041-1043, 1115, 1200, 1208, 1252, 4909, 4932, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5211, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
904	37, 500, 1114, 1200, 1208, 5210, 8229, 13488, 17584, 25480, 28709
905	37, 256, 437, 500, 737, 775, 819, 850, 852, 857, 860-865, 920, 1026, 1112, 1122, 1200, 1208, 1252, 1254, 1281, 4946, 4948, 4953, 4960, 8229, 9044, 9049, 9056, 13488, 17248, 17584, 28709
912	37, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 916, 920, 1025-1027, 1041-1043, 1047, 1200, 1208, 1250, 1252, 1282, 4909, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5346, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
914	37, 437, 500, 819, 850, 1200, 1208, 1252, 1257, 4946, 8229, 13488, 17584, 28709
915	37, 259, 437, 500, 819, 850, 855, 866, 870, 878, 880, 1025, 1131, 1200, 1208, 1251-1252, 1283, 4946, 4951, 5347, 8229, 13488, 17584, 28709
916	37, 273, 277-278, 280, 284-285, 297, 423-424, 437, 500, 803, 813, 819, 838, 850, 852, 856-857, 860-863, 869-871, 874-875, 880, 897, 903, 912, 920, 1025-1027, 1041-1043, 1200, 1208, 1252, 1255, 4909, 4934, 4946, 4948, 4952-4953, 4970-4971, 5012, 5123, 5351, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
918	864, 868, 1006, 1200, 1208, 4960, 9056, 13488, 17248, 17584
920	37, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 1025-1026, 1200, 1208, 1252, 1254, 1281, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5350, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 28709
921	37, 437, 500, 819, 850, 922, 1112, 1122, 1200, 1208, 1252, 1257, 4946, 5353, 8229, 13488, 17584, 28709
922	37, 437, 500, 819, 850, 921, 1112, 1122, 1200, 1208, 1252, 1257, 4946, 5353, 8229, 13488, 17584, 28709

Table 314. IBM MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
923	37, 273, 277-278, 280, 284-285, 297, 437, 500, 808, 819, 850, 858, 860-861, 865, 871-872, 901-902, 924, 1047, 1051, 1140-1149, 1153-1158, 1160-1162, 1164, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
924	37, 273, 277-278, 280, 284-285, 297, 437, 500, 808, 819, 848-850, 858, 860-861, 865, 871-872, 901-902, 923, 1047, 1051, 1140-1149, 1153-1157, 1160-1164, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
926	834, 951, 9026
927	835, 947, 1200, 1208, 4931, 9027, 13488, 17584, 21427
928	837, 1200, 1208, 1380, 13488, 17584
930	931-932, 939, 942-943, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
931	930, 932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
932	930-931, 939, 942-943, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
933	934, 944, 949, 1200, 1208, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13488, 13651, 17317, 17584, 25510, 25520, 25525, 29616, 29621, 33717, 37813
934	933, 949, 5029, 5045, 5460, 9125, 13221, 17317, 25510, 25525, 29621, 33717, 37813
935	936, 946, 1200, 1208, 1381, 1386, 1388, 5031, 5477, 5482, 5484, 9127, 13223, 13488, 17584, 25512
936	935, 946, 1381, 5031, 5477, 5484, 9127, 13223, 25512
937	938, 948, 950, 1200, 1208, 1370, 5033, 5046, 9142, 13488, 17584, 25514, 25524, 29620
938	937, 950, 1370, 5033, 5046, 9142, 25514
939	930-932, 942-943, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
941	300-301, 1200, 1208, 1351, 4396, 8492, 13488, 16684, 17584
942	930-932, 939, 943, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
943	930-932, 939, 942, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
944	933, 949, 1200, 1208, 5029, 5045, 5460, 9125, 13221, 13488, 17317, 17584, 25520, 25525, 29616, 29621, 33717, 37813
946	935-936, 1200, 1208, 5031, 5484, 9127, 13223, 13488, 17584, 25512
947	835, 927, 1200, 1208, 4931, 9027, 13488, 17584, 21427
948	937, 950, 1200, 1208, 1370, 5033, 5046, 9142, 13488, 17584, 25524, 29620
949	933-934, 944, 1200, 1208, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13488, 13651, 17317, 17584, 25510, 25520, 25525, 29616, 29621, 33717, 37813
950	937-938, 948, 1200, 1208, 1370, 5033, 5046, 9142, 13488, 17584, 25514, 25524, 29620
951	834, 926, 1200, 1208, 1362, 4930, 9026, 13488, 17584
1004	500, 819, 850, 1200, 1208, 4946, 13488, 17584

Table 314. IBM MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
1006	868, 918, 1200, 1208, 13488, 17584
1008	420, 864, 1200, 1208, 4960, 5104, 8612, 9056, 13488, 16804, 17248, 17584
1009	37, 273, 277-278, 280, 284, 290, 297, 367, 423, 500, 833, 836, 870-871, 875, 880, 1025-1026, 1200, 1208, 4386, 4929, 4932, 4971, 8229, 8482, 9025, 13121, 13488, 17584, 28709
1010	500, 1200, 1208, 13488, 17584
1011	500, 1200, 1208, 13488, 17584
1012	500, 1200, 1208, 13488, 17584
1013	500, 1140, 1200, 1208, 13488, 17584
1014	500, 1200, 1208, 13488, 17584
1015	500, 1200, 1208, 13488, 17584
1016	500, 1200, 1208, 13488, 17584
1017	500, 1200, 1208, 13488, 17584
1018	500, 1200, 1208, 13488, 17584
1019	500, 1200, 1208, 13488, 17584
1020	500
1021	500
1023	500
1025	37, 256, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-866, 869-871, 874-875, 878, 880, 897, 903, 912, 915-916, 920, 1009, 1026-1027, 1040-1043, 1051, 1088, 1112, 1122, 1131, 1200, 1208, 1251-1252, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5347, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
1026	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-865, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1009, 1025, 1027, 1040-1043, 1047, 1088, 1112, 1122, 1200, 1208, 1252, 1254, 1281, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5350, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
1027	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-865, 869-871, 874-875, 880, 895-897, 903, 912, 916, 1025-1026, 1040-1043, 1047, 1088, 1112, 1122, 1139, 1200, 1208, 1252, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 4992, 5012, 5123, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
1040	37, 273, 277-278, 280, 284-285, 290, 297, 437, 500, 833, 836, 850, 852, 855, 857, 870-871, 1025-1027, 1041-1043, 1088, 1200, 1208, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 5123, 8229, 8482, 9025, 9044, 9049, 13121, 13488, 17584, 25617, 25619, 25664, 28709
1041	37, 273, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 895-897, 903, 912, 916, 1025-1027, 1040, 1042-1043, 1088, 1200, 1208, 1252, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 4992, 5012, 5123, 8229, 8482, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 17584, 25473, 25479, 25617, 25619, 25664, 28709

Table 314. IBM MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
1042	37, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 1025-1027, 1040, 1041, 1043, 1088, 1200, 1208, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 8229, 8482, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 17584, 25473, 25479, 25617, 25619, 25664, 28709
1043	37, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 1025-1027, 1040, 1041, 1042, 1088, 1114, 1200, 1208, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5210, 8229, 8482, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 17584, 25473, 25479, 25617, 25619, 25664, 28709
1046	420, 500, 864, 1089, 1127, 1200, 1208, 1256, 4960, 5142, 5352, 8612, 9056, 9238, 13488, 16804, 17248, 17584
1047	37, 273-275, 277-278, 280, 281, 282, 284-285, 290, 297, 437, 500, 819, 850, 852, 858, 870-871, 875, 912, 923-924, 1026-1027, 1140-1149, 1200, 1208, 1252, 1254, 4946, 4948, 5123, 8229, 8482, 13488, 17584, 28709
1051	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 871, 923-924, 1025, 1097, 1140-1149, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1088	37, 273, 277-278, 280, 284-285, 290, 297, 367, 500, 819, 833, 836, 850, 852, 855, 857, 870-871, 875, 891, 1025-1027, 1040-1043, 1126, 1200, 1208, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4971, 5123, 8229, 8482, 9025, 9044, 9049, 13121, 13488, 17584, 25617, 25619, 25664, 28709
1089	420, 500, 819, 850, 864, 1046, 1127, 1200, 1208, 1256, 4946, 4960, 5142, 5352, 8612, 9056, 9238, 13488, 16804, 17248, 17584
1097	37, 437, 500, 737, 775, 819, 850, 852, 857, 860-865, 1051, 1098, 1112, 1122, 1200, 1208, 1252, 4946, 4948, 4953, 4960, 8229, 9044, 9049, 9056, 13488, 17248, 17584, 28709
1098	259, 420, 437, 819, 850, 1097, 1200, 1208, 1252, 4946, 8612, 13488, 16804, 17584
1100	37, 273, 277-278, 280, 284-285, 297, 500, 850, 4946, 8229, 28709
1101	500
1102	500
1103	500
1104	500
1105	500
1106	500
1107	500
1112	37, 256, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 775, 819, 833, 836, 838, 850, 870-871, 875, 880, 905, 921-922, 1025-1027, 1097, 1122, 1200, 1208, 1252, 1257, 4386, 4929, 4932, 4934, 4946, 4971, 5123, 5353, 8229, 8482, 8612, 9025, 9030, 13121, 13488, 16804, 17584, 28709
1114	37, 437, 500, 819, 836, 850, 904, 1043, 1115, 1200, 1208, 4932, 4946, 5210-5211, 8229, 13488, 17584, 25480, 25619, 28709
1115	37, 367, 437, 500, 836, 903, 1114, 1200, 1208, 4932, 5210-5211, 8229, 13488, 17584, 25479, 28709
1122	37, 256, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 775, 819, 833, 836, 838, 850, 870-871, 875, 880, 905, 921-922, 1025-1027, 1097, 1112, 1200, 1208, 1252, 1257, 4386, 4929, 4932, 4934, 4946, 4971, 5123, 5353, 8229, 8482, 8612, 9025, 9030, 13121, 13488, 16804, 17584, 28709
1123	819, 1124-1125, 1148, 1200, 1208, 1251-1252, 1283, 5347, 13488, 17584

Table 314. IBM MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
1124	37, 500, 1123, 1125, 1200, 1208, 1251, 1283, 5347, 8229, 13488, 17584, 28709
1125	500, 1123, 1124, 1200, 1208, 1251, 1283, 5347, 13488, 17584
1126	37, 367, 437, 500, 819, 833, 850, 1088, 1200, 1208, 1252, 4929, 4946, 8229, 9025, 13121, 13488, 17584, 25664, 28709
1127	420, 864, 1046, 1089, 1256, 4960, 5142, 8612, 9056, 9238, 16804, 17248
1129	500, 1130, 1200, 1208, 1258, 5354, 13488, 17584
1130	37, 500, 819, 850, 1129, 1200, 1208, 1252, 1258, 4946, 5354, 8229, 13488, 17584, 28709
1131	37, 500, 878, 915, 1025, 1200, 1208, 1251, 1283, 5347, 8229, 13488, 17584, 28709
1132	37, 500, 819, 850, 1133, 1200, 1208, 1252, 4946, 8229, 13488, 17584, 28709
1133	500, 1132, 1200, 1208, 13488, 17584
1137	37, 500, 819, 1200, 1208, 8229, 13488, 17584, 28709
1139	290, 1027, 4386, 5123, 8482
1140	37, 273, 277-278, 280, 284-285, 297, 437, 500, 808, 819, 850, 858, 860, 863, 871-872, 901-902, 923-924, 1013, 1047, 1051, 1141-1149, 1153-1157, 1160-1162, 1164, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1141	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 871-872, 923-924, 1047, 1051, 1140, 1142-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1142	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 865, 871-872, 923-924, 1047, 1051, 1140-1141, 1143-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1143	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 865, 871-872, 923-924, 1047, 1051, 1140-1142, 1144-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1144	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 871-872, 923-924, 1047, 1051, 1140-1143, 1145-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1145	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 860, 863, 871-872, 923-924, 1047, 1051, 1140-1144, 1146-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1146	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 860, 863, 871-872, 923-924, 1047, 1051, 1140-1145, 1147-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1147	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 871-872, 923-924, 1047, 1051, 1140-1146, 1148-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1148	37, 273, 277-278, 280, 284-285, 297, 437, 500, 808, 819, 848-850, 858, 860-861, 863, 865, 871-872, 901-902, 923-924, 1047, 1051, 1123, 1140-1147, 1149, 1153-1164, 1200, 1208, 1252, 1275, 4899, 4946, 5348, 5349, 8229, 12712, 13488, 17584, 28709
1149	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 861, 863, 871-872, 923-924, 1047, 1051, 1140-1148, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1153	808, 858-859, 867, 872, 923-924, 1140-1149, 1154-1157, 1160-1162, 1200, 1208, 5348, 9044, 13488, 17584
1154	808, 849, 858-859, 867, 872, 923-924, 1140-1149, 1153, 1155-1157, 1160-1162, 1200, 1208, 5347, 5348, 13488, 17584

Table 314. IBM MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
1155	858-859, 867, 872, 923-924, 1140-1149, 1153-1154, 1156-1157, 1160-1162, 1200, 1208, 5348, 5350, 9049, 13488, 17584
1156	858-859, 901-902, 923-924, 1140-1149, 1153-1155, 1157, 1160, 1200, 1208, 5348, 5353, 12712, 13488, 17584
1157	858-859, 901-902, 923-924, 1140-1149, 1153-1156, 1160, 1200, 1208, 5348, 5353, 12712, 13488, 17584
1158	848, 923, 1148, 1200, 1208, 5347, 5348, 13488, 17584
1159	1148, 1200, 1208, 13488, 17584
1160	858-859, 867, 923-924, 1140-1149, 1153-1157, 1161-1162, 1200, 1208, 5348, 13488, 17584
1161	259, 858-859, 923-924, 1140-1149, 1153-1155, 1160, 5348, 17584
1162	259, 858-859, 923-924, 1140-1149, 1153-1155, 1160, 5348, 17584
1163	924, 1148, 1164, 5354, 17584
1164	858-859, 923-924, 1140, 1148, 1163, 1200, 1208, 5348, 5354, 13488, 17584
1166	1200,1208,13488,17584
1200	37, 256, 259, 273, 275, 277-278, 280, 282, 284-285, 290, 293, 297, 300-301, 367, 420, 423-424, 437, 500, 720, 737, 775, 803, 806, 808, 813, 819, 833-838, 848-852, 855-872, 874-875, 878, 880, 891, 895-897, 901-905, 912, 914-916, 918, 920-924, 927-928, 930, 932-933, 935, 937, 939, 941-944, 946-951, 1004, 1006, 1008-1019, 1025-1027, 1040-1043, 1046-1047, 1051, 1088-1089, 1097-1098, 1112, 1114-1115, 1122-1126, 1129-1133, 1137, 1140-1149, 1153-1160, 1164, 1166, 1208, 1250-1258, 1275-1277, 1280-1285, 1351, 1362-1364, 1370-1371, V 9.0.0 1374-1379,1380-1381, 1385-1386, 1388, 1390, 1399, 4899, 4909, 4930, 4933, 4948, 4951-4952, 4960, 4971, 5012, 5039, 5104, 5123, 5142, 5210, 5346-5354, 8482, 8612, 9027, 9030, 9044, 9048-9049, 9056, 9061, 9066, 9238, 12712, 13121, 13218, 13488, 16684, 16804, 17248, 17584, 21427, 28709
1208	37, 256, 259, 273, 275, 277-278, 280, 282, 284-285, 290, 293, 297, 300-301, 367, 420, 423-424, 437, 500, 720, 737, 775, 803, 806, 808, 813, 819, 833-838, 848-852, 855-872, 874-875, 878, 880, 891, 895-897, 901-905, 912, 914-916, 918, 920-924, 927-928, 930, 932-933, 935, 937, 939, 941-944, 946-951, 1004, 1006, 1008-1019, 1025-1027, 1040-1043, 1046-1047, 1051, 1088-1089, 1097-1098, 1112, 1114-1115, 1122-1126, 1129-1133, 1137, 1140-1149, 1153-1160, 1164, 1166, 1200, 1250-1258, 1275-1277, 1280-1285, 1351, 1362-1364, 1370-1371, V 9.0.0 1374-1379,1380-1381, 1385-1386, 1388, 1390, 1399, 4899, 4909, 4930, 4933, 4948, 4951-4952, 4960, 4971, 5012, 5026, 5035, 5039, 5104, 5123, 5142, 5210, 5346-5354, 8482, 8612, 9027, 9030, 9044, 9048-9049, 9056, 9061, 9066, 9238, 12712, 13121, 13218, 13488, 16684, 16804, 17248, 17584, 21427, 28709
1250	37, 259, 273, 500, 819, 850, 852, 855, 870, 912, 1200, 1208, 1252, 1282, 4946, 4948, 4951, 5346, 8229, 9044, 13488, 17584, 28709
1251	37, 256, 259, 500, 819, 850, 855, 866, 878, 880, 915, 1025, 1123-1125, 1131, 1200, 1208, 1252, 1283, 4946, 4951, 5347, 8229, 13488, 17584, 28709
1252	37, 256, 259, 273, 275, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 737, 775, 803, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-866, 869-871, 874-875, 880, 897, 903, 905, 912, 914-916, 920-924, 1025-1027, 1041, 1047, 1051, 1097-1098, 1112, 1122-1123, 1126, 1130, 1132, 1140-1149, 1200, 1208, 1250-1251, 1254-1255, 1257, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 16804, 17248, 17584, 25473, 25479, 25617, 28709

Table 314. IBM MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
1253	37, 259, 423, 500, 737, 813, 819, 850, 869, 875, 1200, 1208, 1280, 4909, 4946, 4971, 5349, 8229, 9061, 13488, 17584, 28709
1254	37, 259, 500, 819, 850, 857, 869, 905, 920, 1026, 1047, 1200, 1208, 1252, 1281, 4946, 4953, 5350, 8229, 9049, 9061, 13488, 17584, 28709
1255	37, 259, 424, 500, 803, 819, 850, 856, 862, 916, 1200, 1208, 1252, 1281, 4946, 4952, 5012, 5351, 8229, 13488, 17584, 28709
1256	259, 420, 500, 720, 850, 864, 1046, 1089, 1127, 1200, 1208, 4946, 4960, 5142, 5352, 8612, 9056, 9238, 13488, 16804, 17248, 17584
1257	37, 259, 437, 500, 775, 819, 850, 914, 921-922, 1112, 1122, 1200, 1208, 1252, 4946, 5353, 8229, 13488, 17584, 28709
1258	37, 259, 500, 819, 1129-1130, 1200, 1208, 5354, 8229, 13488, 17584, 28709
1275	37, 256, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 871, 923-924, 1051, 1140-1149, 1200, 1208, 1252, 4946, 5348, 8229, 13488, 17584, 28709
1276	1200, 1208, 13488, 17584
1277	1200, 1208, 13488, 17584
1280	37, 423, 437, 500, 737, 813, 819, 850, 869, 875, 1200, 1208, 1252-1253, 4909, 4946, 4971, 5349, 8229, 9061, 13488, 17584, 28709
1281	37, 437, 500, 819, 850, 857, 905, 920, 1026, 1200, 1208, 1252, 1254-1255, 4946, 4953, 5350, 8229, 9049, 13488, 17584, 28709
1282	500, 852, 870, 912, 1200, 1208, 1250, 4948, 5346, 9044, 13488, 17584
1283	37, 437, 500, 819, 850, 855, 866, 878, 880, 915, 1025, 1123-1125, 1131, 1200, 1208, 1251-1252, 4946, 4951, 5347, 8229, 13488, 17584, 28709
1284	1200, 1208, 13488, 17584
1285	1200, 1208, 13488, 17584
1351	300-301, 941, 1200, 1208, 4396, 8492, 13488, 16684, 17584
1362	834, 951, 1200, 1208, 4930, 9026, 13488, 17584
1363	933, 949, 1200, 1208, 1364, 5029, 5045, 5460, 9125, 9555, 13221, 13488, 13651, 17317, 17584, 25525, 29621, 33717, 37813
1364	933, 949, 1200, 1208, 1363, 5029, 5045, 5460, 9125, 9555, 13221, 13488, 13651, 17317, 17584, 25525, 29621, 33717, 37813
1370	937-938, 948, 950, 1200, 1208, 1371, 5033, 5046, 9142, 13488, 17584, 25514, 25524, 29620
1371	1200, 1208, 1370, 13488, 17584
 1374	1200, 1208
 1375	1200, 1208
 1376	1200, 1208
 1377	1200, 1208
 1378	1200, 1208
 1379	1200, 1208
1380	837, 928, 1200, 1208, 1385, 4933, 13488, 17584
1381	935-936, 1200, 1208, 1386, 1388, 5031, 5477, 5482, 5484, 9127, 13223, 13488, 17584, 25512

Table 314. IBM MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
1385	837, 1200, 1208, 1380, 4933, 13488, 17584
1386	935, 1200, 1208, 1381, 1388, 5031, 5477, 5482, 5484, 9127, 13223, 13488, 17584
1388	935, 1200, 1208, 1381, 1386, 5031, 5477, 5482, 5484, 5488, 9127, 13223, 13488, 17584
1390	930-932, 939, 942-943, 1200, 1208, 1399, 5026, 5028, 5035, 5038-5039, 5055, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
1399	930-932, 939, 942-943, 1200, 1208, 1390, 5026, 5028, 5035, 5038-5039, 5050, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
4386	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 860-865, 870-871, 895-897, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1139, 1252, 4929, 4932, 4946, 4948, 4951, 4953, 4960, 4992, 5123, 8229, 8482, 9025, 9044, 9049, 9056, 13121, 17248, 25473, 25617, 25619, 25664, 28709
4396	300-301, 941, 1351, 8492, 16684
4899	867, 1148, 1200, 1208, 5351, 9048, 12712, 13488, 17584
4909	37, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252-1253, 1280, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5349, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
4929	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 860-865, 870-871, 891, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1126, 1252, 4386, 4932, 4946, 4948, 4951, 4953, 4960, 5123, 8229, 8482, 9025, 9044, 9049, 9056, 13121, 17248, 25617, 25619, 25664, 28709
4930	834, 951, 1200, 1208, 1362, 9026, 13488, 17584
4931	835, 927, 947, 9027, 21427
4932	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 424, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 870-871, 875, 903, 1009, 1025-1027, 1040-1043, 1088, 1112, 1114-1115, 1122, 1252, 4386, 4929, 4946, 4948, 4951, 4953, 4971, 5123, 5210-5211, 8229, 8482, 9025, 9044, 9049, 13121, 25479, 25617, 25619, 25664, 28709
4933	837, 1200, 1208, 1380, 1385, 13488, 17584
4934	37, 256, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 775, 813, 819, 838, 850, 852, 857, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1112, 1122, 1252, 4909, 4946, 4948, 4953, 4960, 4970-4971, 5012, 5123, 8229, 9030, 9044, 9049, 9056, 9061, 9066, 17248, 25473, 25479, 25617, 25619, 28709
4946	37, 256, 259, 273, 275, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 737, 775, 803, 813, 819, 833, 836, 838, 850, 852, 855-858, 860-866, 869-871, 874-875, 880, 897, 903, 905, 912, 914-916, 920-924, 1004, 1025-1027, 1040-1043, 1047, 1051, 1088-1089, 1097-1098, 1100, 1112, 1114, 1122, 1126, 1130, 1132, 1140-1149, 1250-1257, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4948, 4951-4953, 4960, 4970-4971, 5012, 5123, 5210, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 16804, 17248, 25473, 25479, 25617, 25619, 25664, 28709
4948	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1047, 1088, 1097, 1200, 1208, 1250, 1252, 1282, 4386, 4909, 4929, 4932, 4934, 4946, 4951, 4953, 4970-4971, 5012, 5123, 5346, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 25664, 28709

Table 314. IBM MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
4951	37, 259, 273, 277-278, 280, 284-285, 290, 297, 437, 500, 819, 833, 836, 850, 852, 855, 857, 866, 870-871, 878, 880, 912, 915, 1025-1027, 1040-1043, 1088, 1200, 1208, 1250-1252, 1283, 4386, 4929, 4932, 4946, 4948, 4953, 5123, 5346, 5347, 8229, 8482, 9025, 9044, 9049, 13121, 13488, 17584, 25617, 25619, 25664, 28709
4952	259, 273, 424, 500, 803, 850, 856, 862, 916, 1200, 1208, 1255, 4946, 5012, 5351, 13488, 17584
4953	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1088, 1097, 1252, 1254, 1281, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4970-4971, 5012, 5123, 5350, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 16804, 25473, 25479, 25617, 25619, 25664, 28709
4960	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 720, 819, 833, 838, 850, 864, 870-871, 875, 880, 905, 918, 1008, 1025-1027, 1046, 1089, 1097, 1127, 1200, 1208, 1252, 1256, 4386, 4929, 4934, 4946, 4971, 5104, 5123, 5142, 5352, 8229, 8482, 8612, 9025, 9030, 9056, 9238, 13121, 13488, 16804, 17248, 17584, 28709
4970	37, 259, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1252, 4909, 4934, 4946, 4948, 4953, 4971, 5012, 5123, 8229, 9030, 9044, 9049, 9061, 9066, 25473, 25479, 25617, 25619, 28709
4971	37, 256, 273, 277-278, 280, 284-285, 297, 367, 423, 437, 500, 737, 775, 813, 819, 836, 838, 850-852, 857, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1009, 1025-1027, 1041-1043, 1047, 1088, 1112, 1122, 1200, 1208, 1252-1253, 1280, 4909, 4932, 4934, 4946, 4948, 4953, 4960, 4970, 5012, 5123, 5349, 8229, 9030, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
4992	290, 896, 1027, 1041, 4386, 5123, 8482, 25617
5012	37, 273, 277-278, 280, 284-285, 297, 423-424, 437, 500, 803, 813, 819, 838, 850, 852, 856-857, 860-863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252, 1255, 4909, 4934, 4946, 4948, 4952-4953, 4970-4971, 5123, 5351, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
5026	930-932, 939, 942-943, 1390, 1399, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 1208, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
5028	930-932, 939, 942-943, 1390, 1399, 5026, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
5029	933-934, 944, 949, 1363-1364, 5045, 5460, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 33717, 37813
5031	935-936, 946, 1381, 1386, 1388, 5477, 5482, 5484, 9127, 13223, 25512
5033	937-938, 948, 950, 1370, 5046, 9142, 25514, 25524, 29620
5035	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5038-5039, 9122, 9124, 9131, 9135, 1208, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
5038	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
5039	930-932, 939, 942-943, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
5045	933-934, 944, 949, 1363-1364, 5029, 5460, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 33717, 37813
5046	937-938, 948, 950, 1370, 5033, 9142, 25514, 25524, 29620

Table 314. IBM MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
5104	420, 864, 1008, 1200, 1208, 4960, 8612, 9056, 13488, 16804, 17248, 17584
5123	290, 367, 423, 437, 819, 1027, 1041, 1047, 1140-1149, 1156, 1157, 1160, 1200, 1208, 1252, 4948, 5348, 8482, 13488
5142	420, 500, 864, 1046, 1089, 1127, 1200, 1208, 1256, 4960, 5352, 8612, 9056, 9238, 13488, 16804, 17248, 17584
5210	37, 437, 500, 819, 836, 850, 904, 1043, 1114-1115, 1200, 1208, 4932, 4946, 5211, 8229, 13488, 17584, 25480, 25619, 28709
5211	37, 367, 437, 500, 836, 903, 1114-1115, 4932, 5210, 8229, 25479, 28709
5346	37, 259, 273, 500, 819, 850, 852, 855, 870, 912, 1200, 1208, 1250, 1252, 1282, 4946, 4948, 4951, 8229, 9044, 13488, 17584, 28709
5347	808, 848-849, 855, 866, 872, 878, 880, 915, 1025, 1123-1125, 1131, 1154, 1158, 1200, 1208, 1251, 1283, 4951, 13488, 17584
5348	37, 259, 273, 275, 277-278, 280, 284-285, 297, 437, 500, 808, 819, 850, 858, 860-861, 863, 865, 871-872, 901-902, 923-924, 1051, 1140-1149, 1153-1158, 1160-1162, 1164, 1200, 1208, 1252, 1275, 4946, 8229, 13488, 17584, 28709
5349	813, 869, 875, 1148, 1200, 1208, 1253, 1280, 4909, 4971, 9061, 13488, 17584
5350	857, 920, 1026, 1155, 1200, 1208, 1254, 1281, 4953, 9049, 13488, 17584
5351	424, 856, 862, 867, 916, 1200, 1208, 1255, 4899, 4952, 5012, 9048, 12712, 13488, 17584
5352	420, 864, 1046, 1089, 1200, 1208, 1256, 4960, 5142, 8612, 9056, 9238, 13488, 16804, 17248, 17584
5353	901-902, 921-922, 1112, 1122, 1156-1157, 1200, 1208, 1257, 13488, 17584
5354	1129-1130, 1163, 1164, 1200, 1208, 1258, 13488, 17584
5460	933-934, 944, 949, 1363-1364, 5029, 5045, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 33717, 37813
5477	935-936, 1381, 1386, 1388, 5031, 5482, 5484, 9127, 13223, 25512
5482	935, 1381, 1386, 1388, 5031, 5477, 5484, 9127, 13223
5484	935-936, 946, 1381, 1386, 1388, 5031, 5477, 5482, 9127, 13223, 25512
5488	1388
8229	37, 256, 273, 275, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 720, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-866, 869-871, 874-875, 880, 897, 903-905, 912, 914-916, 920-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1097, 1100, 1112, 1114-1115, 1122, 1124, 1126, 1130-1132, 1137, 1140-1149, 1250-1255, 1257-1258, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5210-5211, 5346, 5348, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 16804, 17248, 25473, 25479, 25480, 25617, 25619, 25664, 28709
8482	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 860-865, 870-871, 895-897, 1009, 1025-1027, 1040-1043, 1047, 1088, 1112, 1122, 1139, 1200, 1208, 1252, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4960, 4992, 5123, 8229, 9025, 9044, 9049, 9056, 13121, 13488, 17248, 17584, 25473, 25617, 25619, 25664, 28709
8492	300-301, 941, 1351, 4396, 16684
8612	37, 256, 420, 424, 437, 500, 720, 737, 775, 819, 850, 852, 857, 860-865, 1008, 1046, 1089, 1098, 1112, 1122, 1127, 1200, 1208, 1252, 1256, 4946, 4948, 4953, 4960, 5104, 5142, 5352, 8229, 9044, 9049, 9056, 9238, 13488, 16804, 17248, 17584, 28709

Table 314. IBM MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
9025	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 860-865, 870-871, 891, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1126, 1252, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4960, 5123, 8229, 8482, 9044, 9049, 9056, 13121, 17248, 25617, 25619, 25664, 28709
9026	834, 926, 951, 1362, 4930
9027	835, 927, 947, 1200, 1208, 4931, 13488, 17584, 21427
9030	37, 256, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 775, 813, 819, 838, 850, 852, 857, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1112, 1122, 1200, 1208, 1252, 4909, 4934, 4946, 4948, 4953, 4960, 4970-4971, 5012, 5123, 8229, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 28709
9044	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1047, 1088, 1097, 1153, 1200, 1208, 1250, 1252, 1282, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5346, 8229, 8482, 8612, 9025, 9030, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 25664, 28709
9048	867, 1200, 1208, 4899, 5351, 12712, 13488, 17584
9049	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1088, 1097, 1155, 1200, 1208, 1252, 1254, 1281, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5350, 8229, 8482, 8612, 9025, 9030, 9044, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 25664, 28709
9056	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 720, 819, 833, 838, 850, 864, 870-871, 875, 880, 905, 918, 1008, 1025-1027, 1046, 1089, 1097, 1127, 1200, 1208, 1252, 1256, 4386, 4929, 4934, 4946, 4960, 4971, 5104, 5123, 5142, 5352, 8229, 8482, 8612, 9025, 9030, 9238, 13121, 13488, 16804, 17248, 17584, 28709
9061	37, 256, 259, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252-1254, 1280, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5349, 8229, 9030, 9044, 9049, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
9066	37, 259, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 8229, 9030, 9044, 9049, 9061, 13488, 17584, 25473, 25479, 25617, 25619, 28709
9122	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
9124	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
9125	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 33717, 37813
9127	935-936, 946, 1381, 1386, 1388, 5031, 5477, 5482, 5484, 13223, 25512
9131	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
9135	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
9142	937-938, 948, 950, 1370, 5033, 5046, 25514, 25524, 29620

Table 314. IBM MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
9238	420, 500, 864, 1046, 1089, 1127, 1200, 1208, 1256, 4960, 5142, 5352, 8612, 9056, 13488, 16804, 17248, 17584
9555	933, 949, 1363-1364, 5029, 5045, 5460, 9125, 13221, 13651, 17317, 25525, 29621, 33717, 37813
12712	862, 867, 1148, 1156-1157, 1200, 1208, 4899, 5351, 9048, 13488, 17584
13121	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 860-865, 870-871, 891, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1126, 1200, 1208, 1252, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4960, 5123, 8229, 8482, 9025, 9044, 9049, 9056, 13488, 17248, 17584, 25617, 25619, 25664, 28709
13218	930-932, 939, 942-943, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
13219	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
13221	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 33717, 37813
13223	935-936, 946, 1381, 1386, 1388, 5031, 5477, 5482, 5484, 9127, 25512
13231	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 17314, 25508, 25518, 29614, 33698-33700, 37796
13488	37, 256, 259, 273, 275, 277-278, 280, 282, 284-285, 290, 293, 297, 300-301, 367, 420, 423-424, 437, 500, 720, 737, 775, 803, 806, 808, 813, 819, 833-838, 848-852, 855-872, 874-875, 878, 880, 891, 895-897, 901-905, 912, 914-916, 918, 920-924, 927-928, 930, 932-933, 935, 937, 939, 941-944, 946-951, 1004, 1006, 1008-1019, 1025-1027, 1040-1043, 1046-1047, 1051, 1088-1089, 1097-1098, 1112, 1114-1115, 1122-1126, 1129-1133, 1137, 1140-1149, 1153-1160, 1164, 1166, 1200, 1208, 1250-1258, 1275-1277, 1280-1285, 1351, 1362-1364, 1370-1371, 1380-1381, 1385-1386, 1388, 1390, 1399, 4899, 4909, 4930, 4933, 4948, 4951-4952, 4960, 4971, 5012, 5039, 5104, 5123, 5142, 5210, 5346-5354, 8482, 8612, 9027, 9030, 9044, 9048-9049, 9056, 9061, 9066, 9238, 12712, 13121, 13218, 16684, 16804, 17248, 17584, 21427, 28709
13651	933, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 17317, 25525, 29621, 33717, 37813
16684	300-301, 941, 1200, 1208, 1351, 4396, 8492, 13488, 17584
16804	37, 256, 420, 424, 437, 500, 720, 737, 775, 819, 850, 852, 857, 860-865, 1008, 1046, 1089, 1098, 1112, 1122, 1127, 1200, 1208, 1252, 1256, 4946, 4948, 4953, 4960, 5104, 5142, 5352, 8229, 8612, 9044, 9049, 9056, 9238, 13488, 17248, 17584, 28709
17248	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 720, 819, 833, 838, 850, 864, 870-871, 875, 880, 905, 918, 1008, 1025-1027, 1046, 1089, 1097, 1127, 1200, 1208, 1252, 1256, 4386, 4929, 4934, 4946, 4960, 4971, 5104, 5123, 5142, 5352, 8229, 8482, 8612, 9025, 9030, 9056, 9238, 13121, 13488, 16804, 17584, 28709
17314	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 25508, 25518, 29614, 33698-33700, 37796
17317	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13651, 25510, 25520, 25525, 29616, 29621, 33717, 37813

Table 314. IBM MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
17584	37, 256, 259, 273, 275, 277-278, 280, 282, 284-285, 290, 293, 297, 300-301, 367, 420, 423-424, 437, 500, 720, 737, 775, 803, 806, 808, 813, 819, 833-838, 848-852, 855-872, 874-875, 878, 880, 891, 895-897, 901-905, 912, 914-916, 918, 920-924, 927-928, 930, 932-933, 935, 937, 939, 941-944, 946-951, 1004, 1006, 1008-1019, 1025-1027, 1040-1043, 1046-1047, 1051, 1088-1089, 1097-1098, 1112, 1114-1115, 1122-1126, 1129-1133, 1137, 1140-1149, 1153-1160, 1164, 1166, 1200, 1208, 1250-1258, 1275-1277, 1280-1285, 1351, 1362-1364, 1370-1371, 1380-1381, 1385-1386, 1388, 1390, 1399, 4899, 4909, 4930, 4933, 4948, 4951-4952, 4960, 4971, 5012, 5039, 5104, 5123, 5142, 5210, 5346-5354, 8482, 8612, 9027, 9030, 9044, 9048-9049, 9056, 9061, 9066, 9238, 12712, 13121, 13218, 13488, 16684, 16804, 17248, 21427, 28709
21427	835, 927, 947, 1200, 1208, 4931, 9027, 13488, 17584
25473	37, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1252, 4386, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 8229, 8482, 9030, 9044, 9049, 9061, 9066, 25479, 25617, 25619, 28709
25479	37, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 836, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1115, 1252, 4909, 4932, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5211, 8229, 9030, 9044, 9049, 9061, 9066, 25473, 25617, 25619, 28709
25480	37, 500, 904, 1114, 5210, 8229, 28709
25508	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25518, 29614, 33698-33700, 37796
25510	933-934, 949, 5029, 5045, 5460, 9125, 13221, 17317, 25525, 29621, 33717, 37813
25512	935-936, 946, 1381, 5031, 5477, 5484, 9127, 13223
25514	937-938, 950, 1370, 5033, 5046, 9142
25518	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 29614, 33698-33700, 37796
25520	933, 944, 949, 5029, 5045, 5460, 9125, 13221, 17317, 25525, 29616, 29621, 33717, 37813
25524	937, 948, 950, 1370, 5033, 5046, 9142, 29620
25525	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 29616, 29621, 33717, 37813
25617	37, 273, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 895-897, 903, 912, 916, 1025-1027, 1040-1043, 1088, 1252, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 4992, 5012, 5123, 8229, 8482, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 25473, 25479, 25619, 25664, 28709
25619	37, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 1025-1027, 1040-1043, 1088, 1114, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5210, 8229, 8482, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 25473, 25479, 25617, 25664, 28709
25664	37, 273, 277-278, 280, 284-285, 290, 297, 367, 500, 819, 833, 836, 850, 852, 855, 857, 870-871, 875, 891, 1025-1027, 1040-1043, 1088, 1126, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4971, 5123, 8229, 8482, 9025, 9044, 9049, 13121, 25617, 25619, 28709

Table 314. IBM MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
28709	37, 256, 273, 275, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 720, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-866, 869-871, 874-875, 880, 897, 903-905, 912, 914-916, 920-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1097, 1100, 1112, 1114-1115, 1122, 1124, 1126, 1130-1132, 1137, 1140-1149, 1200, 1208, 1250-1255, 1257-1258, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5210-5211, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 16804, 17248, 17584, 25473, 25479, 25480, 25617, 25619, 25664
29614	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 33698-33700, 37796
29616	933, 944, 949, 5029, 5045, 5460, 9125, 13221, 17317, 25520, 25525, 29621, 33717, 37813
29620	937, 948, 950, 1370, 5033, 5046, 9142, 25524
29621	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 33717, 37813
33698	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33699-33700, 37796
33699	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698, 33700, 37796
33700	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33699, 37796
33717	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 37813
37796	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700
37813	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 33717

IBM i conversion support: IBM i

A full list of CCSIDs, and conversions supported by IBM i, can be found in the appropriate IBM i publication.

The supported code pages are listed in Supported CCSID mappings.

Unicode conversion support:

V 9.0.0 Some platforms support the conversion of user data to or from Unicode encoding. The two forms of Unicode encoding supported are UTF-16 (CCSIDs 1200, 13488, and 17584) and UTF-8 (CCSID 1208). You should use CCSIDs 1200 or 1208, as they represent the most recent Unicode version supported.

V 9.0.0
UTF-16 surrogate pairs (a pair of 2-byte UTF-16 characters in the range X'D800' through to X'DFFF' that represent a Unicode code point above U+FFFF) are supported. If a target CCSID does not contain a mapping for a code point represented by a UTF-16 surrogate pair, the pair of characters convert to a single substitution character.

Combining character sequences are supported by IBM MQ. This means that, in some cases, a precomposed character in the source CCSID will be converted to a combining character sequence in the target CCSID, or the other way round.

Note: IBM MQ does not support UTF-16 queue manager CCSIDs so message header data cannot be encoded in UTF-16.

AIX

IBM MQ AIX support for Unicode

On IBM MQ for AIX conversion to and from Unicode CCSIDs is supported for the CCSIDs in the following table.

037	273	278	280	284	285
297	423	437	500	813	819
850	852	856	857	858	860
861	865	867	869	875	878
880	901	902	912	915	916
920	923	924	932	933	935
937	938	939	942	943	948
949	950	954	964	970	1026
1046	1089	1129	1130	1131	1132
1133	1140	1141	1142	1143	1144
1145	1146	1147	1148	1149	1200
1153	1156	1157	1208	1250	1251
1253	1254	1258	1280	1281	1282
1283	1284	1285	1363	1364	1381
1383	1386	1388	4899	5026	5035
5050	5346	5347	5348	5349	5350
5351	5352	5353	5354	5488	9044
9048	9449	12712	13488	17584	33722

HP-UX

IBM MQ HP-UX support for Unicode

On IBM MQ for HP-UX conversion to and from Unicode CCSIDs is supported for the CCSIDs listed in the following table.

437	737	813	819	850	852
855	857	861	864	865	866
869	874	912	915	916	920
932	938	950	954	964	970
1051	1089	1140	1141	1142	1143
1144	1145	1146	1147	1148	1149
1200	1208	1250	1251	1252	1253
1254	1255	1256	1257	1258	1381
5050	5488	13488	33722		

Windows Solaris Linux

IBM MQ for Windows,Solaris, and Linux support for Unicode

On IBM MQ for Windows Solaris , IBM MQ for Solaris, and IBM MQ for Linux conversion to, and from, Unicode CCSIDs is supported for the CCSIDs in the following table.

037	277	278	280	284	285
290	297	300	301	420	424
437	500	813	819	833	835
836	837	838	850	852	855
856	857	858	860	861	862
863	864	865	866	867	868
869	870	871	874	875	878
880	891	897	901	902	903
904	912	913 (5)	915	916	918
920	921	922	923	924	927
928	930	931 (1)	932 (2)	933	935
937	938 (3)	939	941	942	943
947	948	949	950	951	954 (4)
964	970	1006	1025	1026	1027
1040	1041	1042	1043	1046	1047
1051	1088	1089	1097	1098	1112
1114	1115	1122	1123	1124	1129
1130	1132	1133	1140	1141	1142
1143	1144	1145	1146	1147	1148
1149	1153	1156	1157	1200	1208
1250	1251	1252	1253	1254	1255
1256	1257	1258	1275	1280	1281
V 9.0.0 1282	1283	1363	1364	1374	1375
V 9.0.0 1376	1377	1378	1379	1380	1381
1383	1386	1388	4899	5050	5346
5347	5348	5349	5350	5351	5352
5353	5354	5488 (5)	9044	9048	9449
12712	13488	17584	33722 (4)		

Notes:

1. 931 uses 939 for conversion.
2. 932 uses 942 for conversion.
3. 938 uses 948 for conversion.
4. 954 and 33722 use 5050 for conversion.
5. On Windows, Linux, and Solaris only.

IBM i

IBM i support for Unicode

For details on UNICODE support refer to the appropriate IBM i publication relating to your operating system.

z/OS

IBM MQ for z/OS support for Unicode

On IBM MQ for z/OS conversion to and from the Unicode CCSIDs is supported for the following CCSIDs:

37	256	259	273	275	277
278	280	282	284	285	290
293	297	300	301	367	420
423	424	437	500	720	737
775	803	806	808	813	819
833	834	835	836	837	838
848	849	850	851	852	855
856	857	858	859	860	861
862	863	864	865	866	867
868	869	870	871	872	874
875	878	880	891	895	896
897	901	902	903	904	905
912	914	915	916	918	920
921	922	923	924	927	928
930	932	933	935	937	939
941	942	943	944	946	947
948	949	950	951	1004	1006
1008	1009	1010	1011	1012	1013
1014	1015	1016	1017	1018	1019
1025	1026	1027	1040	1041	1042
1043	1046	1047	1051	1088	1089
1097	1098	1112	1114	1115	1122
1123	1124	1125	1126	1129	1130
1131	1132	1133	1137	1140	1141
1142	1143	1144	1145	1146	1147
1148	1149	1153	1154	1155	1156
1157	1158	1159	1160	1161	1162
1164	1200	1208	1250	1251	1252
1253	1254	1255	1256	1257	1258
1275	1276	1277	1280	1281	1282
1283	1284	1285	1351	1362	1363
1364	1370	1371	1380	1381	1385
1386	1388	1390	1399	4899	4909
4930	4933	4948	4951	4952	4960
4971	5012	5039	5104	5123	5142
5210	5346	5347	5348	5349	5350
5351	5352	5353	5354	5488	8482
8612	9027	9030	9044	9048	9049
9056	9061	9066	9238	9449	1166
V 9.0.0 1374	1375	1376	1377	1378	1379
12712	13121	13218	13488	16684	16804
17248	17584	21427	28709		

Coding standards on 64-bit platforms

Use this information to learn about coding standards on 64-bit platforms and the preferred data types.

Preferred data types

These types never change size and are available on both 32-bit and 64-bit IBM MQ platforms:

Name	Length
MQLONG	4 bytes
MQULONG	4 bytes
MQINT32	4 bytes
MQUINT32	4 bytes
MQINT64	8 bytes
MQUINT64	8 bytes

Standard data types on UNIX, Linux, and Windows: ULW

Learn about standard data types on 32-bit UNIX and Linux, 64-bit UNIX and Linux, and 64-bit Windows applications.

UNIX Linux

32-bit UNIX and Linux applications

This section is included for comparison and is based on Solaris. Any differences with other UNIX platforms are noted:

Name	Length
char	1 byte
short	2 bytes
int	4 bytes
long	4 bytes
float	4 bytes
double	8 bytes
long double	16 bytes
	AIX Linux Note that on AIX and Linux PPC a long double is 8 bytes.
pointer	4 bytes
ptrdiff_t	4 bytes
size_t	4 bytes
time_t	4 bytes
clock_t	4 bytes
wchar_t	4 bytes
	AIX Note that on AIX a wchar_t is 2 bytes.

UNIX Linux

64-bit UNIX and Linux applications

This section is based on Solaris. Any differences with other UNIX platforms are noted:

Name	Length
char	1 byte
short	2 bytes
int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes
long double	16 bytes
	AIX Linux Note that on AIX and Linux PPC a long double is 8 bytes.
pointer	8 bytes
ptrdiff_t	8 bytes
size_t	8 bytes
time_t	8 bytes
clock_t	8 bytes
	Note that on the other UNIX platform a clock_t is 4 bytes.
wchar_t	4 bytes
	AIX Note that on AIX a wchar_t is 2 bytes.

Windows

Windows 64-bit applications

Name	Length
char	1 byte
short	2 bytes
int	4 bytes
long	4 bytes
float	4 bytes
double	8 bytes
long double	8 bytes
pointer	8 bytes
	Note that all pointers are 8 bytes.
ptrdiff_t	8 bytes
size_t	8 bytes
time_t	8 bytes
clock_t	4 bytes
wchar_t	2 bytes
WORD	2 bytes
DWORD	4 bytes
HANDLE	8 bytes
HFILE	4 bytes

Windows

Coding considerations on Windows

HANDLE hf;

Use

```
hf = CreateFile((LPCTSTR) FileName,
                Access,
                ShareMode,
```

```
                xihSecAttsNTRestrict,
                Create,
                AttrAndFlags,
                NULL);
```

Do not use

```
HFILE hf;
hf = (HFILE) CreateFile((LPCTSTR) FileName,
                        Access,
                        ShareMode,
                        xihSecAttsNTRestrict,
                        Create,
                        AttrAndFlags,
                        NULL);
```

as this produces an error.

size_t len fgets

Use

```
size_t len
while (fgets(string1, (int) len, fp) != NULL)
len = strlen(buffer);
```

Do not use

```
int len;

while (fgets(string1, len, fp) != NULL)
len = strlen(buffer);
```

printf

Use

```
printf("My struc pointer: %p", pMyStruc);
```

Do not use

```
printf("My struc pointer: %x", pMyStruc);
```

If you need hexadecimal output, you have to print the upper and lower 4 bytes separately.

char *ptr

Use

```
char * ptr1;
char * ptr2;
size_t bufLen;

bufLen = ptr2 - ptr1;
```

Do not use

```
char *ptr1;
char *ptr2;
UINT32 bufLen;

bufLen = ptr2 - ptr1;
```

alignBytes

Use

```
alignBytes = (unsigned short) ((size_t) address % 16);
```

Do not use

```
void *address;
unsigned short alignBytes;

alignBytes = (unsigned short) ((UINT32) address % 16);
```

len

Use

```
len = (UINT32) ((char *) address2 - (char *) address1);
```

Do not use

```
void *address1;
void *address2;
UINT32 len;

len = (UINT32) ((char *) address2 - (char *) address1);
```

sscanf

Use

```
MQLONG SBCSprt;

sscanf(line, "%d", &SBCSprt);
```

Do not use

```
MQLONG SBCSprt;

sscanf(line, "%ld", &SBCSprt);
```

`%ld` tries to put an 8-byte type into a 4-byte type; only use `%l` if you are dealing with an actual `long` data type. MQLONG, UINT32 and INT32 are defined to be four bytes, the same as an `int` on all IBM MQ platforms:

IBM i Application Programming Reference (ILE/RPG)

Application programming for IBM i

Use the following topics to help you develop applications for IBM i:

Related information:
Developing applications

Data type descriptions on IBM i

IBM i

This collection of topics provides descriptions of data types used in IBM i programming.

Conventions used in the description of data types

For each elementary data type, this information gives a description of its usage, in a form that is independent of the programming language. This is followed by typical declarations in the ILE version of the RPG programming language. The definitions of elementary data types are included here to provide consistency. RPG uses 'D' specifications where working fields can be declared using whatever attributes you need. You can, however, do this in the calculation specifications where the field is used.

To use the elementary data types, you create:

- A /COPY member containing all the data types, or
- An external data structure (PF) containing all the data types. You then need to specify your working fields with attributes 'LIKE' the appropriate data type field.

The benefits of the second option are that the definitions can be used as a 'FIELD REFERENCE FILE' for other IBM i objects. If an IBM MQ data type definition changes, it is a relatively simple matter to re-create these objects.

Elementary data types:

All of the other data types described in this section equate either directly to these elementary data types, or to aggregates of these elementary data types (arrays or structures).

Table 315. Elementary data types

Data type	Representation
MQBOOL	10-digit signed integer
MQBYTE	1-byte alphanumeric field
MQBYTE16	16-byte alphanumeric field
MQBYTE24	24-byte alphanumeric field
MQBYTE32	32-byte alphanumeric field
MQBYTE64	64-byte alphanumeric field
MQCHAR	1-byte alphanumeric field
MQCHAR4	4-byte alphanumeric field
MQCHAR8	8-byte alphanumeric field
MQCHAR12	12-byte alphanumeric field
MQCHAR16	16-byte alphanumeric field
MQCHAR20	20-byte alphanumeric field
MQCHAR28	28-byte alphanumeric field
MQCHAR32	32-byte alphanumeric field
MQCHAR48	48-byte alphanumeric field
MQCHAR64	64-byte alphanumeric field
MQCHAR128	128-byte alphanumeric field

Table 315. Elementary data types (continued)

Data type	Representation
MQCHAR256	256-byte alphanumeric field
MQFLOAT32	4-byte floating-point number
MQFLOAT64	8-byte floating-point number
MQHCONFIG	Configuration handle
MQHCONN	10-digit signed integer
MQHMSG	Message handle that gives access to a message
MQHOBJ	10-digit signed integer
MQINT8	8-bit signed integer
MQINT16	16-bit signed integer
MQINT32	32-bit signed integer
MQINT64	64-bit signed integer
MQLONG	32-bit signed integer
MQPID	Process identifier
MQPTR	Pointer
MQTID	Thread identifier
MQUINT8	8-bit unsigned integer
MQUINT16	16-bit unsigned integer
MQUINT32	32-bit unsigned integer
MQUINT64	64-bit unsigned integer
MQULONG	32-bit unsigned integer
PMQACH	Pointer to a data structure of type MQACH
PMQAIR	Pointer to a data structure of type MQAIR
PMQAXC	Pointer to a data structure of type MQAXC
PMAXP	Pointer to a data structure of type MAXP
PMQBMHO	Pointer to a data structure of type MQBMHO
PMQBO	Pointer to a data structure of type MQBO
PMQBOOL	Pointer to data of type MQBOOL
PMQBYTE	Pointer to data of type MQBYTE
PMQBYTEn	Pointer to data of type MQBYTEn
PMQCBC	Pointer to a data structure of type MQCBC
PMQCBD	Pointer to a data structure of type MQCBD
PMQCHAR	Pointer to a data structure of type MQCHAR
PMQCHARV	Pointer to a data structure of type MQCHARV
PMQCHARn	Pointer to data of type MQCHARn
PMQCIH	Pointer to a data structure of type MQCIH
PMQCMHO	Pointer to a data structure of type MQCMHO
PMQCNO	Pointer to a data structure of type MQCNO
PMQCSP	Pointer to a data structure of type MQCSP
PMQCTLO	Pointer to a data structure of type MQCTLO
PMQDH	Pointer to a data structure of type MQDH

Table 315. Elementary data types (continued)

Data type	Representation
PMQDHO	Pointer to a data structure of type MQDHO
PMQDLH	Pointer to a data structure of type MQDLH
PMQDMHO	Pointer to a data structure of type MQDMHO
PMQDMPO	Pointer to a data structure of type MQDMPO
PMQEPH	Pointer to a data structure of type MQEPH
PMQFLOAT32	Pointer to data of type MQFLOAT32
PMQFLOAT64	Pointer to data of type MQFLOAT64
PMQFUNC	Pointer to a function
PMQGMO	Pointer to a data structure of type MQGMO
PMQHCONFIG	Pointer to data of type MQHCONFIG
PMQHCONN	Pointer to data of type MQHCONN
PMQHMSG	Pointer to data of type MQHMSG
PMQHOBJ	Pointer to data of type MQHOBJ
PMQIIH	Pointer to a data structure of type MQIIH
PMQIMPO	Pointer to a data structure of type MQIMPO
PMQINT8	Pointer to data of type MQINT8
PMQINT16	Pointer to data of type MQINT16
PMQINT32	Pointer to data of type MQINT32
PMQINT64	Pointer to data of type MQINT64
PMQLONG	Pointer to data of type MQLONG
PMQMD	Pointer to a data structure of type MQMD
PMQMDE	Pointer to a data structure of type MQMDE
PMQMD1	Pointer to a data structure of type MQMD1
PMQMD2	Pointer to a data structure of type MQMD2
PMQMHBO	Pointer to a data structure of type MQMHBO
PMQOD	Pointer to a data structure of type MQOD
PMQOR	Pointer to a data structure of type MQOR
PMQPD	Pointer to a data structure of type MQPD
PMQPID	Pointer to a process identifier MQPID
PMQPMO	Pointer to a data structure of type MQPMO
PMQPTR	Pointer to data of type MQPTR
PMQRFH	Pointer to a data structure of type MQRFH
PMQRFH2	Pointer to a data structure of type MQRFH2
PMQRMH	Pointer to a data structure of type MQRMH
PMQRR	Pointer to a data structure of type MQRR
PMQSCO	Pointer to a data structure of type MQSCO
PMQSD	Pointer to a data structure of type MQSD
PMQSMPO	Pointer to a data structure of type MQSMPO
PMQSRO	Pointer to a data structure of type MQSRO
PMQSTS	Pointer to a data structure of type MQSTS

Table 315. Elementary data types (continued)

Data type	Representation
PMQTID	Pointer to a thread identifier MQTID
PMQTM	Pointer to a data structure of type MQTM
PMQTMC2	Pointer to a data structure of type MQTMC2
PMQUINT8	Pointer to data of type MQUINT8
PMQUINT16	Pointer to data of type MQUINT16
PMQUINT32	Pointer to data of type MQUINT32
PMQUINT64	Pointer to data of type MQUINT64
PMQULONG	Pointer to data of type MQULONG
PMQVOID	Pointer
PMQWIH	Pointer to a data structure of type MQWIH
PMQXQH	Pointer to a data structure of type MQXQH

MQBOOL on IBM i: IBM i

The MQBOOL data type represents a boolean value. The value 0 represents false. Any other value represents true.

An MQBOOL must be aligned as for the MQLONG data type.

MQBYTE on IBM i: IBM i

The MQBYTE data type represents a single byte of data.

No particular interpretation is placed on the byte-it is treated as a string of bits, and not as a binary number or character. No special alignment is required.

An array of MQBYTE is sometimes used to represent an area of main storage with a nature that is not known to the queue manager. For example, the area might contain application message data or a structure. The boundary alignment of this area must be compatible with the nature of the data contained within it.

MQBYTEn (String of n bytes) on IBM i: IBM i

Each MQBYTEn data type represents a string of *n* bytes.

Where *n* can take one of the following values:

- 16, 24, 32, or 64.

Each byte is described by the MQBYTE data type. No special alignment is required.

If the data in the string is shorter than the defined length of the string, the data must be padded with nulls to fill the string.

When the queue manager returns byte strings to the application (for example, on the MQGET call), the queue manager always pads with nulls to the defined length of the string.

Constants are available that define the lengths of byte string fields.

MQCHAR (character) on IBM i: IBM i

The MQCHAR data type represents a single character.

The coded character set identifier of the character is that of the queue manager (see the **CodedCharSetId** attribute in topic CodedCharSetId). No special alignment is required.

Note: Application message data specified on the MQGET, MQPUT, and MQPUT1 calls is described by the MQBYTE data type, not the MQCHAR data type.

MQCHARn (String of n characters) on IBM i: IBM i

Each MQCHARn data type represents a string of *n* characters.

Where *n* can take one of the following values:

- 4, 8, 12, 16, 20, 28, 32, 48, 64, 128, or 256

Each character is described by the MQCHAR data type. No special alignment is required.

If the data in the string is shorter than the defined length of the string, the data must be padded with blanks to fill the string. In some cases a null character can be used to end the string prematurely, instead of padding with blanks; the null character and characters following it are treated as blanks, up to the defined length of the string. The places where a null can be used are identified in the call and data type descriptions.

When the queue manager returns character strings to the application (for example, on the MQGET call), the queue manager always pads with blanks to the defined length of the string; the queue manager does not use the null character to delimit the string.

Constants are available that define the lengths of character string fields.

MQFLOAT32 on IBM i: IBM i

The MQFLOAT32 data type is a 32-bit floating-point number represented using the standard IEEE floating-point format.

An MQFLOAT32 must be aligned on a 4-byte boundary.

MQFLOAT64 on IBM i: IBM i

The MQFLOAT64 data type is a 64-bit floating-point number represented using the standard IEEE floating-point format.

An MQFLOAT64 must be aligned on an 8-byte boundary.

MQHCONFIG - configuration handle:

The MQHCONFIG data type represents a configuration handle, that is, the component that is being configured for a particular installable service. A configuration handle must be aligned on its natural boundary.

Note: Applications must test variables of this type for equality only.

MQHCONN (Connection handle) on IBM i: 

The MQHCONN data type represents a connection handle, that is, the connection to a particular queue manager.

A connection handle must be aligned on its natural boundary.

Note: Applications must test variables of this type for equality only.

MQHMSG (Message handle) on IBM i: 

The **MQHMSG** data type represents a message handle that gives access to a message.

A message handle must be aligned on an 8-byte boundary.

Note: Applications must test variables of this type for equality only.

MQHOBJ (Object handle) on IBM i: 

The MQHOBJ data type represents an object handle that gives access to an object.

An object handle must be aligned on its natural boundary.

Note: Applications must test variables of this type for equality only.

MQINT8 (8-bit signed integer) on IBM i: 

The MQINT8 data type is an 8-bit signed integer that can take any value in the range -128 to +127, unless otherwise restricted by the context.

MQINT16 (16-bit signed integer) on IBM i: 

The MQINT16 data type is a 16-bit signed integer that can take any value in the range -32 768 to +32 767, unless otherwise restricted by the context.

An MQINT16 must be aligned on a 2-byte boundary.

MQINT32 (32-bit integer) on IBM i: 

The MQINT32 data type is a 32-bit signed integer.

It is equivalent to MQLONG.

MQINT64 (64-bit integer) on IBM i: 

The MQINT64 data type is a 64-bit signed integer that can take any value in the range -9 223 372 036 854 775 808 through +9 223 372 036 854 775 807, unless otherwise restricted by the context.

For COBOL, the valid range is limited to -999 999 999 999 999 999 through +999 999 999 999 999 999. An MQINT64 should be aligned on an 8-byte boundary.

MQLONG (Long integer) on IBM i: 

The MQLONG data type is a 32-bit signed binary integer that can take any value in the range -2 147 483 648 through +2 147 483 647, unless otherwise restricted by the context, aligned on its natural boundary.

MQPID - process identifier:

The IBM MQ process identifier.

This is the same identifier used in IBM MQ trace and FFST dumps, but might be different from the operating system process identifier.

MQPTR - pointer:

The MQPTR data type is the address of data of any type. A pointer must be aligned on its natural boundary; this is a 16-byte boundary on IBM i.

Some programming languages support typed pointers; the MQI also uses these in a few cases.

MQTID - thread identifier:

The MQ thread identifier.

This is the same identifier used in MQ trace and FFST dumps, but might be different from the operating system thread identifier.

MQUINT8 (8-bit unsigned integer) on IBM i: 

The MQUINT8 data type is an 8-bit unsigned integer that can take any value in the range 0 to +255, unless otherwise restricted by the context.

MQUINT16 - 16-bit unsigned integer:

The MQUINT16 data type is a 16-bit unsigned integer that can take any value in the range 0 through +65 535, unless otherwise restricted by the context.

An MQUINT16 must be aligned on a 2-byte boundary.

MQUINT32 (32-bit unsigned integer) on IBM i: 

The MQUINT32 data type is a 32-bit unsigned integer. It is equivalent to MQULONG.

MQUINT64 - 64-bit unsigned integer:

The MQUINT64 data type is a 64-bit unsigned integer that can take any value in the range 0 through +18 446 744 073 709 551 615 unless otherwise restricted by the context.

For COBOL, the valid range is limited to 0 through +999 999 999 999 999 999. An MQUINT64 should be aligned on a 8-byte boundary.

MQULONG - 32-bit unsigned integer:

The MQULONG data type is a 32-bit unsigned binary integer that can take any value in the range 0 through +4 294 967 294, unless otherwise restricted by the context.

An MQULONG must be aligned on a 4-byte boundary.

PMQACH - pointer to a data structure of type MQACH:

A pointer to a data structure of type MQACH.

PMQAIR - pointer to a data structure of type MQAIR:

A pointer to a data structure of type MQAIR.

PMQAXC - pointer to a data structure of type MQAXC:

A pointer to a data structure of type MQAXC.

PMQAXP - pointer to a data structure of type MQAXP:

A pointer to a data structure of type MQAXP.

PMQBMHO - pointer to a data structure of type MQBMHO:

A pointer to a data structure of type MQBMHO.

PMQBO - pointer to a data structure of type MQBO:

A pointer to a data structure of type MQBO.

PMQBOOL - pointer to data of type MQBOOL:

A pointer to data of type MQBOOL.

A pointer to data of type MQBOOL.

PMQBYTE - pointer to a data type of MQBYTE:

A pointer to a data type of MQBYTE.

PMQBYTEn - pointer to a data structure of type MQBYTEn:

A pointer to a data structure of type MQBYTEn, where n can be 8, 12, 16, 24, 32, 40, 48 or 128.

PMQCBC - pointer to a data structure of type MQCBC:

A pointer to a data structure of type MQCBC.

PMQCBD - pointer to a data structure of type MQCBD:

A pointer to a data structure of type MQCBD.

PMQCHAR - pointer to data of type MQCHAR:

A pointer to data of type MQCHAR.

PMQCHARV - pointer to a data structure of type MQCHARV:

A pointer to a data structure of type MQCHARV.

PMQCHARn - pointer to a data type of MQCHARn:

A pointer to a data type of MQCHARn, where n can be 4, 8, 12, 20, 28, 32, 64, 128, 256, 264.

PMQCIH - pointer to a data structure of type of MQCIH:

A pointer to a data structure of type of MQCIH.

PMQCMHO - pointer to a data structure of type MQCMHO:

A pointer to a data structure of type MQCMHO.

PMQCNO - pointer to a data structure of type of MQCNO:

A pointer to a data structure of type of MQCNO.

PMQCSP - pointer to a data structure of type MQCSP:

A pointer to a data structure of type MQCSP.

PMQCTLO - pointer to a data structure of type MQCTLO:

A pointer to a data structure of type MQCTLO.

PMQDH - pointer to a data structure of type MQDH:

A pointer to a data structure of type MQDH.

PMQDHO - pointer to a data structure of type MQDHO:

A pointer to a data structure of type MQDHO.

PMQDLH - pointer to a data structure of type of MQDLH:

A pointer to a data structure of type of MQDLH.

PMQDMHO - pointer to a data structure of type MQDMHO:

A pointer to a data structure of type MQDMHO.

PMQDMPO - pointer to a data structure of type MQDMPO:

A pointer to a data structure of type MQDMPO.

A pointer to a data structure of type MQDMPO.

PMQEPH - pointer to a data structure of type MQEPH:

A pointer to a data structure of type MQEPH.

PMQFLOAT32 - pointer to data of type MQFLOAT32:

A pointer to data of type MQFLOAT32.

PMQFLOAT64 - pointer to data of type MQFLOAT64:

A pointer to data of type MQFLOAT64.

PMQFUNC - pointer to a function:

A pointer to a function.

PMQGMO - pointer to a data structure of type MQGMO:

A pointer to a data structure of type MQGMO.

PMQHCONFIG - pointer to a data type of MQHCONFIG:

A pointer to a data type of MQHCONFIG.

PMQHCONN - pointer to a data type of MQHCONN:

A pointer to a data type of MQHCONN.

PMQHMSG - pointer to a data type of MQHMSG:

A pointer to a data type of MQHMSG.

PMQHOBJ - pointer to data of type MQHOBJ:

A pointer to data of type MQSMPO.

PMQIIH - pointer to a data structure of type MQIIH:

A pointer to a data structure of type MQIIH.

PMQIMPO - pointer to a data structure of type MQIMPO:

A pointer to a data structure of type MQIMPO.

PMQINT8 - pointer to data of type MQINT8:

A pointer to data of type MQINT8.

PMQINT16 - pointer to data of type MQINT16:

A pointer to data of type MQINT16.

PMQINT32 (Pointer to data of type MQINT32) on IBM i: 

The PMQINT32 data type is a pointer to data of type MQINT32. It is equivalent to PMQLONG.

PMQINT64 (Pointer to data of type MQINT64) on IBM i: 

The PMQINT64 data type is a pointer to data of type MQINT64.

PMQLONG - pointer to data of type MQLONG:

A pointer to data of type MQLONG.

PMQMD - pointer to structure of type MQMD:

A pointer to structure of type MQMD.

PMQMDE - pointer to a data structure of type MQMDE:

A pointer to a data structure of type MQMDE.

PMQMDI - pointer to a data structure of type MQMDI:

A pointer to a data structure of type MQMDI.

PMQMD2 - pointer to a data structure of type MQMD2:

A pointer to a data structure of type MQMD2

PMQMHBO - pointer to a data structure of type MQMHBO:

A pointer to a data structure of type MQMHBO.

PMQOD - pointer to a data structure of type MQOD:

A pointer to a data structure of type MQOD.

PMQOR - pointer to a data structure of type MQOR:

A pointer to a data structure of type MQOR.

PMQPD - pointer to a data structure of type MQPD:

A pointer to a data structure of type MQPD.

PMQPID - pointer to a process identifier:

A pointer to a process identifier.

PMQPMO - pointer to a data structure of type MQPMO:

A pointer to a data structure of type MQPMO.

PMQPTR - pointer to data of type MQPTR:

A pointer to data of type MQPTR.

PMQRFH - pointer to a data structure of type MQRFH:

A pointer to a data structure of type MQRFH.

PMQRFH2 - pointer to a data structure of type MQRFH2:

A pointer to a data structure of type MQRFH2.

.

PMQRMH - pointer to a data structure of type MQRMH:

A pointer to a data structure of type MQRMH.

PMQRR - pointer to a data structure of type MQRR:

A pointer to a data structure of type MQRR.

PMQSCO - pointer to a data structure of type MQSCO:

A pointer to a data structure of type MQSCO.

.

PMQSD - pointer to a data structure of type MQSD:

A pointer to a data structure of type MQSD.

PMQSMPO - pointer to a data structure of type MQSMPO:

A pointer to a data structure of type MQSMPO.

PMQSRO - pointer to a data structure of type MQSRO:

A pointer to a data structure of type MQSRO.

PMQSTS - pointer to a data structure of type MQSTS:

A pointer to a data structure of type MQSTS.

PMQTID - pointer to a data structure of type MQTID:

A pointer to a data structure of type MQTID.

PMQTM - pointer to a data structure of type MQTM:

A pointer to a data structure of type MQTM.

PMQTMC2 - pointer to a data structure of type MQTMC2:

A pointer to a data structure of type MQTMC2.

PMQUINT8 - pointer to data of type MQUINT8:

A pointer to data of type MQUINT8.

PMQUINT16 - pointer to data of type MQUINT16:

A pointer to data of type MQUINT16.

PMQUINT32 (Pointer to data of type MQUINT32) on IBM i: 

The PMQUINT32 data type is a pointer to data of type MQUINT32. It is equivalent to PMQULONG.

PMQUINT64 (Pointer to data of type MQUINT64) on IBM i: 

The PMQUINT64 data type is a pointer to data of type MQUINT64.

PMQULONG - pointer to data of type MQULONG:

A pointer to data of type MQULONG.

PMQVOID - pointer:

A pointer.

PMQWIH - pointer to a data structure of type MQWIH:

A pointer to a data structure of type MQWIH.

PMQXQH - pointer to a data structure of type MQXQH:

A pointer to a data structure of type MQXQH.

Language considerations:

This topic contains information to help you use the MQI from the RPG programming language.

Some of these language considerations are:
- "COPY files"
- "Calls" on page 3017
- "Call parameters" on page 3017
- "Structures" on page 3017
- "Named constants" on page 3017
- "MQI procedures" on page 3017
- "Threading considerations" on page 3018
- "Commitment control" on page 3018
- "Coding the bound calls" on page 3018
- "Notational conventions" on page 3019

COPY files

Various COPY files are provided to assist with the writing of RPG application programs that use message queuing. There are three sets of COPY files:
- COPY files with names ending with the letter *G* are for use with programs that use static linkage. These files are initialized with the exceptions stated in "Structures" on page 3017.
- COPY files with names ending with the letter *H* are for use with programs that use static linkage, but are **not** initialized.
- COPY files with names ending with the letter *R* are for use with programs that use dynamic linkage. These files are initialized with the exceptions stated in "Structures" on page 3017.

The COPY files reside in QRPGLESRC in the QMQM library.

For each set of COPY files, there are two files containing named constants, and one file for each of the structures. The COPY files are summarized in Table 316.

Table 316. RPG COPY files

File name (static linkage, initialized, CMQ*G)	File name (static linkage, not initialized, CMQ*H)	File name (dynamic linkage, initialized, CMQ*R)	Contents
CMQBOG	CMQBOH	-	Begin options structure
CMQCDG	CMQCDH	CMQCDR	Channel definition structure
CMQCFBFG	CMQCFBFH	-	PCF bit filter parameter
CMQCFG	-	-	Constants for PCF and events
CMQCFBSG	CMQCFBSH	-	PCF byte string
CMQCFGRG	CMQCFGRH	-	PCF group parameter
CMQCFIFG	CMQCFIFH	-	PCF integer filter parameter
CMQCFHG	CMQCFHH	-	PCF header
CMQCFILG	CMQCFILH	-	PCF integer list parameter structure

Table 316. RPG COPY files (continued)

File name (static linkage, initialized, CMQ*G)	File name (static linkage, not initialized, CMQ*H)	File name (dynamic linkage, initialized, CMQ*R)	Contents
CMQCFING	CMQCFINH	-	PCF integer parameter structure
CMQCFSFG	CMQCFSFH	-	PCF string filter parameter
CMQCFSLG	CMQCFSLH	-	PCF string list parameter structure
CMQCFSTG	CMQCFSTH	-	PCF string parameter structure
CMQCFXLG	CMQCFXLH	-	PCF short name for CFIL64
CMQCFXNG	CMQCFXNH	-	PCF short name for CFIN64
CMQCIHG	CMQCIHH	-	CICS information header structure
CMQCNOG	CMQCNOH	-	Connect options structure
CMQCSPG	CMQCSPH	-	Security parameters
CMQCXPG	CMQCXPH	CMQCXPR	Channel exit parameter structure
CMQDHG	CMQDHH	CMQDHR	Distribution header structure
CMQDLHG	CMQDLHH	CMQDLHR	Dead letter header structure
CMQDXPG	CMQDXPH	CMQDXPR	Data conversion exit parameter structure
CMQEPHG	CMQEPHH	-	Embedded PCF header structure
CMQG	-	CMQR	Named constants for main MQI
CMQGMOG	CMQGMOH	CMQGMOR	Get message options structure
CMQIIHG	CMQIIHH	CMQIIHR	IMS information header structure
CMQMDEG	CMQMDEH	CMQMDER	Message descriptor extension structure
CMQMDG	CMQMDH	CMQMDR	Message descriptor structure
CMQMD1G	CMQMD1H	CMQMD1R	Message descriptor structure version 1
CMQMD2G	CMQMD2H	-	Message descriptor structure version 2
CMQODG	CMQODH	CMQODR	Object descriptor structure
CMQORG	CMQORH	CMQORR	Object record structure
CMQPMOG	CMQPMOH	CMQPMOR	Put message options structure
CMQPSG	-	-	Constants for publish/subscribe
CMQRFHG	CMQRFHH	-	Rules and formatting header structure
CMQRFH2G	CMQRFH2H	-	Rules and formatting header 2 structure
CMQRMHG	CMQRMHH	CMQRMHR	Reference message header structure
CMQRRG	CMQRRH	CMQRRR	Response record structure
CMQTMCG	CMQTMCH	CMQTMCR	Trigger message structure (character format)
CMQTMC2G	CMQTMC2H	CMQTMC2R	Trigger message structure (character format) version 2
CMQTMG	CMQTMH	CMQTMR	Trigger message structure
CMQWIHG	CMQWIHH	-	Work information header structure
CMQXG	-	CMQXR	Named constants for data conversion exit
CMQXQHG	CMQXQHH	CMQXQHR	Transmission queue header structure

Calls

Calls are described using their individual names.

Call parameters

Some parameters passed to the MQI can have more than one concurrent function. This is because the integer value passed is often tested on the setting of individual bits within the field, and not on its total value. This allows you to 'add' several functions together and pass them as a single parameter.

Structures

All IBM MQ structures are defined with initial values for the fields, with the following exceptions:

- Any structure with a suffix of H.
- MQTMC
- MQTMC2

These initial values are defined in the relevant table for each structure.

The structure declarations do not contain `DS` statements. This allows the application to declare either a single data structure or a multiple-occurrence data structure, by coding the `DS` statement and then using the `/COPY` statement to copy in the remainder of the declaration:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7
D* Declare an MQMD data structure with 5 occurrences
DMYMD        DS                          5
D/COPY CMQMDR
```

Named constants

There are many integer and character values that provide data interchange between your application program and the queue manager. To facilitate a more readable and consistent approach to using these values, named constants are defined for them. You can use these named constants and not the values they represent, as this improves the readability of the program source code.

When the COPY file CMQG is included in a program to define the constants, the RPG compiler will issue many severity-zero messages for the constants that are not used by the program; these messages are benign, and can safely be ignored.

MQI procedures

When using the ILE bound calls, you must bind to the MQI procedures when you create your program. These procedures are exported from the following service programs as appropriate:

QMQM/LIBMQM
: This service program contains the single-threaded bindings for version 5.1 and above. See the following section for special considerations when writing threaded applications.

QMQM/LIBMQM_R
: This service program contains the multi-threaded bindings for version 5.1 and above. See the following section for special considerations when writing threaded applications.

QMQM/LIBMQIC
: This service program is for binding non-threaded client applications.

QMQM/LIBMQIC_R
: This service program is for binding threaded client applications.

Use the CRTPGM command to create your programs. For example, the following command creates a single-threaded program that uses the ILE bound calls:

```
CRTPGM PGM(MYPROGRAM) BNDSRVPGM(QMQM/LIBMQM)
```

Threading considerations

The RPG compiler used for IBM i is part of the WebSphere Development Toolset and WebSphere Development Studio for IBM i and is known as the ILE RPG IV Compiler.

In general, RPG programs should not use the multi-threaded service programs. Exceptions are RPG programs created using the ILE RPG IV Compiler, and containing the THREAD(*SERIALIZE) keyword in the control specification. However, even though these programs are thread safe, careful consideration must be given to the overall application design, as THREAD(*SERIALIZE) forces serialization of RPG procedures at the module level, and this might have an adverse affect on overall performance.

Where RPG programs are used as data-conversion exits, they must be made thread-safe, and should be recompiled using the version 4.4 ILE RPG compiler or above, with THREAD(*SERIALIZE) specified in the control specification.

For further information about threading, see the *IBM i IBM MQ Development Studio: ILE RPG Reference*, and the *IBM i IBM MQ Development Studio: ILE RPG Programmer's Guide*.

Commitment control

The MQI syncpoint functions MQCMIT and MQBACK are available to ILE RPG programs running in normal mode; these calls allow the program to commit and back out changes to MQ resources.

Coding the bound calls

MQI ILE procedures are listed in Table 317.

Table 317. ILE RPG bound calls supported by each service program

Name of call	LIBMQM and LIBMQM_R	LIBMQIC and LIBMQIC_R
MQBACK	Y	Y
MQBEGIN	Y	Y
MQCMIT	Y	Y
MQCLOSE	Y	Y
MQCONN	Y	Y
MQCONNX	Y	Y
MQDISC	Y	Y
MQGET	Y	Y
MQINQ	Y	Y
MQOPEN	Y	Y
MQPUT	Y	Y
MQPUT1	Y	Y
MQSET	Y	Y
MQXCNVC	Y	Y

To use these procedures you need to:

1. Define the external procedures in your 'D' specifications. These are all available within the COPY file member CMQG containing the named constants.
2. Use the CALLP operation code to call the procedure along with its parameters.

For example the MQOPEN call requires the inclusion of the following code:

```
D*********************************************************************
D**  MQOPEN Call -- Open Object  (From COPY file CMQG)           **
D*********************************************************************
D*
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQOPEN           PR                  EXTPROC('MQOPEN')
D* Connection handle
D HCONN                          10I 0 VALUE
D* Object descriptor
D OBJDSC                        224A
D* Options that control the action of MQOPEN
D OPTS                           10I 0 VALUE
D* Object handle
D HOBJ                           10I 0
D* Completion code
D CMPCOD                         10I 0
D* Reason code qualifying CMPCOD
D REASON                         10I 0
D*
```

To call the procedure, after initializing the various parameters, you need the following code:

```
...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+....8
    C                   CALLP     MQOPEN(HCONN : MQOD : OPTS : HOBJ :
    C                             CMPCOD : REASON)
```

Here, the structure MQOD is defined using the COPY member CMQODG which breaks it down into its components.

Notational conventions

The latter topics in this section show how the:

- Calls should be invoked
- Parameters should be declared
- Various data types should be declared

In a number of cases, parameters are arrays or character strings with a size that is not fixed. For these, a lowercase “n” is used to represent a numeric constant. When the declaration for that parameter is coded, the “n” must be replaced by the numeric value required.

MQAIR (Authentication information record) on IBM i: IBM i

The MQAIR structure represents the authentication information record.

Overview

Purpose: The MQAIR structure allows an application running as an IBM MQ client to specify information about an authenticator that is to be used for the client connection. The structure is an input parameter on the MQCONNX call.

Character set and encoding: Data in MQAIR must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT.

Fields

The MQAIR structure contains the following fields; the fields are described in **alphabetical order**:

AICN (10-digit signed integer)

This is either the host name or the network address of a host on which the LDAP server is running. This can be followed by an optional port number, enclosed in parentheses.

If the value is shorter than the length of the field, terminate the value with a null character, or pad it with blanks to the length of the field. If the value is not valid, the call fails with reason code RC2387.

The default port number is 389.

This is an input field. The length of this field is given by LNAICN. The initial value of this field is blank characters.

AITYP (10-digit signed integer)

This is the type of authentication information contained in the record.

The value must be:

AITLDP
Certificate revocation using LDAP server.

If the value is not valid, the call fails with reason code RC2386.

This is an input field. The initial value of this field is AITLDP.

AIPW (10-digit signed integer)

This is the password needed to access the LDAP CRL server.

If the value is shorter than the length of the field, terminate the value with a null character, or pad it with blanks to the length of the field. If the LDAP server does not require a password, or you omit the LDAP user name, *AIPW* must be null or blank. If you omit the LDAP user name and *AIPW* is not null or blank, the call fails with reason code RC2390.

This is an input field. The length of this field is given by LNLDPW. The initial value of this field blank characters.

AILUL (10-digit signed integer)

This is the length in bytes of the LDAP user name addressed by the *AILUP* or *AILUO* field. The value must be in the range zero through LNDISN. If the value is not valid, the call fails with reason code RC2389.

If the LDAP server involved does not require a user name, set this field to zero.

This is an input field. The initial value of this field is 0.

AILUO (10-digit signed integer)

This is the offset in bytes of the LDAP user name from the start of the MQAIR structure.

The offset can be positive or negative. The field is ignored if *LDAPUserNameLength* is zero.

You can use either *LDAPUserNamePtr* or *LDAPUserNameOffset* to specify the LDAP user name, but not both; see the description of the *LDAPUserNamePtr* field for details.

This is an input field. The initial value of this field is 0.

AILUP (10-digit signed integer)

This is the LDAP user name.

It consists of the Distinguished Name of the user who is attempting to access the LDAP CRL server. If the value is shorter than the length specified by *AILUL*, terminate the value with a null character, or pad it with blanks to the length *AILUL*. The field is ignored if *AILUL* is zero.

You can supply the LDAP user name in one of two ways:

- By using the pointer field *AILUP*

  In this case, the application can declare a string that is separate from the MQAIR structure, and set *AILUP* to the address of the string.

  Consider using *AILUP* for programming languages that support the pointer data type in a fashion that is portable to different environments (for example, the C programming language).
- By using the offset field *AILUO*

  In this case, the application must declare a compound structure containing the MQSCO structure followed by the array of MQAIR records followed by the LDAP user name strings, and set *AILUO* to the offset of the appropriate name string from the start of the MQAIR structure. Ensure that this value is correct, and has a value that can be accommodated within an MQLONG (the most restrictive programming language is COBOL, for which the valid range is -999 999 999 through +999 999 999).

  Consider using *AILUO* for programming languages that do not support the pointer data type, or that implement the pointer data type in a fashion that might not be portable to different environments (for example, the COBOL programming language).

Whichever technique is chosen, use only one of *AILUP* and *AILUO* ; the call fails with reason code RC2388.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

AISID (10-digit signed integer)

The value must be:

AISIDV
: Identifier for the authentication information record.

This is always an input field. The initial value of this field is AISIDV.

AIVER (10-digit signed integer)

The value must be:

AIVER1
Version-1 authentication information record.

The following constant specifies the version number of the current version:

AIRVERC
Current version of authentication information record.

This is always an input field. The initial value of this field is AIVER1.

Initial values

Table 318. Initial values of fields in MQAIR for MQAIR

Field name	Name of constant	Value of constant
AISID	AISIDV	'AIR¬'
AIVER	AIVERC	1
AITYP	AITLDP	1
AICN	None	Null string or blanks
AILUP	None	Null pointer or null bytes
AILUO	None	0
AILUL	None	0
AIPW	None	Null string or blanks

Notes:

1. The symbol ¬ represents a single blank character.

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQAIR Structure
D*
D* Structure identifier
D  AISID                  1      4    INZ('AIR ')
D* Structure version number
D  AIVER                  5      8I 0 INZ(1)
D* Type of authentication information
D  AITYP                  9     12I 0 INZ(1)
D* Connection name of CRL LDAP server
D  AICN                  13    276    INZ
D* Address of LDAP user name
D  AILUP                277    292*   INZ(*NULL)
D* Offset of LDAP user name from start of MQAIR structure
D  AILUO                293    296I 0 INZ(0)
D* Length of LDAP user name
D  AILUL                297    300I 0 INZ(0)
D* Password to access LDAP server
D  AIPW                 301    332    INZ
```

MQBMHO (Buffer to message handle options) on IBM i: IBM i

Structure defining the buffer to message handle options.

Overview

Purpose: The MQBMHO structure allows applications to specify options that control how message handles are produced from buffers. The structure is an input parameter on the MQBUFMH call.

Character set and encoding: Data in MQBMHO must be in the character set of the application and encoding of the application (ENNAT).

- "Fields"
- "Initial values"
- "RPG declaration" on page 3024

Fields

The MQBMHO structure contains the following fields; the fields are described in **alphabetical order**:

BMSID (10-digit signed integer)

Buffer to message handle structure - StrucId field.

This is the structure identifier. The value must be:

BMSIDV
Identifier for buffer to message handle structure.

This is always an input field. The initial value of this field is BMSIDV.

BMVER (10-digit signed integer)

Buffer to message handle structure - Version field.

This is the structure version number. The value must be:

BMVER1
Version number for buffer to message handle structure.

The following constant specifies the version number of the current version:

BMVERVC
Current version of buffer to message handle structure.

This is always an input field. The initial value of this field is BMVER1.

BMOPT (10-digit signed integer)

Buffer to message handle structure - Options field.

The value can be:

BMDLPR
Properties that are added to the message handle are deleted from the buffer. If the call fails no properties are deleted.

Default options: If you do not need the option described, use the following option:

BMNONE
No options specified.

This is always an input field. The initial value of this field is BMDLPR.

Initial values

Table 319. Initial values of fields in MQBMHO

Field name	Name of constant	Value of constant
BMSID	BMSIDV	'BMHO'
BMVER	BMVER1	1
BMOPT	BMNONE	0

RPG declaration

```
D* MQBMHO Structure
D*
D*
D* Structure identifier
D  BMSID                  1      4    INZ('BMHO')
D*
D* Structure version number
D  BMVER                  5      8I 0 INZ(1)
D*
D* Options that control the action of MQBUFMH
D  BMOPT                  9     12I 0 INZ(1)
```

MQBO (Begin options) on IBM i: IBM i

The MQBO structure allows the application to specify options relating to the creation of a unit of work.

Overview

Purpose: The structure is an input/output parameter on the MQBEGIN call.

Character set and encoding: Data in MQBO must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT.

- "Fields"
- "Initial values" on page 3025
- "RPG declaration" on page 3025

Fields

The MQBO structure contains the following fields; the fields are described in **alphabetical order**:

BOOPT (10-digit signed integer)

Options that control the action of MQBEGIN.

The value must be:

BONONE
No options specified.

This is always an input field. The initial value of this field is BONONE.

BOSID (4-byte character string)

Structure identifier.

The value must be:

BOSIDV
Identifier for begin-options structure.

This is always an input field. The initial value of this field is BOSIDV.

BOVER (10-digit signed integer)

Structure version number.

The value must be:

BOVER1
Version number for begin-options structure.

The following constant specifies the version number of the current version:

BOVERC
Current version of begin-options structure.

This is always an input field. The initial value of this field is BOVER1.

Initial values

Table 320. Initial values of fields in MQBO

Field name	Name of constant	Value of constant
BOSID	BOSIDV	'BO¬¬'
BOVER	BOVER1	1
BOOPT	BONONE	0

Notes:

1. The symbol ¬ represents a single blank character.

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQBO Structure
D*
D* Structure identifier
D  BOSID                  1      4    INZ('BO  ')
D* Structure version number
D  BOVER                  5      8I 0 INZ(1)
D* Options that control the action of MQBEGIN
D  BOOPT                  9     12I 0 INZ(0)
```

MQCBC (Callback context) on IBM i: IBM i

Structure describing the callback routine.

Overview

Purpose
The MQCBC structure is used to specify context information that is passed to a callback function.

The structure is an input/output parameter on the call to a message consumer routine.

Version
The current version of MQCBC is CBCV2.

Character set and encoding
Data in MQCBC is in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT. However, if the application is running as an IBM MQ client, the structure is in the character set and encoding of the client.

- "Fields" on page 3026
- "Initial values" on page 3031
- "RPG declaration" on page 3031

Fields

The MQCBC structure contains the following fields; the fields are described in alphabetical order:

CBCBUFFLEN (10 digit signed integer)

The buffer can be larger than both the MaxMsgLength value defined for the consumer and the ReturnedLength value in the MQGMO.

Callback context structure - BufferLength field.

This is the length in bytes of the message buffer that has been passed to this function.

The actual message length is supplied in DataLength field.

The application can use the entire buffer for its own purposes for the duration of the callback function.

This is an input field to the message consumer function; it is not relevant to an exception handler function.

CBCCALLBA (10 digit signed integer)

Callback context structure - CallbackArea field.

This is a field that is available for the callback function to use.

The queue manager makes no decisions based on the contents of this field and it is passed unchanged from the CBDCALLBA field in the MQCBD structure, which is a parameter on the MQCB call used to define the callback function.

Changes to the *CBCCALLBA* are preserved across the invocations of the callback function for an *CBCHOBJ*. This field is not shared with callback functions for other handles.

This is an input/output field to the callback function. The initial value of this field is a null pointer or null bytes.

CBCCALLT (10 digit signed integer)

Callback Context structure - CallType field.

Field containing information about why this function has been called. The following call types are defined.

Message delivery call types: These call types contain information about a message. The **CBCLEN** and **CBCBUFFLEN** parameters are valid for these call types.

CBCTMR

The message consumer function has been invoked with a message that has been destructively removed from the object handle.

If the value of *CBCCC* is CCWARN, the value of the *Reason* field is RC2079 or one of the codes indicating a data conversion problem.

CBCTMN

The message consumer function has been invoked with a message that has not yet been destructively removed from the object handle. The message can be destructively removed from the object handle using the *MsgToken*.

The message might not have been removed because:

- The MQGMO options requested a browse operation, GMBR*
- The message is larger than the available buffer and the MQGMO options do not specify gmatm

If the value of *CBCCC* is CCWARN, the value of the *Reason* field is RC2080 or one of the codes indicating a data conversion problem.

Callback control call types: These call types contain information about the control of the callback and do not contain details about a message. These call types are requested using CBDOPT in the MQCBD structure.

The **CBCLEN** and **CBCBUFFLEN** parameters are not valid for these call types.

CBCTRC
The purpose of this call type is to allow the callback function to perform some initial setup.

The callback function is invoked is immediately after the callback is registered, that is, upon return from an MQCB call using a value for the *Operation* field of CBREG.

This call type is used both for message consumers and event handlers.

If requested, this is the first invocation of the callback function.

The value of the *CBCREA* field is RCNONE.

CBCTSC
The purpose of this call type is to allow the callback function to perform some setup when it is started, for example, reinstating resources that were cleaned up when it was previously stopped.

The callback function is invoked when the connection is started using either CTLSR or CTLSW.

If a callback function is registered within another callback function, this call type is invoked when the callback returns.

This call type is used for message consumers only.

The value of the *CBCREA* field is RCNONE.

CBCTTC
The purpose of this call type is to allow the callback function to perform some cleanup when it is stopped for a while, for example, cleaning up additional resources that have been acquired during the consuming of messages.

The callback function is invoked when an MQCTL call is issued using a value for the *Operation* field of CTLSP.

This call type is used for message consumers only.

The value of the *CBCREA* field is set to indicate the reason for stopping.

CBCTDC
The purpose of this call type is to allow the callback function to perform final cleanup at the end of the consume process. The callback function is invoked when the:

- Callback function is deregistered using an MQCB call with BCUNR.
- Queue is closed, causing an implicit deregister. In this instance the callback function is passed HOUNUH as the object handle.
- MQDISC call completes - causing an implicit close and, therefore, a deregister. In this case the connection is not disconnected immediately, and any ongoing transaction is not yet committed.

If any of these actions are taken inside the callback function itself, the action is invoked once the callback returns.

This call type is used both for message consumers and event handlers.

If requested, this is the last invocation of the callback function.

The value of the *CBCREA* field is set to indicate the reason for stopping.

CBCTEC

Event handler function

The event handler function has been invoked without a message when:

- An MQCTL call is issued with a value for the *Operation* field of CTLSP, or
- The queue manager or connection stops or quiesces.

This call can be used to take appropriate action for all callback functions.

- **Message consumer function**

  The message consumer function has been invoked without a message when an error (*CBCCC* = CCFAIL) has been detected that is specific to the object handle; for example *CBCREA* code = RC2016 .

The value of the *CBCREA* field is set to indicate the reason for the call.

This is an input field. CBCTMR and CMCTMN are applicable only to message consumer functions.

CBCCC (10 digit signed integer)

Callback context structure - CompCode field.

This is the completion code. It indicates whether there were any problems consuming the message; it is one of the following:

CCOK
Successful completion

CCWARN
Warning (partial completion)

CCFAIL
Call failed

This is an input field. The initial value of this field is CCOK.

CBCCONNAREA (10 digit signed integer)

Callback context structure - ConnectionArea field.

This is a field that is available for the callback function to use.

The queue manager makes no decisions based on the contents of this field and it is passed unchanged from the ConnectionArea field in the MQCTLO structure, which is a parameter on the MQCTL call used to control the callback function.

Any changes made to this field by the callback functions are preserved across the invocations of the callback function. This area can be used to pass information that is to be shared by all callback functions. Unlike *CallbackArea*, this area is common across all callbacks for a connection handle.

This is an input and output field. The initial value of this field is a null pointer or null bytes.

CBCLEN (10 digit signed integer)

This is the length in bytes of the application data in the message. If the value is zero, it means that the message contains no application data.

The CBCLEN field contains the length of the message but not necessarily the length of the message data passed to the consumer. It could be that the message was truncated. Use the GMRL field in the MQGMO to determine how much data has been passed to the consumer.

If the reason code indicates the message has been truncated, you can use the CBCLEN field to determine how large the actual message is. This allows you to determine the size of the buffer

required to accommodate the message data, and then issue an MQCB call to update the CBDMML in the MQCBD with an appropriate value.

If the GMCONV option is specified, the converted message could be larger than the value returned for DataLength. In such cases, the application probably needs to issue an MQCB call to update the CBDMML in the MQCBD to be greater than the value returned by the queue manager for DataLength.

To avoid message truncation problems, specify MaxMsgLength as CBDFM. This causes the queue manager to allocate a buffer for the full message length after data conversion. Be aware, however, that even if this option is specified, it is still possible that sufficient storage is not available to correctly process the request. Applications should always check the returned reason code. For example, if it is not possible to allocate sufficient storage to convert the message, the messages is returned to the application unconverted.

This is an input field to the message consumer function; it is not relevant to an event handler function.

CBCFLG (10 digit signed integer)

Flags containing information about this consumer.

The following option is defined:

CBCFBE

This flag can be returned if a previous MQCLOSE call using the COQSC option failed with a reason code of RC2458.

This code indicated that the last read ahead message is being returned and that the buffer is now empty. If the application issues another MQCLOSE call using the COQSC option, it succeeds.

Note, that an application is not guaranteed to be given a message with this flag set, as there might still be messages in the read-ahead buffer that do not match the current selection criteria. In this instance, the consumer function is invoked with the reason code `RC2019` .

If the read ahead buffer is empty, the consumer is invoked with the CBCFBE flag and the reason code RC2518.

This is an input field to the message consumer function; it is not relevant to an event handler function.

CBCHOBJ (10 digit signed integer)

Callback context structure - CBCHOBJ field.

For a call to a message consumer, this is the handle for the object relating to the message consumer.

For an event handler, this value is HONONE

The application can use this handle and the message token in the Get Message Options block to get the message if a message has not been removed from the queue.

This is always an input field. The initial value of this field is HOUNUH

CBCRCD (10 digit signed integer)

`CBCRCD` indicates how long the queue manager waits before trying to reconnect. The field can be modified by an event handler to change the delay or stop reconnection altogether.

Use the **`CBCRCD`** field only if the value of the **`Reason`** field in the Callback Context is `RC2545`.

On entry to the event handler the value of **`CBCRCD`** is the number of milliseconds the queue manager is going to wait before making a reconnection attempt. Table 321 on page 3030 lists the

values that you can set to modify the behavior of the queue manager on return from the event handler.

Table 321. **CBCRCD** *values*

Value	Description
-1	Make no more reconnection attempts. An error is returned to the application.
0	Try to reconnect immediately.
>0	Wait for this many milliseconds before trying the connection again.

CBCREA (10 digit signed integer)

Callback context structure - Reason field.

This is the reason code qualifying the *CBCCC*

This is an input field. The initial value of this field is RCNONE.

CBCSTATE (10 digit signed integer)

An indication of the state of the current consumer. This field is of most value to an application when a nonzero reason code is passed to the consumer function.

You can use this field to simplify application programming because you do not need to code behavior for each reason code.

This is an input field. The initial value of this field is CSNONE

State	Queue manager action	Value of constant
CSNONE This reason code represents a normal call with no additional reason information	None; this is the normal operation.	0
CSSUST These reason codes represent temporary conditions.	The callback routine is called to report the condition and then suspended. After a period the system might attempt the operation again, which can lead to the same condition being raised again.	1
CSSUSU These reason codes represent conditions where the callback needs to act to resolve the condition.	The consumer is suspended and the callback routine is called to report the condition. The callback routine should resolve the condition if possible and either RESUME or close down the connection.	2
CSSUS These reason codes represent failures that prevent further message callbacks.	The queue manager automatically suspends the callback function. If the callback function is resumed it is likely to receive the same reason code again.	3
CSSTOP These reason codes represent the end of message consumption.	Delivered to the exception handler and to callbacks that specified CBDTC. No further messages can be consumed.	4

CBCSID (10 digit signed integer)

Callback context structure - StrucId field.

This is the structure identifier; the value must be:

CBCSI
Identifier for callback context structure.

This is always an input field. The initial value of this field is CBCSI.

CBCVER (10 digit signed integer)

Callback context structure - Version field.

This is the structure version number; the value must be:

CBCV1
Version-1 callback context structure.

The following constant specifies the version number of the current version:

CBCCV
Current version of the callback context structure.

This is always an input field. The initial value of this field is CBCV1.

Initial values

Table 322. Initial values of fields in MQCBC

Field name	Name of constant	Value of constant
CBCSID	CBCSI	'CBC¬'
CBCVER	CBCV1	1
CBCCALLT	None	0
CBCHOBJ	HOUNUH	-1
CBCCALLBA	None	Null pointer or null bytes
CBCCONNAREA	None	Null pointer or null bytes
CBCCC	CCOK	0
CBCREA	RCNONE	0
CBCSTATE	CSNONE	0
CBCLEN	None	0
CBCBUFFLEN	None	0
CBCFLG	None	0
CBCRCD	none	0

Note:

1. The symbol ¬ represents a single blank character.

RPG declaration

```
D* MQCBC Structure
D*
D*
D* Structure identifier
D  CBCSID                 1      4    INZ('CBC ')
D*
D* Structure version number
D  CBCVER                 5      8I 0 INZ(1)
D*
D* Why Function was called
D  CBCCALLT               9     12I 0 INZ(0)
D*
D* Object Handle
D  CBCHOBJ               13     16I 0 INZ(-1)
D*
D* Callback data passed to the function
D  CBCCALLBA             17     32*   INZ(*NULL)
```

```
D*
D* MQCTL Data area passed to the function
D  CBCCONNAREA           33     48*   INZ(*NULL)
D*
D* Completion Code
D  CBCCC                 49     52I 0 INZ(0)
D*
D* Reason Code
D  CBCREA                53     56I 0 INZ(0)
D*
D* Consumer State
D  CBCSTATE              57     60I 0 INZ(0)
D*
D* Message Data Length
D  CBCLEN                61     64I 0 INZ(0)
D*
D* Buffer Length
D  CBCBUFFLEN            65     68I 0 INZ(0)
D*
** Flags containing information about
D* this consumer
D  CBCFLG                69     72I 0 INZ(0)
D* Ver:1 **
D* Number of milliseconds before reconnect attempt
D  CBCRCD                73     76I 0 INZ(0)
D* Ver:2 **
D*
```

MQCBD (Callback descriptor) on IBM i: IBM i

Structure specifying the callback function.

Overview

Purpose: The MQCBD structure is used to specify a callback function and the options controlling its use by the queue manager.

The structure is an input parameter on the MQCB call.

Version: The current version of MQCBD is CBDV1.

Character set and encoding: Data in MQCBD must be in the character set and encoding of the local queue manager; these are given by the **CodedCharSetId** queue manager attribute and ENNAT. However, if the application is running as an IBM MQ MQI client, the structure must be in the character set and encoding of the client.

- "Fields"
- "Initial values" on page 3036
- "RPG declaration" on page 3036

Fields

The MQCBD structure contains the following fields; the fields are described in **alphabetical order**:

CBDCALLBA (10-digit signed integer)

This is a field that is available for the callback function to use.

The queue manager makes no decisions based on the contents of this field and it is passed unchanged from the field in the MQCBD structure, which is a parameter on the callback function declaration.

The value is used only on an *Operation* having a value CBREG, with no currently defined callback, it does not replace a previous definition.

This is an input and output field to the callback function. The initial value of this field is a null pointer or null bytes.

CBDCALLBF (10-digit signed integer)

The callback function is invoked as a function call.

Use this field to specify a pointer to the callback function.

You must specify either *CallbackFunction* or *CallbackName*. If you specify both, the reason code RC2486 is returned.

If neither *CallbackName* nor *CallbackFunction* is not set, the call fails with the reason code RC2486.

This option is not supported in the following environments:

- CICS on z/OS
- Programming languages and compilers that do not support function-pointer references

In such situations, the call fails with the reason code RC2486.

This is an input field. The initial value of this field is a null pointer or null bytes.

CBDCALLBN (10-digit signed integer)

The callback function is invoked as a dynamically linked program.

You must specify either *CallbackFunction* or *CallbackName*. If you specify both, the reason code RC2486 is returned.

If either *CallbackName* or *CallbackFunction* is not true, the call fails with the reason code RC2486.

The module is loaded when the first callback routine to use is registered, and unloaded when the last callback routine to use it deregisters.

Except where noted in the following text, the name is left-aligned within the field, with no embedded blanks; the name itself is padded with blanks to the length of the field. In the descriptions that follow, square brackets ([]) denote optional information:

IBMi The callback name can be one of the following formats:

- Library "/" Program
- Library "/" ServiceProgram "("FunctionName")"

For example, `MyLibrary/MyProgram(MyFunction)`.

The library name can be *LIBL. Both the library and program names are limited to a maximum of 10 characters.

UNIX The callback name is the name of a dynamically loadable module or library, suffixed with the name of a function residing in that library. The function name must be enclosed in parentheses. The library name can optionally be prefixed with a directory path:

```
[path]library(function)
```

If the path is not specified the system search path is used.

The name is limited to a maximum of 128 characters.

Windows

The callback name is the name of a dynamic-link library, suffixed with the name of a function residing in that library. The function name must be enclosed in parentheses. The library name can optionally be prefixed with a directory path and drive:

```
[d:][path]library(function)
```

If the drive and path are not specified the system search path is used.

The name is limited to a maximum of 128 characters.

z/OS The callback name is the name of a load module that is valid for specification on the EP parameter of the LINK or LOAD macro.

The name is limited to a maximum of 8 characters.

z/OS CICS

The callback name is the name of a load module that is valid for specification on the PROGRAM parameter of the EXEC CICS LINK command macro.

The name is limited to a maximum of 8 characters.

The program can be defined as remote using the REMOTESYTEM option of the installed PROGRAM definition or by the dynamic routing program.

The remote CICS region must be connected to IBM MQ if the program is to use IBM MQ API calls. Note, however, that the field in the MQCBC structure is not valid in a remote system.

If a failure occurs trying to load *CallbackName*, one of the following error codes is returned to the application:

- RC2495
- RC2496
- RC2497

A message is also written to the error log containing the name of the module for which the load was attempted, and the failing reason code from the operating system.

This is an input field. The initial value of this field is a null string or blanks.

CBDCALLBT (10-digit signed integer)

This is the type of the callback function. The value must be one of:

CBTMC

Defines this callback as a message consumer function.

A message consumer callback function is called when a message, meeting the selection criteria specified, is available on an object handle and the connection is started.

CBTEH

Defines this callback as the asynchronous event routine; it is not driven to consume messages for a handle.

Hobj is not required on the MQCB call defining the event handler and is ignored if specified.

The event handler is called for conditions that affect the whole message consumer environment. The consumer function is invoked without a message when an event, for example, a queue manager or connection stopping, or quiescing, occurs. It is not called for conditions that are specific to a single message consumer, for example, RC2016.

Events are delivered to the application, regardless of whether the connection is started or stopped, except in the following environments:

- CICS on z/OS environment
- nonthreaded applications

If the caller does not pass one of these values, the call fails with a reason code of RC2483

This is always an input field. The initial value of this field is CBTMC.

CBDMML (10-digit signed integer)

This is the length in bytes of the longest message that can be read from the handle and given to the callback routine. If a message has a longer length, the callback routine receives *MaxMsgLength* bytes of the message, and reason code:

- RC2080 or
- RC2079 if you specified GMATM.

The actual message length is supplied in the "CBCLEN (10 digit signed integer)" on page 3028 field of the MQCBC structure.

The following special value is defined:

CBDFM
The buffer length is adjusted by the system to return messages without truncation.

If insufficient memory is available to allocate a buffer to receive the message, the system calls the callback function with an RC2071 reason code.

If, for example, you request data conversion, and there is insufficient memory available to convert the message data, the unconverted message is passed to the callback function.

This is an input field. The initial value of the *MaxMsgLength* field is CBDFM.

CBDOPT (10-digit signed integer)

Callback descriptor structure - Options field.

Any one, or all, of the following can be specified. To specify more than one option, either add the values together (do not add the same constant more than once), or combine the values using the bitwise OR operation (if the programming language supports bit operations). Combinations that are not valid are noted; any other combinations are valid.

CBDFQ
The MQCB call fails if the queue manager is in the quiescing state.

On z/OS, this option also forces the MQCB call to fail if the connection (for a CICS or IMS application) is in the quiescing state.

Specify GMFIQ, in the MQGMO options passed on the MQCB call, to cause notification to message consumers when they are quiescing.

Control options: The following options control whether the callback function is called, without a message, when the state of the consumer changes:

CBDRC
The callback function is invoked with call type CBCTRC

.

CBDSC
The callback function is invoked with call type CBCTSC.

CBDTC
The callback function is invoked with call type CBCTTC.

CBDDC
The callback function is invoked with call type CBCTDC.

See "CBCCALLT (10 digit signed integer)" on page 3026 for further details about these call types.

Default option: If you do not need any of the options described, use the following option:

CBDNO
Use this value to indicate that no other options have been specified; all options assume their default values.

CBDNO is defined to aid program documentation; it is not intended that this option is used with any other, but as its value is zero, such use cannot be detected.

This is an input field. The initial value of the *Options* field is CBDNO.

CBDSID (10-digit signed integer)

Callback descriptor structure - StrucId field.

This is the structure identifier; the value must be:

CBDSI
Identifier for callback descriptor structure.

This is always an input field. The initial value of this field is CBDSI.

CBDVER (10-digit signed integer)

Callback descriptor structure - Version field.

This is the structure version number; the value must be:

CBDV1
Version-1 callback descriptor structure.

The following constant specifies the version number of the current version:

CBDCV
Current version of callback descriptor structure.

This is always an input field. The initial value of this field is CBDV1.

Initial values

Table 323. Initial values of fields in MQCBD

Field name	Name of constant	Value of constant
StrucId	CBDSI	'CBD¬'
Version	CBDV1	1
CallBackType	CBTMC	1
Options	CBDNO	0
CallbackArea	None	Null bytes
CallbackFunction	None	Null bytes
CallbackName	None	Blanks
MaxMsgLength	CBDFM	-1

Note:

1. The symbol ¬ represents a single blank character.

RPG declaration

```
D* MQCBD Structure
D*
D*
D* Structure identifier
D  CBDSID                 1      4    INZ('CBD ')
D*
D* Structure version number
D  CBDVER                 5      8I 0 INZ(1)
D*
D* Callback function type
D  CBDCALLBT              9     12I 0 INZ(1)
```

```
D*
** Options controlling message
D* consumption
D  CBDOPT                13     16I 0 INZ(0)
D*
D* User data passed to the function
D  CBDCALLBA             17     32*
D*
D* FP: Callback function pointer
D  CBDCALLBF             33     48*
D*
D* Callback name
D  CBDCALLBN             49    176    INZ('\0')
D*
D* Maximum message length
D  CBDMML               177    180I 0 INZ(-1)
```

MQCHARV (Variable Length String) on IBM i: 

Use the MQCHARV structure to describe a variable length string.

Overview

Character set and encoding: Data in the MQCHARV must be in the encoding of the local queue manager that is given by ENNAT and the character set of the VCHRC field within the structure. If the application is running as an IBM MQ MQI client, the structure must be in the encoding of the client. Some character sets have a representation that depends on the encoding. If VCHRC is one of these character sets, the encoding used is the same encoding as that of the other fields in the MQCHARV. The character set identified by VSCCSID can be a double-byte character set (DBCS).

Usage: The MQCHARV structure addresses data that might be discontiguous with the structure containing it. To address this data, fields declared with the pointer data type can be used.

- "Fields"
- "Initial values" on page 3038
- "RPG declaration" on page 3039
- "Redefinition of CSAPL" on page 3039

Fields

The MQCHARV structure contains the following fields; the fields are described in **alphabetical order**:

VCHRC (10-digit signed integer)

This is the character set identifier of the variable length string addressed by the VCHRP or VCHRO field.

The initial value of this field is CSAPL. This is defined by IBM MQ to indicate that it should be changed by the queue manager to the true character set identifier of the queue manager. This is in the same way as CSQM behaves. As a result, the value CSAPL is never associated with a variable length string. The initial value of this field can be changed by defining a different value for the constant CSAPL for your compile unit by the appropriate means for your application's programming language.

VCHRL (10-digit signed integer)

The length in bytes of the variable length string addressed by the VCHRP or VCHRO field.

The initial value of this field is 0. The value must be either greater than or equal to zero or the following special value which is recognized:

VSNLT

If VSNLT is not specified, VCHRL bytes are included as part of the string. If null characters are present they do not delimit the string.

If VSNLT is specified, the string is delimited by the first null encountered in the string. The null itself is not included as part of that string.

Note: The null character used to terminate a string if VSNLT is specified is a null from the code set specified by VCHRC.

For example, in UTF-16 V 9.0.0 (CCSIDs 1200, 13488, and 17584), this is the 2-byte Unicode encoding where a null is represented by a 16 bit number of all zeros. In UTF-16 it is common to find single bytes set to all zero which are part of characters (7-bit ASCII characters for example), but the strings will only be null terminated when two 'zero' bytes are found on an even byte boundary. It is possible to get two 'zero' bytes on an odd boundary when they are each part of valid characters. For example, x'01' x'00' x'00' x'30' represents two valid Unicode characters and does not null terminate the string.

VCHRO (10-digit signed integer)

The offset in bytes of the variable length string from the start of the MQCHARV, or the structure containing it.

When the MQCHARV structure is embedded within another structure, this value is the offset in bytes of the variable length string from the start of the structure that contains this MQCHARV structure. When the MQCHARV structure is not embedded within another structure, for example, if it is specified as a parameter on a function call, the offset is relative to the start of the MQCHARV structure.

The offset can be positive or negative. You can use either the VCHRP or VCHRO field to specify the variable length string, but not both.

The initial value of this field is 0.

VCHRP (pointer)

This is a pointer to the variable length string.

You can use either the VCHRP or VCHRO field to specify the variable length string, but not both.

The initial value of this field is a null pointer or null bytes.

VCHRS (10-digit signed integer)

The size in bytes of the buffer addressed by the VCHRP or VCHRO field.

When the MQCHARV structure is used as an output field on a function call, this field must be initialized with the length of the buffer provided. If the value of VCHRL is greater than VCHRS then only VCHRS bytes of data will be returned to the caller in the buffer.

The value must be greater than or equal to zero or the following special value which is recognized:

VSUSL
If VSUSL is specified, the length of the buffer is taken from the VCHRL field in the MQCHARV structure. This special value is not appropriate when the structure is used as an output field and a buffer is provided. This is the initial value of this field.

Initial values

Field name	Name of constant	Value of constant
VCHRP	None	Null pointer or null bytes.
VCHRO	None	0
VCHRS	VSUSL	-1
VCHRL	None	0
VCHRC	CSAPL	-3

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQCHARV Structure
D*
D* Address of variable length string
D VCHRP                   1     16*
D* Offset of variable length string
D VCHRO                  17     20I 0
D* Size of buffer
D VCHRS                  21     24I 0
D* Length of variable length string
D VCHRL                  25     28I 0
D* CCSID of variable length string
D VCHRC                  29     32I 0
```

Redefinition of CSAPL

Unlike the programming languages supported on other platforms, RPG does not have a way of redefining a defined constant, so you must set each VCHRC specifically if you want to use a value other than CSAPL.

MQCIH (CICS bridge header) on IBM i: 

The MQCIH structure describes the information that can be present at the start of a message sent to the CICS bridge through IBM MQ for z/OS.

Overview

Format name: FMCICS.

Version: The current version of MQCIH is CIVER2. Fields that exist only in the more-recent version of the structure are identified as such in the descriptions that follow.

The COPY file provided contains the most recent version of MQCIH, with the initial value of the *CIVER* field set to CIVER2.

Character set and encoding: Special conditions apply to the character set and encoding used for the MQCIH structure and application message data:

- Applications that connect to the queue manager that owns the CICS bridge queue must provide an MQCIH structure that is in the character set and encoding of the queue manager. This is because data conversion of the MQCIH structure is not performed in this case.
- Applications that connect to other queue managers can provide an MQCIH structure that is in any of the supported character sets and encodings; conversion of the MQCIH is performed by the receiving message channel agent connected to the queue manager that owns the CICS bridge queue.

Note: There is one exception to this. If the queue manager that owns the CICS bridge queue is using CICS for distributed queuing, the MQCIH must be in the character set and encoding of the queue manager that owns the CICS bridge queue.

- The application message data following the MQCIH structure must be in the same character set and encoding as the MQCIH structure. The *CICSI* and *CIENC* fields in the MQCIH structure cannot be used to specify the character set and encoding of the application message data.

  A data-conversion exit must be provided by the user to convert the application message data if the data is not one of the built-in formats supported by the queue manager.

Usage: If the values required by the application are the same as the initial values shown in Table 325 on page 3049, and the bridge is running with AUTH=LOCAL or AUTH=IDENTIFY, the MQCIH structure can be omitted from the message. In all other cases, the structure must be present.

The bridge accepts either a version-1 or a version-2 MQCIH structure, but for 3270 transactions a version-2 structure must be used.

The application must ensure that fields documented as "request" fields have appropriate values in the message sent to the bridge; these fields are input to the bridge.

Fields documented as "response" fields are set by the CICS bridge in the reply message that the bridge sends to the application. Error information is returned in the *CIRET*, *CIFNC*, *CICC*, *CIREA*, and *CIAC* fields, but not all of them are set in all cases. Table 324 shows which fields are set for different values of *CIRET*.

Table 324. Contents of error information fields in MQCIH structure

CIRET	*CIFNC*	*CICC*	*CIREA*	*CIAC*
CRC000	-	-	-	-
CRC003	-	-	FBC*	-
CRC002 CRC008	IBM MQ call name	IBM MQ *CMPCOD*	IBM MQ *REASON*	-
CRC001 CRC006 CRC007 CRC009	CICS EIBFN	CICS EIBRESP	CICS EIBRESP2	-
CRC004 CRC005	-	-	-	CICS ABCODE

- "Fields"
- "Initial values" on page 3049
- "RPG declaration" on page 3050

Fields

The MQCIH structure contains the following fields; the fields are described in **alphabetical order**:

CIAC (4-byte character string)

Abend code.

The value returned in this field is significant only if the *CIRET* field has the value CRC005 or CRC004. If it does, *CIAC* contains the CICS ABCODE value.

This is a response field. The length of this field is given by LNABNC. The initial value of this field is 4 blank characters.

This is an indicator specifying whether ADS descriptors should be sent on SEND and RECEIVE BMS requests. The following values are defined:

ADNONE
Do not send or receive ADS descriptor.

ADSEND
Send ADS descriptor.

ADRECV
Receive ADS descriptor.

ADMSGF
Use message format for the ADS descriptor.

This causes the ADS descriptor to be sent or received using the long form of the ADS descriptor. The long form has fields that are aligned on 4-byte boundaries.

The *CIADS* field should be set as follows:

- If ADS descriptors are not being used, set the field to ADNONE.
- If ADS descriptors *are* being used, and with the *same* CCSID in each environment, set the field to the sum of ADSEND and ADRECV.
- If ADS descriptors *are* being used, but with *different* CCSIDs in each environment, set the field to the sum of ADSEND, ADRECV, and ADMSGF.

This is a request field used only for 3270 transactions. The initial value of this field is ADNONE.

CIADS (10-digit signed integer)

Send/receive ADS descriptor.

This is an indicator specifying whether ADS descriptors should be sent on SEND and RECEIVE BMS requests. The following values are defined:

ADNONE
Do not send or receive ADS descriptor.

ADSEND
Send ADS descriptor.

ADRECV
Receive ADS descriptor.

ADMSGF
Use message format for the ADS descriptor.

This causes the ADS descriptor to be sent or received using the long form of the ADS descriptor. The long form has fields that are aligned on 4-byte boundaries.

The *CIADS* field should be set as follows:

- If ADS descriptors are not being used, set the field to ADNONE.
- If ADS descriptors *are* being used, and with the *same* CCSID in each environment, set the field to the sum of ADSEND and ADRECV.
- If ADS descriptors *are* being used, but with *different* CCSIDs in each environment, set the field to the sum of ADSEND, ADRECV, and ADMSGF.

This is a request field used only for 3270 transactions. The initial value of this field is ADNONE.

CIAI (4-byte character string)

AID key.

This is the initial value of the AID key when the transaction is started. It is a 1-byte value, left-aligned.

This is a request field used only for 3270 transactions. The length of this field is given by LNATID. The initial value of this field is 4 blanks.

CIAUT (8-byte character string)

Password or passticket.

This is a password or passticket. If user-identifier authentication is active for the CICS bridge, *CIAUT* is used with the user identifier in the MQMD identity context to authenticate the sender of the message.

This is a request field. The length of this field is given by LNAUTH. The initial value of this field is 8 blanks.

CICC (10-digit signed integer)

IBM MQ completion code or CICS EIBRESP.

The value returned in this field is dependent on *CIRET* ; see Table 324 on page 3040.

This is a response field. The initial value of this field is CCOK.

CICNC (4-byte character string)

Abend transaction code.

This is the abend code to be used to terminate the transaction (normally a conversational transaction that is requesting more data). Otherwise this field is set to blanks.

This is a request field used only for 3270 transactions. The length of this field is given by LNCNCL. The initial value of this field is 4 blanks.

CICP (10-digit signed integer)

Cursor position.

This is the initial cursor position when the transaction is started. Later, for conversational transactions, the cursor position is in the RECEIVE vector.

This is a request field used only for 3270 transactions. The initial value of this field is 0. This field is not present if *CIVER* is less than CIVER2.

CICSI (10-digit signed integer)

Reserved.

This is a reserved field; its value is not significant. The initial value of this field is 0.

CICT (10-digit signed integer)

Whether task can be conversational.

This is an indicator specifying whether the task should be allowed to issue requests for more information, or should abend. The value must be one of the following:

CTYES
Task is conversational.

CTNO
Task is not conversational.

This is a request field used only for 3270 transactions. The initial value of this field is CTNO.

CIENC (10-digit signed integer)

Reserved.

This is a reserved field; its value is not significant. The initial value of this field is 0.

CIEO (10-digit signed integer)

Offset of error in message.

This is the position of invalid data detected by the bridge exit. This field provides the offset from the start of the message to the location of the invalid data.

This is a response field used only for 3270 transactions. The initial value of this field is 0. This field is not present if *CIVER* is less than CIVER2.

CIFAC (8-byte bit string)

Bridge facility token.

This is an 8-byte bridge facility token. The purpose of a bridge facility token is to allow multiple transactions in a pseudoconversation to use the same bridge facility (virtual 3270 terminal). In the first, or only, message in a pseudoconversation, a value of FCNONE should be set; this tells CICS to allocate a new bridge facility for this message. A bridge facility token is returned in response messages when a nonzero *CIFKT* is specified on the input message. Subsequent input messages can then use the same bridge facility token.

The following special value is defined:

FCNONE
No BVT token specified.

This is both a request and a response field used only for 3270 transactions. The length of this field is given by LNFAC. The initial value of this field is FCNONE.

CIFKT (10-digit signed integer)

Bridge facility release time.

This is the length of time in seconds that the bridge facility will be kept after the user transaction has ended. For nonconversational transactions, the value should be zero.

This is a request field used only for 3270 transactions. The initial value of this field is 0.

CIFL (4-byte character string)

Terminal emulated attributes.

This is the name of an installed terminal that is to be used as a model for the bridge facility. A value of blanks means that *CIFL* is taken from the bridge transaction profile definition, or a default value is used.

This is a request field used only for 3270 transactions. The length of this field is given by LNFACL. The initial value of this field is 4 blanks.

CIFLG (10-digit signed integer)

Flags.

The value must be:

CIFNON
No flags.

This is a request field. The initial value of this field is CIFNON.

CIFMT (8-byte character string)

IBM MQ format name of data that follows MQCIH.

This specifies the IBM MQ format name of the data that follows the MQCIH structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *MDFMT* field in MQMD.

This format name is also used for the reply message, if the *CIRFM* field has the value FMNONE.

- For DPL requests, *CIFMT* must be the format name of the COMMAREA.
- For 3270 requests, *CIFMT* must be `CSQCBDCI`, and *CIRFM* must be `CSQCBDCO`.

The data-conversion exits for these formats must be installed on the queue manager where they are to run.

If the request message results in the generation of an error reply message, the error reply message has a format name of FMSTR.

This is a request field. The length of this field is given by LNFMT. The initial value of this field is FMNONE.

CIFNC (4-byte character string)

IBM MQ call name or CICS EIBFN function.

The value returned in this field is dependent on *CIRET* ; see Table 324 on page 3040. The following values are possible when *CIFNC* contains an IBM MQ call name:

CFCONN
MQCONN call.

CFGET
MQGET call.

CFINQ
MQINQ call.

CFOPEN
MQOPEN call.

CFPUT
MQPUT call.

CFPUT1
MQPUT1 call.

CFNONE
No call.

This is a response field. The length of this field is given by LNFUNC. The initial value of this field is CFNONE.

CIGWI (10-digit signed integer)

Wait interval for MQGET call issued by bridge task.

This field is applicable only when *CIUOW* has the value CUFRST. It allows the sending application to specify the approximate time in milliseconds that the MQGET calls issued by the bridge should wait for second and subsequent request messages for the unit of work started by this message. This overrides the default wait interval used by the bridge. The following special values may be used:

WIDFLT
Default wait interval.

This causes the CICS bridge to wait for the period specified when the bridge was started.

WIULIM
Unlimited wait interval.

This is a request field. The initial value of this field is WIDFLT.

CIII (10-digit signed integer)

Reserved.

This is a reserved field. The value must be 0. This field is not present if *CIVER* is less than CIVER2.

CILEN (10-digit signed integer)

Length of MQCIH structure.

The value must be one of the following:

CILEN1
Length of version-1 CICS information header structure.

CILEN2
Length of version-2 CICS information header structure.

The following constant specifies the length of the current version:

CILENC
Length of current version of CICS information header structure.

This is a request field. The initial value of this field is CILEN2.

CILT (10-digit signed integer)

Link type.

This indicates the type of object that the bridge should try to link. The value must be one of the following:

LTPROG
DPL program.

LTTRAN
3270 transaction.

This is a request field. The initial value of this field is LTPROG.

CINTI (4-byte character string)

Next transaction to attach.

This is the name of the next transaction returned by the user transaction (typically by EXEC CICS RETURN TRANSID). If there is no next transaction, this field is set to blanks.

This is a response field used only for 3270 transactions. The length of this field is given by LNTRID. The initial value of this field is 4 blanks.

CIODL (10-digit signed integer)

Output COMMAREA data length.

This is the length of the user data to be returned to the client in a reply message. This length includes the 8-byte program name. The length of the COMMAREA passed to the linked program is the maximum of this field and the length of the user data in the request message, minus 8.

Note: The length of the user data in a message is the length of the message *excluding* the MQCIH structure.

If the length of the user data in the request message is smaller than *CIODL*, the DATALENGTH option of the LINK command is used; this allows the LINK to be function-shipped efficiently to another CICS region.

The following special value can be used:

OLINPT
Output length is same as input length.

This value might be needed even if no reply is requested, in order to ensure that the COMMAREA passed to the linked program is of sufficient size.

This is a request field used only for DPL programs. The initial value of this field OLINPT.

CIREA (10-digit signed integer)

IBM MQ reason or feedback code, or CICS EIBRESP2.

The value returned in this field is dependent on *CIRET* ; see Table 324 on page 3040.

This is a response field. The initial value of this field is RCNONE.

CIRET (10-digit signed integer)

Return code from bridge.

This is the return code from the CICS bridge describing the outcome of the processing performed by the bridge. The *CIFNC*, *CICC*, *CIREA*, and *CIAC* fields may contain additional information (see Table 324 on page 3040). The value is one of the following:

CRC000
(0, X'000') No error.

CRC001
(1, X'001') EXEC CICS statement detected an error.

CRC002
(2, X'002') IBM MQ call detected an error.

CRC003
(3, X'003') CICS bridge detected an error.

CRC004
(4, X'004') CICS bridge ended abnormally.

CRC005
(5, X'005') Application ended abnormally.

CRC006
(6, X'006') Security error occurred.

CRC007
(7, X'007') Program not available.

CRC008
(8, X'008') Second or later message within current unit of work not received within specified time.

CRC009
(9, X'009') Transaction not available.

This is a response field. The initial value of this field is CRC000.

CIRFM (8-byte character string)

IBM MQ format name of reply message.

This is the IBM MQ format name of the reply message that will be sent in response to the current message. The rules for coding this are the same as those for the *MDFMT* field in MQMD.

This is a request field used only for DPL programs. The length of this field is given by LNFMT. The initial value of this field is FMNONE.

CIRSI (4-byte character string)

Reserved.

This is a reserved field. The value must be 4 blanks. The length of this field is given by LNRSID.

CIRS1 (8-byte character string)

Reserved.

This is a reserved field. The value must be 8 blanks.

CIRS2 (8-byte character string)

Reserved.

This is a reserved field. The value must be 8 blanks.

CIRS3 (8-byte character string)

Reserved.

This is a reserved field. The value must be 8 blanks.

CIRS4 (10-digit signed integer)

Reserved.

This is a reserved field. The value must be 0. This field is not present if *CIVER* is less than CIVER2.

CIRTI (4-byte character string)

Reserved.

This is a reserved field. The value must be 4 blanks. The length of this field is given by LNTRID.

CISC (4-byte character string)

Transaction start code.

This is an indicator specifying whether the bridge emulates a terminal transaction or a START transaction. The value must be one of the following:

SCSTRT
Start.

SCDATA
Start data.

SCTERM
Terminate input.

SCNONE
None.

In the response from the bridge, this field is set to the start code appropriate to the next transaction ID contained in the *CINTI* field. The following start codes are possible in the response:

- SCSTRT
- SCDATA
- SCTERM

For CICS Transaction Server Version 1.2, this field is a request field only; its value in the response is undefined.

For CICS Transaction Server Version 1.3 and subsequent releases, this is both a request and a response field.

This field is used only for 3270 transactions. The length of this field is given by LNSTCO. The initial value of this field is SCNONE.

CISID (4-byte character string)

Structure identifier.

The value must be:

CISIDV
Identifier for CICS information header structure.

This is a request field. The initial value of this field is CISIDV.

CITES (10-digit signed integer)

Status at end of task.

This field shows the status of the user transaction at end of task. One of the following values is returned:

TENOSY
Not synchronized.

The user transaction has not yet completed and has not syncpointed. The *MDMT* field in MQMD is MTRQST in this case.

TECMIT
Commit unit of work.

The user transaction has not yet completed, but has syncpointed the first unit of work. The *MDMT* field in MQMD is MTDGRM in this case.

TEBACK
Back out unit of work.

The user transaction has not yet completed. The current unit of work will be backed out. The *MDMT* field in MQMD is MTDGRM in this case.

TEENDT
End task.

The user transaction has ended (or abended). The *MDMT* field in MQMD is MTRPLY in this case.

This is a response field used only for 3270 transactions. The initial value of this field is TENOSY.

CITI (4-byte character string)

Transaction to attach.

If *CILT* has the value LTTRAN, *CITI* is the transaction identifier of the user transaction to be run; a nonblank value must be specified in this case.

If *CILT* has the value LTPROG, *CITI* is the transaction code under which all programs within the unit of work are to be run. If the value specified is blank, the CICS DPL bridge default transaction code (CKBP) is used. If the value is nonblank, it must have been defined to CICS as a local TRANSACTION with an initial program of CSQCBP00. This field is applicable only when *CIUOW* has the value CUFRST or CUONLY.

This is a request field. The length of this field is given by LNTRID. The initial value of this field is 4 blanks.

CIUOW (10-digit signed integer)

Unit-of-work control.

This controls the unit-of-work processing performed by the CICS bridge. You can request the bridge to run a single transaction, or one or more programs within a unit of work. The field indicates whether the CICS bridge should start a unit of work, perform the requested function within the current unit of work, or end the unit of work by committing it or backing it out. Various combinations are supported, to optimize the data transmission flows.

The value must be one of the following:

CUONLY
Start unit of work, perform function, then commit the unit of work (DPL and 3270).

CUCONT
Additional data for the current unit of work (3270 only).

CUFRST
Start unit of work and perform function (DPL only).

CUMIDL
Perform function within current unit of work (DPL only).

CULAST
Perform function, then commit the unit of work (DPL only).

CUCMIT
Commit the unit of work (DPL only).

CUBACK
Back out the unit of work (DPL only).

This is a request field. The initial value of this field is CUONLY.

CIVER (10-digit signed integer)

Structure version number.

The value must be one of the following:

CIVER1
Version-1 CICS information header structure.

CIVER2
Version-2 CICS information header structure.

Fields that exist only in the more-recent version of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

CIVERC
Current version of CICS information header structure.

This is a request field. The initial value of this field is CIVER2.

Initial values

Table 325. Initial values of fields in MQCIH

Field name	Name of constant	Value of constant
CISID	CISIDV	'CIH¬'
CIVER	CIVER2	2
CILEN	CILEN2	180
CIENC	None	0
CICSI	None	0
CIFMT	FMNONE	Blanks
CIFLG	CIFNON	0
CIRET	CRC000	0
CICC	CCOK	0
CIREA	RCNONE	0
CIUOW	CUONLY	273
CIGWI	WIDFLT	-2
CILT	LTPROG	1
CIODL	OLINPT	-1
CIFKT	None	0
CIADS	ADNONE	0
CICT	CTNO	0

Table 325. Initial values of fields in MQCIH (continued)

Field name	Name of constant	Value of constant
CITES	TENOSY	0
CIFAC	FCNONE	Nulls
CIFNC	CFNONE	Blanks
CIAC	None	Blanks
CIAUT	None	Blanks
CIRS1	None	Blanks
CIRFM	FMNONE	Blanks
CIRSI	None	Blanks
CIRTI	None	Blanks
CITI	None	Blanks
CIFL	None	Blanks
CIAI	None	Blanks
CISC	SCNONE	Blanks
CICNC	None	Blanks
CINTI	None	Blanks
CIRS2	None	Blanks
CIRS3	None	Blanks
CICP	None	0
CIEO	None	0
CIII	None	0
CIRS4	None	0

Notes:

1. The symbol ¬ represents a single blank character.

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQCIH Structure
D*
D* Structure identifier
D  CISID                  1      4    INZ('CIH ')
D* Structure version number
D  CIVER                  5      8I 0 INZ(2)
D* Length of MQCIH structure
D  CILEN                  9     12I 0 INZ(180)
D* Reserved
D  CIENC                 13     16I 0 INZ(0)
D* Reserved
D  CICSI                 17     20I 0 INZ(0)
D* MQ format name of data that followsMQCIH
D  CIFMT                 21     28    INZ('        ')
D* Flags
D  CIFLG                 29     32I 0 INZ(0)
D* Return code from bridge
D  CIRET                 33     36I 0 INZ(0)
D* MQ completion code or CICSEIBRESP
D  CICC                  37     40I 0 INZ(0)
D* MQ reason or feedback code, or CICSEIBRESP2
D  CIREA                 41     44I 0 INZ(0)
```

```
D* Unit-of-work control
D  CIUOW                 45     48I 0 INZ(273)
D* Wait interval for MQGET call issuedby bridge task
D  CIGWI                 49     52I 0 INZ(-2)
D* Link type
D  CILT                  53     56I 0 INZ(1)
D* Output COMMAREA data length
D  CIODL                 57     60I 0 INZ(-1)
D* Bridge facility release time
D  CIFKT                 61     64I 0 INZ(0)
D* Send/receive ADS descriptor
D  CIADS                 65     68I 0 INZ(0)
D* Whether task can beconversational
D  CICT                  69     72I 0 INZ(0)
D* Status at end of task
D  CITES                 73     76I 0 INZ(0)
D* Bridge facility token
D  CIFAC                 77     84    INZ(X'00000000000000-
D                                     00')
D* MQ call name or CICS EIBFNfunction
D  CIFNC                 85     88    INZ('    ')
D* Abend code
D  CIAC                  89     92    INZ
D* Password or passticket
D  CIAUT                 93    100    INZ
D* Reserved
D  CIRS1                101    108    INZ
D* MQ format name of reply message
D  CIRFM                109    116    INZ('        ')
D* Remote CICS system ID to use
D  CIRSI                117    120    INZ
D* CICS RTRANSID to use
D  CIRTI                121    124    INZ
D* Transaction to attach
D  CITI                 125    128    INZ
D* Terminal emulated attributes
D  CIFL                 129    132    INZ
D* AID key
D  CIAI                 133    136    INZ
D* Transaction start code
D  CISC                 137    140    INZ('    ')
D* Abend transaction code
D  CICNC                141    144    INZ
D* Next transaction to attach
D  CINTI                145    148    INZ
D* Reserved
D  CIRS2                149    156    INZ
D* Reserved
D  CIRS3                157    164    INZ
D* Cursor position
D  CICP                 165    168I 0 INZ(0)
D* Offset of error in message
D  CIEO                 169    172I 0 INZ(0)
D* Reserved
D  CIII                 173    176I 0 INZ(0)
D* Reserved
D  CIRS4                177    180I 0 INZ(0)
D*
```

MQCMHO (Create message handle options) on IBM i: IBM i

The **MQCMHO** structure allows applications to specify options that control how message handles are created.

Overview

Purpose

The structure is an input parameter on the **MQCRTMH** call.

Character set and encoding

Data in **MQCMHO** must be in the character set of the application and encoding of the application (ENNAT).

Fields

The MQCMHO structure contains the following fields; the fields are described in alphabetical order:

CMOPT (10 digit signed integer)

One of the following options can be specified:

CMVAL

When **MQSETMP** is called to set a property in this message handle, the property name is validated to ensure that it:

- contains no invalid characters.
- does not begin "JMS" or "usr.JMS" except for the following:
  - JMSCorrelationID
  - JMSReplyTo
  - JMSType
  - JMSXGroupID
  - JMSXGroupSeq

  These names are reserved for JMS properties.
- is not one of the following keywords, in any mixture of upper or lowercase:
  - "AND"
  - "BETWEEN"
  - "ESCAPE"
  - "FALSE"
  - "IN"
  - "IS"
  - "LIKE"
  - "NOT"
  - "NULL"
  - "OR"
  - "TRUE"
- does not begin "Body." or "Root." (except for "Root.MQMD.").

If the property is MQ-defined ("mq.*") and the name is recognized, the property descriptor fields are set to the correct values for the property. If the property is not recognized, the *Support* field of the property descriptor is set to **PDSUPO** (for more information, see PDSUP).

CMDEFV

This specifies that the default level of validation of property names occurs.

The default level of validation is equivalent to that specified by **CMVAL**.

In a future release an administrative option might be defined which will change the level of validation that will occur when **CMDEFV** is defined.

This is the default value.

CMNOVA

No validation on the property name occurs. See the description of **CMVAL**.

Default option: If none of the options previously described in this section is required, the following option can be used:

CMNONE

All options assume their default values. Use this value to indicate that no other options have been specified. **CMNONE** aids program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

This is always an input field. The initial value of this field is **CMDEFV**.

CMSID (10 digit signed integer)

This is the structure identifier; the value must be:

CMSIDV
Identifier for create message handle options structure.

This is always an input field. The initial value of this field is **CMSIDV**.

CMVER (10 digit signed integer)

This is the structure version number; the value must be:

CMVER1
Version-1 create message handle options structure.

The following constant specifies the version number of the current version:

CMVERC
Current version of create message handle options structure.

This is always an input field. The initial value of this field is **CMVER1**.

Initial values

Table 326. Initial values of fields in MQCMHO

Field name	Name of constant	Value of constant
CMSID	CMSIDV	'CMHO'
CMVER	CMVER1	1
CMOPT	CMDEFV	0

RPG declaration

```
D* MQCMHO Structure
D*
D*
D* Structure identifier
D  CMSID                  1      4    INZ('CMHO')
D*
D* Structure version number
D  CMVER                  5      8I 0 INZ(1)
D*
D* Options that control the action of MQCRTMH
D  CMOPT                  9     12I 0 INZ(0)
```

MQCNO (Connect options) on IBM i: IBM i

The MQCNO structure allows the application to specify options relating to the connection to the local queue manager.

Overview

Purpose: The structure is an input/output parameter on the MQCONNX call.

Version: The current version of MQCNO is V 9.0.0 CNVER6. Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions that follow.

The COPY file provided contains the most recent version of MQCNO that is supported by the environment, but with the initial value of the *CNVER* field set to CNVER1. To use fields that are not present in the version-1 structure, the application must set the *CNVER* field to the version number of the version required.

Character set and encoding: Data in MQCNO must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT.

- "Fields"
- "Initial values" on page 3060
- "RPG declaration" on page 3060

Fields

The MQCNO structure contains the following fields; the fields are described in **alphabetical order**:

V 9.0.0 **CCDTUL (10-digit signed integer)**

CCDTUL is the length of the string identified by either CCDTUP or CCDTUO which contains a URL that identifies the location of the client connection channel table to use for the connection.

Use CCDTUL only when the application issuing the MQCONNX call is running as an IBM MQ MQI client.

This is a programmatic alternative to setting the MQCHLLIB and MQCHLTAB environment variables.

If the application is not running as an client, CCDTUL is ignored.

This field is ignored if CNVER is less than CNVER6.

V 9.0.0 CCDTUO (10-digit signed integer)

CCDTUO is the offset in bytes, from the start of the MQCNO structure, to a string which contains a URL that identifies the location of the client connection channel table to use for the connection. The offset can be positive or negative.

Use CCDTUL only when the application issuing the MQCONNX call is running as an IBM MQ MQI client.

Important: You can use only one of CCDTUP and CCDTUO. The call fails with reason code RC2600 if both fields are nonzero.

This is a programmatic alternative to setting the MQCHLLIB and MQCHLTAB environment variables.

If the application is not running as an client, CCDTUO is ignored.

This field is ignored if CNVER is less than CNVER6.

V 9.0.0 CCDTUP (pointer)

CCDTUP is an optional pointer to a string which contains a URL, to identify the location of the client connection channel table to use for the connection..

Use CCDTUP only when the application issuing the MQCONNX call is running as an IBM MQ MQI client.

Important: You can use only one of CCDTUP and CCDTUO. The call fails with reason code RC2600 if both fields are nonzero.

This is a programmatic alternative to setting the MQCHLLIB and MQCHLTAB environment variables.

If the application is not running as an client, CCDTUP is ignored.

This field is ignored if CNVER is less than CNVER6.

CNCCO (10-digit signed integer)

This is the offset in bytes of an MQCD channel definition structure from the start of the MQCNO structure.

CNCCP (pointer)

This is a pointer to an MQCD channel definition structure.

CNCONID (24-byte character string)

Unique connection identifier. This field allows the queue manager to reliably identify an application process by assigning it a unique identifier when it first connects to the queue manager.

Applications use the connection identifier for correlation purposes when making PUT and GET calls. All connections are assigned an identifier by the queue manager, no matter how the connection was established.

It is possible to use the connection identifier to force the end of a long running unit of work. To do this, specifying the connection identifier using the PCF command 'Stop Connection', or the MQSC command STOP CONN. For more information about using these commands, see the related links.

The initial value of the field is 24 null bytes.

CNCT (128-byte bit string)

This is a tag that the queue manager associates with the resources that are affected by the application during this connection.

Queue manager connection tag.

Each application or application instance must use a different value for the tag, so that the queue manager can correctly serialize access to the affected resources. See the descriptions of the CN*CT* options for further details. The tag ceases to be valid when the application terminates, or issues the MQDISC call.

Use the following special value if no tag is required:

CTNONE
No connection tag specified.

The value is binary zero for the length of the field.

This is an input field. The length of this field is given by LNCTAG. The initial value of this field is CTNONE. This field is ignored if *CNVER* is less than CNVER3.

Use the field ConnTag when connecting to a z/OS queue manager.

CNOPT (10 digit signed integer)

Options that control the action of MQCONNX.

Binding options

The binding options control the type of IBM MQ binding that is used; specify only one of these options:

CNSBND Standard binding.

The standard binding option causes the application and the local queue manager agent to run in separate units of execution, typically in separate processes. The arrangement maintains the integrity of the queue manager; that is, it protects the queue manager from errant programs.

Use CNSBND in situations where the application might not have been fully tested, or might be unreliable or untrustworthy. CNSBND is the default.

CNSBND is defined to aid program documentation. Do not use this option with any other option controlling the type of binding used; but because its value is zero, such use cannot be detected.

This option is supported in all environments.

CNFBND Fast path binding.

The fast path binding option causes the application and the local queue manager agent to be part of the same unit of execution. Fast path is in contrast to the standard binding, where the application and the local queue manager agent run in separate units of execution.

CNFBND is ignored if the queue manager does not support this type of binding; processing continues as though the option had not been specified.

CNFBND can be of advantage in situations where multiple processes consume more resources than the overall resource used by the application. An application that uses the fast path binding is known as a *trusted application*.

Consider the following important points when deciding whether to use the fast path binding:

- **Using the CNFBND option does not prevent an application altering or corrupting messages and other data areas belonging to the queue manager. Use this option only in situations where you have fully evaluated these issues.**
- The application must not use asynchronous signals or timer interrupts (such as sigkill) with CNFBND. There are also restrictions on the use of shared memory segments.
- The application must not have more than one thread connected to the queue manager at any one time.
- The application must use the MQDISC call to disconnect from the queue manager.
- The application must finish before ending the queue manager with the endmqm command.

The following points apply to the use of CNFBND in the environments indicated:

- On IBM i, the job must run under user profile QMQM that belongs to the QMQMADM group. Also, the program must not terminate abnormally, otherwise unpredictable results might occur.

For more information about the implications of using trusted applications, see Connecting to a queue manager using the MQCONNX call and Restrictions for trusted applications.

CNSHBD Shared Bindings.

The shared bindings option causes the application and the local queue manager agent to run in separate units of execution, typically in separate processes. The arrangement maintains the integrity of the queue manager; that is, it protects the queue manager from errant programs. However some resources are shared between the application and the local queue manager agent. CNSHBD is ignored if the queue manager does not support this type of binding. Processing continues as though the option had not been specified.

CNIBND Isolated Bindings.

The isolated bindings option causes the application and the local queue manager agent to run in separate units of execution, typically in separate processes. The arrangement maintains the integrity of the queue manager; that is, it protects the queue manager from errant programs. The application process and the local queue manager agent are isolated from each other in that they do not share resources. CNIBND is ignored if the queue manager does not support this type of binding. Processing continues as though the option had not been specified.

Handle-sharing options

The following options control the sharing of handles between different threads (units of parallel processing) within the same process. Only one of these options can be specified.

CNHSN No handle sharing between threads.

The no handle sharing between threads option indicates that connection and object handles can be used only by the thread that caused the handle to be allocated; that is, the thread that issued the MQCONN,

MQCONNX, or MQOPEN call. The handles cannot be used by other threads belonging to the same process.

CNHSB Serial handle sharing between threads, with call blocking.

The serial handle sharing between threads, with call blocking, option indicates that connection and object handles allocated by one thread of a process can be used by other threads belonging to the same process. However, only one thread at a time can use any particular handle, that is, only serial use of a handle is permitted. If a thread tries to use a handle that is already in use by another thread, the call blocks (waits) until the handle becomes available.

CNHSNB Serial handle sharing between threads, without call blocking.

The serial handle sharing between threads, *without* call blocking, option is the same as the " *with* blocking" option, except that, if the handle is in use by another thread, the call completes immediately with `CCFAIL` and `RC2219` instead of blocking until the handle becomes available.

A thread can have zero or one nonshared handles, plus zero or more shared handles:

- Each MQCONN or MQCONNX call that specifies `CNHSN` returns a new nonshared handle on the first call, and the same nonshared handle on subsequent calls (assuming no intervening MQDISC call). The reason code is RC2002 for the second and later calls.
- Each MQCONNX call that specifies `CNHSB` or `CNHSNB` returns a new shared handle on each call.

Object handles inherit the same sharing properties as the connection handle specified on the MQOPEN call that created the object handle. Also, units of work inherit the same sharing properties as the connection handle used to start the unit of work; if the unit of work is started in one thread using a shared handle, the unit of work can be updated in another thread using the same handle.

If you do not specify a handle-sharing option, the default is determined by the environment:

- In the Microsoft Transaction Server (MTS) environment, the default is the same as `CNHSB`.
- In other environments, the default is the same as `CNHSN`.

Reconnection options

Reconnection options determine if a connection is reconnectable. Only client connections are reconnectable.

CNRCDF The reconnection option is resolved to its default value. If no default is set, the value of this option resolves to `DISABLED`. The value of the option is passed to the server, and can be queried by **PCF** and **MQSC**.

CNRC The application can be reconnected to any queue manager consistent with the value of the MQCONNX **QMNAME** parameter. Use the `CNRC` option only if there is no affinity between the client application and the queue manager with which it initially established a connection. The value of the option is passed to the server, and can be queried by **PCF** and **MQSC**.

CNRCD The application cannot be reconnected. The value of the option is not passed to the server.

CNRCQM The application can only be reconnected to the queue manager with which it originally connected. Use this value if a client can be

reconnected, but there is an affinity between the client application, and the queue manager with which it originally established a connection. Choose this value if you want a client to automatically reconnect to the standby instance of a highly available queue manager. The value of the option is passed to the server, and can be queried by **PCF** and **MQSC**.

Use the options CNRC, CNRCD, and CNRCQM only for client connections. If the options are used for a binding connection, MQCONNX fails with completion code, MQCC_FAILED and reason code, MQRC_OPTIONS_ERROR.

Default option: If none of the options described is required, the following option can be used:

CNNONE No options are specified.

CNNONE is defined to aid program documentation. It is not intended that this option is used with any other CN* option, but because its value is zero, such use cannot be detected.

CNSCO (10-digit signed integer)

This is the offset in bytes of an MQSCO structure from the start of the MQCNO structure.

This field is ignored if *CNVER* is less than CNVER4.

CNSCP (pointer)

This is the address of an MQSCO structure.

This field is ignored if *CNVER* is less than CNVER4.

CNSECPO (10-digit signed integer)

Security parameters offset. The offset of the MQCSP structure used for specifying a user ID and password.

The value may be positive or negative. The initial value of this field is 0.

This field is ignored if *CNVER* is less than CNVER5.

CNSECPP (pointer)

Security parameters pointer. Address of the MQCSP structure used for specifying a user ID and a password.

The initial value of this field is a null pointer or null bytes.

This field is ignored if *CNVER* is less than CNVER5.

CNSID (4-byte character string)

The structure identifier for the MQCNO structure.

The value must be:

CNSIDV
Identifier for connect-options structure.

This is always an input field. The initial value of this field is CNSIDV.

CNVER (10-digit signed integer)

The structure version number for the MQCNO structure.

The value must be:

V 9.0.0 **CNVER6**
Version-6 connect-options structure.

This version is supported in all environments.

The following constant specifies the version number of the current version:

CNVERC
Current version of connect-options structure.

This is always an input field. The initial value of this field is V 9.0.0 CNVER6.

Initial values

Table 327. Initial values of fields in MQCNO

Field name	Name of constant	Value of constant
CNSID	CNSIDV	'CNO¬
CNVER	CNVER5	1
CNOPT	CNNONE	0
CNCCO	None	0
CNCCP	None	Null pointer or null bytes
CNCT	CTNONE	Nulls
CNSCP	None	Null pointer or null bytes
CNSCO	None	0
CNCONID	None	Nulls
CNSECPO	None	0
CNSECPP	None	Null pointer or null bytes
V 9.0.0 *CCDTUL*	None	0
V 9.0.0 *CCDTUO*	None	0
V 9.0.0 *CCDTUP*	None	Null pointer or null bytes

Notes:

1. The symbol ¬ represents a single blank character.

RPG declaration V 9.0.0

```
D*****************************************************************
     D**                                                            **
     D**                   IBM MQ for IBM i                         **
     D**                                                            **
     D**  FILE NAME:      CMQCNOG                                   **
     D**                                                            **
     D**  DESCRIPTION:    MQCNO Structure -- Connect Options        **
     D**                                                            **
     D*****************************************************************
     D**  <N_OCO_COPYRIGHT>                                         **
     D**  Licensed Materials - Property of IBM                      **
     D**                                                            **
     D**  5724-H72                                                  **
     D**  (c) Copyright IBM Corp. 1993, 2013 All Rights Reserved.   **
     D**                                                            **
     D**  US Government Users Restricted Rights - Use, duplication or **
     D**  disclosure restricted by GSA ADP Schedule Contract with   **
     D**  IBM Corp.                                                 **
     D**  <NOC_COPYRIGHT>                                           **
     D*****************************************************************
     D**                                                            **
```

```
D**  FUNCTION:       This file declares the structure MQCNO,     **
D**                  which is used by the main MQI.              **
D**                                                              **
D**  PROCESSOR:      RPG (ILE)                                   **
D**                                                              **
D******************************************************************
D*
D*
D******************************************************************
D** <BEGIN_BUILDINFO>                                            **
D** Generated on:  08/02/16 13:50                                **
D** Build Level:   L000000                                       **
D** Build Type:    Production                                    **
D** Pointer Size:  128 Bit                                       **
D** Source File:                                                 **
D** CMQCNOG                                                      **
D** <END_BUILDINFO>                                              **
D******************************************************************
D*
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D*
D*
D* MQCNO Structure
D*
D* Structure identifier
D  CNSID                  1      4    INZ('CNO ')
D* Structure version number
D  CNVER                  5      8I 0 INZ(1)
D* Options that control the action of MQCONNX
D  CNOPT                  9     12I 0 INZ(0)
D* Ver:1 **
D* Offset of MQCD structure for client connection
D  CNCCO                 13     16I 0 INZ(0)
D* Address of MQCD structure for client connection
D  CNCCP                 17     32*   INZ(*NULL)
D* Ver:2 **
D* Queue managerconnection tag
D  CNCT                  33    160    INZ(X'00000000000000000-
D                                     00000000000000000000000000-
D                                     00000000000000000000000000-
D                                     00000000000000000000000000-
D                                     00000000000000000000000000-
D                                     00000000000000000000000000-
D                                     00000000000000000000000000-
D                                     00000000000000000000000000-
D                                     00000000000000000000000000-
D                                     00000000000000000000000000-
D                                     00000000000000')
D* Ver:3 **
D* Address of MQSCO structure for client connection
D  CNSCP                 161    176*   INZ(*NULL)
D* Offset of MQSCO structure for client connection
D  CNSCO                 177    180I 0 INZ(0)
D* Ver:4 **
D* Unique Connection Identifier
D  CNCONID               181    204    INZ(X'00000000000000000-
D                                     00000000000000000000000000-
D                                     000000')
D* Offset of MQCSP structure
D  CNSECPO               205    208I 0 INZ(0)
D* Address of MQCSP structure
D  CNSECPP               209    224*   INZ(*NULL)
D* Ver:5 **
D* Address of CCDT URL string
D  CNCCDTUP              225    240*   INZ(*NULL)
D* Offset of CCDT URL string
D  CNCCDTUO              241    244I 0 INZ(0)
```

```
D* Length of CCDT URL
D  CNCCDTUL             245    248I 0 INZ(0)
D* Ver:6 **
D*
D******************************************************************
D**  End of CMQCNOG                                              **
D******************************************************************
```

MQCSP (Security parameters) on IBM i: IBM i

Summary of the MQCSP structure for IBM i.

Overview

Purpose: The MQCSP structure enables the authorization service to authenticate a user ID and password. You specify the MQCSP connection security parameters structure on an MQCONNX call.

Character set and encoding: Data in MQCSP must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT.

- "Fields"
- "Initial values" on page 3063
- "RPG declaration" on page 3064

Fields

The MQCSP structure contains the following fields; the fields are described in **alphabetical order**:

CSAUTHT (10-digit signed integer)

This is the type of authentication to perform.

Valid values are:

CSAN
Do not use user ID and password fields.

CSAUIAP
Authenticate user ID and password fields.

This is an input field. The initial value of this field is CSAN.

CSCPPL (10-digit signed integer)

This is the length of the password to be used in authentication.

The maximum length of the password is not dependent on the platform. If the length of the password is greater than that allowed, the authentication request fails with an RC2035.

This is an input field. The initial value of this field is 0.

CSCPPO (10-digit signed integer)

This is the offset in bytes of the password to be used in authentication.

The offset can be positive or negative.

This is an input field. The initial value of this field is 0.

CSCPPP (pointer)

This is the address of the password to be used in authentication.

This is an input field. The initial value of this field is the null pointer.

CSCSPUIL (10-digit signed integer)

This is the length of the user ID to be used in authentication.

The maximum length of the user ID is not dependent on the platform. If the length of the user ID is greater than that allowed, the authentication request fails with an RC2035.

This is an input field. The initial value of this field is 0.

CSCSPUIO (10-digit signed integer)

This is the offset in bytes of the user ID to be used in authentication.

The offset can be positive or negative.

This is an input field. The initial value of this field is 0.

CSCSPUIP (pointer)

This is the address of the user ID to be used in authentication.

This is an input field. The initial value of this field is the null pointer. This field is ignored if CSVER is less than CSVER5.

CSRE1 (4-byte character string)

A reserved field, required for pointer alignment on IBM i.

This is an input field. The initial value of this field is all null.

CSRS2 (8-byte character string)

A reserved field, required for pointer alignment on IBM i.

This is an input field. The initial value of this field is all null.

CSSID (4-byte character string)

Structure identifier.

The value must be:

CSSIDV
Identifier for the security parameters structure.

CSVER (10-digit signed integer)

Structure version number.

The value must be:

CSVER1
Version-1 security parameters structure.

The following constant specifies the version number of the current version:

CSVERC
Current version of security parameters structure.

This is always an input field. The initial value of this field is CSVER1.

Initial values

Table 328. Initial values of fields in MQCNO

Field name	Name of constant	Value of constant	
CSSID	CSSIDV	'CSP¬'	
CSVER	CSVER1	1	
CSAUTHT	None	0	
CSRE1	None	Nulls	
CSCSPUIP	None	Null pointer	
CSCSPUIO	None	0	
CSCSPUIL	None	0	
CSRS2	None	Nulls	
CSCPPP	None	Null pointer	
CSCPPO	None	0	
CSCPPL	None	0	

Note:

1. The symbol ¬ represents a single blank character.

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
      D*
      D* MQCSP Structure
      D*
      D* Structure identifier
      D  CSSID                  1      4    INZ('CSP ')
      D* Structure version number
      D  CSVER                  5      8I 0 INZ(1)
      D* Type of authentication
      D  CSAUTHT                9     12I 0 INZ(0)
      D* Reserved
      D  CSRE1                 13     16    INZ(X'00000000')
      D* Address of user ID
      D  CSCSPUIP              17     32*   INZ(*NULL)
      D* Offset of user ID
      D  CSCSPUIO              33     36I 0 INZ(0)
      D* Length of user ID
      D  CSCSPUIL              37     40I 0 INZ(0)
      D* Reserved
      D  CSRS2                 41     48    INZ(X'0000000000000000')
      D* Address of password
      D  CSCPPP                49     64*   INZ(*NULL)
      D* Offset of password
      D  CSCPPO                65     68I 0 INZ(0)
      D* Length of password
      D  CSCPPL                69     72I 0 INZ(0)
```

MQCTLO (Control callback options structure) on IBM i: 

Structure specifying the control callback function.

Overview

Purpose

The MQCTLO structure is used to specify options relating to a control callbacks function.

The structure is an input and output parameter on the MQCTL call.

Version

The current version of MQCTLO is CTLV1.

Character set and encoding

Data in MQCTLO must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT. However, if the application is running as an IBM MQ client, the structure must be in the character set and encoding of the client.

- "Fields"
- "Initial values" on page 3066
- "RPG declaration" on page 3066

Fields

The MQCTLO structure contains the following fields; the fields are described in alphabetical order:

COCONNAREA (10 digit signed integer)

Control options structure - ConnectionArea field.

This is a field that is available for the callback function to use.

The queue manager makes no decisions based on the contents of this field and it is passed unchanged from the CBCCONNAREA field in the MQCBC structure, which is a parameter on the MQCB call.

This field is ignored for all operations other than CTLSR and CTLSW.

This is an input and output field to the callback function. The initial value of this field is a null pointer or null bytes.

COOPT (10 digit signed integer)

Options that control the action of MQCTLO.

CTLFQ

Force the MQCTLO call to fail if the queue manager or connection is in the quiescing state.

Specify GMFIQ, in the MQGMO options passed on the MQCB call, to cause notification to message consumers when they are quiescing.

CTLTHR

This option informs the system that the application requires that all message consumers, for the same connection, are called on the same thread.

Default option: If you do not need any of the options described, use the following option:

CTLNO

Use this value to indicate that no other options have been specified; all options assume

their default values. CTLNO is defined to aid program documentation; it is not intended that this option is used with any other, but as its value is zero, such use cannot be detected.

This is an input field. The initial value of the *COOPT* field is CTLNO.

CORSV (10 digit signed integer)

This is a reserved field. The initial value of this field is a blank character.

COSID (10 digit signed integer)

Control options structure - StrucId field.

This is the structure identifier; the value must be:

CTLSI Identifier for Control Options structure.

This is always an input field. The initial value of this field is CTLSI.

COVER (10 digit signed integer)

Control options structure - Version field.

This is the structure version number; the value must be:

CTLV1
Version-1 Control options structure.

The following constant specifies the version number of the current version:

CTLCV
Current version of Control options structure.

This is always an input field. The initial value of this field is CTLV1.

Initial values

Table 329. Initial values of fields in MQCTLO

Field name	Name of constant	Value of constant
COSID	CTLSI	'CTLO'
COVER	CTLV1	1
COOPT	CTLNO	Nulls
CORSV	Reserved field	
COCONNAREA	None	Null pointer or null bytes

RPG declaration

```
D* MQCTLO Structure
D*
D*
D* Structure identifier
D  COSID                  1      4    INZ('CTLO')
D*
D* Structure version number
D  COVER                  5      8I 0 INZ(1)
D*
D* Options that control the action of MQCTL
D  COOPT                  9     12I 0 INZ(0)
D*
D* Reserved
```

```
D  CORSV                 13     16I 0 INZ(-1)
D*
D* MQCTL Data area passed to the function
D  COCONNAREA            17     32*   INZ(*NULL)
```

MQDH (Distribution header) on IBM i: 

The MQDH structure describes the additional data that is present in a message when that message is a distribution-list message stored on a transmission queue.

Overview

Purpose: A distribution-list message is a message that is sent to multiple destination queues. The additional data consists of the MQDH structure followed by an array of MQOR records and an array of MQPMR records.

This structure is for use by specialized applications that put messages directly on transmission queues, or which remove messages from transmission queues (for example: message channel agents).

This structure should not be used by normal applications which simply want to put messages to distribution lists. Those applications should use the MQOD structure to define the destinations in the distribution list, and the MQPMO structure to specify message properties or receive information about the messages sent to the individual destinations.

Character set and encoding: Data in MQDH must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT for the C programming language.

The character set and encoding of the MQDH must be set into the *MDCSI* and *MDENC* fields in:

- The MQMD (if the MQDH structure is at the start of the message data), or
- The header structure that precedes the MQDH structure (all other cases).

Usage: When an application puts a message to a distribution list, and some or all of the destinations are remote, the queue manager prefixes the application message data with the MQXQH and MQDH structures, and places the message on the relevant transmission queue. The data therefore occurs in the following sequence when the message is on a transmission queue:

- MQXQH structure
- MQDH structure plus arrays of MQOR and MQPMR records
- Application message data

Depending on the destinations, more than one such message might be generated by the queue manager, and placed on different transmission queues. In this case, the MQDH structures in those messages identify different subsets of the destinations defined by the distribution list opened by the application.

An application that puts a distribution-list message directly on a transmission queue must conform to the sequence described previously, and must ensure that the MQDH structure is correct. If the MQDH structure is not valid, the queue manager may choose to fail the MQPUT or MQPUT1 call with reason code RC2135.

Messages can be stored on a queue in distribution-list form only if the queue is defined as being able to support distribution list messages (see the **DistLists** queue attribute described in "Attributes for queues" on page 3385). If an application puts a distribution-list message directly on a queue that does not support distribution lists, the queue manager splits the distribution list message into individual messages, and places those on the queue instead.

- "Fields" on page 3068

- "Initial values" on page 3071
- "RPG declaration" on page 3071

Fields

The MQDH structure contains the following fields; the fields are described in **alphabetical order**:

DHCNT (10-digit signed integer)

Number of MQOR records present.

This defines the number of destinations. A distribution list must always contain at least one destination, so *DHCNT* must always be greater than zero.

The initial value of this field is 0.

DHCSI (10-digit signed integer)

Character set identifier of data that follows the MQOR and MQPMR records.

This specifies the character set identifier of the data that follows the arrays of MQOR and MQPMR records; it does not apply to character data in the MQDH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

CSINHT
Inherit character-set identifier of this structure.

Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value CSINHT is not returned by the MQGET call.

CSINHT cannot be used if the value of the *MDPAT* field in MQMD is ATBRKR.

The initial value of this field is CSUNDF.

DHENC (10-digit signed integer)

Numeric encoding of data that follows the MQOR and MQPMR records.

This specifies the numeric encoding of the data that follows the arrays of MQOR and MQPMR records; it does not apply to numeric data in the MQDH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is 0.

DHFLG (10-digit signed integer)

General flags.

The following flag can be specified:

DHFNEW
Generate new message identifiers.

This flag indicates that a new message identifier is to be generated for each destination in the distribution list. This can be set only when there are no put-message records present, or when the records are present but they do not contain the *PRMID* field.

Using this flag defers generation of the message identifiers until the last possible moment, namely the moment when the distribution-list message is finally split into individual messages. This minimizes the amount of control information that must flow with the distribution-list message.

When an application puts a message to a distribution list, the queue manager sets DHFNEW in the MQDH it generates when both of the following statements are true:

- There are no put-message records provided by the application, or the records provided do not contain the *PRMID* field.
- The *MDMID* field in MQMD is MINONE, or the *PMOPT* field in MQPMO includes PMNMID

If no flags are needed, the following can be specified:

DHFNON
No flags.

This constant indicates that no flags have been specified. DHFNON is defined to aid program documentation. It is not intended that this constant is used with any other, but as its value is zero, such use cannot be detected.

The initial value of this field is DHFNON.

DHFMT (8-byte character string)

Format name of data that follows the MQOR and MQPMR records.

This specifies the format name of the data that follows the arrays of MQOD and MQPMR records (whichever occurs last).

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *MDFMT* field in MQMD.

The initial value of this field is FMNONE.

DHLEN (10-digit signed integer)

Length of MQDH structure plus following MQOR and MQPMR records.

This is the number of bytes from the start of the MQDH structure to the start of the message data following the arrays of MQOR and MQPMR records. The data occurs in the following sequence:

- MQDH structure
- Array of MQOR records
- Array of MQPMR records
- Message data

The arrays of MQOR and MQPMR records are addressed by offsets contained within the MQDH structure. If these offsets result in unused bytes between one or more of the MQDH structure, the arrays of records, and the message data, those unused bytes must be included in the value of *DHLEN*, but the content of those bytes is not preserved by the queue manager. It is valid for the array of MQPMR records to precede the array of MQOR records.

The initial value of this field is 0.

DHORO (10-digit signed integer)

Offset of first MQOR record from start of MQDH.

This field gives the offset in bytes of the first record in the array of MQOR object records containing the names of the destination queues. There are *DHCNT* records in this array. These records (plus any bytes skipped between the first object record and the previous field) are included in the length given by the *DHLEN* field.

A distribution list must always contain at least one destination, so *DHORO* must always be greater than zero.

The initial value of this field is 0.

DHPRF (10-digit signed integer)

Flags indicating which MQPMR fields are present.

Zero or more of the following flags can be specified:

PFMID
Message-identifier field is present.

PFCID
Correlation-identifier field is present.

PFGID
Group-identifier field is present.

PFFB Feedback field is present.

PFACC
Accounting-token field is present.

If no MQPMR fields are present, the following can be specified:

PFNONE
No put-message record fields are present.

PFNONE is defined to aid program documentation. It is not intended that this constant be used with any other, but as its value is zero, such use cannot be detected.

The initial value of this field is PFNONE.

DHPRO (10-digit signed integer)

Offset of first MQPMR record from start of MQDH.

This field gives the offset in bytes of the first record in the array of MQPMR put message records containing the message properties. If present, there are *DHCNT* records in this array. These records (plus any bytes skipped between the first put message record and the previous field) are included in the length given by the *DHLEN* field.

Put message records are optional; if no records are provided, *DHPRO* is zero, and *DHPRF* has the value PFNONE.

The initial value of this field is 0.

DHSID (4-byte character string)

Structure identifier.

The value must be:

DHSIDV
Identifier for distribution header structure.

The initial value of this field is DHSIDV.

DHVER (10-digit signed integer)

Structure version number.

The value must be:

DHVER1
Version number for distribution header structure.

The following constant specifies the version number of the current version:

DHVERC
Current version of distribution header structure.

The initial value of this field is DHVER1.

Initial values

Table 330. Initial values of fields in MQDH

Field name	Name of constant	Value of constant
DHSID	DHSIDV	'DH¬¬'
DHVER	DHVER1	1
DHLEN	None	0
DHENC	None	0
DHCSI	CSUNDF	0
DHFMT	FMNONE	Blanks
DHFLG	DHFNON	0
DHPRF	PFNONE	0
DHCNT	None	0
DHORO	None	0
DHPRO	None	0

Notes:

1. The symbol ¬ represents a single blank character.

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQDH Structure
D*
D* Structure identifier
D  DHSID                  1      4    INZ('DH  ')
D* Structure version number
D  DHVER                  5      8I 0 INZ(1)
D* Length of MQDH structure plusfollowing MQOR and MQPMR records
D  DHLEN                  9     12I 0 INZ(0)
D* Numeric encoding of data that followsthe MQOR and MQPMR records
D  DHENC                 13     16I 0 INZ(0)
D* Character set identifier of data thatfollows the MQOR and MQPMR
D* records
D  DHCSI                 17     20I 0 INZ(0)
D* Format name of data that follows theMQOR and MQPMR records
D  DHFMT                 21     28    INZ('        ')
D* General flags
D  DHFLG                 29     32I 0 INZ(0)
D* Flags indicating which MQPMR fieldsare present
D  DHPRF                 33     36I 0 INZ(0)
D* Number of MQOR records present
D  DHCNT                 37     40I 0 INZ(0)
D* Offset of first MQOR record from startof MQDH
D  DHORO                 41     44I 0 INZ(0)
D* Offset of first MQPMR record fromstart of MQDH
D  DHPRO                 45     48I 0 INZ(0)
```

MQDLH (Dead-letter header) on IBM i: IBM i

Overview

Purpose

The MQDLH structure describes the information that prefixes the application message data of messages on the dead-letter (undelivered-message) queue. A message can arrive on the dead-letter queue because the queue manager or message channel agent redirected it to the queue. An application might put the message directly on the queue.

Format name

`FMDLH`

Character set and encoding

The MQDLH might be at the start of the application message data. If so, the fields in the MQDLH structure are in the character set and encoding given by the `MDCSI` and `MDENC` fields. If not, the character set and encoding are set by the `MDCSI` and `MDENC` fields in the header structure that precedes the MQDLH.

The character set must be one that has single-byte characters for the characters that are valid in queue names.

Usage Applications that put messages directly on the dead-letter queue must prefix the message data with an MQDLH structure, and initialize the fields with appropriate values. However, the queue manager does not require that an MQDLH structure is present, or that valid values are specified for the fields.

If a message is too long to put on the dead-letter queue, the application must consider doing one of the following things:

- Truncate the message data to fit on the dead-letter queue.
- Record the message on auxiliary storage and place an exception report message on the dead-letter queue indicating the message is too long.
- Discard the message and return an error to its originator. If the message is a critical message. Discard the message only if it is known that the originator still has a copy of the message. For example, a message received by a message channel agent from a communication channel.

Which of the choices is appropriate depends on the design of the application.

The queue manager performs special processing when a message which is a segment is put with an MQDLH structure at the front. See the description of the MQMDE structure for further details.

Putting messages on the dead-letter queue

If a message is put on the dead-letter queue, the MQMD structure used for the MQPUT or MQPUT1 call must be identical to the MQMD associated with the message. The MQMD is typically the one returned by the MQGET call, except for the following cases:

- The `MDCSI` and `MDENC` fields must be set to whatever character set and encoding are used for fields in the MQDLH structure.
- The `MDFMT` field must be set to `FMDLH` to indicate that the data begins with an MQDLH structure.
- The context fields, `MDACC`, `MDAID`, `MDAOD`, `MDPAN`, `MDPAT`, `MDPD`, `MDPT`, and `MDUID` must be set by using a context option appropriate to the circumstances:

- An application putting on the dead-letter queue a message that is not related to any preceding message must use the PMDEFC option. The PMDEFC option causes the queue manager to set all of the context fields in the message descriptor to their default values.
- A server application putting on the dead-letter queue a message it received must use the PMPASA option, in order to preserve the original context information.
- A server application putting on the dead-letter queue a reply to message it received must use the PMPASI option. The PMPASI option preserves the identity information but sets the origin information to be that of the server application.
- A message channel agent putting on the dead-letter queue a message it received from its communication channel must use the PMSETA option. The PMSETA option preserves the original context information.

In the MQDLH structure itself, the fields must be set as follows:

- The DLCSI, DLENC, and *DLFMT* fields must be set to the values that describe the data that follows the MQDLH structure. These values are typically the values from the original message descriptor.
- The context fields DLPAT, DLPAN, DLPD, and DLPT must be set to values appropriate to the application that is putting the message on the dead-letter queue. These values are not related to the original message.
- Other fields must be set as appropriate.

The application must ensure that all fields have valid values, and that character fields are padded with blanks to the defined length of the field. The character data must not be terminated prematurely by using a null character. The queue manager does not convert the null and subsequent characters to blanks in the MQDLH structure.

Getting messages from the dead-letter queue

Applications that get messages from the dead-letter queue must verify that the messages begin with an MQDLH structure. The application can determine whether an MQDLH structure is present by examining the MDFMT field in the message descriptor MQMD. If the field has the value FMDLH, the message data begins with an MQDLH structure. Messages on the dead-letter queue might be truncated if they were originally too long for the queue they were intended for.

Fields

The MQDLH structure contains the following fields; the fields are described in alphabetical order:

DLCSI (10-digit signed integer)

Character set identifier of data that follows MQDLH.

DLCSI specifies the character set identifier of the data that follows the MQDLH structure. The data is typically from the original message. It does not apply to character data in the MQDLH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

CSINHT Inherit character-set identifier of this structure.

Character data in the data following this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value CSINHT is not returned by the MQGET call.

CSINHT cannot be used if the value of the MDPAT field in MQMD is ATBRKR.

The initial value of this field is CSUNDF.

DLDM (48-byte character string)

Name of original destination queue manager.

This is the name of the queue manager that was the original destination for the message.

The length of this field is given by LNQMN. The initial value of this field is 48 blank characters.

DLDQ (48-byte character string)

Name of original destination queue.

This is the name of the message queue that was the original destination for the message.

The length of this field is given by LNQN. The initial value of this field is 48 blank characters.

DLENC (10-digit signed integer)

Numeric encoding of data that follows MQDLH.

DLENC specifies the numeric encoding of the data that follows the MQDLH structure. The data is typically from the original message. It does not apply to numeric data in the MQDLH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is 0.

DLFMT (8-byte character string)

Format name of data that follows MQDLH.

This specifies the format name of the data that follows the MQDLH structure (typically the data from the original message).

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as the rules for the MDFMT field in MQMD.

The length of this field is given by LNFMT. The initial value of this field is FMNONE.

DLPAN (28-byte character string)

Name of application that put message on dead-letter (undelivered-message) queue.

The format of the name depends on the DLPAT field. See the description of the MDPAN field in "MQMD (Message descriptor) on IBM i" on page 3118.

If it is the queue manager that redirects the message to the dead-letter queue, DLPAN contains the first 28 characters of the queue manager name. The name is padded with blanks if necessary.

The length of this field is given by LNPAN. The initial value of this field is 28 blank characters.

DLPAT (10-digit signed integer)

Type of application that put message on dead-letter (undelivered-message) queue.

This field has the same meaning as the MDPAT field in the message descriptor MQMD (see "MQMD (Message descriptor) on IBM i" on page 3118 for details).

If it is the queue manager that redirects the message to the dead-letter queue, DLPAT has the value ATQM.

The initial value of this field is 0.

DLPD (8-byte character string)

Date when message was put on dead-letter (undelivered-message) queue.

The format used for the date when this field is generated by the queue manager is:

- YYYYMMDD

where the characters represent:

YYYY year (four numeric digits)

MM month of year (01 through 12)

DD day of month (01 through 31)

Greenwich Mean Time (GMT) is used for the DLPD and DLPT fields, subject to the system clock being set accurately to GMT.

The length of this field is given by LNPDAT. The initial value of this field is eight blank characters.

DLPT (8-byte character string)

Time when message was put on the dead-letter (undelivered-message) queue.

The format used for the time when this field is generated by the queue manager is:

- HHMMSSTH

where the characters represent (in order):

HH hours (00 through 23)

MM minutes (00 through 59)

SS seconds (00 through 59; see note later in this topic)

T tenths of a second (0 through 9)

H hundredths of a second (0 through 9)

Note: If the system clock is synchronized to an accurate time standard, it is possible for 60 or 61 to be returned for the seconds in DLPT. The extra second occurs when leap seconds are inserted into the global time standard.

Greenwich Mean Time (GMT) is used for the DLPD and DLPT fields, subject to the system clock being set accurately to GMT.

The length of this field is given by LNPTIM. The initial value of this field is eight blank characters.

DLREA (10-digit signed integer)

Reason message arrived on dead-letter (undelivered-message) queue.

This identifies the reason why the message was placed on the dead-letter queue instead of on the original destination queue. It must be one of the FB* or RC* values (for example, RC2053). See the description of the *MDFB* field in "MQMD (Message descriptor) on IBM i" on page 3118 for details of the common FB* values that can occur.

If the value is in the range FBIFST through FBILST, the actual IMS error code can be determined by subtracting FBIERR from the value of the *DLREA* field.

Some FB* values occur only in this field. They relate to repository messages, trigger messages, or transmission-queue messages that are transferred to the dead-letter queue. These values are:

FBABEG Application cannot be started.

An application processing a trigger message was unable to start the application named in the TMAI field of the trigger message; see "MQTM - Trigger message" on page 3248.

FBATYP Application type error.

An application processing a trigger message was unable to start the application because the TMAT field of the trigger message is not valid; see "MQTM - Trigger message" on page 3248.

FBBOCD Cluster-receiver channel deleted.

The message was on a cluster transmission queue intended for a cluster queue that was opened with the FBIERR option. The remote cluster-receiver channel to be used to transmit the message to the destination queue was deleted before the message could be sent. Because FBIERR was specified, only the channel selected when the queue was opened can be used to transmit the message. As this channel is not longer available, the message was placed on the dead-letter queue.

FBNARM Message is not a repository message.

FBSBCX Message stopped by channel auto-definition exit.

FBSBMX Message stopped by channel message exit.

FBTM MQTM structure not valid or missing.

The MDFMT field in MQMD specifies FMTM, but the message does not begin with a valid MQTM structure. For example, the *TMSID* mnemonic eye-catcher might not be valid. The *TMVER* might not be recognized. The length of the trigger message might be insufficient to contain the MQTM structure.

FBXQME Message on transmission queue not in correct format.

A message channel agent found that a message on the transmission queue is not in the correct format. The message channel agent puts the message on the dead-letter queue using this feedback code.

The initial value of this field is RCNONE.

DLSID (4-byte character string)

Structure identifier.

The value must be:

DLSIDV Identifier for dead-letter header structure.

The initial value of this field is DLSIDV.

DLVER (10-digit signed integer)

Structure version number.

The value must be:

DLVER1 Version number for dead-letter header structure.

The following constant specifies the version number of the current version:

DLVERC Current version of dead-letter header structure.

The initial value of this field is DLVER1.

Initial values

Table 331. Initial values of fields in MQDLH.

Lists the names of the MQDLH constants and their values.

Field name	Name of constant	Value of constant
DLSID	DLSIDV	'DLH¬'
DLVER	DLVER1	1
DLREA	RCNONE	0
DLDQ	None	Blanks
DLDM	None	Blanks
DLENC	None	0

Table 331. Initial values of fields in MQDLH (continued).

Lists the names of the MQDLH constants and their values.

Field name	Name of constant	Value of constant
DLCSI	CSUNDF	0
DLFMT	FMNONE	Blanks
DLPAT	None	0
DLPAN	None	Blanks
DLPD	None	Blanks
DLPT	None	Blanks

Notes:

1. The symbol ¬ represents a single blank character.

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQDLH Structure
D*
D* Structure identifier
D  DLSID                  1      4    INZ('DLH ')
D* Structure version number
D  DLVER                  5      8I 0 INZ(1)
D* Reason message arrived on dead-letter(undelivered-message) queue
D  DLREA                  9     12I 0 INZ(0)
D* Name of original destination queue
D  DLDQ                  13     60    INZ
D* Name of original destination queue manager
D  DLDM                  61    108    INZ
D* Numeric encoding of data that followsMQDLH
D  DLENC                 109    112I 0 INZ(0)
D* Character set identifier of data thatfollows MQDLH
D  DLCSI                 113    116I 0 INZ(0)
D* Format name of data that followsMQDLH
D  DLFMT                 117    124    INZ('        ')
D* Type of application that put messageon dead-letter
D* (undelivered-message)queue
D  DLPAT                 125    128I 0 INZ(0)
D* Name of application that put messageon dead-letter
D* (undelivered-message)queue
D  DLPAN                 129    156    INZ
D* Date when message was put ondead-letter (undelivered-message)queue
D  DLPD                  157    164    INZ
D* Time when message was put on thedead-letter (undelivered-message)queue
D  DLPT                  165    172    INZ
```

MQDMHO (Delete message handle options) on IBM i: IBM i

The **MQDMHO** structure allows applications to specify options that control how message handles are deleted.

Overview

Purpose: The structure is an input parameter on the **MQDLTMH** call.

Character set and encoding: Data in **MQDMHO** must be in the character set of the application and encoding of the application (ENNAT).

- "Fields"
- "Initial values"
- "RPG declaration" on page 3079

Fields

The MQDMHO structure contains the following fields; the fields are described in **alphabetical order**:

DMOPT (10-digit signed integer)

The value must be:

DMNONE

No options specified.

This is always an input field. The initial value of this field is **DMNONE**.

DMSID (10-digit signed integer)

This is the structure identifier; the value must be:

DMSIDV

Identifier for delete message handle options structure.

This is always an input field. The initial value of this field is **DMSIDV**.

DMVER (10-digit signed integer)

This is the structure version number; the value must be:

DMVER1
Version-1 delete message handle options structure.

The following constant specifies the version number of the current version:

DMVERC
Current version of delete message handle options structure.

This is always an input field. The initial value of this field is **DMVER1**.

Initial values

Table 332. Initial values of fields in MQDMHO

Field name	Name of constant	Value of constant
DMSID	DMSIDV	'DMHO'
DMVER	DMVER1	1
DMOPT	DMNONE	0

RPG declaration

```
D* MQDMHO Structure
D*
D*
D* Structure identifier
D  DMSID                  1      4    INZ('DMHO')
D*
D* Structure version number
D  DMVER                  5      8I 0 INZ(1)
D*
D* Options that control the action of MQDLTMH
D  DMOPT                  9     12I 0 INZ(0)
```

MQDMPO (Delete message property options) on IBM i: 

Structure defining the delete message property options.

Overview

Purpose: The MQDMPO structure allows applications to specify options that control how properties of messages are deleted. The structure is an input parameter on the MQDLTMP call.

Character set and encoding: Data in MQDMPO must be in the character set of the application and encoding of the application (ENNAT).

- "Fields"
- "Initial values" on page 3080
- "RPG declaration" on page 3080

Fields

The MQDMPO structure contains the following fields; the fields are described in alphabetic order:

DPOPT (10-digit signed integer)

Delete message property options structure - DPOPT field.

Location options: The following options relate to the relative location of the property compared to the property cursor.

DPDELF
Deletes the first property that matches the specified name.

DPDELC
Deletes the property pointed to by the property cursor; that is the property that was last inquired by using either the IPINQF or the IPINQN option.

The property cursor is reset when the message handle is reused. It is also reset when the message handle is specified in the *HMSG* field of the MQGMO on an MQGET call, or MQPMO structure on an MQPUT call.

The property cursor is reset when the message handle is reused, or when the message handle is specified in the *HMSG* field of the MQGMO structure on an MQGET structure on an MQGET call or MQPMO structure on an MQPUT call.

The call fails with completion code CCFAIL and reason RC2471 if this option is used when the property cursor has not yet been established. It also fails with these codes if the property pointed to by the property cursor has already been deleted..

If neither of these options is required, the following option can be used:

DPNONE
No options specified.

The initial value of this input field is DPDELF.

DPSID (10-digit signed integer)

Delete message property options structure - DPSID field.

This is the structure identifier. The value must be:

DPSIDV
Identifier for delete message property options structure.

This field is always an input field. The initial value of this field is DPSIDV.

DPVER (10-digit signed integer)

Delete message property options structure - DPVER field.

This is the structure version number. The value must be:

DPVER1
Version number for delete message property options structure.

The following constant specifies the version number of the current version:

DPVERC
Current version of delete message property options structure.

This field is always an input field. The initial value of this field is DPVER1.

Initial values

Table 333. Initial values of fields in MQDPMO

Field name	Name of constant	Value of constant
DPSID	DPSIDV	'DMPO'
DPVER	DPVER1	1
DPOPT	Options that control the action of MQDLTMP	DPNONE

RPG declaration

```
D* MQDMPO Structure
D*
D*
D* Structure identifier
D  DPSID                  1      4    INZ('DMPO')
D*
D* Structure version number
D  DPVER                  5      8I 0 INZ(1)
D*
** Options that control the action of
D* MQDLTMP
D  DPOPT                  9     12I 0 INZ(0)
```

MQEPH (Embedded PCF header) on IBM i: IBM i

Overview

Purpose

The MQEPH structure describes the additional data that is present in a message when that message is a programmable command format (PCF) message. The *EPPFH* field defines the PCF parameters that follow this structure and this allows you to follow the PCF message data with other headers.

Format name

EPFMT

Character set and encoding

Data in MQEPH must be in the character set and encoding of the local queue manager; this is given by the **CCSID** queue manager attribute.

Set the character set and encoding of the MQEPH into the *MDCSI* and *MDENC* fields in:

- The MQMD (if the MQEPH structure is at the start of the message data), or
- The header structure that precedes the MQEPH structure (all other cases).

Usage

You cannot use MQEPH structures to send commands to the command server or any other queue manager PCF-accepting server.

Similarly, the command server or any other queue manager PCF-accepting server do not generate responses or events containing MQEPH structures.

- "Fields"
- "Initial values" on page 3083
- "RPG declaration" on page 3083

Fields

The MQEPH structure contains the following fields; the fields are described in **alphabetical order**:

EPCSI (10-digit signed integer)

This is the character set identifier of the data that follows the MQEPH structure and the associated PCF parameters; it does not apply to character data in the MQEPH structure itself.

The initial value of this field is EPCUND.

EPENC (10-digit signed integer)

This is the numeric encoding of the data that follows the MQEPH structure and the associated PCF parameters; it does not apply to character data in the MQEPH structure itself.

The initial value of this field is 0.

EPFLG (10-digit signed integer)

The following values are available:

EPNONE

No flags have been specified. *MDCSI* EPNONE is defined to aid program documentation. It is not intended that this constant be used with any other, but as its value is zero, such use cannot be detected.

EPCSEM

The character set of the parameters containing character data is specified individually within the *CCSID* field in each structure. The character set of the *EPSID* and *EPFMT* fields are defined by the *CCSID* in the header structure that precedes the MQEPH structure, or by the *MDCSI* field in the MQMD if the MQEPH is at the start of the message.

The initial value of this field is EPNONE.

EPFMT (8-byte character string)

This is the format name of the data that follows the MQEPH structure and the associated PCF parameters.

The initial value of this field is EPFMNO.

EPLEN (10-digit signed integer)

This is the amount of data preceding the next header structure. It includes:

- The length of the MQEPH header
- The length of all PCF parameters following the header
- Any blank padding following those parameters

EPLEN must be a multiple of 4.

The fixed-length part of the structure is defined by EPSTLF.

The initial value of this field is 68.

EPPCFH (MQCFH)

This is the programmable command format (PCF) header, defining the PCF parameters that follow the MQEPH structure. This enables you to follow the PCF message data with other headers.

The PCF header is initially defined with the following values:

Table 334. Initial values of fields in EPPCFH

Field name	Name of constant	Value of constant
EP3TYP	CFTNON	0
EP3LEN	FHLENV	36
EP3VER	FHVER3	3
EP3CMD	CMNONE	0
EP3SEQ	None	1
EP3CTL	CFCLST	1
EEP3CC	CCOK	0
EP3REA	RCNONE	0
EP3CNT	None	0

The application must change EP3TYP from CFTNON to a valid structure type for the use it is making of the embedded PCF header.

EPSID (4-byte character string)

The value must be:

EPSTID
Identifier for the Embedded PCF header structure.

The initial value of this field is EPSTID.

EPVER (10-digit signed integer)

The value can be:

EPVER1
Version number for embedded PCF header structure.

The following constant specifies the version number of the current version:

EPVER3
Current version of embedded PCF header structure.

The initial value of this field is EPVER3.

Initial values

Table 335. Initial values of fields in MQEPH

Field name	Name of constant	Value of constant
EPSID	EPSTID	'EP¬¬'
EPVER	EPVER1	1
EPLEN	EPSTLF	68
EPENC	None	0
EPCSI	EPCUND	0
EPFMT	EPFMNO	Blanks
EPFLG	EPNONE	0
EPPCFH	Names and values as defined in Table 334 on page 3082	0

Note:

1. The symbol ¬ represents a single blank character.

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQEPH Structure
D*
D* Structure identifier
D  EPSID                  1      4
D* Structure version number
D  EPVER                  5      8I 0
D* Total lenght of MQEPH including MQCFHand parameter structures
D* that follow
D  EPLEN                  9     12I 0
D* Numeric encoding of data that follows last PCF parameter structure
D  EPENC                 13     16I 0
D* Character set identifier of data that follows last PCF parameter
D* structure
D  EPCSI                 17     20I 0
D* Format name of data that follows last PCF parameter structure
D  EPFMT                 21     28
D* Flags
D  EPFLG                 29     32I 0
D* Programmable Command Format Header
D  EP3TYP                33     36I 0
D  EP3LEN                37     40I 0
D  EP3VER                41     44I 0
D  EP3CMD                45     48I 0
D  EP3SEQ                49     52I 0
D  EP3CTL                53     56I 0
D  EP3CC                 57     60I 0
D  EP3REA                61     64I 0
D  EP3CNT                65     68I 0
```

MQGMO (Get-message options) on IBM i: IBM i

The MQGMO structure allows the application to specify options that control how messages are removed from queues.

Overview

Purpose
The structure is an input/output parameter on the MQGET call.

Version
The current version of MQGMO is GMVER4. Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions that follow.

The COPY file provided contains the most recent version of MQGMO that is supported by the environment, but with the initial value of the *GMVER* field set to GMVER1. To use fields that are not present in the version-1 structure, the application must set the *GMVER* field to the version number of the version required.

Character set and encoding
Data in MQGMO must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT. However, if the application is running as an IBM MQ client, the structure must be in the character set and encoding of the client.

- "Fields"
- "Initial values" on page 3105
- "RPG declaration" on page 3105

Fields

The MQGMO structure contains the following fields; the fields are described in alphabetical order:

GMGST (1 byte character string)

Flag indicating whether message retrieved is in a group.

It has one of the following values:

GSNIG
Message is not in a group.

GSMIG
Message is in a group, but is not the last in the group.

GSLMIG
Message is the last in the group.

This value is also the value returned if the group consists of only one message.

This field is an output field. The initial value of this field is GSNIG. This field is ignored if *GMVER* is less than GMVER2.

GMMH (10 digit signed integer)

Message Handle

If the GMPRAQ option is specified and the PRPCTL queue attribute is not set to PRPRFH then this is the handle to a message which is populated with the properties of the message being retrieved from the queue. The handle is created by an MQCRTMH call. Any properties already associated with the handle are cleared before retrieving a message.

The following value can also be specified:
MQHM_NONE
No message handle supplied.

No message descriptor is required on the MQGET call if a valid message handle is supplied and used on output to contain the message properties, the message descriptor associated with the message handle is used for input fields.

If a message descriptor is specified on the MQGET call, it always takes precedence over the message descriptor associated with a message handle.

If GMPRRF is specified, or the GMPRAQ is specified and the PRPCTL queue attribute is PRPRFH then the call fails with reason code RC2026 when no message descriptor parameter is specified.

On return from the MQGET call, the properties and message descriptor associated with this message handle are updated to reflect the state of the message retrieved (as well as the message descriptor if one was supplied on the MQGET call). The properties of the message can then be inquired using the MQINQMP call.

Except for message descriptor extensions, when present, a property that can be inquired with the MQINQMP call is not contained in the message data; if the message on the queue contained properties in the message data these are removed from the message data before the data is returned to the application.

If no message handle is provided or Version is less than GMVER4 then you must supply a valid message descriptor on the MQGET call. Any message properties (except those properties contained in the message descriptor) are returned in the message data subject to the value of the property options in the MQGMO structure and the PRPCTL queue attribute.

This field is an always an input field. The initial value of this field is HMNONE. This field is ignored if *GMVER* is less than GMVER4.

GMMO (10 digit signed integer)

Options controlling selection criteria used for MQGET.

These options allow the application to choose which fields in the **MSGDSC** parameter is used to select the message returned by the MQGET call. The application sets the required options in this field, and then sets the corresponding fields in the **MSGDSC** parameter to the values required for those fields. Only messages that have those values in the MQMD for the message are candidates for retrieval using that **MSGDSC** parameter on the MQGET call. Fields for which the corresponding match option is not specified are ignored when selecting the message to be returned. If no selection criteria are to be used on the MQGET call (that is, any message is acceptable), *GMMO* should be set to MONONE.

If GMLOGO is specified, only certain messages are eligible for return by the next MQGET call:

- If there is no current group or logical message, only messages that have *MDSEQ* equal to 1 and *MDOFF* equal to 0 are eligible for return. In this situation, one or more of the following options can be used to select which of the eligible messages is the one returned:
  - MOMSGI
  - MOCORI
  - MOGRPI
- If there is a current group or logical message, only the next message in the group or next segment in the logical message is eligible for return, and this cannot be altered by specifying MO* options.

In both cases, match options which are not applicable can still be specified, but the value of the relevant field in the **MSGDSC** parameter must match the value of the corresponding field in the message to be returned; the call fails with reason code RC2247 is this condition is not satisfied.

GMMO is ignored if either GMMUC or GMBRWC is specified.

One or more of the following options can be specified:

MOMSGI
Retrieve message with specified message identifier.

This option specifies that the message to be retrieved must have a message identifier that matches the value of the *MDMID* field in the **MSGDSC** parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the correlation identifier).

If this option is not specified, the *MDMID* field in the **MSGDSC** parameter is ignored, and any message identifier matches.

Note: The message identifier MINONE is a special value that matches any message identifier in the MQMD for the message. Therefore, specifying MOMSGI with MINONE is the same as not specifying MOMSGI.

MOCORI
Retrieve message with specified correlation identifier.

This option specifies that the message to be retrieved must have a correlation identifier that matches the value of the *MDCID* field in the **MSGDSC** parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the message identifier).

If this option is not specified, the *MDCID* field in the **MSGDSC** parameter is ignored, and any correlation identifier matches.

Note: The correlation identifier CINONE is a special value that matches any correlation identifier in the MQMD for the message. Therefore, specifying MOCORI with CINONE is the same as not specifying MOCORI.

MOGRPI
Retrieve message with specified group identifier.

This option specifies that the message to be retrieved must have a group identifier that matches the value of the *MDGID* field in the **MSGDSC** parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the correlation identifier).

If this option is not specified, the *MDGID* field in the **MSGDSC** parameter is ignored, and any group identifier matches.

Note: The group identifier GINONE is a special value that matches any group identifier in the MQMD for the message. Therefore, specifying MOGRPI with GINONE is the same as not specifying MOGRPI.

MOSEQN
Retrieve message with specified message sequence number.

This option specifies that the message to be retrieved must have a message sequence number that matches the value of the *MDSEQ* field in the **MSGDSC** parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the group identifier).

If this option is not specified, the *MDSEQ* field in the **MSGDSC** parameter is ignored, and any message sequence number matches.

MOOFFS
Retrieve message with specified offset.

This option specifies that the message to be retrieved must have an offset that matches the value of the *MDOFF* field in the **MSGDSC** parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the message sequence number).

If this option is not specified, the *MDOFF* field in the **MSGDSC** parameter is ignored, and any offset matches.

If none of the options described is specified, the following option can be used:

MONONE
No matches.

This option specifies that no matches are to be used in selecting the message to be returned; therefore, all messages on the queue are eligible for retrieval (but subject to control by the GMAMSA, GMASGA, and GMCMPM options).

MONONE is defined to aid program documentation. It is not intended that this option in used with any other MO* option, but as its value is zero, such use cannot be detected.

This field is an input field. The initial value of this field is MOMSGI with MOCORI. This field is ignored if *GMVER* is less than GMVER2.

Note: The initial value of the *GMMO* field is defined for compatibility with earlier version queue managers. However, when reading a series of messages from a queue without using selection criteria, this initial value requires the application to reset the *MDMID* and *MDCID* fields to MINONE and CINONE before each MQGET call. The need to reset *MDMID* and *MDCID* can be avoided by setting *GMVER* to GMVER2, and *GMMO* to MONONE.

GMOPT (10 digit signed integer)

Options that control the action of MQGET.

Zero or more of the following described options can be specified. If more than one is required the values can be added (do not add the same constant more than once). Combinations of options that are not valid are noted; all other combinations are valid.

Wait options: The following options relate to waiting for messages to arrive on the queue:

GMWT
Wait for message to arrive.

The application is to wait until a suitable message arrives. The maximum time the application waits is specified in *GMWI*.

If MQGET requests are inhibited, or MQGET requests become inhibited while waiting, the wait is canceled and the call completes with CCFAIL and reason code RC2016, regardless of whether there are suitable messages on the queue.

This option can be used with the GMBRWF or GMBRWN options.

If several applications are waiting on the same shared queue, the application, or applications, that are activated when a suitable message arrives are described later in this section.

Note: In the following description, a browse MQGET call is one which specifies one of the browse options, but not GMLK; an MQGET call specifying the GMLK option is treated as a nonbrowse call.

- If one or more nonbrowse MQGET calls is waiting, but no browse MQGET calls are waiting, one is activated.
- If one or more browse MQGET calls is waiting, but no nonbrowse MQGET calls are waiting, all are activated.

- If one or more nonbrowse MQGET calls, and one or more browse MQGET calls are waiting, one nonbrowse MQGET call is activated, and none, some, or all the browse MQGET calls. (The number of browse MQGET calls activated cannot be predicted, because it depends on the scheduling considerations of the operating system, and other factors.)

If more than one nonbrowse MQGET call is waiting on the same queue, only one is activated; in this situation the queue manager attempts to give priority to waiting nonbrowse calls in the following order:

1. Specific get-wait requests that can be satisfied only by certain messages, for example, ones with a specific *MDMID* or *MDCID* (or both).
2. General get-wait requests that can be satisfied by any message.

The following points must be noted:

- Within the first category, no additional priority is given to more specific get-wait requests, for example those that specify both *MDMID* and *MDCID*.
- Within either category, it cannot be predicted which application is selected. In particular, the application waiting longest is not necessarily the one selected.
- Path length, and priority-scheduling considerations of the operating system, can mean that a waiting application of lower operating system priority than expected retrieves the message.
- It might also happen that an application that is not waiting retrieves the message in preference to one that is.

GMWT is ignored if specified with GMBRWC or GMMUC; no error is raised.

GMNWT
Return immediately if no suitable message.

The application is not to wait if no suitable message is available. This is the opposite of the GMWT option, and is defined to aid program documentation. It is the default if neither is specified.

GMFIQ
Fail if queue manager is quiescing.

This option forces the MQGET call to fail if the queue manager is in the quiescing state.

If this option is specified together with GMWT, and the wait is outstanding at the time the queue manager enters the quiescing state:

- The wait is canceled and the call returns completion code CCFAIL with reason code `RC2161` .

If GMFIQ is not specified and the queue manager enters the quiescing state, the wait is not canceled.

Syncpoint options: The following options relate to the participation of the MQGET call within a unit of work:

GMSYP
Get message with syncpoint control.

The request is to operate within the normal unit-of-work protocols. The message is marked as being unavailable to other applications, but it is deleted from the queue only when the unit of work is committed. The message is made available again if the unit of work is backed out.

If this option or GMNSYP is not specified, the get request is not within a unit of work.

This option is not valid with any of the following options:

- GMBRWF

- GMBRWC
- GMBRWN
- GMLK
- GMNSYP
- GMPSYP
- GMUNLK

GMPSYP
Get message with syncpoint control if message is persistent.

The request is to operate within the normal unit-of-work protocols, but only if the message retrieved is persistent. A persistent message has the value PEPER in the *MDPER* field in MQMD.

- If the message is persistent, the queue manager processes the call as though the application had specified GMSYP.
- If the message is not persistent, the queue manager processes the call as though the application had specified GMNSYP (see the following section for details).

This option is not valid with any of the following options:

- GMBRWF
- GMBRWC
- GMBRWN
- GMCMPM
- GMNSYP
- GMSYP
- GMUNLK

GMNSYP
Get message without syncpoint control.

The request is to operate outside the normal unit-of-work protocols. The message is deleted from the queue immediately (unless this is a browse request). The message cannot be made available again by backing out the unit of work.

This option is assumed if GMBRWF or GMBRWN is specified.

If this option and GMSYP are not specified, the get request is not within a unit of work.

This option is not valid with any of the following options:

- GMSYP
- GMPSYP

Browse options: The following options relate to browsing messages on the queue:

GMBRWF
Browse from start of queue.

When a queue is opened with the OOBRW option, a browse cursor is established, positioned logically before the first message on the queue. Subsequent MQGET calls specifying the GMBRWF, GMBRWN, or GMBRWC option can be used to retrieve messages from the queue nondestructively. The browse cursor marks the position, within the messages on the queue, from which the next MQGET call with GMBRWN searches for a suitable message.

An MQGET call with GMBRWF causes the previous position of the browse cursor to be ignored. The first message on the queue that satisfies the conditions specified in the message descriptor is retrieved. The message remains on the queue, and the browse cursor is positioned on this message.

After this call, the browse cursor is positioned on the message that has been returned. If the message is removed from the queue before the next MQGET call with GMBRWN is issued, the browse cursor remains at the position in the queue that the message occupied, even though that position is now empty.

The GMMUC option can then be used with a nonbrowse MQGET call if required, to remove the message from the queue.

The browse cursor is not moved by a nonbrowse MQGET call using the same *HOBJ* handle. Nor is it moved by a browse MQGET call that returns a completion code of CCFAIL, or a reason code of RC2080 .

The GMLK option can be specified together with this option, to cause the message that is browsed to be locked.

GMBRWF can be specified with any valid combination of the GM* and MO* options that control the processing of messages in groups and segments of logical messages.

If GMLOGO is specified, the messages are browsed in logical order. If that option is omitted, the messages are browsed in physical order. When GMBRWF is specified, it is possible to switch between logical order and physical order, but subsequent MQGET calls using GMBRWN must browse the queue in the same order as the most recent call that specified GMBRWF for the queue handle.

The group and segment information that the queue manager retains for MQGET calls that browse messages on the queue, is separate from the group and segment information that the queue manager retains for MQGET calls that remove messages from the queue. When GMBRWF is specified, the queue manager ignores the group and segment information for browsing, and scans the queue as though there were no current group and no current logical message. If the MQGET call is successful (completion code CCOK or CCWARN), the group and segment information for browsing is set to that of the message returned; if the call fails, the group and segment information remains the same as it was before the call.

This option is not valid with any of the following options:

- GMBRWC
- GMBRWN
- GMMUC
- GMSYP
- GMPSYP
- GMUNLK

It is also an error if the queue was not opened for browse.

GMBRWN

Browse from current position in queue.

The browse cursor is advanced to the next message on the queue that satisfies the selection criteria specified on the MQGET call. The message is returned to the application, but remains on the queue.

After a queue has been opened for browse, the first browse call using the handle has the same effect whether it specifies the GMBRWF or GMBRWN option.

If the message is removed from the queue before the next MQGET call with GMBRWN is issued, the browse cursor logically remains at the position in the queue that the message occupied, even though that position is now empty.

Messages are stored on the queue in one of two ways:

- FIFO within priority (MSPRIO), or

- FIFO regardless of priority (MSFIFO)

The **MsgDeliverySequence** queue attribute indicates which method applies (see "Attributes for queues" on page 3385 for details).

If the queue has a *MsgDeliverySequence* of MSPRIO, and a message arrives on the queue that is of a higher priority than the one currently pointed to by the browse cursor, that message is not found during the current sweep of the queue using GMBRWN. It can only be found after the browse cursor has been reset with GMBRWF (or by reopening the queue).

The GMMUC option can later be used with a nonbrowse MQGET call if required, to remove the message from the queue.

The browse cursor is not moved by nonbrowse MQGET calls using the same *HOBJ* handle.

The GMLK option can be specified together with this option, to cause the message that is browsed to be locked.

GMBRWN can be specified with any valid combination of the GM* and MO* options that control the processing of messages in groups and segments of logical messages.

If GMLOGO is specified, the messages are browsed in logical order. If that option is omitted, the messages are browsed in physical order. When GMBRWF is specified, it is possible to switch between logical order and physical order, but subsequent MQGET calls using GMBRWN must browse the queue in the same order as the most recent call that specified GMBRWF for the queue handle. The call fails with reason code RC2259 if this condition is not satisfied.

Note: Special care is needed if an MQGET call is used to browse beyond the end of a message group (or logical message not in a group) when GMLOGO is not specified. For example, if the last message in the group happens to precede the first message in the group on the queue, using GMBRWN to browse beyond the end of the group, specifying MOSEQN with *MDSEQ* set to 1 (to find the first message of the next group) would return again the first message in the group already browsed. This could happen immediately, or a number of MQGET calls later (if there are intervening groups).

The possibility of an infinite loop can be avoided by opening the queue twice for browse:

- Use the first handle to browse only the first message in each group.
- Use the second handle to browse only the messages within a specific group.
- Use the MO* options to move the second browse cursor to the position of the first browse cursor, before browsing the messages in the group.
- Do not use GMBRWN to browse beyond the end of a group.

The group and segment information that the queue manager retains for MQGET calls that browse messages on the queue, is separate from the group and segment information that it retains for MQGET calls that remove messages from the queue.

This option is not valid with any of the following options:

- GMBRWF
- GMBRWC
- GMMUC
- GMSYP
- GMPSYP
- GMUNLK

It is also an error if the queue was not opened for browse.

GMBRWC

Browse message under browse cursor.

This option causes the message pointed to by the browse cursor to be retrieved nondestructively, regardless of the MO* options specified in the *GMMO* field in MQGMO.

The message pointed to by the browse cursor is the one that was last retrieved using either the GMBRWF or the GMBRWN option. The call fails if neither of these calls has been issued for this queue since it was opened, or if the message that was under the browse cursor has since been retrieved destructively.

The position of the browse cursor is not changed by this call.

The GMMUC option can then be used with a nonbrowse MQGET call if required, to remove the message from the queue.

The browse cursor is not moved by a nonbrowse MQGET call using the same *HOBJ* handle. Nor is it moved by a browse MQGET call that returns a completion code of CCFAIL, or a reason code of RC2080.

If GMBRWC is specified with GMLK:

- If there is already a message locked, it must be the one under the cursor, so that is returned without unlocking and relocking it; the message remains locked.
- If there is no locked message, the message under the browse cursor (if there is one) is locked and returned to the application; if there is no message under the browse cursor the call fails.

If GMBRWC is specified without GMLK:

- If there is already a message locked, it must be the one under the cursor. This message is returned to the application and then unlocked. Because the message is now unlocked, there is no guarantee that it can be browsed again, or retrieved destructively (it might be retrieved destructively by another application getting messages from the queue).
- If there is no locked message, the message under the browse cursor (if there is one) is returned to the application; if there is no message under the browse cursor the call fails.

If GMCMPM is specified with GMBRWC, the browse cursor must identify a message with a *MDOFF* field in MQMD that is zero. If this condition is not satisfied, the call fails with reason code RC2246 .

The group and segment information that the queue manager retains for MQGET calls that browse messages on the queue, is separate from the group and segment information that it retains for MQGET calls that remove messages from the queue.

This option is not valid with any of the following options:

- GMBRWF
- GMBRWN
- GMMUC
- GMSYP
- GMPSYP
- GMUNLK

It is also an error if the queue was not opened for browse.

GMMUC

Get message under browse cursor.

This option causes the message pointed to by the browse cursor to be retrieved, regardless of the MO* options specified in the *GMMO* field in MQGMO. The message is removed from the queue.

The message pointed to by the browse cursor is the one that was last retrieved using either the GMBRWF or the GMBRWN option.

If GMCMPM is specified with GMMUC, the browse cursor must identify a message with a *MDOFF* field in MQMD that is zero. If this condition is not satisfied, the call fails with reason code RC2246 .

This option is not valid with any of the following options:

- GMBRWF
- GMBRWC
- GMBRWN
- GMUNLK

It is also an error if the queue was not opened both for browse and for input. If the browse cursor is not currently pointing to a retrievable message, an error is returned by the MQGET call.

Lock options: The following options relate to locking messages on the queue:

GMLK

Lock message.

This option locks the message that is browsed, so that the message becomes invisible to any other handle open for the queue. The option can be specified only if one of the following options is also specified:

- GMBRWF
- GMBRWN
- GMBRWC

Only one message can be locked per queue handle, but this can be a logical message or a physical message:

- If GMCMPM is specified, all the message segments that make up the logical message are locked to the queue handle (if they are all present on the queue and available for retrieval).
- If GMCMPM is not specified, only a single physical message is locked to the queue handle. If this message happens to be a segment of a logical message, the locked segment prevents other applications using GMCMPM to retrieve or browse the logical message.

The locked message is always the one under the browse cursor, and the message can be removed from the queue by a later MQGET call that specifies the GMMUC option. Other MQGET calls using the queue handle can also remove the message (for example, a call that specifies the message identifier of the locked message).

If the call returns completion code CCFAIL, or CCWARN with reason code RC2080, no message is locked.

If the application decides not to remove the message from the queue, the lock is released by:

- Issuing another MQGET call for this handle, with either GMBRWF or GMBRWN specified (with or without GMLK); the message is unlocked if the call completes with CCOK or CCWARN, but remains locked if the call completes with CCFAIL. However, the following exceptions apply:
  - The message is not unlocked if CCWARN is returned with RC2080.

– The message is unlocked if CCFAIL is returned with RC2033.

If GMLK is also specified, the message returned is locked. If GMLK is not specified, there is no locked message after the call.

If GMWT is specified, and no message is immediately available, the unlock on the original message occurs before the start of the wait (providing the call is otherwise free from error).

- Issuing another MQGET call for this handle, with GMBRWC (without GMLK); the message is unlocked if the call completes with CCOK or CCWARN, but remains locked if the call completes with CCFAIL. However, the following exception applies:
  – The message is not unlocked if CCWARN is returned with RC2080.
- Issuing another MQGET call for this handle with GMUNLK.
- Issuing an MQCLOSE call for this handle (either explicitly, or implicitly by the application ending).

No special open option is required to specify this option, other than OOBRW, which is needed in order to specify the accompanying browse option.

This option is not valid with any of the following options:

- GMSYP
- GMPSYP
- GMUNLK

GMUNLK

Unlock message.

The message to be unlocked must have been previously locked by an MQGET call with the GMLK option. If there is no message locked for this handle, the call completes with CCWARN and `RC2209` .

The **MSGDSC**, **BUFLEN**, **BUFFER**, and **DATLEN** parameters are not checked or altered if GMUNLK is specified. No message is returned in *BUFFER*.

No special open option is required to specify this option (although OOBRW is needed to issue the lock request in the first place).

This option is not valid with any options except the following:

- GMNWT
- GMNSYP

Both of these options are assumed whether specified or not.

Message-data options: The following options relate to the processing of the message data when the message is read from the queue:

GMATM

Allow truncation of message data.

If the message buffer is too small to hold the complete message, this option allows the MQGET call to fill the buffer with as much of the message as the buffer can hold, issue a warning completion code, and complete its processing. This means:

- When browsing messages, the browse cursor is advanced to the returned message.
- When removing messages, the returned message is removed from the queue.
- Reason code `RC2079` is returned if no other error occurs.

Without this option, the buffer is still filled with as much of the message as it can hold, a warning completion code is issued, but processing is not completed. This means:

- When browsing messages, the browse cursor is not advanced.
- When removing messages, the message is not removed from the queue.

- Reason code RC2080 is returned if no other error occurs.

GMCONV

Convert message data.

This option requests that the application data in the message is converted, to conform to the *MDCSI* and *MDENC* values specified in the **MSGDSC** parameter on the MQGET call, before the data is copied to the **BUFFER** parameter.

The *MDFMT* field specified when the message was put is assumed by the conversion process to identify the nature of the data in the message. Conversion of the message data is by the queue manager for built-in formats, and by a user-written exit for other formats.

- If conversion is performed successfully, the *MDCSI* and *MDENC* fields specified in the **MSGDSC** parameter are unchanged on return from the MQGET call.
- If conversion cannot be performed successfully (but the MQGET call otherwise completes without error), the message data is returned unconverted, and the *MDCSI* and *MDENC* fields in *MSGDSC* are set to the values for the unconverted message. The completion code is CCWARN in this case.

In either case, therefore, these fields describe the character-set identifier and encoding of the message data that is returned in the **BUFFER** parameter.

See the *MDFMT* field described in "MQMD (Message descriptor) on IBM i" on page 3118 for a list of format names for which the queue manager performs the conversion.

Group and segment options: The following options relate to the processing of messages in groups and segments of logical messages. These definitions might be of help in understanding the options:

Physical message

This is the smallest unit of information that can be placed on or removed from a queue; it often corresponds to the information specified or retrieved on a single MQPUT, MQPUT1, or MQGET call. Every physical message has its own message descriptor (MQMD). Generally, physical messages are distinguished by differing values for the message identifier (*MDMID* field in MQMD), although this is not enforced by the queue manager.

Logical message

This is a single unit of application information. In the absence of system constraints, a logical message would be the same as a physical message. But where logical messages are large, system constraints might make it advisable or necessary to split a logical message into two or more physical messages, called segments.

A logical message that has been segmented consists of two or more physical messages that have the same nonnull group identifier (*MDGID* field in MQMD), and the same message sequence number (*MDSEQ* field in MQMD). The segments are distinguished by differing values for the segment offset (*MDOFF* field in MQMD), which gives the offset of the data in the physical message from the start of the data in the logical message. Because each segment is a physical message, the segments in a logical message typically have differing message identifiers.

A logical message that has not been segmented, but for which segmentation has been permitted by the sending application, also has a nonnull group identifier, although in this case there is only one physical message with that group identifier if the logical message does not belong to a message group. Logical messages for which segmentation has been inhibited by the sending application have a null group identifier (GINONE), unless the logical message belongs to a message group.

Message group

This is a set of one or more logical messages that have the same nonnull group identifier. The logical messages in the group are distinguished by differing values for the message sequence number, which is an integer in the range 1 through n, where n is the number of

logical messages in the group. If one or more of the logical messages is segmented, there are more than n physical messages in the group.

GMLOGO

Messages in groups and segments of logical messages are returned in logical order.

This option controls the order in which messages are returned by successive MQGET calls for the queue handle. The option must be specified on each of those calls in order to have an effect.

If GMLOGO is specified for successive MQGET calls for the queue handle, messages in groups are returned in the order given by their message sequence numbers, and segments of logical messages are returned in the order given by their segment offsets. This order might be different from the order in which those messages and segments occur on the queue.

Note: Specifying GMLOGO has no adverse consequences on messages that do not belong to groups and that are not segments. In effect, such messages are treated as though each belonged to a message group consisting of only one message. Thus it is perfectly safe to specify GMLOGO when retrieving messages from queues that might contain a mixture of messages in groups, message segments, and unsegmented messages not in groups.

To return the messages in the required order, the queue manager retains the group and segment information between successive MQGET calls. This information identifies the current message group and current logical message for the queue handle, the current position within the group and logical message, and whether the messages are being retrieved within a unit of work. Because the queue manager retains this information, the application does not need to set the group and segment information before each MQGET call. Specifically, it means that the application does not need to set the *MDGID*, *MDSEQ*, and *MDOFF* fields in MQMD. However, the application does need to set the GMSYP or GMNSYP option correctly on each call.

When the queue is opened, there is no current message group and no current logical message. A message group becomes the current message group when a message that has the MFMIG flag is returned by the MQGET call. With GMLOGO specified on successive calls, that group remains the current group until a message is returned that has:

- MFLMIG without MFSEG (that is, the last logical message in the group is not segmented), or
- MFLMIG with MFLSEG (that is, the message returned is the last segment of the last logical message in the group).

When such a message is returned, the message group is terminated, and on successful completion of that MQGET call there is no longer a current group. In a similar way, a logical message becomes the current logical message when a message that has the MFSEG flag is returned by the MQGET call, and that logical message is terminated when the message that has the MFLSEG flag is returned.

If no selection criteria are specified, successive MQGET calls return (in the correct order) the messages for the first message group on the queue, then the messages for the second message group, and so on, until there are no more messages available. It is possible to select the particular message groups returned by specifying one or more of the following options in the *GMMO* field:

- MOMSGI
- MOCORI
- MOGRPI

However, these options are effective only when there is no current message group or logical message; see the *GMMO* field described in this topic.

Table 336 shows the values of the *MDMID*, *MDCID*, *MDGID*, *MDSEQ*, and *MDOFF* fields that the queue manager looks for when attempting to find a message to return on the MQGET call. This applies both to removing messages from the queue, and browsing messages on the queue. The columns in the table have the following meanings:

LOG ORD
Indicates whether the GMLOGO option is specified on the call.

Cur grp
Indicates whether a current message group exists before the call.

Cur log msg
Indicates whether a current logical message exists before the call.

Other columns
Show the values that the queue manager looks for. "Previous" denotes the value returned for the field in the previous message for the queue handle.

Table 336. MQGET options relating to messages in groups and segments of logical messages

Options you specify	Group and log-msg status before call		Values the queue manager looks for				
LOG ORD	**Cur grp**	**Cur log msg**	*MDMID*	*MDCID*	*MDGID*	*MDSEQ*	*MDOFF*
Yes	No	No	Controlled by *GMMO*	Controlled by *GMMO*	Controlled by *GMMO*	1	0
Yes	No	Yes	Any message identifier	Any correlation identifier	Previous group identifier	1	Previous offset + previous segment length
Yes	Yes	No	Any message identifier	Any correlation identifier	Previous group identifier	Previous sequence number + 1	0
Yes	Yes	Yes	Any message identifier	Any correlation identifier	Previous group identifier	Previous sequence number	Previous offset + previous segment length
No	Either	Either	Controlled by *GMMO*	Controlled by *GMMO*	Controlled by *GMMO*	Controlled by *GMMO*	Controlled by *GMMO*

When multiple message groups are present on the queue and eligible for return, the groups are returned in the order determined by the position on the queue of the first segment of the first logical message in each group (that is, the physical messages that have message sequence numbers of 1, and offsets of 0, determine the order in which eligible groups are returned).

The GMLOGO option affects units of work as follows:

- If the first logical message or segment in a group is retrieved within a unit of work, all the other logical messages and segments in the group must be retrieved within a unit of work, if the same queue handle is used. However, they need not be retrieved within the same unit of work. This allows a message group consisting of many physical messages to be split across two or more consecutive units of work for the queue handle.
- If the first logical message or segment in a group is not retrieved within a unit of work, none of the other logical messages and segments in the group can be retrieved within a unit of work, if the same queue handle is used.

If these conditions are not satisfied, the MQGET call fails with reason code `RC2245` .

When GMLOGO is specified, the MQGMO supplied on the MQGET call must not be less than GMVER2, and the MQMD must not be less than MDVER2. If this condition is not satisfied, the call fails with reason code RC2256 or RC2257, as appropriate.

If GMLOGO is not specified for successive MQGET calls for the queue handle, messages are returned without regard for whether they belong to message groups, or whether they are segments of logical messages. This means that messages or segments from a particular group or logical message might be returned out of order, or they might be intermingled with messages or segments from other groups or logical messages, or with messages that are not in groups and are not segments. In this situation, the particular messages that are returned by successive MQGET calls is controlled by the MO* options specified on those calls (see the *GMMO* field described in "MQGMO (Get-message options) on IBM i" on page 3084 for details of these options).

This is the technique that can be used to restart a message group or logical message in the middle, after a system failure has occurred. When the system restarts, the application can set the *MDGID*, *MDSEQ*, *MDOFF*, and *GMMO* fields to the appropriate values, and then issue the MQGET call with GMSYP or GMNSYP set as needed, but without specifying GMLOGO. If this call is successful, the queue manager retains the group and segment information, and subsequent MQGET calls using that queue handle can specify GMLOGO as normal.

The group and segment information that the queue manager retains for the MQGET call is separate from the group and segment information that it retains for the MQPUT call. In addition, the queue manager retains separate information for:

- MQGET calls that remove messages from the queue.
- MQGET calls that browse messages on the queue.

For any given queue handle, the application is free to mix MQGET calls that specify GMLOGO with MQGET calls that do not, but the following points must be noted:

- If GMLOGO is not specified, each successful MQGET call causes the queue manager to set the saved group and segment information to the values corresponding to the message returned; this replaces the existing group and segment information retained by the queue manager for the queue handle. Only the information appropriate to the action of the call (browse or remove) is modified.
- If GMLOGO is not specified, the call does not fail if there is a current message group or logical message; the call might however succeed with a CCWARN completion code. Table 337 on page 3099 shows the various cases that can arise. In these cases, if the completion code is not CCOK, the reason code is one of the following:
  - RC2241
  - RC2242
  - RC2245

  Note: The queue manager does not check the group and segment information when browsing a queue, or when closing a queue that was opened for browse but not input; in those cases the completion code is always CCOK (assuming no other errors).

Table 337. Outcome when MQGET or MQCLOSE call is not consistent with group and segment information

Current call is	Previous call was MQGET with GMLOGO	Previous call was MQGET without GMLOGO
MQGET with GMLOGO	CCFAIL	CCFAIL
MQGET without GMLOGO	CCWARN	CCOK
MQCLOSE with an unterminated group or logical message	CCWARN	CCOK

Applications that simply want to retrieve messages and segments in logical order are recommended to specify GMLOGO, as this is the simplest option to use. This option relieves the application of the need to manage the group and segment information, because the queue manager manages that information. However, specialized applications might need more control than provided by the GMLOGO option, and this can be achieved by not specifying that option. If this is done, the application must ensure that the *MDMID*, *MDCID*, *MDGID*, *MDSEQ*, and *MDOFF* fields in MQMD, and the MO* options in *GMMO* in MQGMO, are set correctly, before each MQGET call.

For example, an application that wants to forward physical messages that it receives, without regard for whether those messages are in groups or segments of logical messages, should not specify GMLOGO. This is because in a complex network with multiple paths between sending and receiving queue managers, the physical messages might arrive out of order. By not specifying GMLOGO and the corresponding PMLOGO on the MQPUT call, the forwarding application can retrieve and forward each physical message as soon as it arrives, without having to wait for the next one in logical order to arrive.

GMLOGO can be specified with any of the other GM* options, and with various of the MO* options in appropriate circumstances.

GMCMPM

Only complete logical messages are retrievable.

This option specifies that only a complete logical message can be returned by the MQGET call. If the logical message is segmented, the queue manager reassembles the segments and returns the complete logical message to the application; the fact that the logical message was segmented is not apparent to the application retrieving it.

Note: This is the only option that causes the queue manager to reassemble message segments. If not specified, segments are returned individually to the application if they are present on the queue (and they satisfy the other selection criteria specified on the MQGET call). Applications that do not want to receive individual segments should therefore always specify GMCMPM.

To use this option, the application must provide a buffer which is large enough to accommodate the complete message, or specify the GMATM option.

If the queue contains segmented messages with some of the segments missing (perhaps because they have been delayed in the network and have not yet arrived), specifying GMCMPM prevents the retrieval of segments belonging to incomplete logical messages. However, those message segments still contribute to the value of the **CurrentQDepth** queue attribute; this means that there might be no retrievable logical messages, even though *CurrentQDepth* is greater than zero.

For persistent messages, the queue manager can reassemble the segments only within a unit of work:

- If the MQGET call is operating within a user-defined unit of work, that unit of work is used. If the call fails part way through the reassembly process, the queue manager

reinstates on the queue any segments that were removed during reassembly. However, the failure does not prevent the unit of work being committed successfully.

- If the call is operating outside a user-defined unit of work, and there is no user-defined unit of work in existence, the queue manager creates a unit of work just for the duration of the call. If the call is successful, the queue manager commits the unit of work automatically (the application does not need to do this). If the call fails, the queue manager backs out the unit of work.
- If the call is operating outside a user-defined unit of work, but a user-defined unit of work does exist, the queue manager is unable to perform reassembly. If the message does not require reassembly, the call can still succeed. But if the message does require reassembly, the call fails with reason code RC2255 .

For nonpersistent messages, the queue manager does not require a unit of work to be available in order to perform reassembly.

Each physical message that is a segment has its own message descriptor. For the segments constituting a single logical message, most of the fields in the message descriptor is the same for all segments in the logical message - typically it is only the *MDMID*, *MDOFF*, and *MDMFL* fields that differ between segments in the logical message. However, if a segment is placed on a dead-letter queue at an intermediate queue manager, the DLQ handler retrieves the message specifying the GMCONV option, and this might result in the character set or encoding of the segment being changed. If the DLQ handler successfully sends the segment on its way, the segment might have a character set or encoding that differs from the other segments in the logical message when the segment finally arrives at the destination queue manager.

A logical message consisting of segments in which the *MDCSI*, *MDENC*, or both fields differ cannot be reassembled by the queue manager into a single logical message. Instead, the queue manager reassembles and returns the first few consecutive segments at the start of the logical message that have the same character-set identifiers and encodings, and the MQGET call completes with completion code CCWARN and reason code RC2243 or RC2244, as appropriate. This happens regardless of whether GMCONV is specified. To retrieve the remaining segments, the application must reissue the MQGET call without the GMCMPM option, retrieving the segments one by one. GMLOGO can be used to retrieve the remaining segments in order.

It is also possible for an application which puts segments to set other fields in the message descriptor to values that differ between segments. However, there is no advantage in doing this if the receiving application uses GMCMPM to retrieve the logical message. When the queue manager reassembles a logical message, it returns in the message descriptor the values from the message descriptor for the first segment; the only exception is the *MDMFL* field, which the queue manager sets to indicate that the reassembled message is the only segment.

If GMCMPM is specified for a report message, the queue manager performs special processing. The queue manager checks the queue to see if all the report messages of that report type relating to the different segments in the logical message are present on the queue. If they are, they can be retrieved as a single message by specifying GMCMPM. For this to be possible, either the report messages must be generated by a queue manager or MCA which supports segmentation, or the originating application must request at least 100 bytes of message data (that is, the appropriate RO*D or RO*F options must be specified). If less than the full amount of application data is present for a segment, the missing bytes are replaced by nulls in the report message returned.

If GMCMPM is specified with GMMUC or GMBRWC, the browse cursor must be positioned on a message with a *MDOFF* field in MQMD that has a value of 0. If this condition is not satisfied, the call fails with reason code RC2246 .

GMCMPM implies GMASGA, which need not therefore be specified.

GMCMPM can be specified with any of the other GM* options apart from GMPSYP, and with any of the MO* options apart from MOOFFS.

GMAMSA

All messages in group must be available.

This option specifies that messages in a group become available for retrieval only when all messages in the group are available. If the queue contains message groups with some of the messages missing (perhaps because they have been delayed in the network and have not yet arrived), specifying GMAMSA prevents retrieval of messages belonging to incomplete groups. However, those messages still contribute to the value of the **CurrentQDepth** queue attribute; this means that there might be no retrievable message groups, even though **CurrentQDepth** is greater than zero. If there are no other messages that are retrievable, reason code RC2033 is returned after the specified wait interval (if any) has expired.

The processing of GMAMSA depends on whether GMLOGO is also specified:

- If both options are specified, GMAMSA affects only when there is no current group or logical message. If there is a current group or logical message, GMAMSA is ignored. This means that GMAMSA can remain on when processing messages in logical order.
- If GMAMSA is specified without GMLOGO, GMAMSA always has an effect. This means that the option must be turned off after the first message in the group has been removed from the queue, in order to be able to remove the remaining messages in the group.

Successful completion of an MQGET call specifying GMAMSA means that at the time that the MQGET call was issued, all the messages in the group were on the queue. However, be aware that other applications are still able to remove messages from the group (the group is not locked to the application that retrieves the first message in the group).

If this option is not specified, messages belonging to groups can be retrieved even when the group is incomplete.

GMAMSA implies GMASGA, which need not therefore be specified.

GMAMSA can be specified with any of the other GM* options, and with any of the MO* options.

GMASGA

All segments in a logical message must be available.

This option specifies that segments in a logical message become available for retrieval only when all segments in the logical message are available. If the queue contains segmented messages with some of the segments missing (perhaps because they have been delayed in the network and have not yet arrived), specifying GMASGA prevents retrieval of segments belonging to incomplete logical messages. However those segments still contribute to the value of the **CurrentQDepth** queue attribute; this means that there might be no retrievable logical messages, even though **CurrentQDepth** is greater than zero. If there are no other messages that are retrievable, reason code RC2033 is returned after the specified wait interval (if any) has expired.

The processing of GMASGA depends on whether GMLOGO is also specified:

- If both options are specified, GMASGA has an effect only when there is no current logical message. If there is a current logical message, GMASGA is ignored. This means that GMASGA can remain on when processing messages in logical order.
- If GMASGA is specified without GMLOGO, GMASGA always has an effect. This means that the option must be turned off after the first segment in the logical message has been removed from the queue, in order to be able to remove the remaining segments in the logical message.

If this option is not specified, message segments can be retrieved even when the logical message is incomplete.

While both GMCMPM and GMASGA require all segments to be available before any of them can be retrieved, the former returns the complete message, whereas the latter allows the segments to be retrieved one by one.

If GMASGA is specified for a report message, the queue manager performs special processing. The queue manager checks the queue to see if there is at least one report message for each of the segments that make up the complete logical message. If there is, the GMASGA condition is satisfied. However, the queue manager does not check the type of the report messages present, and so there might be a mixture of report types in the report messages relating to the segments of the logical message. As a result, the success of GMASGA does not imply that GMCMPM succeeds. If there is a mixture of report types present for the segments of a particular logical message, those report messages must be retrieved one by one.

GMASGA can be specified with any of the other GM* options, and with any of the MO* options.

Default option: If none of the options described are required, the following option can be used:

GMNONE
No options specified.

This value can be used to indicate that no other options have been specified; all options assume their default values. GMNONE is defined to aid program documentation; it is not intended that this option is used with any other, but as its value is zero, such use cannot be detected.

The initial value of the *GMOPT* field is GMNWT.

GMRE1 (1 byte character string)

Reserved.

This is a reserved field. The initial value of this field is a blank character. This field is ignored if *GMVER* is less than GMVER2.

GMRL (10 digit signed integer)

Length of message data returned (bytes).

This is an output field that is set by the queue manager to the length in bytes of the message data returned by the MQGET call in the **BUFFER** parameter. If the queue manager does not support this capability, *GMRL* is set to the value RLUNDF.

When messages are converted between encodings or character sets, the message data can sometimes change size. On return from the MQGET call:

- If *GMRL* is not RLUNDF, the number of bytes of message data returned is given by *GMRL*.
- If *GMRL* has the value RLUNDF, the number of bytes of message data returned is typically given by the smaller of *BUFLEN* and *DATLEN*, but can be less than this if the MQGET call completes with reason code RC2079 . If this happens, the insignificant bytes in the **BUFFER** parameter are set to nulls.

The following special value is defined:

RLUNDF
Length of returned data not defined.

The initial value of this field is RLUNDF. This field is ignored if *GMVER* is less than GMVER3.

GMRQN (48 byte character string)

Resolved name of destination queue.

This is an output field which is set by the queue manager to the local name of the queue from which the message was retrieved, as defined to the local queue manager. This is different from the name used to open the queue if:

- An alias queue was opened (in which case, the name of the local queue to which the alias resolved is returned), or
- A model queue was opened (in which case, the name of the dynamic local queue is returned).

The length of this field is given by LNQN. The initial value of this field is 48 blank characters.

GMRS2 (1 byte character string)

Reserved.

This is a reserved field. The initial value of this field is a blank character. This field is ignored if *GMVER* is less than GMVER4.

GMSEG (1 byte character string)

Flag indicating whether further segmentation is allowed for the message retrieved.

It has one of the following values:

SEGIHB
Segmentation not allowed.

SEGALW
Segmentation allowed.

This is an output field. The initial value of this field is SEGIHB. This field is ignored if *GMVER* is less than GMVER2.

GMSG1 (10 digit signed integer)

Signal.

This is a reserved field; its value is not significant. The initial value of this field is 0.

GMSG2 (10 digit signed integer)

Signal identifier.

This is a reserved field; its value is not significant.

GMSID (4 byte character string)

Structure identifier.

The value must be:

GMSIDV
Identifier for get-message options structure.

This field is always an input field. The initial value of this field is GMSIDV.

GMSST (1 byte character string)

Flag indicating whether message retrieved is a segment of a logical message.

It has one of the following values:

SSNSEG
Message is not a segment.

SSSEG
Message is a segment, but is not the last segment of the logical message.

SSLSEG
Message is the last segment of the logical message.

This is also the value returned if the logical message consists of only one segment.

This field is an output field. The initial value of this field is SSNSEG. This field is ignored if *GMVER* is less than GMVER2.

GMTOK (16 byte bit string)

Message token.

This is a reserved field; its value is not significant. The following special value is defined:

MTKNON
No message token.

The value is binary zero for the length of the field.

The length of this field is given by LNMTOK. The initial value of this field is MTKNON. This field is ignored if *GMVER* is less than GMVER3.

GMVER (10 digit signed integer)

Structure version number.

The value must be one of the following:

GMVER1
Version-1 get-message options structure.

GMVER2
Version-2 get-message options structure.

GMVER3
Version-3 get-message options structure.

GMVER4
Version-4 get-message options structure.

Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

GMVERC
Current version of get-message options structure.

This field is always an input field. The initial value of this field is GMVER1.

GMVER (10 digit signed integer)

Structure version number.

The value must be one of the following:

GMVER1
Version-1 get-message options structure.

GMVER2
Version-2 get-message options structure.

GMVER3
Version-3 get-message options structure.

GMVER4
Version-4 get-message options structure.

Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

GMVERC
Current version of get-message options structure.

This field is always an input field. The initial value of this field is GMVER1.

GMWI (10 digit signed integer)

Wait interval.

This is the approximate time, expressed in milliseconds, that the MQGET call waits for a suitable message to arrive (that is, a message satisfying the selection criteria specified in the **MSGDSC** parameter of the MQGET call; see the *MDMID* field described in "MQMD (Message descriptor) on IBM i" on page 3118 for more details). If no suitable message has arrived after this time has elapsed, the call completes with CCFAIL and reason code RC2033.

GMWI is used with the GMWT option. It is ignored if this option is not specified. If it is specified, *GMWI* must be greater than or equal to zero, or the following special value:

WIULIM
Unlimited wait interval.

The initial value of this field is 0.

Initial values

Table 338. Initial values of fields in MQGMO

Field name	Name of constant	Value of constant
GMSID	GMSIDV	'GMO¬'
GMVER	GMVER1	1
GMOPT	GMNWT	0
GMWI	None	0
GMSG1	None	0
GMSG2	None	0
GMRQN	None	Blanks
GMMO	MOMSGI + MOCORI	3
GMGST	GSNIG	'⌂'
GMSST	SSNSEG	'⌂'
GMSEG	SEGIHB	'⌂'
GMRE1	None	'⌂'
GMTOK	MTKNON	Nulls
GMRL	RLUNDF	-1
GMRS2	None	'⌂'
GMMH	HMNONE	0

Notes:

1. The symbol ¬ represents a single blank character.

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D*
D* MQGMO Structure
D*
D* Structure identifier
D  GMSID                 1      4    INZ('GMO ')
```

```
D* Structure version number
D  GMVER                  5      8I 0 INZ(1)
D* Options that control the action ofMQGET
D  GMOPT                  9     12I 0 INZ(0)
D* Wait interval
D  GMWI                  13     16I 0 INZ(0)
D* Signal
D  GMSG1                 17     20I 0 INZ(0)
D* Signal identifier
D  GMSG2                 21     24I 0 INZ(0)
D* Resolved name of destination queue
D  GMRQN                 25     72    INZ
D* Options controlling selection criteriaused for MQGET
D  GMMO                  73     76I 0 INZ(3)
D* Flag indicating whether messageretrieved is in a group
D  GMGST                 77     77    INZ(' ')
D* Flag indicating whether messageretrieved is a segment of a
D* logicalmessage
D  GMSST                 78     78    INZ(' ')
D* Flag indicating whether furthersegmentation is allowed for themessage
D* retrieved
D  GMSEG                 79     79    INZ(' ')
D* Reserved
D  GMRE1                 80     80    INZ
D* Message token
D  GMTOK                 81     96    INZ(X'00000000000000-
D                                     0000000000000000')
D* Length of message data returned(bytes)
D  GMRL                  97    100I 0 INZ(-1)
D* Reserved
D  GMRS2                101    104I 0 INZ(0)
D* Message handle
D  GMMH                 105    112I 0 INZ(0)
```

MQIIH (IMS information header) on IBM i: 

The MQIIH structure describes the information that must be present at the start of a message sent to the IMS bridge through IBM MQ for z/OS.

Overview

Format name: FMIMS.

Character set and encoding: Special conditions apply to the character set and encoding used for the MQIIH structure and application message data:

- Applications that connect to the queue manager that owns the IMS bridge queue must provide an MQIIH structure that is in the character set and encoding of the queue manager. This is because data conversion of the MQIIH structure is not performed in this case.
- Applications that connect to other queue managers can provide an MQIIH structure that is in any of the supported character sets and encodings; conversion of the MQIIH is performed by the receiving message channel agent connected to the queue manager that owns the IMS bridge queue.

  Note: There is one exception to this. If the queue manager that owns the IMS bridge queue is using CICS for distributed queuing, the MQIIH must be in the character set and encoding of the queue manager that owns the IMS bridge queue.
- The application message data following the MQIIH structure must be in the same character set and encoding as the MQIIH structure. The *IICSI* and *IIENC* fields in the MQIIH structure cannot be used to specify the character set and encoding of the application message data.

  A data-conversion exit must be provided by the user to convert the application message data if the data is not one of the built-in formats supported by the queue manager.

Authenticating passtickets for IMS bridge applications

It is now possible for IBM MQ administrators to specify the application name to be used for authenticating passtickets, for IMS bridge applications. To do this, the application name is specified as a new attribute PTKTAPPL for the STGCLASS object definition, as a 1 to 8 character alphanumeric string.

A blank value means that authentication occurs as with previous releases of IBM MQ, that is, no application name flows on the authentication request, and the MVSxxxx value to is used instead.

A value of 1 - 8 alphanumeric characters must follow the rules for passticket application names as described in the RACF publications.

IBM MQ Administrators and RACF administrators must both agree on the valid application names to be used. The RACF administrator must create a profile in the PTKTDATA class giving READ access to the user IDs of all applications that are to be granted access. The IBM MQ administrator must create or alter the required STGCLASS definitions that specify the application name to be used for passticket authentication.

For related information, see the *Script (MQSC) Command Reference*.

Fields

The MQIIH structure contains the following fields; the fields are described in **alphabetical order**:

IIAUT (8-byte character string)

RACF password or passticket.

This is optional; if specified, it is used with the user ID in the MQMD security context to build a UTOKEN that is sent to IMS to provide a security context. If it is not specified, the user ID is used without verification. This depends on the setting of the RACF switches, which may require an authenticator to be present.

This is ignored if the first byte is blank or null. The following special value may be used:

IAUNON
No authentication.

The length of this field is given by LNAUTH. The initial value of this field is IAUNON.

IICMT (1-byte character string)

Commit mode.

See the *OTMA Reference* for more information about IMS commit modes. The value must be one of the following:

ICMCTS
Commit then send.

This mode implies double queuing of output, but shorter region occupancy times. Fast-path and conversational transactions cannot run with this mode.

ICMSTC
Send then commit.

The initial value of this field is ICMCTS.

IICSI (10-digit signed integer)

Reserved.

This is a reserved field; its value is not significant. The initial value of this field is 0.

IIENC (10-digit signed integer)

Reserved.

This is a reserved field; its value is not significant. The initial value of this field is 0.

IIFLG (10-digit signed integer)

Flags.

The value must be:

IINONE
No flags.

The initial value of this field is IINONE.

IIFMT (8-byte character string)

IBM MQ format name of data that follows MQIIH.

This specifies the IBM MQ format name of the data that follows the MQIIH structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *MDFMT* field in MQMD.

The length of this field is given by LNFMT. The initial value of this field is FMNONE.

IILEN (10-digit signed integer)

Length of MQIIH structure.

The value must be:

IILEN1
Length of IMS information header structure.

The initial value of this field is IILEN1.

IILTO (8-byte character string)

Logical terminal override.

This is placed in the IO PCB field. It is optional; if it is not specified the TPIPE name is used. It is ignored if the first byte is blank, or null.

The length of this field is given by LNLTOV. The initial value of this field is 8 blank characters.

IIMMN (8-byte character string)

Message format services map name.

This is placed in the IO PCB field. It is optional. On input it represents the MID, on output it represents the MOD. It is ignored if the first byte is blank or null.

The length of this field is given by LNMFMN. The initial value of this field is 8 blank characters.

IIRFM (8-byte character string)

IBM MQ format name of reply message.

This is the IBM MQ format name of the reply message that will be sent in response to the current message. The rules for coding this are the same as those for the *MDFMT* field in MQMD.

The length of this field is given by LNFMT. The initial value of this field is FMNONE.

IIRSV (1-byte character string)

Reserved.

This is a reserved field; it must be blank.

IISEC (1-byte character string)

Security scope.

This indicates the required IMS security processing. The following values are defined:

ISSCHK
Check security scope.

An ACEE is built in the control region, but not in the dependent region.

ISSFUL
Full security scope.

A cached ACEE is built in the control region and a non-cached ACEE is built in the dependent region. If you use ISSFUL, you must ensure that the user ID for which the ACEE is built has access to the resources used in the dependent region.

If ISSCHK and ISSFUL are not specified for this field, ISSCHK is assumed.

The initial value of this field is ISSCHK.

IISID (4-byte character string)

Structure identifier.

The value must be:

IISIDV
Identifier for IMS information header structure.

The initial value of this field is IISIDV.

IITID (16-byte bit string)

Transaction instance identifier.

This field is used by output messages from IMS so is ignored on first input. If *IITST* is set to ITSIC, this must be provided in the next input, and all subsequent inputs, to enable IMS to correlate the messages to the correct conversation. The following special value may be used:

ITINON
No transaction instance ID.

The length of this field is given by LNTIID. The initial value of this field is ITINON.

IITST (1-byte character string)

Transaction state.

This indicates the IMS conversation state. This is ignored on first input because no conversation exists. On subsequent inputs it indicates whether a conversation is active or not. On output it is set by IMS. The value must be one of the following:

ITSIC In conversation.

ITSNIC
Not in conversation.

ITSARC
Return transaction state data in architected form.

This value is used only with the IMS /DISPLAY TRAN command. It causes the transaction state data to be returned in the IMS architected form instead of character form. See Writing IMS transaction programs through IBM MQ for further details.

The initial value of this field is ITSNIC.

IIVER (10-digit signed integer)

Structure version number.

The value must be:

IIVER1
Version number for IMS information header structure.

The following constant specifies the version number of the current version:

IIVERC
Current version of IMS information header structure.

The initial value of this field is IIVER1.

Initial values

Table 339. Initial values of fields in MQIIH

Field name	Name of constant	Value of constant
IISID	IISIDV	'IIH¬'
IIVER	IIVER1	1
IILEN	IILEN1	84
IIENC	None	0
IICSI	None	0
IIFMT	FMNONE	Blanks
IIFLG	IINONE	0
IILTO	None	Blanks
IIMMN	None	Blanks
IIRFM	FMNONE	Blanks
IIAUT	IAUNON	Blanks
IITID	ITINON	Nulls
IITST	ITSNIC	'⌂'
IICMT	ICMCTS	'0'
IISEC	ISSCHK	'C'
IIRSV	None	'⌂'

Notes:

1. The symbol ¬ represents a single blank character.

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D*
D* MQIIH Structure
D*
D* Structure identifier
D  IISID                 1      4    INZ('IIH ')
D* Structure version number
D  IIVER                 5      8I 0 INZ(1)
```

```
D* Length of MQIIH structure
D  IILEN                  9     12I 0 INZ(84)
D* Reserved
D  IIENC                 13     16I 0 INZ(0)
D* Reserved
D  IICSI                 17     20I 0 INZ(0)
D* MQ format name of data that followsMQIIH
D  IIFMT                 21     28    INZ('        ')
D* Flags
D  IIFLG                 29     32I 0 INZ(0)
D* Logical terminal override
D  IILTO                 33     40    INZ
D* Message format services map name
D  IIMMN                 41     48    INZ
D* MQ format name of reply message
D  IIRFM                 49     56    INZ('        ')
D* RACF password or passticket
D  IIAUT                 57     64    INZ('        ')
D* Transaction instance identifier
D  IITID                 65     80    INZ(X'00000000000000-
D                                     000000000000000000')
D* Transaction state
D  IITST                 81     81    INZ(' ')
D* Commit mode
D  IICMT                 82     82    INZ('0')
D* Security scope
D  IISEC                 83     83    INZ('C')
D* Reserved
D  IIRSV                 84     84    INZ
```

MQIMPO (Inquire message property options) on IBM i: 

The MQIMPO structure allows applications to specify options that control how properties of messages are inquired.

Overview

Purpose: The structure is an input parameter on the MQINQMP call.

Character set and encoding: Data in MQIMPO must be in the character set of the application and encoding of the application (ENNAT).

- "Fields"
- "Initial values" on page 3117
- "RPG declaration" on page 3117

Fields

The MQIMPO structure contains the following fields; the fields are described in **alphabetical order**:

IPOPT (10-digit signed integer)

The following options control the action of MQINQMP. You can specify one or more of these options. To specify more than one option, either add the values together (do not add the same constant more than once), or combine the values using the bitwise OR operation (if the programming language supports bit operations). Combinations of options that are not valid are noted; all other combinations are valid.

Value data options: The following options relate to the processing of the value data when the property is retrieved from the message.

IPCVAL

This option requests that the value of the property be converted to conform to the *IPREQCSI* and *IPREQENC* values specified before the MQINQMP call returns the property value in the *Value* area.

- If conversion is successful, the *IPRETCSI* and *IPRETENC* fields are set to the same as *IPREQCSI* and *IPREQENC* on return from the MQINQMP call.
- If conversion fails, but the MQINQMP call otherwise completes without error, the property value is returned unconverted.

  If the property is a string, the *IPRETCSI* and *IPRETENC* fields are set to the character set and encoding of the unconverted string. The completion code is CCWARN in this case, with reason code RC2466. The property cursor is advanced to the returned property.

If the property value expands during conversion, and exceeds the size of the **Value** parameter, the value is returned unconverted, with completion code CCFAIL; the reason code is set to RC2469.

The **DataLength** parameter of the MQINQMP call returns the length that the property value would have converted to, in order to allow the application to determine the size of the buffer required to accommodate the converted property value. The property cursor is unchanged.

This option also requests that:

- If the property name contains a wildcard, and
- The *IPRETNAMECHRP* field is initialized with an address or offset for the returned name,

then the returned name is converted to conform to the *IPREQCSI* and *IPREQENC* values.

- If conversion is successful, the *VSCCSID* field of *IPRETNAMECHRP* and the encoding of the returned name are set to the input value of *IPREQCSI* and *IPREQENC*.
- If conversion fails, but the MQINQMP call otherwise completes without error or warning, the returned name is unconverted. The completion code is CCWARN in this case, with reason code RC2492.

  The property cursor is advanced to the returned property. RC2466 is returned if both the value and the name are not converted.

If the returned name expands during conversion, and exceeds the size of the *VSBufsize* field of the *RequestedName*, the returned string is left unconverted, with completion code CCFAIL and the reason code is set to RC2465.

The *VSLength* field of the MQCHARV structure returns the length that the property value would have converted to, in order to allow the application to determine the size of the buffer required to accommodate the converted property value. The property cursor is unchanged.

IPCTYP

This option requests that the value of the property be converted from its current data type, into the data type specified on the **Type** parameter of the MQINQMP call.

- If conversion is successful, the **Type** parameter is unchanged on return of the MQINQMP call.
- If conversion fails, but the MQINQMP call otherwise completes without error, the call fails with reason RC2470. The property cursor is unchanged.

If the conversion of the data type causes the value to expand during conversion, and the converted value exceeds the size of the **Value** parameter, the value is returned unconverted, with completion code CCFAIL and the reason code is set to RC2469.

The **DataLength** parameter of the MQINQMP call returns the length that the property value would have converted to, in order to allow the application to determine the size of the buffer required to accommodate the converted property value. The property cursor is unchanged.

If the value of the **Type** parameter of the MQINQMP call is not valid, the call fails with reason RC2473.

If the requested data type conversion is not supported, the call fails with reason RC2470. The following data type conversions are supported:

Property data type	Supported target data types
TYPBOL	TYPSTR, TYPI8, TYPI16, TYPI32, TYPI64
TYPBST	TYPSTR
TYPI8	TYPSTR, TYPI16, TYPI32, TYPI64
TYPI16	TYPSTR, TYPI32, TYPI64
TYPI32	TYPSTR, TYPI64
TYPI64	TYPSTR
TYPF32	TYPSTR, TYPF64
TYPF64	TYPSTR
TYPSTR	TYPBOL, TYPI8, TYPI16, TYPI32, TYPI64, TYPF32, TYPF64
TYPNUL	None

The general rules governing the supported conversions are as follows:

- Numeric property values can be converted from one data type to another, provided that no data is lost during the conversion.

  For example, the value of a property with data type TYPI32 can be converted into a value with data type TYPI64, but cannot be converted into a value with data type TYPI16.
- A property value of any data type can be converted into a string.
- A string property value can be converted to any other data type provided the string is formatted correctly for the conversion. If an application attempts to convert a string property value that is not formatted correctly, IBM MQ returns reason code RC2472.
- If an application attempts a conversion that is not supported, IBM MQ returns reason code RC2470.

The specific rules for converting a property value from one data type to another are as follows:

- When converting a TYPBOL property value to a string, the value TRUE is converted to the string "TRUE", and the value false is converted to the string "FALSE".
- When converting a TYPBOL property value to a numeric data type, the value TRUE is converted to one, and the value FALSE is converted to zero.
- When converting a string property value to a TYPBOL value, the string "TRUE" , or "1" , is converted to TRUE, and the string "FALSE", or "0", is converted to FALSE.

  Note that the terms "TRUE" and "FALSE" are not case sensitive.

  Any other string cannot be converted; IBM MQ returns reason code RC2472.
- When converting a string property value to a value with data type TYPI8, TYPI16, TYPI32 or TYPI64, the string must have the following format:

  ```
  [blanks][sign]digits
  ```

  The meanings of the components of the string are as follows:

blanks Optional leading blank characters

sign An optional plus sign (+) or minus sign (-) character.

digits A contiguous sequence of digit characters (0-9). At least one digit character must be present.

After the sequence of digit characters, the string can contain other characters that are not digit characters, but the conversion stops as soon as the first of these characters is reached. The string is assumed to represent a decimal integer.

IBM MQ returns reason code RC2472 if the string is not formatted correctly.

- When converting a string property value to a value with data type TYPF32 or TYPF64, the string must have the following format:

  ```
  [blanks][sign]digits[.digits][e_char[e_sign]e_digits]
  ```

  The meanings of the components of the string are as follows:

  blanks Optional leading blank characters

  sign An optional plus sign (+) or minus sign (-) character.

  digits A contiguous sequence of digit characters (0-9). At least one digit character must be present.

  e_char An exponent character, which is either "E" or "e".

  e_sign An optional plus sign (+) or minus sign (-) character for the exponent.

  e_digits
  A contiguous sequence of digit characters (0-9) for the exponent. At least one digit character must be present if the string contains an exponent character.

  After the sequence of digit characters, or the optional characters representing an exponent, the string can contain other characters that are not digit characters, but the conversion stops as soon as the first of these characters is reached. The string is assumed to represent a decimal floating point number with an exponent that is a power of 10.

  IBM MQ returns reason code RC2472 if the string is not formatted correctly.

- When converting a numeric property value to a string, the value is converted to the string representation of the value as a decimal number, not the string containing the ASCII character for that value. For example, the integer 65 is converted to the string "65", not the string "A".
- When converting a byte string property value to a string, each byte is converted to the two hexadecimal characters that represent the byte. For example, the byte array {0xF1, 0x12, 0x00, 0xFF} is converted to the string "F11200FF".

IPQLEN

Query the type and length of the property value. The length is returned in the **DataLength** parameter of the MQINQMP call. The property value is not returned.

If a *ReturnedName* buffer is specified, the *VSLength* field of the MQCHARV structure is filled in with the length of the property name. The property name is not returned.

Iteration options: The following options relate to iterating over properties, using a name with a wildcard character

IPINQF

Inquire on the first property that matches the specified name. After this call, a cursor is established on the property that is returned.

This is the default value.

The IPINQC option can subsequently be used with an MQINQMP call, if required, to inquire on the same property again.

Note that there is only one property cursor; therefore, if the property name, specified in the MQINQMP call, changes the cursor is reset.

This option is not valid with either of the following options:

IPINQN

IPINQC

IPINQN

Inquires on the next property that matches the specified name, continuing the search from the property cursor. The cursor is advanced to the property that is returned.

If this is the first MQINQMP call for the specified name, then the first property that matches the specified name is returned.

The IPINQC option can subsequently be used with an MQINQMP call if required, to inquire on the same property again.

If the property under the cursor has been deleted, MQINQMP returns the next matching property following the one that has been deleted.

If a property is added that matches the wildcard, while an iteration is in progress, the property might or might not be returned during the completion of the iteration. The property is returned once the iteration restarts using IPINQF.

A property matching the wildcard that was deleted, while the iteration was in progress, is not returned subsequent to its deletion.

This option is not valid with either of the following options:

IPINQF

IPINQC

IPINQC

Retrieve the value of the property pointed to by the property cursor. The property pointed to by the property cursor is the one that was last inquired, using either the IPINQF or the IPINQN option.

The property cursor is reset when the message handle is reused, when the message handle is specified in the *MsgHandle* field of the MQGMO on an MQGET call, or when the message handle is specified in *OriginalMsgHandle* or *NewMsgHandle* fields of the MQPMO structure on an MQPUT call.

If this option is used when the property cursor has not yet been established, or if the property pointed to by the property cursor has been deleted, the call fails with completion code CCFAIL and reason RC2471.

This option is not valid with either of the following options:

IPINQF

IPINQN

If none of the options previously described is required, the following option can be used:

IPNONE

Use this value to indicate that no other options have been specified; all options assume their default values.

IPNONE aids program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

This is always an input field. The initial value of this field is IPINQF.

IPREQCSI (10-digit signed integer)

The character set that the inquired property value is to be converted into if the value is a character string. This is also the character set into which the *ReturnedName* is to be converted when IPCVAL or IPCTYP is specified.

The initial value of this field is CSAPL.

IPREQENC (10-digit signed integer)

This is the encoding into which the inquired property value is to be converted when IPCVAL or IPCTYP is specified.

The initial value of this field is ENNAT.

IPRE1 (10-digit signed integer)

This is a reserved field. The initial value of this field is a blank character.

IPRETCSI (10-digit signed integer)

On output, this is the character set of the value returned if the **Type** parameter of the MQINQMP call is TYPSTR.

If the IPCVAL option is specified and conversion was successful, the *ReturnedCCSID* field, on return, is the same value as the value passed in.

The initial value of this field is zero.

IPRETENC (10-digit signed integer)

On output, this is the encoding of the value returned.

If the IPCVAL option is specified and conversion was successful, the *ReturnedEncoding* field, on return, is the same value as the value passed in.

The initial value of this field is ENNAT.

IPRETNAMCHRP (10-digit signed integer)

The actual name of the inquired property.

On input a string buffer can be passed in using the *VSPtr* or *VSOffset* field of the MQCHARV structure. The length of the string buffer is specified using the *VSBufsize* field of the MQCHARV structure.

On return from the MQINQMP call, the string buffer is completed with the name of the property that was inquired, provided the string buffer was long enough to fully contain the name. The *VSLength* field of the MQCHARV structure is filled in with the length of the property name. The *VSCCSID* field of the MQCHARV structure is filled in to indicate the character set of the returned name, whether or not conversion of the name failed.

This is an input/output field. The initial value of this field is MQCHARV_DEFAULT.

IPSID (10-digit signed integer)

This is the structure identifier. The value must be:

IPSIDV
Identifier for inquire message property options structure.

This is always an input field. The initial value of this field is IPSIDV.

IPTYP (10-digit signed integer)

A string representation of the data type of the property.

If the property was specified in an MQRFH2 header and the MQRFH2 `dt` attribute is not recognized, this field can be used to determine the data type of the property. *TypeString* is returned in coded character set 1208 (UTF-8), and is the first eight bytes of the value of the `dt` attribute of the property that failed to be recognized

This is always an output field. The initial value of this field is the null string in the C programming language, and 8 blank characters in other programming languages.

IPVER (10-digit signed integer)

This is the structure version number. The value must be:

IPVER1
Version number for inquire message property options structure.

The following constant specifies the version number of the current version:

IPVERC
Current version of inquire message property options structure.

This is always an input field. The initial value of this field is IPVER1.

Initial values

Table 340. Initial values of fields in MQIPMO

Field name	Name of constant	Value of constant
IPSID	IPSIDV	'IMPO'
IPVER	IPVER1	1
IPOPT	IPINQF	
IPREQENC	ENNAT	
IPREQCSI	CSAPL	
IPRETENC	ENNAT	
IPRETCSI	0	
IPRE1	0	
IPRETNAMCHRP		
IPTYP		blanks

RPG declaration

```
D* MQIMPO Structure
D*
D*
D* Structure identifier
D IPSID         1   4  INZ('IMPO')
D*
D* Structure version number
D IPVER         5   8I 0 INZ(1)
D*
** Options that control the action of
D* MQINQMP
D IPOPT         9   12I 0 INZ(0)
D*
D* Requested encoding of Value
D IPREQENC       13   16I 0 INZ(273)
D*
** Requested character set identifier
D* of Value
D IPREQCSI       17   20I 0 INZ(-3)
D*
D* Returned encoding of Value
D IPRETENC       21   24I 0 INZ(273)
D*
** Returned character set identifier of
D* Value
D IPRETCSI       25   28I 0 INZ(0)
```

```
D*
D* Reserved
D IPRE1         29   32I 0 INZ(0)
D*
D* Returned property name
D* Address of variable length string
D IPRETNAMCHRP     33   48*  INZ(*NULL)
D* Offset of variable length string
D IPRETNAMCHRO     49   52I 0 INZ(0)
D* Size of buffer
D IPRETNAMVSBS     53   56I 0 INZ(-1)
D* Length of variable length string
D IPRETNAMCHRL     57   60I 0 INZ(0)
D* CCSID of variable length string
D IPRETNAMCHRC     61   64I 0 INZ(-3)
D*
D* Property data type as a string
D IPTYP         65   72  INZ
```

MQMD (Message descriptor) on IBM i: 

Overview

Purpose: The MQMD structure contains the control information that accompanies the application data when a message travels between the sending and receiving applications. The structure is an input/output parameter on the MQGET, MQPUT, and MQPUT1 calls.

Version: The current version of MQMD is MDVER2. Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions that follow.

The COPY file provided contains the most recent version of MQMD that is supported by the environment, but with the initial value of the *MDVER* field set to MDVER1. To use fields that are not present in the version-1 structure, the application must set the *MDVER* field to the version number of the version required.

A declaration for the version-1 structure is available with the name MQMD1.

Character set and encoding: Data in MQMD must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT. However, if the application is running as an IBM MQ MQI client, the structure must be in the character set and encoding of the client.

If the sending and receiving queue managers use different character sets or encodings, the data in MQMD is converted automatically. It is not necessary for the application to convert the MQMD.

Using different versions of MQMD

A version-2 MQMD is generally equivalent to using a version-1 MQMD and prefixing the message data with an MQMDE structure. However, if all of the fields in the MQMDE structure have their default values, the MQMDE can be omitted. A version-1 MQMD plus MQMDE are used as described later in this section.

- On the MQPUT and MQPUT1 calls, if the application provides a version-1 MQMD, the application can optionally prefix the message data with an MQMDE, setting the *MDFMT* field in MQMD to FMMDE to indicate that an MQMDE is present. If the application does not provide an MQMDE, the queue manager assumes default values for the fields in the MQMDE.

  Note: Several of the fields that exist in the version-2 MQMD but not the version-1 MQMD are input/output fields on the MQPUT and MQPUT1 calls. However, the queue manager does not return any values in the equivalent fields in the MQMDE on output from the MQPUT and MQPUT1 calls; if the application requires those output values, it must use a version-2 MQMD.
- On the MQGET call, if the application provides a version-1 MQMD, the queue manager prefixes the message returned with an MQMDE, but only if one or more of the fields in the MQMDE has a non-default value. The *MDFMT* field in MQMD will have the value FMMDE to indicate that an MQMDE is present.

The default values that the queue manager used for the fields in the MQMDE are the same as the initial values of those fields, shown in Table 341 on page 3161.

When a message is on a transmission queue, some of the fields in MQMD are set to particular values; see "MQXQH (Transmission-queue header) on IBM i" on page 3258 for details.

Message context

Certain fields in MQMD contain the message context. Typically:

- **Identity** context relates to the application that *originally* put the message
- **Origin** context relates to the application that *most recently* put the message
- **User** context relates to the application that *originally* put the message.

These two applications can be the same application, but they can also be different applications (for example, when a message is forwarded from one application to another).

Although identity and origin context usually have the meanings described previously, the content of both types of context fields in MQMD actually depends on the PM* options that are specified when the message is put. As a result, identity context does not necessarily relate to the application that originally put the message, and origin context does not necessarily relate to the application that most recently put the message - it depends on the design of the application suite.

There is one class of application that never alters message context, namely the message channel agent (MCA). MCAs that receive messages from remote queue managers use the context option PMSETA on the MQPUT or MQPUT1 call. This allows the receiving MCA to preserve exactly the message context that travelled with the message from the sending MCA. However, the result is that the origin context does not relate to the application that most recently put the message (the receiving MCA), but instead relates to an earlier application that put the message (possibly the originating application itself).

For more information see Message context.

Message expiry

Messages that have expired on a loaded queue (a queue that has been opened) are automatically removed from the queue within a reasonable period of time after their expiry. Some other new features of this release of IBM MQ can lead to loaded queues being scanned less frequently than in the previous product version, however expired messages on loaded queues are always removed within a reasonable period of their expiry.

Fields

The MQMD structure contains the following fields; the fields are described in alphabetical order:

MDACC (32-byte bit string)

Accounting token.

This is part of the **identity context** of the message. For more information about message context, see Message context and Controlling context information.

MDACC allows an application to cause work done as a result of the message to be appropriately charged. The queue manager treats this information as a string of bits and does not check its content.

When the queue manager generates this information, it is set as follows:

- The first byte of the field is set to the length of the accounting information present in the bytes that follow; this length is in the range zero through 30, and is stored in the first byte as a binary integer.
- The second and subsequent bytes (as specified by the length field) are set to the accounting information appropriate to the environment.
  - On z/OS the accounting information is set to:
    - For z/OS batch, the accounting information from the JES JOB card or from a JES ACCT statement in the EXEC card (comma separators are changed to X'FF'). This information is truncated, if necessary, to 31 bytes.
    - For TSO, the user's account number.
    - For CICS, the LU 6.2 unit of work identifier (UEPUOWDS) (26 bytes).
    - For IMS, the 8-character PSB name concatenated with the 16-character IMS recovery token.
  - On IBM i, the accounting information is set to the accounting code for the job.
  - On HP Integrity NonStop Server, and UNIX, the accounting information is set to the numeric user identifier, in ASCII characters.
  - On Windows, the accounting information is set to a Windows NT security identifier (SID) in a compressed format. The SID uniquely identifies the user identifier stored in the *MDUID* field. When the SID is stored in the *MDACC* field, the 6-byte Identifier Authority (located in the third and subsequent bytes of the SID) is omitted. For example, if the Windows NT SID is 28 bytes long, 22 bytes of SID information are stored in the *MDACC* field.
- The last byte is set to the accounting-token type, one of the following values:

  ATTCIC
  CICS LUOW identifier.

  ATTDOS
  PC DOS default accounting token.

  ATTWNT
  Windows security identifier.

  ATT400
  IBM i accounting token.

  ATTUNX
  UNIX numeric identifier.

  ATTUSR
  User-defined accounting token.

  ATTUNK
  Unknown accounting-token type.

The accounting-token type is set to an explicit value only in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ MQI clients connected to these systems. In other environments, the accounting-token type is set to the value ATTUNK. In these environments the *MDPAT* field can be used to deduce the type of accounting token received.

- All other bytes are set to binary zero.

For the MQPUT and MQPUT1 calls, this is an input/output field if PMSETI or PMSETA is specified in the **PMO** parameter. If neither PMSETI nor PMSETA is specified, this field is ignored on input and is an output-only field. For more information on message context, see Message context and Controlling context information.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *MDACC* that was transmitted with the message if it was put to a queue. This will be the value of *MDACC* that is kept with the message if it is retained (see description of PMRET in "MQPMO (Put-message options) on IBM i" on page 3185 for more details about retained publications) but is not used as the *MDACC* when the message is sent as a publication to subscribers since they provide a value to override *MDACC* in all publications sent to them. If the message has no context, the field is entirely binary zero.

This is an output field for the MQGET call.

This field is not subject to any translation based on the character set of the queue manager-the field is treated as a string of bits, and not as a string of characters.

The queue manager does nothing with the information in this field. The application must interpret the information if it wants to use the information for accounting purposes.

The following special value may be used for the *MDACC* field:

ACNONE
No accounting token is specified.

The value is binary zero for the length of the field.

The length of this field is given by LNACCT. The initial value of this field is ACNONE.

MDAID (32-byte character string)

Application data relating to identity.

This is part of the **identity context** of the message. For more information about message context, see Message context and Controlling context information.

MDAID is information that is defined by the application suite, and can be used to provide additional information about the message or its originator. The queue manager treats this information as character data, but does not define the format of it. When the queue manager generates this information, it is entirely blank.

For the MQPUT and MQPUT1 calls, this is an input/output field if PMSETI or PMSETA is specified in the **PMO** parameter. If a null character is present, the null and any following characters are converted to blanks by the queue manager. If neither PMSETI nor PMSETA is specified, this field is ignored on input and is an output-only field. For more information on message context, see Message context and Controlling context information.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *MDAID* that was transmitted with the message if it was put to a queue. This will be the value of *MDAID* that is kept with the message if it is retained (see description of PMRET for more details about retained publications) but is not used as the *MDAID* when the message is sent as a publication to subscribers since they provide a value to override *MDAID* in all publications sent to them. If the message has no context, the field is entirely blank.

This is an output field for the MQGET call. The length of this field is given by LNAIDD. The initial value of this field is 32 blank characters.

MDAOD (4-byte character string)

Application data relating to origin.

This is part of the **origin context** of the message. For more information about message context, see Message context and Controlling context information.

MDAOD is information that is defined by the application suite that can be used to provide additional information about the origin of the message. For example, it could be set by applications running with suitable user authority to indicate whether the identity data is trusted.

The queue manager treats this information as character data, but does not define the format of it. When the queue manager generates this information, it is entirely blank.

For the MQPUT and MQPUT1 calls, this is an input/output field if PMSETA is specified in the **PMO** parameter. Any information following a null character within the field is discarded. The null character and any following characters are converted to blanks by the queue manager. If PMSETA is not specified, this field is ignored on input and is an output-only field.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *MDAOD* that was transmitted with the message if it was put to a queue. This will be the value of *MDAOD* that is kept with the message if it is retained (see description of PMRET for more details about retained publications) but is not used as the *MDAOD* when the message is sent as a publication to subscribers since they provide a value to override *MDAOD* in all publications sent to them. If the message has no context, the field is entirely blank.

This is an output field for the MQGET call. The length of this field is given by LNAORD. The initial value of this field is 4 blank characters.

MDBOC (10-digit signed integer)

Backout counter.

This is a count of the number of times the message has been previously returned by the MQGET call as part of a unit of work, and subsequently backed out. It is provided as an aid to the application in detecting processing errors that are based on message content. The count excludes MQGET calls that specified any of the GMBRW* options.

The accuracy of this count is affected by the **HardenGetBackout** queue attribute; see "Attributes for queues" on page 3385.

This is an output field for the MQGET call. It is ignored for the MQPUT and MQPUT1 calls. The initial value of this field is 0.

MDCID (24-byte bit string)

Correlation identifier.

This is a byte string that the application can use to relate one message to another, or to relate the message to other work that the application is performing. The correlation identifier is a permanent property of the message, and persists across restarts of the queue manager. Because the correlation identifier is a byte string and not a character string, the correlation identifier is not converted between character sets when the message flows from one queue manager to another.

For the MQPUT and MQPUT1 calls, the application can specify any value. The queue manager transmits this value with the message and delivers it to the application that issues the get request for the message.

If the application specifies PMNCID, the queue manager generates a unique correlation identifier which is sent with the message, and also returned to the sending application on output from the MQPUT or MQPUT1 call.

This generated correlation identifier is kept with the message if it is retained and is used as the correlation identifier when the message is sent as a publication to subscribers who specify CINONE in the *SDCID* field in the MQSD passed on the MQSUB call.

See "MQPMO (Put-message options) on IBM i" on page 3185 for more details about retained publications

When the queue manager or a message channel agent generates a report message, it sets the *MDCID* field in the way specified by the *MDREP* field of the original message, either ROCMTC or ROPCI. Applications which generate report messages should also do this.

For the MQGET call, *MDCID* is one of the five fields that can be used to select a particular message to be retrieved from the queue. See the description of the *MDMID* field for details of how to specify values for this field.

Specifying CINONE as the correlation identifier has the same effect as not specifying MOCORI, that is, any correlation identifier will match.

If the GMMUC option is specified in the **GMO** parameter on the MQGET call, this field is ignored.

On return from an MQGET call, the *MDCID* field is set to the correlation identifier of the message returned (if any).

The following special values may be used:

CINONE
No correlation identifier is specified.

The value is binary zero for the length of the field.

CINEWS
Message is the start of a new session.

This value is recognized by the CICS bridge as indicating the start of a new session, that is, the start of a new sequence of messages.

For the MQGET call, this is an input/output field. For the MQPUT and MQPUT1 calls, this is an input field if PMNCID is not specified, and an output field if PMNCID is specified. The length of this field is given by LNCID. The initial value of this field is CINONE.

MDCSI (10-digit signed integer)

This specifies the character set identifier of character data in the message.

Note: Character data in MQMD and the other IBM MQ data structures that are parameters on calls must be in the character set of the queue manager. This is defined by the queue manager's **CodedCharSetId** attribute; see "Attributes for the queue manager on IBM i" on page 3420 for details of this attribute.

The following special values can be used:

CSQM
Queue manager's character set identifier.

Character data in the message is in the queue manager's character set.

On the MQPUT and MQPUT1 calls, the queue manager changes this value in the MQMD sent with the message to the true character-set identifier of the queue manager. As a result, the value CSQM is never returned by the MQGET call.

CSINHT
Inherit character-set identifier of this structure.

Character data in the message is in the same character set as this structure; this is the queue manager's character set. (For MQMD only, CSINHT has the same meaning as CSQM).

The queue manager changes this value in the MQMD sent with the message to the actual character-set identifier of MQMD. Provided no error occurs, the value CSINHT is not returned by the MQGET call.

CSINHT cannot be used if the value of the *MDPAT* field in MQMD is ATBRKR.

CSEMBD
Embedded character set identifier.

Character data in the message is in a character set with the identifier that is contained within the message data itself. There can be any number of character-set identifiers embedded within the message data, applying to different parts of the data. This value must be used for PCF messages that contain data in a mixture of character sets. PCF messages have a format name of FMPCF.

Specify this value only on the MQPUT and MQPUT1 calls. If it is specified on the MQGET call, it prevents conversion of the message.

On the MQPUT and MQPUT1 calls, the queue manager changes the values CSQM and CSINHT in the MQMD sent with the message as described previously, but does not change the MQMD specified on the MQPUT or MQPUT1 call. No other check is carried out on the value specified.

Applications that retrieve messages should compare this field against the value the application is expecting; if the values differ, the application may need to convert character data in the message.

If the GMCONV option is specified on the MQGET call, this field is an input/output field. The value specified by the application is the coded character-set identifier to which the message data should be converted if necessary. If conversion is successful or unnecessary, the value is unchanged (except that the value CSQM or CSINHT is converted to the actual value). If conversion is unsuccessful, the value after the MQGET call represents the coded character-set identifier of the unconverted message that is returned to the application.

Otherwise, this is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is CSQM.

MDENC (10-digit signed integer)

Numeric encoding of message data.

This specifies the numeric encoding of numeric data in the message; it does not apply to numeric data in the MQMD structure itself. The numeric encoding defines the representation used for binary integers, packed-decimal integers, and floating-point numbers.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that the field is valid. The following special value is defined:

ENNAT
Native machine encoding.

The encoding is the default for the programming language and machine on which the application is running.

Note: The value of this constant depends on the programming language and environment. For this reason, applications must be compiled using the header, macro, COPY, or INCLUDE files appropriate to the environment in which the application will run.

Applications that put messages should normally specify ENNAT. Applications that retrieve messages should compare this field against the value ENNAT; if the values differ, the application may need to convert numeric data in the message. The GMCONV option can be used to request the queue manager to convert the message as part of the processing of the MQGET call.

If the GMCONV option is specified on the MQGET call, this field is an input/output field. The value specified by the application is the encoding to which the message data should be converted

if necessary. If conversion is successful or unnecessary, the value is unchanged. If conversion is unsuccessful, the value after the MQGET call represents the encoding of the unconverted message that is returned to the application.

In other cases, this is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is ENNAT.

MDEXP (10-digit signed integer)

Message lifetime.

This is a period of time expressed in tenths of a second, set by the application that puts the message. The message becomes eligible to be discarded if it has not been removed from the destination queue before this period of time elapses.

The value is decremented to reflect the time the message spends on the destination queue, and also on any intermediate transmission queues if the put is to a remote queue. It may also be decremented by message channel agents to reflect transmission times, if these are significant. Likewise, an application forwarding this message to another queue might decrement the value if necessary, if it has retained the message for a significant time. However, the expiration time is treated as approximate, and the value need not be decremented to reflect small time intervals.

When the message is retrieved by an application using the MQGET call, the *MDEXP* field represents the amount of the original expiry time that still remains.

After a message's expiry time has elapsed, it becomes eligible to be discarded by the queue manager. In the current implementations, the message is discarded when a browse or nonbrowse MQGET call occurs that would have returned the message had it not already expired. For example, a nonbrowse MQGET call with the *GMMO* field in MQGMO set to MONONE reading from a FIFO ordered queue will cause all the expired messages to be discarded up to the first unexpired message. With a priority ordered queue, the same call will discard expired messages of higher priority and messages of an equal priority that arrived on the queue before the first unexpired message.

A message that has expired is never returned to an application (either by a browse or a non-browse MQGET call), so the value in the *MDEXP* field of the message descriptor after a successful MQGET call is either greater than zero, or the special value EIULIM.

If a message is put on a remote queue, the message may expire (and be discarded) while it is on an intermediate transmission queue, before the message reaches the destination queue.

A report is generated when an expired message is discarded, if the message specified one of the ROEXP* report options. If none of these options is specified, no such report is generated; the message is assumed to be no longer relevant after this time period (perhaps because a later message has superseded it).

Any other program that discards messages based on expiry time must also send an appropriate report message if one was requested.

Note:

1. If a message is put with an *MDEXP* time of zero, the MQPUT or MQPUT1 call fails with reason code RC2013; no report message is generated in this case.
2. Since a message with an expiry time that has elapsed may not actually be discarded until later, there may be messages on a queue that have passed their expiry time, and which are not therefore eligible for retrieval. These messages nevertheless count towards the number of messages on the queue for all purposes, including depth triggering.
3. An expiration report is generated, if requested, when the message is actually discarded, not when it becomes eligible for discarding.

4. Discarding of an expired message, and the generation of an expiration report if requested, are never part of the application's unit of work, even if the message was scheduled for discarding as a result of an MQGET call operating within a unit of work.
5. If a nearly-expired message is retrieved by an MQGET call within a unit of work, and the unit of work is subsequently backed out, the message may become eligible to be discarded before it can be retrieved again.
6. If a nearly-expired message is locked by an MQGET call with GMLK, the message may become eligible to be discarded before it can be retrieved by an MQGET call with GMMUC; reason code RC2034 is returned on this subsequent MQGET call if that happens.
7. When a request message with an expiry time greater than zero is retrieved, the application can take one of the following actions when it sends the reply message:
   - Copy the remaining expiry time from the request message to the reply message.
   - Set the expiry time in the reply message to an explicit value greater than zero.
   - Set the expiry time in the reply message to EIULIM.

   The action to take depends on the design of the application suite. However, the default action for putting messages to a dead-letter (undelivered-message) queue should be to preserve the remaining expiry time of the message, and to continue to decrement it.
8. Trigger messages are always generated with EIULIM.
9. A message (normally on a transmission queue) which has a *MDFMT* name of FMXQH has a second message descriptor within the MQXQH. It therefore has two *MDEXP* fields associated with it. The following additional points should be noted in this case:
   - When an application puts a message on a remote queue, the queue manager places the message initially on a local transmission queue, and prefixes the application message data with an MQXQH structure. The queue manager sets the values of the two *MDEXP* fields to be the same as that specified by the application.

     If an application puts a message directly on a local transmission queue, the message data must already begin with an MQXQH structure, and the format name must be FMXQH (but the queue manager does not enforce this). In this case the application need not set the values of these two *MDEXP* fields to be the same. (The queue manager does not check that the *MDEXP* field within the MQXQH contains a valid value, or even that the message data is long enough to include it.)
   - When a message with a *MDFMT* name of FMXQH is retrieved from a queue (whether this is a normal or a transmission queue), the queue manager decrements *both* these *MDEXP* fields with the time spent waiting on the queue. No error is raised if the message data is not long enough to include the *MDEXP* field in the MQXQH.
   - The queue manager uses the *MDEXP* field in the separate message descriptor (that is, not the one in the message descriptor embedded within the MQXQH structure) to test whether the message is eligible for discarding.
   - If the initial values of the two *MDEXP* fields were different, it is therefore possible for the *MDEXP* time in the separate message descriptor when the message is retrieved to be greater than zero (so the message is not eligible for discarding), while the time according to the *MDEXP* field in the MQXQH has elapsed. In this case the *MDEXP* field in the MQXQH is set to zero.

The following special value is recognized:

EIULIM
: Unlimited lifetime.

  The message has an unlimited expiration time.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is EIULIM.

MDFB (10-digit signed integer)

Feedback or reason code.

This is used with a message of type MTRPRT to indicate the nature of the report, and is only meaningful with that type of message. The field can contain one of the FB* values, or one of the RC* values. Feedback codes are grouped as follows:

FBNONE
No feedback provided.

FBSFST
Lowest value for system-generated feedback.

FBSLST
Highest value for system-generated feedback.

The range of system-generated feedback codes FBSFST through FBSLST includes the general feedback codes listed later in this section(FB*), and also the reason codes (RC*) that can occur when the message cannot be put on the destination queue.

FBAFST
Lowest value for application-generated feedback.

FBALST
Highest value for application-generated feedback.

Applications that generate report messages should not use feedback codes in the system range (other than FBQUIT), unless they want to simulate report messages generated by the queue manager or message channel agent.

On the MQPUT or MQPUT1 calls, the value specified must either be FBNONE, or be within the system range or application range. This is checked whatever the value of *MDMT*.

General feedback codes:

FBCOA
Confirmation of arrival on the destination queue (see ROCOA).

FBCOD
Confirmation of delivery to the receiving application (see ROCOD).

FBEXP
Message expired.

Message was discarded because it had not been removed from the destination queue before its expiry time had elapsed.

FBPAN
Positive action notification (see ROPAN).

FBNAN
Negative action notification (see RONAN).

FBQUIT
Application should end.

This can be used by a workload scheduling program to control the number of instances of an application program that are running. Sending an MTRPRT message with this feedback code to an instance of the application program indicates to that instance that it should stop processing. However, adherence to this convention is a matter for the application; it is not enforced by the queue manager.

IMS-bridge feedback codes: When the IMS bridge receives a nonzero IMS-OTMA sense code, the IMS bridge converts the sense code from hexadecimal to decimal, adds the value FBIERR (300),

and places the result in the *MDFB* field of the reply message. This results in the feedback code having a value in the range FBIFST (301) through FBILST (399) when an IMS-OTMA error has occurred.

The following feedback codes can be generated by the IMS bridge:

FBDLZ
Data length zero.

A segment length was zero in the application data of the message.

FBDLN
Data length negative.

A segment length was negative in the application data of the message.

FBDLTB
Data length too big.

A segment length was too big in the application data of the message.

FBBUFO
Buffer overflow.

The value of one of the length fields would cause the data to overflow the message buffer.

FBLOB1
Length in error by one.

The value of one of the length fields was one byte too short.

FBIIH MQIIH structure not valid or missing.

The *MDFMT* field in MQMD specifies FMIMS, but the message does not begin with a valid MQIIH structure.

FBNAFI
User ID not authorized for use in IMS.

The user ID contained in the message descriptor MQMD, or the password contained in the *IIAUT* field in the MQIIH structure, failed the validation performed by the IMS bridge. As a result the message was not passed to IMS.

FBIERR
Unexpected error returned by IMS.

An unexpected error was returned by IMS. Consult the IBM MQ error log on the system on which the IMS bridge resides for more information about the error.

FBIFST
Lowest value for IMS-generated feedback.

IMS-generated feedback codes occupy the range FBIFST (300) through FBILST (399). The IMS-OTMA sense code itself is *MDFB* minus FBIERR.

FBILST
Highest value for IMS-generated feedback.

CICS-bridge feedback codes: The following feedback codes can be generated by the CICS bridge:

FBCAAB
Application abended.

The application program specified in the message abended. This feedback code occurs only in the *DLREA* field of the MQDLH structure.

FBCANS
Application cannot be started.

The EXEC CICS LINK for the application program specified in the message failed. This feedback code occurs only in the *DLREA* field of the MQDLH structure.

FBCBRF
CICS bridge terminated abnormally without completing normal error processing.

FBCCSE
Character set identifier not valid.

FBCIHE
CICS information header structure missing or not valid.

FBCCAE
Length of CICS commarea not valid.

FBCCIE
Correlation identifier not valid.

FBCDLQ
Dead-letter queue not available.

The CICS bridge task was unable to copy a reply to this request to the dead-letter queue. The request was backed out.

FBCENE
Encoding not valid.

FBCINE
CICS bridge encountered an unexpected error.

This feedback code occurs only in the *DLREA* field of the MQDLH structure.

FBCNTA
User identifier not authorized or password not valid.

This feedback code occurs only in the *DLREA* field of the MQDLH structure.

FBCUBO
Unit of work backed out.

The unit of work was backed out, for one of the following reasons:

- A failure was detected while processing another request within the same unit of work.
- A CICS abend occurred while the unit of work was in progress.

FBCUWE
Unit-of-work control field *CIUOW* not valid.

MQ reason codes: For exception report messages, *MDFB* contains an MQ reason code. Among possible reason codes are:

RC2051
(2051, X'803') Put calls inhibited for the queue.

RC2053
(2053, X'805') Queue already contains maximum number of messages.

RC2035
(2035, X'7F3') Not authorized for access.

RC2056
(2056, X'808') No space available on disk for queue.

RC2048
(2048, X'800') Queue does not support persistent messages.

RC2031
(2031, X'7EF') Message length greater than maximum for queue manager.

RC2030
(2030, X'7EE') Message length greater than maximum for queue.

This is an output field for the MQGET call, and an input field for MQPUT and MQPUT1 calls. The initial value of this field is FBNONE.

MDFMT (8-byte character string)

Format name of message data.

This is a name that the sender of the message may use to indicate to the receiver the nature of the data in the message. Any characters that are in the queue manager's character set may be specified for the name, but it is recommended that the name be restricted to the following:

- Uppercase A through Z
- Numeric digits 0 through 9

If other characters are used, it may not be possible to translate the name between the character sets of the sending and receiving queue managers.

The name should be padded with blanks to the length of the field, or a null character used to terminate the name before the end of the field; the null and any subsequent characters are treated as blanks. Do not specify a name with leading or embedded blanks. For the MQGET call, the queue manager returns the name padded with blanks to the length of the field.

The queue manager does not check that the name complies with the recommendations described previously.

Names beginning "MQ" in upper, lower, and mixed case have meanings that are defined by the queue manager; you should not use names beginning with these letters for your own formats. The queue manager built-in formats are:

FMNONE
No format name.

The nature of the data is undefined. This means that the data cannot be converted when the message is retrieved from a queue using the GMCONV option.

If GMCONV is specified on the MQGET call, and the character set or encoding of data in the message differs from that specified in the **MSGDSC** parameter, the message is returned with the following completion and reason codes (assuming no other errors):

- Completion code CCWARN and reason code RC2110 if the FMNONE data is at the beginning of the message.
- Completion code CCOK and reason code RCNONE if the FMNONE data is at the end of the message (that is, preceded by one or more MQ header structures). The MQ header structures are converted to the requested character set and encoding in this case.

FMADMN
Command server request/reply message.

The message is a command-server request or reply message in programmable command format (PCF). Messages of this format can be converted if the GMCONV option is specified on the MQGET call. For more information about using programmable command format messages, see Using Programmable Command Formats.

FMCICS
CICS information header.

The message data begins with the CICS information header MQCIH, which is followed by the application data. The format name of the application data is given by the *CIFMT* field in the MQCIH structure.

FMCMD1
Type 1 command reply message.

The message is an MQSC command-server reply message containing the object count, completion code, and reason code. Messages of this format can be converted if the GMCONV option is specified on the MQGET call.

FMCMD2
Type 2 command reply message.

The message is an MQSC command-server reply message containing information about the object(s) requested. Messages of this format can be converted if the GMCONV option is specified on the MQGET call.

FMDLH
Dead-letter header.

The message data begins with the dead-letter header MQDLH. The data from the original message immediately follows the MQDLH structure. The format name of the original message data is given by the *DLFMT* field in the MQDLH structure; see "MQDLH (Dead-letter header) on IBM i" on page 3072 for details of this structure. Messages of this format can be converted if the GMCONV option is specified on the MQGET call.

COA and COD reports are not generated for messages which have a *MDFMT* of FMDLH.

FMDH
Distribution-list header.

The message data begins with the distribution-list header MQDH; this includes the arrays of MQOR and MQPMR records. The distribution-list header may be followed by additional data. The format of the additional data (if any) is given by the *DHFMT* field in the MQDH structure; see "MQDH (Distribution header) on IBM i" on page 3067 for details of this structure. Messages with format FMDH can be converted if the GMCONV option is specified on the MQGET call.

FMEVNT
Event message.

The message is an MQ event message that reports an event that occurred. Event messages have the same structure as programmable commands; for more information about this structure, see Structures for commands and responses. For information about events, see Event monitoring.

Version-1 event messages can be converted if the GMCONV option is specified on the MQGET call.

FMIMS
IMS information header.

The message data begins with the IMS information header MQIIH, which is followed by the application data. The format name of the application data is given by the *IIFMT* field in the MQIIH structure. Messages of this format can be converted if the GMCONV option is specified on the MQGET call.

FMIMVS
IMS variable string.

The message is an IMS variable string, which is a string of the form `llzzccc`, where:

ll is a 2-byte length field specifying the total length of the IMS variable string item.

This length is equal to the length of ll (2 bytes), plus the length of zz (2 bytes), plus the length of the character string itself. ll is a 2-byte binary integer in the encoding specified by the *MDENC* field.

zz is a 2-byte field containing flags that are significant to IMS. zz is a byte string consisting of two 1-byte bit string fields, and is transmitted without change from sender to receiver (that is, zz is not subject to any conversion).

ccc is a variable-length character string containing ll-4 characters. ccc is in the character set specified by the *MDCSI* field.

Messages of this format can be converted if the GMCONV option is specified on the MQGET call.

FMMDE
Message-descriptor extension.

The message data begins with the message-descriptor extension MQMDE, and is optionally followed by other data (usually the application message data). The format name, character set, and encoding of the data which follows the MQMDE is given by the *MEFMT*, *MECSI*, and *MEENC* fields in the MQMDE. See "MQMDE (Message descriptor extension) on IBM i" on page 3163 for details of this structure. Messages of this format can be converted if the GMCONV option is specified on the MQGET call.

FMPCF
User-defined message in programmable command format (PCF).

The message is a user-defined message that conforms to the structure of a programmable command format (PCF) message. Messages of this format can be converted if the GMCONV option is specified on the MQGET call. See Using Programmable Command Formats for more information about using programmable command format messages.

FMRMH
Reference message header.

The message data begins with the reference message header MQRMH, and is optionally followed by other data. The format name, character set, and encoding of the data is given by the *RMFMT*, *RMCSI*, and *RMENC* fields in the MQRMH. See "MQRMH (Reference message header) on IBM i" on page 3211 for details of this structure. Messages of this format can be converted if the GMCONV option is specified on the MQGET call.

FMRFH
Rules and formatting header.

The message data begins with the rules and formatting header MQRFH, and is optionally followed by other data. The format name, character set, and encoding of the data (if any) is given by the *RFFMT*, *RFCSI*, and *RFENC* fields in the MQRFH. Messages of this format can be converted if the GMCONV option is specified on the MQGET call.

FMRFH2
Rules and formatting header version 2.

The message data begins with the version-2 rules and formatting header MQRFH2, and is optionally followed by other data. The format name, character set, and encoding of the optional data (if any) is given by the *RF2FMT*, *RF2CSI*, and *RF2ENC* fields in the MQRFH2. Messages of this format can be converted if the GMCONV option is specified on the MQGET call.

FMSTR
Message consisting entirely of characters.

The application message data can be either an SBCS string (single-byte character set), or a DBCS string (double-byte character set). Messages of this format can be converted if the GMCONV option is specified on the MQGET call.

FMTM
Trigger message.

The message is a trigger message, described by the MQTM structure; see "MQTM - Trigger message" on page 3248 for details of this structure. Messages of this format can be converted if the GMCONV option is specified on the MQGET call.

FMWIH
Work information header.

The message data begins with the work information header MQWIH, which is followed by the application data. The format name of the application data is given by the *WIFMT* field in the MQWIH structure.

FMXQH
Transmission queue header.

The message data begins with the transmission queue header MQXQH. The data from the original message immediately follows the MQXQH structure. The format name of the original message data is given by the *MDFMT* field in the MQMD structure which is part of the transmission queue header MQXQH. See "MQXQH (Transmission-queue header) on IBM i" on page 3258 for details of this structure.

COA and COD reports are not generated for messages which have a *MDFMT* of FMXQH.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The length of this field is given by LNFMT. The initial value of this field is FMNONE.

MDGID (24-byte bit string)

Group identifier.

This is a byte string that is used to identify the particular message group or logical message to which the physical message belongs. *MDGID* is also used if segmentation is allowed for the message. In all of these cases, *MDGID* has a non-null value, and one or more of the following flags is set in the *MDMFL* field:

- MFMIG
- MFLMIG
- MFSEG
- MFLSEG
- MFSEGA

If none of these flags is set, *MDGID* has the special null value GINONE.

This field need not be set by the application on the MQPUT or MQGET call if:

- On the MQPUT call, PMLOGO is specified.
- On the MQGET call, MOGRPI is not specified.

Consider using these calls for messages that are not report messages. However, if the application requires more control, or the call is MQPUT1, the application must ensure that *MDGID* is set to an appropriate value.

Message groups and segments can be processed correctly only if the group identifier is unique. For this reason, *applications should not generate their own group identifiers* ; instead, applications should do one of the following:

- If PMLOGO is specified, the queue manager automatically generates a unique group identifier for the first message in the group or segment of the logical message, and uses that group

identifier for the remaining messages in the group or segments of the logical message, so the application does not need to take any special action. Consider using this procedure.

- If PMLOGO is not specified, the application should request the queue manager to generate the group identifier, by setting *MDGID* to GINONE on the first MQPUT or MQPUT1 call for a message in the group or segment of the logical message. The group identifier returned by the queue manager on output from that call should then be used for the remaining messages in the group or segments of the logical message. If a message group contains segmented messages, the same group identifier must be used for all segments and messages in the group.

  When PMLOGO is not specified, messages in groups and segments of logical messages can be put in any order (for example, in reverse order), but the group identifier must be allocated by the *first* MQPUT or MQPUT1 call that is issued for any of those messages.

On input to the MQPUT and MQPUT1 calls, the queue manager uses the value detailed in PMOPT. On output from the MQPUT and MQPUT1 calls, the queue manager sets this field to the value that was sent with the message if the object opened is a single queue and not a distribution list, but leaves it unchanged if the object opened is a distribution list. In the latter case, if the application needs to know the group identifiers generated, the application must provide MQPMR records containing the *PRGID* field.

On input to the MQGET call, the queue manager uses the value detailed in Table 1. On output from the MQGET call, the queue manager sets this field to the value for the message retrieved.

The following special value is defined:

GINONE
No group identifier specified.

The value is binary zero for the length of the field. This is the value that is used for messages that are not in groups, not segments of logical messages, and for which segmentation is not allowed.

The length of this field is given by LNGID. The initial value of this field is GINONE. This field is ignored if *MDVER* is less than MDVER2.

MDMFL (10-digit signed integer)

Message flags.

These are flags that specify attributes of the message, or control its processing. The flags are divided into the following categories:

- Segmentation flag
- Status flags

These are described in turn.

Segmentation flags: When a message is too big for a queue, an attempt to put the message on the queue usually fails. Segmentation is a technique whereby the queue manager or application splits the message into smaller pieces called segments, and places each segment on the queue as a separate physical message. The application which retrieves the message can either retrieve the segments one by one, or request the queue manager to reassemble the segments into a single message which is returned by the MQGET call. The latter is achieved by specifying the GMCMPM option on the MQGET call, and supplying a buffer that is big enough to accommodate the complete message. (See "MQGMO (Get-message options) on IBM i" on page 3084 for details of the GMCMPM option.) Segmentation of a message can occur at the sending queue manager, at an intermediate queue manager, or at the destination queue manager.

You can specify one of the following to control the segmentation of a message:

MFSEGI
Segmentation inhibited.

This option prevents the message being broken into segments by the queue manager. If specified for a message that is already a segment, this option prevents the segment being broken into smaller segments.

The value of this flag is binary zero. This is the default.

MFSEGA

Segmentation allowed.

This option allows the message to be broken into segments by the queue manager. If specified for a message that is already a segment, this option allows the segment to be broken into smaller segments. MFSEGA can be set without either MFSEG or MFLSEG being set.

When the queue manager segments a message, the queue manager turns on the MFSEG flag in the copy of the MQMD that is sent with each segment, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call. For the last segment in the logical message, the queue manager also turns on the MFLSEG flag in the MQMD that is sent with the segment.

Note: Care is needed when messages are put with MFSEGA but without PMLOGO. If the message is:

- Not a segment, and
- Not in a group, and
- Not being forwarded,

the application must remember to reset the *MDGID* field to GINONE before *each* MQPUT or MQPUT1 call, in order to cause a unique group identifier to be generated by the queue manager for each message. If this is not done, unrelated messages could inadvertently end up with the same group identifier, which might lead to incorrect processing subsequently. See the descriptions of the *MDGID* field and the PMLOGO option for more information about when the *MDGID* field must be reset.

The queue manager splits messages into segments as necessary in order to ensure that the segments (plus any header data that may be required) fit on the queue. However, there is a lower limit for the size of a segment generated by the queue manager, and only the last segment created from a message can be smaller than this limit. (The lower limit for the size of an application-generated segment is one byte.) Segments generated by the queue manager may be of unequal length. The queue manager processes the message as follows:

- User-defined formats are split on boundaries which are multiples of 16 bytes. This means that the queue manager will not generate segments that are smaller than 16 bytes (other than the last segment).
- Built-in formats other than FMSTR are split at points appropriate to the nature of the data present. However, the queue manager never splits a message in the middle of an MQ header structure. This means that a segment containing a single MQ header structure cannot be split further by the queue manager, and as a result the minimum possible segment size for that message is greater than 16 bytes.

  The second or later segment generated by the queue manager will begin with one of the following:

  – An MQ header structure
  – The start of the application message data
  – Part-way through the application message data
- FMSTR is split without regard for the nature of the data present (SBCS, DBCS, or mixed SBCS/DBCS). When the string is DBCS or mixed SBCS/DBCS, this may result in

segments which cannot be converted from one character set to another. The queue manager never splits FMSTR messages into segments that are smaller than 16 bytes (other than the last segment).

- The *MDFMT*, *MDCSI*, and *MDENC* fields in the MQMD of each segment are set by the queue manager to describe correctly the data present at the *start* of the segment; the format name will be either the name of a built-in format, or the name of a user-defined format.
- The *MDREP* field in the MQMD of segments with *MDOFF* greater than zero are modified as follows:
  - For each report type, if the report option is RO*D, but the segment cannot possibly contain any of the first 100 bytes of user data (that is, the data following any MQ header structures that may be present), the report option is changed to RO*.

The queue manager follows the previously rules, but otherwise splits messages unpredictably; do not make assumptions about where a message is split

For *persistent* messages, the queue manager can perform segmentation only within a unit of work:

- If the MQPUT or MQPUT1 call is operating within a user-defined unit of work, that unit of work is used. If the call fails partway through the segmentation process, the queue manager removes any segments that were placed on the queue as a result of the failing call. However, the failure does not prevent the unit of work being committed successfully.
- If the call is operating outside a user-defined unit of work, and there is no user-defined unit of work in existence, the queue manager creates a unit of work just for the duration of the call. If the call is successful, the queue manager commits the unit of work automatically (the application does not need to do this). If the call fails, the queue manager backs out the unit of work.
- If the call is operating outside a user-defined unit of work, but a user-defined unit of work *does* exist, the queue manager is unable to perform segmentation. If the message does not require segmentation, the call can still succeed. But if the message *does* require segmentation, the call fails with reason code RC2255.

For *nonpersistent* messages, the queue manager does not require a unit of work to be available in order to perform segmentation.

Special consideration must be given to data conversion of messages which may be segmented:

- If data conversion is performed only by the receiving application on the MQGET call, and the application specifies the GMCMPM option, the data-conversion exit will be passed the complete message for the exit to convert, and the fact that the message was segmented will not be apparent to the exit.
- If the receiving application retrieves one segment at a time, the data-conversion exit will be invoked to convert one segment at a time. The exit must therefore be capable of converting the data in a segment independently of the data in any of the other segments.

  If the nature of the data in the message is such that arbitrary segmentation of the data on 16-byte boundaries may result in segments which cannot be converted by the exit, or the format is FMSTR and the character set is DBCS or mixed SBCS/DBCS, the sending application should itself create and put the segments, specifying MFSEGI to suppress further segmentation. In this way, the sending application can ensure that each segment contains sufficient information to allow the data-conversion exit to convert the segment successfully.

- If sender conversion is specified for a sending message channel agent (MCA), the MCA converts only messages which are not segments of logical messages; the MCA never attempts to convert messages which are segments.

This flag is an input flag on the MQPUT and MQPUT1 calls, and an output flag on the MQGET call. On the latter call, the queue manager also echoes the value of the flag to the *GMSEG* field in MQGMO.

The initial value of this flag is MFSEGI.

Status flags: These are flags that indicate whether the physical message belongs to a message group, is a segment of a logical message, both, or neither. One or more of the following can be specified on the MQPUT or MQPUT1 call, or returned by the MQGET call:

MFMIG
Message is a member of a group.

MFLMIG
Message is the last logical message in a group.

If this flag is set, the queue manager turns on MFMIG in the copy of MQMD that is sent with the message, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call.

It is valid for a group to consist of only one logical message. If this is the case, MFLMIG is set, but the *MDSEQ* field has the value one.

MFSEG
Message is a segment of a logical message.

When MFSEG is specified without MFLSEG, the length of the application message data in the segment (*excluding* the lengths of any MQ header structures that may be present) must be at least one. If the length is zero, the MQPUT or MQPUT1 call fails with reason code RC2253.

MFLSEG
Message is the last segment of a logical message.

If this flag is set, the queue manager turns on MFSEG in the copy of MQMD that is sent with the message, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call.

It is valid for a logical message to consist of only one segment. If this is the case, MFLSEG is set, but the *MDOFF* field has the value zero.

When MFLSEG is specified, it is permissible for the length of the application message data in the segment (*excluding* the lengths of any header structures that may be present) to be zero.

The application must ensure that these flags are set correctly when putting messages. If PMLOGO is specified, or was specified on the preceding MQPUT call for the queue handle, the settings of the flags must be consistent with the group and segment information retained by the queue manager for the queue handle. The following conditions apply to *successive* MQPUT calls for the queue handle when PMLOGO is specified:

- If there is no current group or logical message, all of these flags (and combinations of them) are valid.
- Once MFMIG has been specified, it must remain on until MFLMIG is specified. The call fails with reason code RC2241 if this condition is not satisfied.
- Once MFSEG has been specified, it must remain on until MFLSEG is specified. The call fails with reason code RC2242 if this condition is not satisfied.
- Once MFSEG has been specified without MFMIG, MFMIG must remain *off* until after MFLSEG has been specified. The call fails with reason code RC2242 if this condition is not satisfied.

Table 1 shows the valid combinations of the flags, and the values used for various fields.

These flags are input flags on the MQPUT and MQPUT1 calls, and output flags on the MQGET call. On the latter call, the queue manager also echoes the values of the flags to the *GMGST* and *GMSST* fields in MQGMO.

Default flags: The following can be specified to indicate that the message has default attributes:

MFNONE
No message flags (default message attributes).

This inhibits segmentation, and indicates that the message is not in a group and is not a segment of a logical message. MFNONE is defined to aid program documentation. It is not intended that this flag be used with any other, but as its value is zero, such use cannot be detected.

The *MDMFL* field is partitioned into subfields; for details see "Report options and message flags on IBM i" on page 3460.

The initial value of this field is MFNONE. This field is ignored if *MDVER* is less than MDVER2.

MDMID (24-byte bit string)

Message identifier.

This is a byte string that is used to distinguish one message from another. Generally, no two messages should have the same message identifier, although this is not disallowed by the queue manager. The message identifier is a permanent property of the message, and persists across restarts of the queue manager. Because the message identifier is a byte string and not a character string, the message identifier is not converted between character sets when the message flows from one queue manager to another.

For the MQPUT and MQPUT1 calls, if MINONE or PMNMID is specified by the application, the queue manager generates a unique message identifier[3] when the message is put, and places it in the message descriptor sent with the message. The queue manager also returns this message identifier in the message descriptor belonging to the sending application. The application can use this value to record information about particular messages, and to respond to queries from other parts of the application.

If the message is being put to a topic, the queue manager generates unique message identifiers as necessary for each message published. If PMNMID is specified by the application, the queue manager generates a unique message identifier to return on output. If MINONE is specified by the application, the value of the *MDMID* field in the MQMD is unchanged on return from the call.

See the description of PMRET in PMOPT for more details about retained publications.

If the message is being put to a distribution list, the queue manager generates unique message identifiers as necessary, but the value of the *MDMID* field in MQMD is unchanged on return from the call, even if MINONE or PMNMID was specified. If the application needs to know the message identifiers generated by the queue manager, the application must provide MQPMR records containing the *PRMID* field.

3. An *MDMID* generated by the queue manager consists of a 4-byte product identifier (AMQ¬ or CSQ¬ in either ASCII or EBCDIC, where ¬ represents a single blank character), followed by a product-specific implementation of a unique string. In IBM MQ this contains the first 12 characters of the queue manager name, and a value derived from the system clock. All queue managers that can intercommunicate must therefore have names that differ in the first 12 characters, to ensure that message identifiers are unique. The ability to generate a unique string also depends upon the system clock not being changed backward. To eliminate the possibility of a message identifier generated by the queue manager duplicating one generated by the application, the application should avoid generating identifiers with initial characters in the range A through I in ASCII or EBCDIC (X'41' through X'49' and X'C1' through X'C9'). However, the application is not prevented from generating identifiers with initial characters in these ranges.

The sending application can also specify a particular value for the message identifier, other than MINONE; this stops the queue manager generating a unique message identifier. An application that is forwarding a message can use this facility to propagate the message identifier of the original message.

The queue manager does not itself make any use of this field except to:

- Generate a unique value if requested, as described previously
- Deliver the value to the application that issues the get request for the message
- Copy the value to the *MDCID* field of any report message that it generates about this message (depending on the *MDREP* options)

When the queue manager or a message channel agent generates a report message, it sets the *MDMID* field in the way specified by the *MDREP* field of the original message, either RONMI or ROPMI. Applications that generate report messages should also do this.

For the MQGET call, *MDMID* is one of the five fields that can be used to select a particular message to be retrieved from the queue. Normally the MQGET call returns the next message on the queue, but if a particular message is required, this can be obtained by specifying one or more of the five selection criteria, in any combination; these fields are:

- *MDMID*
- *MDCID*
- *MDGID*
- *MDSEQ*
- *MDOFF*

The application sets one or more of these field to the values required, and then sets the corresponding MO* match options in the *GMMO* field in MQGMO to indicate that those fields should be used as selection criteria. Only messages that have the specified values in those fields are candidates for retrieval. The default for the *GMMO* field (if not altered by the application) is to match both the message identifier and the correlation identifier.

Normally, the message returned is the *first* message on the queue that satisfies the selection criteria. But if GMBRWN is specified, the message returned is the *next* message that satisfies the selection criteria; the scan for this message starts with the message *following* the current cursor position.

Note: The queue is scanned sequentially for a message that satisfies the selection criteria, so retrieval times will be slower than if no selection criteria are specified, especially if many messages have to be scanned before a suitable one is found.

See Table 1 for more information about how selection criteria are used in various situations.

Specifying MINONE as the message identifier has the same effect as not specifying MOMSGI, that is, any message identifier will match.

This field is ignored if the GMMUC option is specified in the **GMO** parameter on the MQGET call.

On return from an MQGET call, the *MDMID* field is set to the message identifier of the message returned (if any).

The following special value may be used:

MINONE
No message identifier is specified.

The value is binary zero for the length of the field.

This is an input/output field for the MQGET, MQPUT, and MQPUT1 calls. The length of this field is given by LNMID. The initial value of this field is MINONE.

MDMT (10-digit signed integer)

Message type.

This indicates the type of the message. Message types are grouped as follows:

MTSFST
Lowest value for system-defined message types.

MTSLST
Highest value for system-defined message types.

The following values are currently defined within the system range:

MTDGRM
Message not requiring a reply.

The message is one that does not require a reply.

MTRQST
Message requiring a reply.

The message is one that requires a reply.

The name of the queue to which the reply should be sent must be specified in the *MDRQ* field. The *MDREP* field indicates how the *MDMID* and *MDCID* of the reply are to be set.

MTRPLY
Reply to an earlier request message.

The message is the reply to an earlier request message (MTRQST). The message should be sent to the queue indicated by the *MDRQ* field of the request message. The *MDREP* field of the request should be used to control how the *MDMID* and *MDCID* of the reply are set.

Note: The queue manager does not enforce the request-reply relationship; this is an application responsibility.

MTRPRT
Report message.

The message is reporting on some expected or unexpected occurrence, usually related to some other message (for example, a request message was received which contained data that was not valid). The message should be sent to the queue indicated by the *MDRQ* field of the message descriptor of the original message. The *MDFB* field should be set to indicate the nature of the report. The *MDREP* field of the original message can be used to control how the *MDMID* and *MDCID* of the report message should be set.

Report messages generated by the queue manager or message channel agent are always sent to the *MDRQ* queue, with the *MDFB* and *MDCID* fields set as described previously.

Other values within the system range may be defined in future versions of the MQI, and are accepted by the MQPUT and MQPUT1 calls without error.

Application-defined values can also be used. They must be within the following range:

MTAFST
Lowest value for application-defined message types.

MTALST
Highest value for application-defined message types.

For the MQPUT and MQPUT1 calls, the *MDMT* value must be within either the system-defined range or the application-defined range; if it is not, the call fails with reason code RC2029.

This is an output field for the MQGET call, and an input field for MQPUT and MQPUT1 calls. The initial value of this field is MTDGRM.

MDOFF (10-digit signed integer)

Offset of data in physical message from start of logical message.

This is the offset in bytes of the data in the physical message from the start of the logical message of which the data forms part. This data is called a *segment*. The offset is in the range 0 through 999 999 999. A physical message which is not a segment of a logical message has an offset of zero.

This field need not be set by the application on the MQPUT or MQGET call if:

- On the MQPUT call, PMLOGO is specified.
- On the MQGET call, MOOFFS is not specified.

These are the recommended ways of using these calls for messages that are not report messages. However, if the application does not comply with these conditions, or the call is MQPUT1, the application must ensure that *MDOFF* is set to an appropriate value.

On input to the MQPUT and MQPUT1 calls, the queue manager uses the value detailed in Table 1. On output from the MQPUT and MQPUT1 calls, the queue manager sets this field to the value that was sent with the message.

For a report message reporting on a segment of a logical message, the *MDOLN* field (provided it is not OLUNDF) is used to update the offset in the segment information retained by the queue manager.

On input to the MQGET call, the queue manager uses the value detailed in Table 1. On output from the MQGET call, the queue manager sets this field to the value for the message retrieved.

The initial value of this field is zero. This field is ignored if *MDVER* is less than MDVER2.

MDOLN (10-digit signed integer)

Length of original message.

This field is of relevance only for report messages that are segments. It specifies the length of the message segment to which the report message relates; it does not specify the length of the logical message of which the segment forms part, nor the length of the data in the report message.

Note: When generating a report message for a message that is a segment, the queue manager and message channel agent copy into the MQMD for the report message the *MDGID*, *MDSEQ*, *MDOFF*, and *MDMFL*, fields from the original message. As a result, the report message is also a segment. Applications that generate report messages are recommended to do the same, and to ensure that the *MDOLN* field is set correctly.

The following special value is defined:

OLUNDF
Original length of message not defined.

MDOLN is an input field on the MQPUT and MQPUT1 calls, but the value provided by the application is accepted only in particular circumstances:

- If the message being put is a segment and is also a report message, the queue manager accepts the value specified. The value must be:
  - Greater than zero if the segment is not the last segment
  - Not less than zero if the segment is the last segment
  - Not less than the length of data present in the message

  If these conditions are not satisfied, the call fails with reason code RC2252.
- If the message being put is a segment but not a report message, the queue manager ignores the field and uses the length of the application message data instead.
- In all other cases, the queue manager ignores the field and uses the value OLUNDF instead.

This is an output field on the MQGET call.

The initial value of this field is OLUNDF. This field is ignored if *MDVER* is less than MDVER2.

MDPAN (28-byte character string)

Name of application that put the message.

This is part of the **origin context** of the message. For more information about message context, see Message context and Controlling context information.

The format of the *MDPAN* depends on the value of *MDPAT*.

When this field is set by the queue manager (that is, for all options except PMSETA), it is set to value which is determined by the environment:

- On z/OS, the queue manager uses:
  - For z/OS batch, the 8-character job name from the JES JOB card
  - For TSO, the 7-character TSO user identifier
  - For CICS, the 8-character applid, followed by the 4-character tranid
  - For IMS, the 8-character IMS system identifier, followed by the 8-character PSB name
  - For XCF, the 8-character XCF group name, followed by the 16-character XCF member name
  - For a message generated by a queue manager, the first 28 characters of the queue manager name
  - For distributed queuing without CICS, the 8-character jobname of the channel initiator followed by the 8-character name of the module putting to the dead-letter queue followed by an 8-character task identifier.
  - For MQSeries Java language bindings processing with IBM MQ for z/OS the 8-character jobname of the address space created for the UNIX System Services environment. Typically, this will be a TSO user identifier with a single numeric character appended.

  The name or names are each padded to the right with blanks, as is any space in the remainder of the field. Where there is more than one name, there is no separator between them.
- On PC DOS, and Windows systems, the queue manager uses:
  - For a CICS application, the CICS transaction name
  - For a non-CICS application, the rightmost 28 characters of the fully-qualified name of the executable
- On IBM i, the queue manager uses the fully-qualified job name.
- On HP Integrity NonStop Server, the queue manager uses: the rightmost 28 characters of the fully-qualified name of the executable, if this is available to the queue manager, and blanks otherwise
- On UNIX, the queue manager uses:
  - For a CICS application, the CICS transaction name
  - For a non-CICS application, the rightmost 14 characters of the fully-qualified name of the executable if this is available to the queue manager, and blanks otherwise (for example, on AIX)
- On VSE/ESA, the queue manager uses the 8-character applid, followed by the 4-character tranid.

For the MQPUT and MQPUT1 calls, this is an input/output field if PMSETA is specified in the **PMO** parameter. Any information following a null character within the field is discarded. The null character and any following characters are converted to blanks by the queue manager. If PMSETA is not specified, this field is ignored on input and is an output-only field.

This is an output field for the MQGET call. The length of this field is given by LNPAN. The initial value of this field is 28 blank characters.

MDPAT (10-digit signed integer)

Type of application that put the message.

This is part of the **origin context** of the message. For more information about message context, see Message context and Controlling context information.

MDPAT may have one of the following standard types. User-defined types can also be used but should be restricted to values in the range ATUFST through ATULST.

ATAIX
AIX application (same value as ATUNIX).

ATBRKR
Broker.

ATCICS
CICS transaction.

ATCICB
CICS bridge.

ATVSE
CICS/VSE transaction.

ATDOS
IBM MQ MQI client application on PC DOS.

ATDQM
Distributed queue manager agent.

ATGUAR
Tandem Guardian application (same value as ATNSK).

ATIMS
IMS application.

ATIMSB
IMS bridge.

ATJAVA
Java.

ATMVS
MVS or TSO application (same value as ATZOS).

ATNOTE
Lotus Notes Agent application.

ATNSK
Tandem NonStop Kernel application.

AT390 OS/390 application (same value as ATZOS).

AT400 IBM i application.

ATQM
Queue manager.

ATUNIX
UNIX application.

ATVOS
Stratus VOS application.

ATWIN
16-bit Windows application.

ATWINT
32-bit Windows application.

ATXCF
XCF.

ATZOS
z/OS application.

ATDEF
Default application type.

This is the default application type for the platform on which the application is running.

Note: The value of this constant is environment-specific.

ATUNK
Unknown application type.

This value can be used to indicate that the application type is unknown, even though other context information is present.

ATUFST
Lowest value for user-defined application type.

ATULST
Highest value for user-defined application type.

The following special value can also occur:

ATNCON
No context information present in message.

This value is set by the queue manager when a message is put with no context (that is, the PMNOC context option is specified).

When a message is retrieved, *MDPAT* can be tested for this value to decide whether the message has context (it is recommended that *MDPAT* is never set to ATNCON, by an application using PMSETA, if any of the other context fields are nonblank).

ATSIB Indicates a message originated in another IBM MQ messaging product and arrived via the SIB (Service Integration Bus) bridge.

When the queue manager generates this information as a result of an application put, the field is set to a value that is determined by the environment. Note that on IBM i, it is set to AT400; the queue manager never uses ATCICS on IBM i.

For the MQPUT and MQPUT1 calls, this is an input/output field if PMSETA is specified in the **PMO** parameter. If PMSETA is not specified, this field is ignored on input and is an output-only field.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *MDPAT* that was transmitted with the message if it was put to a queue. This will be the value of *MDPAT* that is kept with the message if it is retained (see description of PMRET for more details about retained publications) but is not used as the *MDPAT* when the message is sent as a publication to subscribers since they provide a value to override *MDPAT* in all publications sent to them. If the message has no context, the field is set to ATNCON.

This is an output field for the MQGET call. The initial value of this field is ATNCON.

MDPD (8-byte character string)

Date when message was put.

This is part of the **origin context** of the message. For more information about message context, see Message context and Controlling context information.

The format used for the date when this field is generated by the queue manager is:

- YYYYMMDD

where the characters represent:

YYYY year (four numeric digits)

MM month of year (01 through 12)

DD day of month (01 through 31)

Greenwich Mean Time (GMT) is used for the *MDPD* and *MDPT* fields, subject to the system clock being set accurately to GMT.

If the message was put as part of a unit of work, the date is that when the message was put, and not the date when the unit of work was committed.

For the MQPUT and MQPUT1 calls, this is an input/output field if PMSETA is specified in the **PMO** parameter. The contents of the field are not checked by the queue manager, except that any information following a null character within the field is discarded. The null character and any following characters are converted to blanks by the queue manager. If PMSETA is not specified, this field is ignored on input and is an output-only field.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *MDPD* that was transmitted with the message if it was put to a queue. This will be the value of *MDPD* that is kept with the message if it is retained (see description of PMRET for more details about retained publications) but is not used as the *MDPD* when the message is sent as a publication to subscribers since they provide a value to override *MDPD* in all publications sent to them. If the message has no context, the field is entirely blank.

This is an output field for the MQGET call. The length of this field is given by LNPDAT. The initial value of this field is 8 blank characters.

MDPER (10-digit signed integer)

Message persistence.

This indicates whether the message survives system failures and restarts of the queue manager. For the MQPUT and MQPUT1 calls, the value must be one of the following:

PEPER
Message is persistent.

This means that the message survives system failures and restarts of the queue manager. Once the message has been put, and the putter's unit of work committed (if the message is put as part of a unit of work), the message is preserved on auxiliary storage. It remains there until the message is removed from the queue, and the getter's unit of work committed (if the message is retrieved as part of a unit of work).

When a persistent message is sent to a remote queue, a store-and-forward mechanism is used to hold the message at each queue manager along the route to the destination, until the message is known to have arrived at the next queue manager.

Persistent messages cannot be placed on:

- Temporary dynamic queues
- Shared queues where the coupling facility structure level is less than three, or the coupling facility structure is not recoverable.

Persistent messages can be placed on permanent dynamic queues, predefined queues, and shared queues where the coupling facility structure level is 3, and the coupling facility is recoverable.

PENPER
Message is not persistent.

This means that the message does not normally survive system failures or restarts of the queue manager. This applies even if an intact copy of the message is found on auxiliary storage during restart of the queue manager.

In the special case of shared queues, nonpersistent messages *do* survive restarts of queue managers in the queue-sharing group, but do not survive failures of the coupling facility used to store messages on the shared queues.

PEQDEF
Message has default persistence.

- If the queue is a cluster queue, the persistence of the message is taken from the **DefPersistence** attribute defined at the destination queue manager that owns the particular instance of the queue on which the message is placed. Usually, all of the instances of a cluster queue have the same value for the **DefPersistence** attribute, although this is not mandated.

  The value of **DefPersistence** is copied into the *MDPER* field when the message is placed on the destination queue. If **DefPersistence** is changed subsequently, messages that have already been placed on the queue are not affected.
- If the queue is not a cluster queue, the persistence of the message is taken from the **DefPersistence** attribute defined at the local queue manager, even if the destination queue manager is remote.

  If there is more than one definition in the queue-name resolution path, the default persistence is taken from the value of this attribute in the first definition in the path. This could be:
  - An alias queue
  - A local queue
  - A local definition of a remote queue
  - A queue manager alias
  - A transmission queue (for example, the *DefXmitQName* queue)

  The value of **DefPersistence** is copied into the *MDPER* field when the message is put. If **DefPersistence** is changed subsequently, messages that have already been put are not affected.

Both persistent and nonpersistent messages can exist on the same queue.

When replying to a message, applications should normally use for the reply message the persistence of the request message.

For an MQGET call, the value returned is either PEPER or PENPER.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is PEQDEF.

MDPRI (10-digit signed integer)

Message priority.

For the MQPUT and MQPUT1 calls, the value must be greater than or equal to zero; zero is the lowest priority. The following special value can also be used:

PRQDEF
Default priority for queue.

- If the queue is a cluster queue, the priority for the message is taken from the **DefPriority** attribute as defined at the destination queue manager that owns the particular instance of the queue on which the message is placed. Usually, all of the instances of a cluster queue have the same value for the **DefPriority** attribute, although this is not mandated.

The value of **DefPriority** is copied into the *MDPRI* field when the message is placed on the destination queue. If **DefPriority** is changed subsequently, messages that have already been placed on the queue are not affected.

- If the queue is not a cluster queue, the priority for the message is taken from the **DefPriority** attribute as defined at the local queue manager, even if the destination queue manager is remote.

  If there is more than one definition in the queue-name resolution path, the default priority is taken from the value of this attribute in the first definition in the path. This could be:

  – An alias queue
  – A local queue
  – A local definition of a remote queue
  – A queue manager alias
  – A transmission queue (for example, the *DefXmitQName* queue)

  The value of **DefPriority** is copied into the *MDPRI* field when the message is put. If **DefPriority** is changed subsequently, messages that have already been put are not affected.

The value returned by the MQGET call is always greater than or equal to zero; the value PRQDEF is never returned.

If a message is put with a priority greater than the maximum supported by the local queue manager (this maximum is given by the **MaxPriority** queue manager attribute), the message is accepted by the queue manager, but placed on the queue at the queue manager's maximum priority; the MQPUT or MQPUT1 call completes with CCWARN and reason code RC2049. However, the *MDPRI* field retains the value specified by the application which put the message.

When replying to a message, applications should normally use for the reply message the priority of the request message. In other situations, specifying PRQDEF allows priority tuning to be carried out without changing the application.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is PRQDEF.

MDPT (8-byte character string)

Time when message was put.

This is part of the **origin context** of the message. For more information about message context, see Message context and Controlling context information.

The format used for the time when this field is generated by the queue manager is:

- HHMMSSTH

where the characters represent (in order):

HH hours (00 through 23)

MM minutes (00 through 59)

SS seconds (00 through 59; see note)

T tenths of a second (0 through 9)

H hundredths of a second (0 through 9)

Note: If the system clock is synchronized to a very accurate time standard, it is possible on rare occasions for 60 or 61 to be returned for the seconds in *MDPT*. This happens when leap seconds are inserted into the global time standard.

Greenwich Mean Time (GMT) is used for the *MDPD* and *MDPT* fields, subject to the system clock being set accurately to GMT.

If the message was put as part of a unit of work, the time is that when the message was put, and not the time when the unit of work was committed.

For the MQPUT and MQPUT1 calls, this is an input/output field if PMSETA is specified in the **PMO** parameter. The contents of the field are not checked by the queue manager, except that any information following a null character within the field is discarded. The null character and any following characters are converted to blanks by the queue manager. If PMSETA is not specified, this field is ignored on input and is an output-only field.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *MDPT* value that was transmitted with the message if it was put to a queue. This will be the value of *MDPT* that is kept with the message if it is retained (see description of PMRET for more details about retained publications) but is not used as the *MDPT* when the message is sent as a publication to subscribers since they provide a value to override *MDPT* in all publications sent to them. If the message has no context, the field is entirely blank.

This is an output field for the MQGET call. The length of this field is given by LNPTIM. The initial value of this field is 8 blank characters.

MDREP (10-digit signed integer)

Options for report messages.

A report message is a message about another message, used to inform an application about expected or unexpected events that relate to the original message. The *MDREP* field enables the application sending the original message to specify which report messages are required, whether the application message data is to be included in them, and also (for both reports and replies) how the message and correlation identifiers in the report or reply message are to be set. Any or all (or none) of the following types of report message can be requested:

- Exception
- Expiration
- Confirm on arrival (COA)
- Confirm on delivery (COD)
- Positive action notification (PAN)
- Negative action notification (NAN)

If more than one type of report message is required, or other report options are needed, the values can be added together (do not add the same constant more than once).

The application that receives the report message can determine the reason the report was generated by examining the *MDFB* field in the MQMD; see the *MDFB* field for more details.

The use of report options when putting a message to a topic can cause zero, one or many report messages to be generated and sent to the application. This is because the publication message may be sent to zero, one or many subscribing applications.

Exception options: You can specify one of the following options to request an exception report message.

ROACTIVITY

Activity reports required

This report option enables an activity report to be generated, whenever a message with this report option set is processed by supporting applications.

Messages with this report option set must be accepted by any queue manager, even if they do not 'understand' the option. This allows the report option to be set on any user

message, even if they are processed by previous queue managers. To achieve this, the report option is placed in the ROAUM subfield.

If a process (either a queue manager or a user process) performs an Activity on a message with ROACT set, it can choose to generate and put an activity report.

The activity report option allows the route of any message to be traced throughout a queue manager network. The report option can be specified on any current user message and instantly they can begin to calculate the route of the message through the network. If the application generating the message cannot enable activity report generation, it can be enabled by using an API crossing exit supplied by queue manager administrators.

Several conditions are applicable to activity reports:

1. The route will be less detailed if there are fewer queue managers in the network which are able to generate activity reports.
2. The activity reports may not be easily 'orderable' in order to determine the route taken.
3. The activity reports may not be able to find a route to their requested destination.

ROEXC

Exception reports required.

This type of report can be generated by a message channel agent when a message is sent to another queue manager and the message cannot be delivered to the specified destination queue. For example, the destination queue or an intermediate transmission queue might be full, or the message might be too big for the queue.

Generation of the exception report message depends on the persistence of the original message, and the speed of the message channel (normal or fast) through which the original message travels:

- For all persistent messages, and for nonpersistent messages traveling through normal message channels, the exception report is generated only if the action specified by the sending application for the error condition can be completed successfully. The sending application can specify one of the following actions to control the disposition of the original message when the error condition arises:
  - RODLQ (this causes the original message to be placed on the dead-letter queue).
  - RODISC (this causes the original message to be discarded).

  If the action specified by the sending application cannot be completed successfully, the original message is left on the transmission queue, and no exception report message is generated.
- For nonpersistent messages traveling through fast message channels, the original message is removed from the transmission queue and the exception report generated *even if* the specified action for the error condition cannot be completed successfully. For example, if RODLQ is specified, but the original message cannot be placed on the dead-letter queue because (say) that queue is full, the exception report message is generated and the original message discarded.

  See Message persistence for more information about normal and fast message channels.

An exception report is not generated if the application that put the original message can be notified synchronously of the problem by means of the reason code returned by the MQPUT or MQPUT1 call.

Applications can also send exception reports, to indicate that a message that it has received cannot be processed (for example, because it is a debit transaction that would cause the account to exceed its credit limit).

Message data from the original message is not included with the report message.

Do not specify more than one of ROEXC, ROEXCD, and ROEXCF.

ROEXCD
Exception reports with data required.

This is the same as ROEXC, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of ROEXC, ROEXCD, and ROEXCF.

ROEXCF
Exception reports with full data required.

This is the same as ROEXC, except that all of the application message data from the original message is included in the report message.

Do not specify more than one of ROEXC, ROEXCD, and ROEXCF.

Expiration options: You can specify one of the following options to request an expiration report message.

ROEXP
Expiration reports required.

This type of report is generated by the queue manager if the message is discarded before delivery to an application because its expiry time has passed (see the *MDEXP* field). If this option is not set, no report message is generated if a message is discarded for this reason (even if one of the ROEXC* options is specified).

Message data from the original message is not included with the report message.

Do not specify more than one of ROEXP, ROEXPD, and ROEXPF.

ROEXPD
Expiration reports with data required.

This is the same as ROEXP, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of ROEXP, ROEXPD, and ROEXPF.

ROEXPF
Expiration reports with full data required.

This is the same as ROEXP, except that all of the application message data from the original message is included in the report message.

Do not specify more than one of ROEXP, ROEXPD, and ROEXPF.

Confirm-on-arrival options: You can specify one of the following options to request a confirm-on-arrival report message.

ROCOA
Confirm-on-arrival reports required.

This type of report is generated by the queue manager that owns the destination queue, when the message is placed on the destination queue. Message data from the original message is not included with the report message.

If the message is put as part of a unit of work, and the destination queue is a local queue, the COA report message generated by the queue manager becomes available for retrieval only if and when the unit of work is committed.

A COA report is not generated if the *MDFMT* field in the message descriptor is FMXQH or FMDLH. This prevents a COA report being generated if the message is put on a transmission queue, or is undeliverable and put on a dead-letter queue.

Do not specify more than one of ROCOA, ROCOAD, and ROCOAF.

ROCOAD
Confirm-on-arrival reports with data required.

This is the same as ROCOA, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of ROCOA, ROCOAD, and ROCOAF.

ROCOAF
Confirm-on-arrival reports with full data required.

This is the same as ROCOA, except that all of the application message data from the original message is included in the report message.

Do not specify more than one of ROCOA, ROCOAD, and ROCOAF.

Discard and expiry options: You can specify the following option to set the expiry time and discard flag for report messages.

ROPDAE
Set report message expiry time and discard flag.

This option ensures that report messages and reply messages inherit the expiry time and discard flag (whether to discard or not), from their original messages. With this option set, report and reply messages:

1. Inherit the RODISC flag (if it was set).
2. Inherit the remaining expiry time of the message, if the message is not an expiry report. If the message is an expiry report, the expiry time is set to 60 seconds.

With this option set, the following applies:

Note:

1. Report and reply messages are generated with a discard flag and an expiry value, and cannot remain within the system.
2. Trace route messages are prevented from reaching destination queues on non-trace route enabled queue managers.
3. Queues are prevented from being filled with reports that cannot be delivered, if communications links are broken.
4. Command server responses inherit the remaining expiry of the request.

Confirm-on-delivery options: You can specify one of the following options to request a confirm-on-delivery report message.

ROCOD
Confirm-on-delivery reports required.

This type of report is generated by the queue manager when an application retrieves the message from the destination queue in a way that causes the message to be deleted from the queue. Message data from the original message is not included with the report message.

If the message is retrieved as part of a unit of work, the report message is generated within the same unit of work, so that the report is not available until the unit of work is committed. If the unit of work is backed out, the report is not sent.

A COD report is not generated if the *MDFMT* field in the message descriptor is FMDLH. This prevents a COD report being generated if the message is undeliverable and put on a dead-letter queue.

ROCOD is not valid if the destination queue is an XCF queue.

Do not specify more than one of ROCOD, ROCODD, and ROCODF.

ROCODD
Confirm-on-delivery reports with data required.

This is the same as ROCOD, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

If GMATM is specified on the MQGET call for the original message, and the message retrieved is truncated, the amount of application message data placed in the report message is the minimum of:

- The length of the original message
- 100 bytes.

ROCODD is not valid if the destination queue is an XCF queue.

Do not specify more than one of ROCOD, ROCODD, and ROCODF.

ROCODF
Confirm-on-delivery reports with full data required.

This is the same as ROCOD, except that all of the application message data from the original message is included in the report message.

ROCODF is not valid if the destination queue is an XCF queue.

Do not specify more than one of ROCOD, ROCODD, and ROCODF.

Action-notification options: You can specify one or both of the following options to request that the receiving application send a positive-action or negative-action report message.

ROPAN
Positive action notification reports required.

This type of report is generated by the application that retrieves the message and acts upon it. It indicates that the action requested in the message has been performed successfully. The application generating the report determines whether any data is to be included with the report.

Other than conveying this request to the application retrieving the message, the queue manager takes no action based upon this option. It is the responsibility of the retrieving application to generate the report if appropriate.

RONAN
Negative action notification reports required.

This type of report is generated by the application that retrieves the message and acts upon it. It indicates that the action requested in the message has not been performed successfully. The application generating the report determines whether any data is to be included with the report. For example, it may be desirable to include some data indicating why the request could not be performed.

Other than conveying this request to the application retrieving the message, the queue manager takes no action based upon this option. It is the responsibility of the retrieving application to generate the report if appropriate.

Determination of which conditions correspond to a positive action and which correspond to a negative action is the responsibility of the application. However, it is recommended that if the request has been only partially performed, a NAN report rather than a PAN report should be generated if requested. It is also recommended that every possible condition should correspond to either a positive action, or a negative action, but not both.

Message-identifier options: You can specify one of the following options to control how the *MDMID* of the report message (or of the reply message) is to be set.

RONMI
New message identifier.

This is the default action, and indicates that if a report or reply is generated as a result of this message, a new *MDMID* is to be generated for the report or reply message.

ROPMI
Pass message identifier.

If a report or reply is generated as a result of this message, the *MDMID* of this message is to be copied to the *MDMID* of the report or reply message.

The *MsgId* of a publication message will be different for each subscriber that receives a copy of the publication and therefore the *MsgId* copied into the report or reply message will be different for each one.

If this option is not specified, RONMI is assumed.

Correlation-identifier options: You can specify one of the following options to control how the *MDCID* of the report message (or of the reply message) is to be set.

ROCMTC
Copy message identifier to correlation identifier.

This is the default action, and indicates that if a report or reply is generated as a result of this message, the *MDMID* of this message is to be copied to the *MDCID* of the report or reply message.

The *MsgId* of a publication message will be different for each subscriber that receives a copy of the publication and therefore the *MsgId* copied into the *CorrelId* of the report or reply message will be different for each one.

ROPCI
Pass correlation identifier.

If a report or reply is generated as a result of this message, the *MDCID* of this message is to be copied to the *MDCID* of the report or reply message.

The *MDCID* of a publication message will be specific to a subscriber unless it uses the SOSCID option and sets the *SCDIC* field in the MQSD to CINONE. Therefore it is possible that the *MDCID* copied into the *MDCID* of the report or reply message will be different for each one.

If this option is not specified, ROCMTC is assumed.

Servers replying to requests or generating report messages are recommended to check whether the ROPMI or ROPCI options were set in the original message. If they were, the servers should take the action described for those options. If neither is set, the servers should take the corresponding default action.

: You can specify one of the following options to control the disposition of the original message when it cannot be delivered to the destination queue. These options apply only to those situations that would result in an exception report message being generated if one had been requested by the sending application. The application can set the disposition options independently of requesting exception reports.

RODLQ
Place message on dead-letter queue.

This is the default action, and indicates that the message should be placed on the dead-letter queue, if the message cannot be delivered to the destination queue. This happens in the following situations:

- When the application that put the original message cannot be notified synchronously of the problem by means of the reason code returned by the MQPUT or MQPUT1 call. An exception report message is generated, if one was requested by the sender.
- When the application that put the original message was putting to a topic

An exception report message will be generated, if one was requested by the sender.

RODISC
Discard message.

This indicates that the message should be discarded if it cannot be delivered to the destination queue. This happens in the following situations:

- When the application that put the original message cannot be notified synchronously of the problem by means of the reason code returned by the MQPUT or MQPUT1 call. An exception report message is generated, if one was requested by the sender.
- When the application that put the original message was putting to a topic

An exception report message will be generated, if one was requested by the sender.

If it is required to return the original message to the sender, without the original message being placed on the dead-letter queue, the sender should specify RODISC with ROEXCF.

Default option: You can specify the following if no report options are required:

RONONE
No reports required.

This value can be used to indicate that no other options have been specified. RONONE is defined to aid program documentation. It is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

General information:

1. All report types required must be specifically requested by the application sending the original message. For example, if a COA report is requested but an exception report is not, a COA report is generated when the message is placed on the destination queue, but no exception report is generated if the destination queue is full when the message arrives there. If no *MDREP* options are set, no report messages are generated by the queue manager or message channel agent (MCA).

   Some report options can be specified even though the local queue manager does not recognize them; this is useful when the option is to be processed by the *destination* queue manager. See "Report options and message flags on IBM i" on page 3460 for more details.

   If a report message is requested, the name of the queue to which the report should be sent must be specified in the *MDRQ* field. When a report message is received, the nature of the report can be determined by examining the *MDFB* field in the message descriptor.
2. If the queue manager or MCA that generates a report message is unable to put the report message on the reply queue (for example, because the reply queue or transmission queue is full), the report message is placed instead on the dead-letter queue. If that *also* fails, or there is no dead-letter queue, the action taken depends on the type of the report message:
   - If the report message is an exception report, the message which caused the exception report to be generated is left on its transmission queue; this ensures that the message is not lost.

- For all other report types, the report message is discarded and processing continues normally. This is done because either the original message has already been delivered safely (for COA or COD report messages), or is no longer of any interest (for an expiration report message).

Once a report message has been placed successfully on a queue (either the destination queue or an intermediate transmission queue), the message is no longer subject to special processing; it is treated just like any other message.

3. When the report is generated, the *MDRQ* queue is opened and the report message put using the authority of the *MDUID* in the MQMD of the message causing the report, except in the following cases:
   - Exception reports generated by a receiving MCA are put with whatever authority the MCA used when it tried to put the message causing the report. The *CDPA* channel attribute determines the user identifier used.
   - COA reports generated by the queue manager are put with whatever authority was used when the message causing the report was put on the queue manager generating the report. For example, if the message was put by a receiving MCA using the MCA's user identifier, the queue manager puts the COA report using the MCA's user identifier.

   Applications generating reports should normally use the same authority as they would have used to generate a reply; this should normally be the authority of the user identifier in the original message.

   If the report has to travel to a remote destination, senders and receivers can decide whether to accept it, in the same way as they do for other messages.
4. If a report message with data is requested:
   - The report message is always generated with the amount of data requested by the sender of the original message. If the report message is too big for the reply queue, the processing described previously occurs; the report message is never truncated in order to fit on the reply queue.
   - If the *MDFMT* of the original message is FMXQH, the data included in the report does not include the MQXQH. The report data starts with the first byte of the data beyond the MQXQH in the original message. This occurs whether the queue is a transmission queue.
5. If a COA, COD, or expiration report message is received at the reply queue, it is guaranteed that the original message arrived, was delivered, or expired, as appropriate. However, if one or more of these report messages is requested and is not received, the reverse cannot be assumed, since one of the following may have occurred:
   a. The report message is held up because a link is down.
   b. The report message is held up because a blocking condition exists at an intermediate transmission queue or at the reply queue (for example, the queue is full or inhibited for puts).
   c. The report message is on a dead-letter queue.
   d. When the queue manager was attempting to generate the report message, it was unable to put it on the appropriate queue, and was also unable to put it on the dead-letter queue, so the report message could not be generated.
   e. A failure of the queue manager occurred between the action being reported (arrival, delivery or expiry), and generation of the corresponding report message. (This does not happen for COD report messages if the application retrieves the original message within a unit of work, as the COD report message is generated within the same unit of work.)

   Exception report messages may be held up in the same way for reasons 1, 2, and 3 previously. However, when an MCA is unable to generate an exception report message (the report message cannot be put either on the reply queue or the dead-letter queue), the original

message remains on the transmission queue at the sender, and the channel is closed. This occurs irrespective of whether the report message was to be generated at the sending or the receiving end of the channel.

6. If the original message is temporarily blocked (resulting in an exception report message being generated and the original message being put on a dead-letter queue), but the blockage clears and an application then reads the original message from the dead-letter queue and puts it again to its destination, the following may occur:
   - Even though an exception report message has been generated, the original message eventually arrives successfully at its destination.
   - More than one exception report message is generated in respect of a single original message, since the original message may encounter another blockage later.

Report messages when putting to a topic:

1. Reports can be generated when putting a message to a topic. This message will be sent to all subscribers to the topic, which could be zero, one or many. This should be taken into account when choosing to use report options as many report messages could be generated as a result.
2. When putting a message to a topic, there may be many destination queues that are to be given a copy of the message. If some of these destination queues have a problem, such as queue full, then the successful completion of the MQPUT depends on the setting of NPMSGDLV or PMSGDLV (depending on the persistence of the message). If the setting is such that message delivery to the destination queue must be successful (for example, it is a persistent message to a durable subscriber and PMSGDLV is set to ALL or ALLDUR), then success is defined as one of the following criteria being met:
   - Successful put to the subscriber queue
   - Use of RODLQ and a successful put to the Dead-letter queue if the subscriber queue cannot take the message
   - Use of RODISC if the subscriber queue cannot take the message.

Report messages for message segments:

1. Report messages can be requested for messages that have segmentation allowed (see the description of the MFSEGA flag). If the queue manager finds it necessary to segment the message, a report message can be generated for each of the segments that subsequently encounters the relevant condition. Applications should therefore be prepared to receive multiple report messages for each type of report message requested. The *MDGID* field in the report message can be used to correlate the multiple reports with the group identifier of the original message, and the *MDFB* field used to identify the type of each report message.
2. If GMLOGO is used to retrieve report messages for segments, be aware that reports of *different types* may be returned by the successive MQGET calls. For example, if both COA and COD reports are requested for a message that is segmented by the queue manager, the MQGET calls for the report messages may return the COA and COD report messages interleaved in an unpredictable fashion. This can be avoided by using the GMCMPM option (optionally with GMATM). GMCMPM causes the queue manager to reassemble report messages that have the same report type. For example, the first MQGET call might reassemble all of the COA messages relating to the original message, and the second MQGET call might reassemble all of the COD messages. Which is reassembled first depends on which type of report message happens to occur first on the queue.
3. Applications that themselves put segments can specify different report options for each segment. However, the following points should be noted:
   - If the segments are retrieved using the GMCMPM option, only the report options in the *first* segment are honored by the queue manager.
   - If the segments are retrieved one by one, and most of them have one of the ROCOD* options, but at least one segment does not, it will not be possible to use the GMCMPM option to retrieve the report messages with a single MQGET call, or use the GMASGA option to detect when all of the report messages have arrived.

4. In an MQ network, it is possible for the queue managers to have differing capabilities. If a report message for a segment is generated by a queue manager or MCA that does not support segmentation, the queue manager or MCA will not by default include the necessary segment information in the report message, and this may make it difficult to identify the original message that caused the report to be generated. This difficulty can be avoided by requesting data with the report message, that is, by specifying the appropriate RO*D or RO*F options. However, be aware that if RO*D is specified, *less than* 100 bytes of application message data may be returned to the application which retrieves the report message, if the report message is generated by a queue manager or MCA that does not support segmentation.

Contents of the message descriptor for a report message: When the queue manager or message channel agent (MCA) generates a report message, it sets the fields in the message descriptor to the following values, and then puts the message in the normal way.

Field in MQMD	Value used
MDSID	MDSIDV
MDVER	MDVER2
MDREP	RONONE
MDMT	MTRPRT
MDEXP	EIULIM
MDFB	As appropriate for the nature of the report (FBCOA, FBCOD, FBEXP, or an RC* value)
MDENC	Copied from the original message descriptor
MDCSI	Copied from the original message descriptor
MDFMT	Copied from the original message descriptor
MDPRI	Copied from the original message descriptor
MDPER	Copied from the original message descriptor
MDMID	As specified by the report options in the original message descriptor
MDCID	As specified by the report options in the original message descriptor
MDBOC	0
MDRQ	Blanks
MDRM	Name of queue manager
MDUID	As set by the PMPASI option
MDACC	As set by the PMPASI option
MDAID	As set by the PMPASI option
MDPAT	ATQM, or as appropriate for the message channel agent
MDPAN	First 28 bytes of the queue manager name or message channel agent name. For report messages generated by the IMS bridge, this field contains the XCF group name and XCF member name of the IMS system to which the message relates.
MDPD	Date when report message is sent
MDPT	Time when report message is sent
MDAOD	Blanks
MDGID	Copied from the original message descriptor
MDSEQ	Copied from the original message descriptor
MDOFF	Copied from the original message descriptor
MDMFL	Copied from the original message descriptor
MDOLN	Copied from the original message descriptor if not OLUNDF, and set to the length of the original message data otherwise

An application generating a report is recommended to set similar values, except for the following:

- The *MDRM* field can be set to blanks (the queue manager will change this to the name of the local queue manager when the message is put).
- The context fields should be set using the option that would have been used for a reply, normally PMPASI.

Analyzing the report field: The *MDREP* field contains subfields; because of this, applications that need to check whether the sender of the message requested a particular report should use one of the techniques described in "Analyzing the report field on IBM i" on page 3462.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is RONONE.

MDRM (48-byte character string)

Name of reply queue manager.

This is the name of the queue manager to which the reply message or report message should be sent. *MDRQ* is the local name of a queue that is defined on this queue manager.

If the *MDRM* field is blank, the local queue manager looks up the **MDRQ** name in its queue definitions. If a local definition of a remote queue exists with this name, the **MDRM** value in the transmitted message is replaced by the value of the **RemoteQMgrName** attribute from the definition of the remote queue, and this value will be returned in the message descriptor when the receiving application issues an MQGET call for the message. If a local definition of a remote queue does not exist, the *MDRM* that is transmitted with the message is the name of the local queue manager.

If the name is specified, it may contain trailing blanks; the first null character and characters following it are treated as blanks. Otherwise, however, no check is made that the name satisfies the naming rules for queue managers, or that this name is known to the sending queue manager; this is also true for the name transmitted, if the **MDRM** is replaced in the transmitted message.

If a reply-to queue is not required, it is recommended (although this is not checked) that the *MDRM* field should be set to blanks; the field should not be left uninitialized.

For the MQGET call, the queue manager always returns the name padded with blanks to the length of the field.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The length of this field is given by LNQMN. The initial value of this field is 48 blank characters.

MDRQ (48-byte character string)

Name of reply queue.

This is the name of the message queue to which the application that issued the get request for the message should send MTRPLY and MTRPRT messages. The name is the local name of a queue that is defined on the queue manager identified by *MDRM*. This queue should not be a model queue, although the sending queue manager does not verify this when the message is put.

For the MQPUT and MQPUT1 calls, this field must not be blank if the *MDMT* field has the value MTRQST, or if any report messages are requested by the *MDREP* field. However, the value specified (or substituted) is passed on to the application that issues the get request for the message, whatever the message type.

If the *MDRM* field is blank, the local queue manager looks up the MDRQ name in its own queue definitions. If a local definition of a remote queue exists with this name, the *MDRQ* value in the transmitted message is replaced by the value of the **RemoteQName** attribute from the definition of the remote queue, and this value will be returned in the message descriptor when the receiving application issues an MQGET call for the message. If a local definition of a remote queue does not exist, *MDRQ* is unchanged.

If the name is specified, it may contain trailing blanks; the first null character and characters following it are treated as blanks. Otherwise, however, no check is made that the name satisfies the naming rules for queues; this is also true for the name transmitted, if the *MDRQ* is replaced in the transmitted message. The only check made is that a name has been specified, if the circumstances require it.

If a reply-to queue is not required, it is recommended (although this is not checked) that the *MDRQ* field should be set to blanks; the field should not be left uninitialized.

For the MQGET call, the queue manager always returns the name padded with blanks to the length of the field.

If a message that requires a report message cannot be delivered, and the report message also cannot be delivered to the queue specified, both the original message and the report message go to the dead-letter (undelivered-message) queue. See the **DeadLetterQName** attribute described in "Attributes for the queue manager on IBM i" on page 3420.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The length of this field is given by LNQN. The initial value of this field is 48 blank characters.

MDSEQ (10-digit signed integer)

Sequence number of logical message within group.

Sequence numbers start at 1, and increase by 1 for each new logical message in the group, up to a maximum of 999 999 999. A physical message which is not in a group has a sequence number of 1.

This field need not be set by the application on the MQPUT or MQGET call if:

- On the MQPUT call, PMLOGO is specified.
- On the MQGET call, MOSEQN is not specified.

These are the recommended ways of using these calls for messages that are not report messages. However, if the application requires more control, or the call is MQPUT1, the application must ensure that *MDSEQ* is set to an appropriate value.

On input to the MQPUT and MQPUT1 calls, the queue manager uses the value detailed in Table 1. On output from the MQPUT and MQPUT1 calls, the queue manager sets this field to the value that was sent with the message.

On input to the MQGET call, the queue manager uses the value detailed in Table 1. On output from the MQGET call, the queue manager sets this field to the value for the message retrieved.

The initial value of this field is one. This field is ignored if *MDVER* is less than MDVER2.

MDSID (4-byte character string)

Structure identifier.

The value must be:

MDSIDV
Identifier for message descriptor structure.

This is always an input field. The initial value of this field is MDSIDV.

MDUID (12-byte character string)

User identifier.

This is part of the **identity context** of the message. For more information about message context, see Message context and Controlling context information.

MDUID specifies the user identifier of the application that originated the message. The queue manager treats this information as character data, but does not define the format of it.

After a message has been received, *MDUID* can be used in the *ODAU* field of the **OBJDSC** parameter of a subsequent MQOPEN or MQPUT1 call, so that the authorization check is performed for the *MDUID* user instead of the application performing the open.

When the queue manager generates this information for an MQPUT or MQPUT1 call, the queue manager uses a user identifier determined from the environment.

When the user identifier is determined from the environment:

- On z/OS, the queue manager uses:
  - For batch, the user identifier from the JES JOB card or started task
  - For TSO, the log on user identifier
  - For CICS, the user identifier associated with the task
  - For IMS, the user identifier depends on the type of application:
    - For:
      - Nonmessage BMP regions
      - Nonmessage IFP regions
      - Message BMP and message IFP regions that have not issued a successful GU call

      the queue manager uses the user identifier from the region JES JOB card or the TSO user identifier. If these are blank or null, it uses the name of the program specification block (PSB).
    - For:
      - Message BMP and message IFP regions that *have* issued a successful GU call
      - MPP regions

      the queue manager uses one of:
      - The signed-on user identifier associated with the message
      - The logical terminal (LTERM) name
      - The user identifier from the region JES JOB card
      - The TSO user identifier
      - The PSB name
- On IBM i, the queue manager uses the name of the user profile associated with the application job.
- On HP Integrity NonStop Server, the queue manager uses the MQSeries principal that is defined for the Tandem user identifier in the MQSeries principal database.
- On UNIX, the queue manager uses:
  - The application's logon name
  - The effective user identifier of the process if no logon is available
  - The user identifier associated with the transaction, if the application is a CICS transaction
- On VSE/ESA, this is a reserved field.
- On Windows, the queue manager uses the first 12 characters of the logged-on user name.

For the MQPUT and MQPUT1 calls, this is an input/output field if PMSETI or PMSETA is specified in the **PMO** parameter. Any information following a null character within the field is discarded. The null character and any following characters are converted to blanks by the queue manager. If PMSETI or PMSETA is not specified, this field is ignored on input and is an output-only field.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *MDUID* that was transmitted with the message if it was put to a queue. This will be the value of *MDUID* that is kept with the message if it is retained (see description of PMRET for more details about retained publications) but is not used as the *MDUID* when the message is sent as a publication to subscribers since they provide a value to override *MDUID* in all publications sent to them. If the message has no context, the field is entirely blank.

This is an output field for the MQGET call. The length of this field is given by LNUID. The initial value of this field is 12 blank characters.

MDVER (10-digit signed integer)

Structure version number.

The value must be one of the following:

MDVER1
Version-1 message descriptor structure.

MDVER2
Version-2 message descriptor structure.

Note: When a version-2 MQMD is used, the queue manager performs additional checks on any MQ header structures that may be present at the beginning of the application message data; for further details see the usage notes for the MQPUT call.

Fields that exist only in the more-recent version of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MDVERC
Current version of message descriptor structure.

This is always an input field. The initial value of this field is MDVER1.

Initial values

Table 341. Initial values of fields in MQMD

Field name	Name of constant	Value of constant
MDSID	MDSIDV	'MD¬¬'
MDVER	MDVER1	1
MDREP	RONONE	0
MDMT	MTDGRM	8
MDEXP	EIULIM	-1
MDFB	FBNONE	0
MDENC	ENNAT	Depends on environment
MDCSI	CSQM	0
MDFMT	FMNONE	Blanks
MDPRI	PRQDEF	-1
MDPER	PEQDEF	2
MDMID	MINONE	Nulls
MDCID	CINONE	Nulls
MDBOC	None	0
MDRQ	None	Blanks
MDRM	None	Blanks
MDUID	None	Blanks
MDACC	ACNONE	Nulls
MDAID	None	Blanks
MDPAT	ATNCON	0
MDPAN	None	Blanks
MDPD	None	Blanks

Table 341. Initial values of fields in MQMD (continued)

Field name	Name of constant	Value of constant
MDPT	None	Blanks
MDAOD	None	Blanks
MDGID	GINONE	Nulls
MDSEQ	None	1
MDOFF	None	0
MDMFL	MFNONE	0
MDOLN	OLUNDF	-1
Notes: 1. The symbol ¬ represents a single blank character.		

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D*
D* MQMD Structure
D*
D* Structure identifier
D  MDSID                  1      4    INZ('MD  ')
D* Structure version number
D  MDVER                  5      8I 0 INZ(1)
D* Options for report messages
D  MDREP                  9     12I 0 INZ(0)
D* Message type
D  MDMT                  13     16I 0 INZ(8)
D* Message lifetime
D  MDEXP                 17     20I 0 INZ(-1)
D* Feedback or reason code
D  MDFB                  21     24I 0 INZ(0)
D* Numeric encoding of message data
D  MDENC                 25     28I 0 INZ(273)
D* Character set identifier of messagedata
D  MDCSI                 29     32I 0 INZ(0)
D* Format name of message data
D  MDFMT                 33     40    INZ('        ')
D* Message priority
D  MDPRI                 41     44I 0 INZ(-1)
D* Message persistence
D  MDPER                 45     48I 0 INZ(2)
D* Message identifier
D  MDMID                 49     72    INZ(X'00000000000000-
D                                     00000000000000000000-
D                                     000000000000')
D* Correlation identifier
D  MDCID                 73     96    INZ(X'00000000000000-
D                                     00000000000000000000-
D                                     000000000000')
D* Backout counter
D  MDBOC                 97    100I 0 INZ(0)
D* Name of reply queue
D  MDRQ                 101    148    INZ
D* Name of reply queue manager
D  MDRM                 149    196    INZ
D* User identifier
D  MDUID                197    208    INZ
D* Accounting token
D  MDACC                209    240    INZ(X'00000000000000-
D                                     00000000000000000000-
D                                     00000000000000000000-
```

```
D                                 000000')
D* Application data relating toidentity
D  MDAID                241    272    INZ
D* Type of application that put themessage
D  MDPAT                273    276I 0 INZ(0)
D* Name of application that put themessage
D  MDPAN                277    304    INZ
D* Date when message was put
D  MDPD                 305    312    INZ
D* Time when message was put
D  MDPT                 313    320    INZ
D* Application data relating toorigin
D  MDAOD                321    324    INZ
D* Group identifier
D  MDGID                325    348    INZ(X'00000000000000-
D                                     00000000000000000000-
D                                     000000000000')
D* Sequence number of logical messagewithin group
D  MDSEQ                349    352I 0 INZ(1)
D* Offset of data in physical messagefrom start of logical message
D  MDOFF                353    356I 0 INZ(0)
D* Message flags
D  MDMFL                357    360I 0 INZ(0)
D* Length of original message
D  MDOLN                361    364I 0 INZ(-1)
```

MQMDE (Message descriptor extension) on IBM i: IBM i

Overview

Purpose: The MQMDE structure describes the data that sometimes occurs preceding the application message data. The structure contains those MQMD fields that exist in the version-2 MQMD, but not in the version-1 MQMD.

Format name: FMMDE.

Character set and encoding: Data in MQMDE must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT for the C programming language.

The character set and encoding of the MQMDE must be set into the *MDCSI* and *MDENC* fields in:

- The MQMD (if the MQMDE structure is at the start of the message data), or
- The header structure that precedes the MQMDE structure (all other cases).

If the MQMDE is not in the queue manager's character set and encoding, the MQMDE is accepted but not honored, that is, the MQMDE is treated as message data.

Usage: Normal applications should use a version-2 MQMD, in which case they will not encounter an MQMDE structure. However, specialized applications, and applications that continue to use a version-1 MQMD, may encounter an MQMDE in some situations. The MQMDE structure can occur in the following circumstances:

- Specified on the MQPUT and MQPUT1 calls
- Returned by the MQGET call
- In messages on transmission queues
- "MQMDE specified on MQPUT and MQPUT1 calls" on page 3164
- "MQMDE returned by MQGET call" on page 3164
- "MQMDE in messages on transmission queues" on page 3165

- "Fields" on page 3165
- "Initial values" on page 3167
- "RPG declaration" on page 3167

MQMDE specified on MQPUT and MQPUT1 calls

On the MQPUT and MQPUT1 calls, if the application provides a version-1 MQMD, the application can optionally prefix the message data with an MQMDE, setting the *MDFMT* field in MQMD to FMMDE to indicate that an MQMDE is present. If the application does not provide an MQMDE, the queue manager assumes default values for the fields in the MQMDE. The default values that the queue manager uses are the same as the initial values for the structure - see Table 343 on page 3167.

If the application provides a version-2 MQMD *and* prefixes the application message data with an MQMDE, the structures are processed as shown in Table 342.

Table 342. Queue manager action when MQMDE specified on MQPUT or MQPUT1

MQMD version	Values of version-2 fields	Values of corresponding fields in MQMDE	Action taken by queue manager
1	-	Valid	MQMDE is honored
2	Default	Valid	MQMDE is honored
2	Not default	Valid	MQMDE is treated as message data
1 or 2	Any	Not valid	Call fails with an appropriate reason code
1 or 2	Any	MQMDE is in the wrong character set or encoding, or is an unsupported version	MQMDE is treated as message data

There is one special case. If the application uses a version-2 MQMD to put a message that is a segment (that is, the MFSEG or MFLSEG flag is set), and the format name in the MQMD is FMDLH, the queue manager generates an MQMDE structure and inserts it *between* the MQDLH structure and the data that follows it. In the MQMD that the queue manager retains with the message, the version-2 fields are set to their default values.

Several of the fields that exist in the version-2 MQMD but not the version-1 MQMD are input/output fields on MQPUT and MQPUT1. However, the queue manager does not return any values in the equivalent fields in the MQMDE on output from the MQPUT and MQPUT1 calls; if the application requires those output values, it must use a version-2 MQMD.

MQMDE returned by MQGET call

On the MQGET call, if the application provides a version-1 MQMD, the queue manager prefixes the message returned with an MQMDE, but only if one or more of the fields in the MQMDE has a nondefault value. The queue manager sets the *MDFMT* field in MQMD to the value FMMDE to indicate that an MQMDE is present.

If the application provides an MQMDE at the start of the **BUFFER** parameter, the MQMDE is ignored. On return from the MQGET call, it is replaced by the MQMDE for the message (if one is needed), or overwritten by the application message data (if the MQMDE is not needed).

If an MQMDE is returned by the MQGET call, the data in the MQMDE is typically in the queue manager's character set and encoding. However the MQMDE may be in some other character set and encoding if:

- The MQMDE was treated as data on the MQPUT or MQPUT1 call (see Table 342 on page 3164 for the circumstances that can cause this).
- The message was received from a remote queue manager connected by a TCP connection, and the receiving message channel agent (MCA) was not set up correctly (see Security of IBM MQ for IBM i objects for further information).

MQMDE in messages on transmission queues

Messages on transmission queues are prefixed with the MQXQH structure, which contains within it a version-1 MQMD. An MQMDE may also be present, positioned between the MQXQH structure and application message data, but it will typically be present only if one or more of the fields in the MQMDE has a nondefault value.

Other IBM MQ header structures can also occur between the MQXQH structure and the application message data. For example, when the dead-letter header MQDLH is present, and the message is not a segment, the order is:

- MQXQH (containing a version-1 MQMD)
- MQMDE
- MQDLH
- Application message data

Fields

The MQMDE structure contains the following fields; the fields are described in **alphabetical order**:

MECSI (10-digit signed integer)

Character-set identifier of data that follows MQMDE.

This specifies the character set identifier of the data that follows the MQMDE structure; it does not apply to character data in the MQMDE structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that this field is valid. The following special value can be used:

CSINHT

Inherit character-set identifier of this structure.

Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value CSINHT is not returned by the MQGET call.

CSINHT cannot be used if the value of the *MDPAT* field in MQMD is ATBRKR.

The initial value of this field is CSUNDF.

MEENC (10-digit signed integer)

MEENC (10-digit signed integer)

This specifies the numeric encoding of the data that follows the MQMDE structure; it does not apply to numeric data in the MQMDE structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that the field is valid. See the *MDENC* field described in "MQMD (Message descriptor) on IBM i" on page 3118 for more information about data encodings.

The initial value of this field is ENNAT.

MEFLG (10-digit signed integer)

General flags.

The following flag can be specified:

MEFNON
No flags.

The initial value of this field is MEFNON.

MEFMT (8-byte character string)

Format name of data that follows MQMDE.

This specifies the format name of the data that follows the MQMDE structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that this field is valid. See the *MDFMT* field described in "MQMD (Message descriptor) on IBM i" on page 3118 for more information about format names.

The initial value of this field is FMNONE.

MEGID (24-byte bit string)

Group identifier.

See the *MDGID* field described in "MQMD (Message descriptor) on IBM i" on page 3118. The initial value of this field is GINONE.

MELEN (10-digit signed integer)

Length of MQMDE structure.

The following value is defined:

MELEN2
Length of version-2 message descriptor extension structure.

The initial value of this field is MELEN2.

MEMFL (10-digit signed integer)

Message flags.

See the *MDMFL* field described in "MQMD (Message descriptor) on IBM i" on page 3118. The initial value of this field is MFNONE.

MEOFF (10-digit signed integer)

Offset of data in physical message from start of logical message.

See the *MDOFF* field described in "MQMD (Message descriptor) on IBM i" on page 3118. The initial value of this field is 0.

MEOLN (10-digit signed integer)

Length of original message.

See the *MDOLN* field described in "MQMD (Message descriptor) on IBM i" on page 3118. The initial value of this field is OLUNDF.

MESEQ (10-digit signed integer)

Sequence number of logical message within group.

See the *MDSEQ* field described in "MQMD (Message descriptor) on IBM i" on page 3118. The initial value of this field is 1.

MESID (4-byte character string)

Structure identifier.

The value must be:

MESIDV
Identifier for message descriptor extension structure.

The initial value of this field is MESIDV.

MEVER (10-digit signed integer)

Structure version number.

The value must be:

MEVER2
Version-2 message descriptor extension structure.

The following constant specifies the version number of the current version:

MEVERC
Current version of message descriptor extension structure.

The initial value of this field is MEVER2.

Initial values

Table 343. Initial values of fields in MQMDE

Field name	Name of constant	Value of constant
MESID	MESIDV	'MDE¬'
MEVER	MEVER2	2
MELEN	MELEN2	72
MEENC	ENNAT	Depends on environment
MECSI	CSUNDF	0
MEFMT	FMNONE	Blanks
MEFLG	MEFNON	0
MEGID	GINONE	Nulls
MESEQ	None	1
MEOFF	None	0
MEMFL	MFNONE	0
MEOLN	OLUNDF	-1
Notes: 1. The symbol ¬ represents a single blank character.		

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D*
D* MQMDE Structure
D*
D* Structure identifier
D  MESID                  1      4    INZ('MDE ')
D* Structure version number
D  MEVER                  5      8I 0 INZ(2)
D* Length of MQMDE structure
D  MELEN                  9     12I 0 INZ(72)
D* Numeric encoding of data that followsMQMDE
```

```
D  MEENC                 13     16I 0 INZ(273)
D* Character-set identifier of data thatfollows MQMDE
D  MECSI                 17     20I 0 INZ(0)
D* Format name of data that followsMQMDE
D  MEFMT                 21     28    INZ('        ')
D* General flags
D  MEFLG                 29     32I 0 INZ(0)
D* Group identifier
D  MEGID                 33     56    INZ(X'00000000000000-
D                                     00000000000000000000-
D                                     000000000000')
D* Sequence number of logical messagewithin group
D  MESEQ                 57     60I 0 INZ(1)
D* Offset of data in physical messagefrom start of logical message
D  MEOFF                 61     64I 0 INZ(0)
D* Message flags
D  MEMFL                 65     68I 0 INZ(0)
D* Length of original message
D  MEOLN                 69     72I 0 INZ(-1)
```

MQMHBO (Message handle to buffer options) on IBM i: IBM i

Structure defining the message handle to buffer options

Overview

Purpose: The MQMHBO structure allows applications to specify options that control how buffers are produced from message handles. The structure is an input parameter on the MQMHBUF call.

Character set and encoding: Data in MQMHBO must be in the character set of the application and encoding of the application (ENNAT).

- "Fields"
- "Initial values" on page 3169
- "RPG declaration" on page 3169

Fields

The MQMHBO structure contains the following fields; the fields are described in **alphabetical order**:

MBOPT (10-digit signed integer)

Message handle to buffer options structure - MBOPT field.

These options control the action of MQMHBUF.

You must specify the following option:

MBPRRF
When converting properties from a message handle into a buffer, convert them into the MQRFH2 format.

Optionally, you can also specify the following option. To specify more than one option, either add the values together (do not add the same constant more than once), or combine the values using the bitwise OR operation (if the programming language supports bit operations).

MBDLPR
Properties that are added to the buffer are deleted from the message handle. If the call fails no properties are deleted.

This is always an input field. The initial value of this field is MBPRRF.

MBSID (10-digit signed integer)

Message handle to buffer options structure - MBSID field.

This is the structure identifier. The value must be:

MBSIDV
Identifier for message handle to buffer options structure.

This is always an input field. The initial value of this field isMBSIDV.

MBVER (10-digit signed integer)

This is the structure version number. The value must be:

MBVER1
Version number for message handle to buffer options structure.

The following constant specifies the version number of the current version:

MBVERC
Current version of message handle to buffer options structure.

This is always an input field. The initial value of this field is MBVER1.

Initial values

Table 344. Initial values of fields in MQMHBO

Field name	Name of constant	Value of constant
MVSID	MBSIDV	'MHBO'
MBVER	MBVER1	1
MBOPT	MBPRRF	

Notes:

1. The value Null string or blanks denotes a blank character.

RPG declaration

```
D* MQMHBO Structure
D*
D*
D* Structure identifier
D  MBSID                  1      4    INZ('MHBO')
D*
D* Structure version number
D  MBVER                  5      8I 0 INZ(1)
D*
D* Options that control the action of MQMHBUF
D  MBOPT                  9     12I 0 INZ(1)
```

MQOD (Object descriptor) on IBM i: IBM i

The MQOD structure is used to specify an object by name.

Overview

Purpose: The following types of object are valid:

- Queue or distribution list
- Namelist
- Process definition
- Queue manager
- Topic

The structure is an input/output parameter on the MQOPEN and MQPUT1 calls.

Version: The current version of MQOD is ODVER4. Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions that follow.

The COPY file provided contains the most recent version of MQOD that is supported by the environment, but with the initial value of the *ODVER* field set to ODVER1. To use fields that are not present in the version-1 structure, the application must set the *ODVER* field to the version number of the version required.

To open a distribution list, *ODVER* must be ODVER2 or greater.

Character set and encoding: Data in MQOD must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT. However, if the application is running as an IBM MQ client, the structure must be in the character set and encoding of the client.

- "Fields"
- "Initial values" on page 3177
- "RPG declaration" on page 3178

Fields

The MQOD structure contains the following fields; the fields are described in **alphabetical order**:

ODASI (40-byte bit string)

Alternate security identifier.

This is a security identifier that is passed with the *ODAU* to the authorization service to allow appropriate authorization checks to be performed. *ODASI* is used only if:

- OOALTU is specified on the MQOPEN call, or
- PMALTU is specified on the MQPUT1 call,

and the *ODAU* field is not entirely blank up to the first null character or the end of the field.

The *ODASI* field has the following structure:

- The first byte is a binary integer containing the length of the significant data that follows; the value excludes the length byte itself. If no security identifier is present, the length is zero.
- The second byte indicates the type of security identifier that is present; the following values are possible:

  SITWNT
  Windows security identifier.

SITNON
No security identifier.

- The third and subsequent bytes up to the length defined by the first byte contain the security identifier itself.
- Remaining bytes in the field are set to binary zero.

The following special value may be used:

SINONE
No security identifier specified.

The value is binary zero for the length of the field.

This is an input field. The length of this field is given by LNSCID. The initial value of this field is SINONE. This field is ignored if *ODVER* is less than ODVER3.

ODAU (12-byte character string)

Alternate user identifier.

If OOALTU is specified for the MQOPEN call, or PMALTU for the MQPUT1 call, this field contains an alternate user identifier that is to be used to check the authorization for the open, in place of the user identifier that the application is currently running under. Some checks, however, are still carried out with the current user identifier (for example, context checks).

If OOALTU and PMALTU are not specified and this field is entirely blank up to the first null character or the end of the field, the open can succeed only if no user authorization is needed to open this object with the options specified.

If neither OOALTU nor PMALTU is specified, this field is ignored.

This is an input field. The length of this field is given by LNUID. The initial value of this field is 12 blank characters.

ODDN (48-byte character string)

Dynamic queue name.

This is the name of a dynamic queue that is to be created by the MQOPEN call. This is of relevance only when *ODON* specifies the name of a model queue; in all other cases *ODDN* is ignored.

The characters that are valid in the name are the same as those for *ODON*, except that an asterisk is also valid. A name that is blank (or one in which only blanks are shown before the first null character) is not valid if *ODON* is the name of a model queue.

If the last nonblank character in the name is an asterisk (`*`), the queue manager replaces the asterisk with a string of characters that guarantees that the name generated for the queue is unique at the local queue manager. To allow a sufficient number of characters for this, the asterisk is valid only in positions 1 through 33. There must be no characters other than blanks or a null character following the asterisk.

It is valid for the asterisk to occur in the first character position, in which case the name consists solely of the characters generated by the queue manager.

This is an input field. The length of this field is given by LNQN. The initial value of this field is `'AMQ.*'`, padded with blanks.

ODIDC (10-digit signed integer)

Number of queues that failed to open.

This is the number of queues in the distribution list that failed to open successfully. If present, this field is also set when opening a single queue which is not in a distribution list.

Note: If present, this field is set only if the **CMPCOD** parameter on the MQOPEN or MQPUT1 call is CCOK or CCWARN; it is not set if the **CMPCOD** parameter is CCFAIL.

This is an output field. The initial value of this field is 0. This field is ignored if *ODVER* is less than ODVER2.

ODKDC (10-digit signed integer)

Number of local queues opened successfully.

This is the number of queues in the distribution list that resolve to local queues and that were opened successfully. The count does not include queues that resolve to remote queues (even though a local transmission queue is used initially to store the message). If present, this field is also set when opening a single queue which is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is ignored if *ODVER* is less than ODVER2.

ODMN (48-byte character string)

Object queue manager name.

This is the name of the queue manager on which the *ODON* object is defined. The characters that are valid in the name are the same as those for *ODON* (see previously). A name that is entirely blank up to the first null character or the end of the field denotes the queue manager to which the application is connected (the local queue manager).

The following points apply to the types of object indicated:

- If *ODOT* is OTTOP, OTNLST, OTPRO, or OTQM, *ODMN* must be blank or the name of the local queue manager.
- If *ODON* is the name of a model queue, the queue manager creates a dynamic queue with the attributes of the model queue, and returns in the *ODMN* field the name of the queue manager on which the queue is created; this is the name of the local queue manager. A model queue can be specified only on the MQOPEN call; a model queue is not valid on the MQPUT1 call.
- If *ODON* is the name of a cluster queue, and *ODMN* is blank, the actual destination of messages sent using the queue handle returned by the MQOPEN call is chosen by the queue manager (or cluster workload exit, if one is installed) as follows:
  - If OOBNDO is specified, the queue manager selects an instance of the cluster queue during the processing of the MQOPEN call, and all messages put using this queue handle are sent to that instance.
  - If OOBNDN is specified, the queue manager may choose a different instance of the destination queue (residing on a different queue manager in the cluster) for each successive MQPUT call that uses this queue handle.

  If the application needs to send a message to a *specific* instance of a cluster queue (that is, a queue instance that resides on a particular queue manager in the cluster), the application should specify the name of that queue manager in the *ODMN* field. This forces the local queue manager to send the message to the specified destination queue manager.
- If the object being opened is a distribution list (that is, *ODREC* is greater than zero), *ODMN* must be blank or the null string. If this condition is not satisfied, the call fails with reason code RC2153.

This is an input/output field for the MQOPEN call when *ODON* is the name of a model queue, and an input-only field in all other cases. The length of this field is given by LNQMN. The initial value of this field is 48 blank characters.

ODON (48-byte character string)

Object name.

This is the local name of the object as defined on the queue manager identified by *ODMN*. The name can contain the following characters:

- Uppercase alphabetic characters (A - Z)
- Lowercase alphabetic characters (a - z)
- Numeric digits (0 - 9)
- Period (.), forward slash (/), underscore (_), percent (%)

The name must not contain leading or embedded blanks, but may contain trailing blanks. A null character can be used to indicate the end of significant data in the name; the null and any characters following it are treated as blanks. The following restrictions apply in the environments indicated:

- On systems that use EBCDIC Katakana, lowercase characters cannot be used.
- On IBM i, names containing lowercase characters, forward slash, or percent, must be enclosed in quotation marks when specified on commands. These quotation marks must not be specified for names that occur as fields in structures or as parameters on calls.

The following points apply to the types of object indicated:

- If *ODON* is the name of a model queue, the queue manager creates a dynamic queue with the attributes of the model queue, and returns in the *ODON* field the name of the queue created. A model queue can be specified only on the MQOPEN call; a model queue is not valid on the MQPUT1 call.
- If the object being opened is a distribution list (that is, *ODREC* is present and greater than zero), *ODON* must be blank or the null string. If this condition is not satisfied, the call fails with reason code RC2152.
- If *ODOT* is OTQM, special rules apply; in this case the name must be entirely blank up to the first null character or the end of the field.
- If ODON is the name of an alias queue with TARGTYPE(TOPIC), a security check is first made on the named alias queue, as is normal for the use of alias queues. If this security check is successful, this MQOPEN call will continue and behaves like an MQOPEN of an OTTOP, including making a security check against the administrative topic object.

This is an input/output field for the MQOPEN call when *ODON* is the name of a model queue, and an input-only field in all other cases. The length of this field is given by LNQN. The initial value of this field is 48 blank characters.

The full topic name can be built from two different fields: *ODON* and *ODOS*. For details of how these two fields are used, see “Using topic strings” on page 2566.

ODORO (10-digit signed integer)

Offset of first object record from start of MQOD.

This is the offset in bytes of the first MQOR object record from the start of the MQOD structure. The offset can be positive or negative. *ODORO* is used only when a distribution list is being opened. The field is ignored if *ODREC* is zero.

When a distribution list is being opened, an array of one or more MQOR object records must be provided in order to specify the names of the destination queues in the distribution list. This can be done in one of two ways:

- By using the offset field *ODORO*

  In this case, the application should declare its own structure containing an MQOD followed by the array of MQOR records (with as many array elements as are needed), and set *ODORO* to the offset of the first element in the array from the start of the MQOD. Care must be taken to ensure that this offset is correct.
- By using the pointer field *ODORP*

  In this case, the application can declare the array of MQOR structures separately from the MQOD structure, and set *ODORP* to the address of the array.

Whichever technique is chosen, one of *ODORO* and *ODORP* must be used; the call fails with reason code RC2155 if both are zero, or both are nonzero.

This is an input field. The initial value of this field is 0. This field is ignored if *ODVER* is less than ODVER2.

ODORP (pointer)

Address of first object record.

This is the address of the first MQOR object record. *ODORP* is used only when a distribution list is being opened. The field is ignored if *ODREC* is zero.

This is an input field. The initial value of this field is the null pointer. Either *ODORP* or *ODORO* can be used to specify the object records, but not both; see the description of the *ODORO* field previously for details. If *ODORP* is not used, it must be set to the null pointer or null bytes. This field is ignored if *ODVER* is less than ODVER2.

ODOS (MQCHARV)

ODOS specifies the long object name to be used.

This field is referenced only for certain values of *ODOT*. See the description of *ODOT* for details of which values indicate that this field is used.

If *ODOS* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code RC2441.

This is an input field. The initial values of the fields in this structure are the same as those in the MQCHARV structure.

The full topic name can be built from two different fields: *ODON* and *ODOS*. For details of how these two fields are used, see "Using topic strings" on page 2566. This field is ignored if *ODVER* is less than ODVER4.

ODOT (10-digit signed integer)

Object type.

Type of object being named in *ODON*. Possible values are:

OTQ Queue. The name of the object is found in *ODON*.

OTNLST
Namelist. The name of the object is found in *ODON*.

OTPRO
Process definition. The name of the object is found in *ODON*.

OTQM
Queue manager. The name of the object is found in *ODON*.

OTTOP
Topic. The full topic name can be built from two different fields: *ODON* and *ODOS*.

For details of how those two fields are used, see "Using topic strings" on page 2566.

If the object identified by the *ODON* field cannot be found, the call will fail with reason code RC2425 even if there is a string specified in *ODOS*.

This is always an input field. The initial value of this field is OTQ.

ODREC (10-digit signed integer)

Number of object records present.

This is the number of MQOR object records that have been provided by the application. If this number is greater than zero, it indicates that a distribution list is being opened, with *ODREC* being the number of destination queues in the list. It is valid for a distribution list to contain only one destination.

The value of *ODREC* must not be less than zero, and if it is greater than zero *ODOT* must be OTQ; the call fails with reason code RC2154 if these conditions are not satisfied.

This is an input field. The initial value of this field is 0. This field is ignored if *ODVER* is less than ODVER2.

ODRMN (48-byte character string)

Resolved queue manager name.

This is the name of the destination queue manager after name resolution has been performed by the local queue manager. The name returned is the name of the queue manager that owns the queue identified by *ODRQN*. *ODRMN* can be the name of the local queue manager.

If *ODRQN* is a shared queue that is owned by the queue-sharing group to which the local queue manager belongs, *ODRMN* is the name of the queue-sharing group. If the queue is owned by some other queue-sharing group, *ODRQN* can be the name of the queue-sharing group or the name of a queue manager that is a member of the queue-sharing group (the nature of the value returned is determined by the queue definitions that exist at the local queue manager).

A nonblank value is returned only if the object is a single queue opened for browse, input, or output (or any combination). If the object opened is any of the following, *ODRMN* is set to blanks:

- Not a queue
- A queue, but not opened for browse, input, or output
- A cluster queue with OOBNDN specified (or with OOBNDQ in effect when the **DefBind** queue attribute has the value BNDNOT)
- A distribution list

This is an output field. The length of this field is given by LNQN. The initial value of this field is the null string in C, and 48 blank characters in other programming languages. This field is ignored if *ODVER* is less than ODVER3.

ODRO (MQCHARV)

ODRO is the long object name after the queue manager resolves the name provided in *ODON*.

This field is returned only for certain types of objects, topics, and queue aliases which reference a topic object.

If the long object name is provided in *ODOS* and nothing is provided in *ODON*, the value returned in this field is the same as provided in *ODOS*.

If this field is omitted (that is ODRO.VSBufSize is zero), the *ODRO* is not returned, but the length is returned in ODRO.VSLength. If the length is shorter than the full *ODRO* then it is truncated and returns as many of the rightmost characters as can fit in the provided length.

If *ODRO* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code RC2520. This field is ignored if *ODVER* is less than ODVER4.

ODRQN (48-byte character string)

Resolved queue name.

This is the name of the destination queue after name resolution has been performed by the local queue manager. The name returned is the name of a queue that exists on the queue manager identified by *ODRMN*.

A nonblank value is returned only if the object is a single queue opened for browse, input, or output (or any combination). If the object opened is any of the following, *ODRQN* is set to blanks:

- Not a queue
- A queue, but not opened for browse, input, or output
- A distribution list
- An alias queue that references a topic object (refer to "ODRO (MQCHARV)" on page 3175 instead)

This is an output field. The length of this field is given by LNQN. The initial value of this field is the null string in C, and 48 blank characters in other programming languages. This field is ignored if *ODVER* is less than ODVER3.

ODRRO (10-digit signed integer)

Offset of first response record from start of MQOD.

This is the offset in bytes of the first MQRR response record from the start of the MQOD structure. The offset can be positive or negative. *ODRRO* is used only when a distribution list is being opened. The field is ignored if *ODREC* is zero.

When a distribution list is being opened, an array of one or more MQRR response records can be provided in order to identify the queues that failed to open (*RRCC* field in MQRR), and the reason for each failure (*RRREA* field in MQRR). The data is returned in the array of response records in the same order as the queue names occur in the array of object records. The queue manager sets the response records only when the outcome of the call is mixed (that is, some queues were opened successfully while others failed, or all failed but for differing reasons); reason code RC2136 from the call indicates this case. If the same reason code applies to all queues, that reason is returned in the **REASON** parameter of the MQOPEN or MQPUT1 call, and the response records are not set. Response records are optional, but if they are supplied there must be *ODREC* of them.

The response records can be provided in the same way as the object records, either by specifying an offset in *ODRRO*, or by specifying an address in *ODRRP* ; see the description of *ODORO* previously for details of how to do this. However, no more than one of *ODRRO* and *ODRRP* can be used; the call fails with reason code RC2156 if both are nonzero.

For the MQPUT1 call, these response records are used to return information about errors that occur when the message is sent to the queues in the distribution list, as well as errors that occur when the queues are opened. The completion code and reason code from the put operation for a queue replace those from the open operation for that queue only if the completion code from the latter was CCOK or CCWARN.

This is an input field. The initial value of this field is 0. This field is ignored if *ODVER* is less than ODVER2.

ODRRP (pointer)

Address of first response record.

This is the address of the first MQRR response record. *ODRRP* is used only when a distribution list is being opened. The field is ignored if *ODREC* is zero.

Either *ODRRP* or *ODRRO* can be used to specify the response records, but not both; see the previous description of the *ODRRO* field for details. If *ODRRP* is not used, it must be set to the null pointer or null bytes.

This is an input field. The initial value of this field is the null pointer. This field is ignored if *ODVER* is less than ODVER2.

ODSID (4-byte character string)

Structure identifier.

The value must be:

ODSIDV
Identifier for object descriptor structure.

This is always an input field. The initial value of this field is ODSIDV.

ODSS (MQCHARV)

ODSS contains the string used to provide the selection criteria used when retrieving messages off a queue.

ODSS must not be provided in the following cases:

- If *ODOT* is not OTQ
- If the queue being opened is not being opened using one of the input options, OOINP*

If *ODSS* is provided in these cases, the call fails with reason code RC2516.

If *ODSS* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code RC2519. This field is ignored if *ODVER* is less than ODVER4.

ODUDC (10-digit signed integer)

Number of remote queues opened successfully

This is the number of queues in the distribution list that resolve to remote queues and that were opened successfully. If present, this field is also set when opening a single queue which is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is ignored if *ODVER* is less than ODVER2.

ODVER (10-digit signed integer)

Structure version number.

The value must be one of the following:

ODVER1
Version-1 object descriptor structure.

ODVER2
Version-2 object descriptor structure.

ODVER3
Version-3 object descriptor structure.

ODVER4
Version-4 object descriptor structure.

Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

ODVERC
Current version of object descriptor structure.

This is always an input field. The initial value of this field is ODVER1.

Initial values

Table 345. Initial values of fields in MQOD

Field name	Name of constant	Value of constant
ODSID	ODSIDV	'OD¬¬'
ODVER	ODVER1	1
ODOT	OTQ	1
ODON	None	Blanks
ODMN	None	Blanks
ODDN	None	'AMQ.*'
ODAU	None	Blanks
ODREC	None	0
ODKDC	None	0
ODUDC	None	0
ODIDC	None	0
ODORO	None	0
ODRRO	None	0
ODORP	None	Null pointer or null bytes
ODRRP	None	Null pointer or null bytes
ODASI	SINONE	Nulls
ODRQN	None	Blanks
ODRMN	None	Blanks
ODOS	As defined for MQCHARV	As defined for MQCHARV
ODRO	As provided in *ODOS*	As provided in *ODOS*
ODSS	None	Blanks
Notes: 1. The symbol ¬ represents a single blank character.		

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D*
D* MQOD Structure
D*
D*
D* Structure identifier
D  ODSID                  1      4    INZ('OD  ')
D*
D* Structure version number
D  ODVER                  5      8I 0 INZ(1)
D*
D* Object type
D  ODOT                   9     12I 0 INZ(1)
D*
D* Object name
D  ODON                  13     60    INZ
D*
D* Object queue manager name
D  ODMN                  61    108    INZ
D*
D* Dynamic queue name
D  ODDN                 109    156    INZ('AMQ.*')
D*
D* Alternate user identifier
```

```
D  ODAU                 157    168    INZ
D*
** Number of object records
D* present
D  ODREC                169    172I 0 INZ(0)
D*
** Number of local queues opened
D* successfully
D  ODKDC                173    176I 0 INZ(0)
D*
** Number of remote queues opened
D* successfully
D  ODUDC                177    180I 0 INZ(0)
D*
** Number of queues that failed to
D* open
D  ODIDC                181    184I 0 INZ(0)
D*
** Offset of first object record
D* from start of MQOD
D  ODORO                185    188I 0 INZ(0)
D*
** Offset of first response record
D* from start of MQOD
D  ODRRO                189    192I 0 INZ(0)
D*
D* Address of first object record
D  ODORP                193    208*   INZ(*NULL)
D*
** Address of first response
D* record
D  ODRRP                209    224*   INZ(*NULL)
D*
D* Alternate security identifier
D  ODASI                225    264    INZ(X'00000000000000000-
D                                     00000000000000000000000000-
D                                     00000000000000000000000000-
D                                     0000000000000')
D*
D* Resolved queue name
D  ODRQN                265    312    INZ
D*
D* Resolved queue manager name
D  ODRMN                313    360    INZ
D*
D* reserved field
D  ODRE1                361    364I 0 INZ(0)
D*
D* reserved field
D  ODRS2                365    368I 0 INZ(0)
D*
D* Object long name
D* Address of variable length string
D  ODOSCHRP             369    384*   INZ(*NULL)
D* Offset of variable length string
D  ODOSCHRO             385    388I 0 INZ(0)
D* Size of buffer
D  ODOSVSBS             389    392I 0 INZ(-1)
D* Length of variable length string
D  ODOSCHRL             393    396I 0 INZ(0)
D* CCSID of variable length string
D  ODOSCHRC             397    400I 0 INZ(-3)
D*
D* Message Selector
D* Address of variable length string
D  ODSSCHRP             401    416*   INZ(*NULL)
D* Offset of variable length string
```

```
D  ODSSCHRO             417    420I 0 INZ(0)
D* Size of buffer
D  ODSSVSBS             421    424I 0 INZ(-1)
D* Length of variable length string
D  ODSSCHRL             425    428I 0 INZ(0)
D* CCSID of variable length string
D  ODSSCHRC             429    432I 0 INZ(-3)
D*
D* Resolved long object name
D* Address of variable length string
D  ODRSOCHRP            433    448*   INZ(*NULL)
D* Offset of variable length string
D  ODRSOCHRO            449    452I 0 INZ(0)
D* Size of buffer
D  ODRSOVSBS            453    456I 0 INZ(-1)
D* Length of variable length string
D  ODRSOCHRL            457    460I 0 INZ(0)
D* CCSID of variable length string
D  ODRSOCHRC            461    464I 0 INZ(-3)
D*
D* Alias queue resolved object type
D  ODRT                 465    468I 0 INZ(0)
```

MQOR (Object record) on IBM i: IBM i

The MQOR structure is used to specify the queue name and queue manager name of a single destination queue.

Overview

Purpose: MQOR is an input structure for the MQOPEN and MQPUT1 calls.

Character set and encoding: Data in MQOR must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT. However, if the application is running as an IBM MQ client, the structure must be in the character set and encoding of the client.

Usage: By providing an array of these structures on the MQOPEN call, it is possible to open a list of queues; this list is called a *distribution list*. Each message put using the queue handle returned by that MQOPEN call is placed on each of the queues in the list, if the queue was opened successfully.

- "Fields"
- "Initial values" on page 3181
- "RPG declaration" on page 3181

Fields

The MQOR structure contains the following fields; the fields are described in **alphabetical order**:

ORMN (48-byte character string)

Object queue manager name.

This is the same as the *ODMN* field in the MQOD structure (see MQOD for details).

This is always an input field. The initial value of this field is 48 blank characters.

ORON (48-byte character string)

Object name.

This is the same as the *ODON* field in the MQOD structure (see MQOD for details), except that:

- It must be the name of a queue.

- It must not be the name of a model queue.

This is always an input field. The initial value of this field is 48 blank characters.

Initial values

Table 346. Initial values of fields in MQOR

Field name	Name of constant	Value of constant
ORON	None	Blanks
ORMN	None	Blanks

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D*
D* MQOR Structure
D*
D* Object name
D  ORON                   1     48    INZ
D* Object queue manager name
D  ORMN                  49     96    INZ
```

MQPD - Property descriptor:

The **MQPD** is used to define the attributes of a property.

Overview

Purpose: The structure is an input/output parameter on the MQSETMP call and an output parameter on the MQINQMP call.

Character set and encoding: Data in MQPD must be in the character set of the application and encoding of the application (ENNAT).

- "Fields"
- "Initial values" on page 3184
- "RPG declaration" on page 3184

Fields

The MQPD structure contains the following fields; the fields are described in **alphabetical order**:

PDCT (10-digit signed integer)

This describes what message context the property belongs to.

When a queue manager receives a message containing an IBM MQ-defined property that the queue manager recognizes as being incorrect. the queue manager corrects the value of the *PDCT* field.

The following option can be specified:

PDUSC

The property is associated with the user context.

No special authorization is required to be able to set a property associated with the user context using the MQSETMP call.

On a IBM WebSphere MQ Version 7.0 queue manager, a property associated with the user context is saved as described for OOSAVA. An MQPUT call with PMPASA specified, causes the property to be copied from the saved context into the new message.

If the option previously described is not required, the following option can be used:

PDNOC
The property is not associated with a message context.

An unrecognized value is rejected with a *PDREA* code of RC2482.

This is an input/output field to the MQSETMP call and an output field from the MQINQMP call. The initial value of this field is PDNOC.

PDCPYOPT (10-digit signed integer)

This describes which type of messages the property should be copied into.

This is an output only field for recognized IBM MQ-defined properties; IBM MQ sets the appropriate value.

When a queue manager receives a message containing an IBM MQ-defined property that the queue manager recognizes as being incorrect. the queue manager corrects the value of the *CopyOptions* field.

You can specify one or more of these options. To specify more than one option, either add the values together (do not add the same constant more than once), or combine the values using the bitwise OR operation (if the programming language supports bit operations).

COPFOR
This property is copied into a message being forwarded.

COPPUB
This property is copied into the message received by a subscriber when a message is being published.

COPREP
This property is copied into a reply message.

COPRP
This property is copied into a report message.

COPALL
This property is copied into all types of subsequent messages.

COPNON
This property is not copied into a message.

Default option: The following option can be specified to supply the default set of copy options:

COPDEF
This property is copied into a message being forwarded, into a report message, or into a message received by a subscriber when a message is being published.

This is equivalent to specifying the combination of options COPFOR, plus COPRP, plus COPPUB.

If none of the options described previously are required, use the following option:

COPNON
Use this value to indicate that no other copy options have been specified; programmatically no relationship exists between this property and subsequent messages. This is always returned for message descriptor properties.

This is an input/output field to the MQSETMP call and an output field from the MQINQMP call. The initial value of this field is COPDEF.

PDOPT (10-digit signed integer)

The value must be:

PDNONE
No options specified

This is always an input field. The initial value of this field is PDNONE.

PDSID (10-digit signed integer)

This is the structure identifier; the value must be:

PSIDV
Identifier for property descriptor structure.

This is always an input field. The initial value of this field is **PSIDV**.

PDSUP (10-digit signed integer)

This field describes what level of support for the message property is required of the queue manager, in order for the message containing this property to be put to a queue. This applies only to IBM MQ-defined properties; support for all other properties is optional.

The field is automatically set to the correct value when the IBM MQ-defined property is known by the queue manager. If the property is not recognized, PDSUPO is assigned. When a queue manager receives a message containing an IBM MQ-defined property that the queue manager recognizes as being incorrect. the queue manager corrects the value of the *PDSUP* field.

When setting an IBM MQ-defined property using the MQSETMP call on a message handle where the CMNOVA option was set, *PDSUP* becomes an input field. This allows an application to put an IBM MQ-defined property, with the correct value, where the property is unsupported by the connected queue manager, but where the message is intended to be processed on another queue manager.

The value PDSUPO is always assigned to properties that are not IBM MQ-defined properties.

If a IBM WebSphere MQ Version 7.0 queue manager, that supports message properties, receives a property that contains an unrecognized *PDSUP* value, the property is treated as if:

- PDSUPR was specified if any of the unrecognized values are contained in the PDRUM.
- PDSUPL was specified if any of the unrecognized values are contained in the PDAUXM
- PDSUPO was specified otherwise.

One of the following values is returned by the MQINQMP call, or one of the values can be specified, when using the MQSETMP call on a message handle where the CMNOVA option is set:

PDSUPO
The property is accepted by a queue manager even if it is not supported. The property can be discarded in order for the message to flow to a queue manager that does not support message properties. This value is also assigned to properties that are not IBM MQ-defined.

PDSUPR
Support for the property is required. The message is rejected by a queue manager that does not support the IBM MQ-defined property. The MQPUT or MQPUT1 call fails with completion code CCFAIL and reason code RC2490.

PDSUPL
The message is rejected by a queue manager that does not support the IBM MQ-defined property if the message is destined for a local queue. The MQPUT or MQPUT1 call fails with completion code CCFAIL and reason code RC2490.

The MQPUT or MQPUT1 call succeeds if the message is destined for a remote queue manager.

This is an output field on the MQINQMP call and an input field on the MQSETMP call if the message handle was created with the CMNOVA option set. The initial value of this field is PDSUPO.

PDVER (10-digit signed integer)

This is the structure version number; the value must be:

PDVER1
Version-1 property descriptor structure.

The following constant specifies the version number of the current version:

PDVERC
Current version of property descriptor structure.

This is always an input field. The initial value of this field is **PDVER1**.

Initial values

Table 347. Initial values of fields in MQPD

Field name	Name of constant	Value of constant
PDSID	PDSIDV	'PD'
PDVER	PDVER1	1
PDOPT	PDNONE	0
PDSUP	PDSUPO	0
PDCT	PDNOC	0
PDCPYOPT	COPDEF	0

RPG declaration

```
D* MQDMHO Structure
D*
D*
D* Structure identifier
D  DMSID                  1      4    INZ('DMHO')
D*
D* Structure version number
D  DMVER                  5      8I 0 INZ(1)
D*
D* Options that control the action of MQDLTMH
D  DMOPT                  9     12I 0 INZ(0)
```

MQPMO (Put-message options) on IBM i: IBM i

The MQPMO structure allows the application to specify options that control how messages are placed on queues or published to topics.

Overview

Purpose

The structure is an input/output parameter on the MQPUT and MQPUT1 calls.

Version

The current version of MQPMO is PMVER2. Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions that follow.

The COPY file provided contains the most recent version of MQPMO that is supported by the environment, but with the initial value of the *PMVER* field set to PMVER1. To use fields that are not present in the version-1 structure, the application must set the *PMVER* field to the version number of the version required.

Character set and encoding

Data in MQPMO must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT. However, if the application is running as an IBM MQ client, the structure must be in the character set and encoding of the client.

- "Fields"
- "Initial values" on page 3199
- "RPG declaration" on page 3199

Fields

The MQPMO structure contains the following fields; the fields are described in alphabetical order:

PMCT (10 digit signed integer)

Object handle of input queue.

If PMPASI or PMPASA is specified, this field must contain the input queue handle from which context information to be associated with the message being put is taken.

If PMPASI and PMPASA are not specified, this field is ignored.

This is an input field. The initial value of this field is `0`.

PMIDC (10 digit signed integer)

Number of messages that could not be sent.

This is the number of messages that could not be sent to queues in the distribution list. The count includes queues that failed to open, and queues that were opened successfully but for which the put operation failed. This field is also set when putting a message to a single queue which is not in a distribution list.

Note: This field is set only if the **CMPCOD** parameter on the MQPUT or MQPUT1 call is CCOK or CCWARN; it is not set if the **CMPCOD** parameter is CCFAIL.

This is an output field. The initial value of this field is 0. This field is not set if *PMVER* is less than PMVER2.

PMKDC (10 digit signed integer)

Number of messages sent successfully to local queues.

This is the number of messages that the current MQPUT or MQPUT1 call has sent successfully to queues in the distribution list that are local queues. The count does not include messages sent to queues that resolve to remote queues (even though a local transmission queue is used initially to store the message). This field is also set when putting a message to a single queue which is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is not set if *PMVER* is less than PMVER2.

PMOPT (10 digit signed integer)

Options that control the action of MQPUT and MQPUT1.

Any or none of the following can be specified. If more than one is required the values can be added (do not add the same constant more than once). Combinations that are not valid are noted; any other combinations are valid.

Publishing options: The following options control the way messages are published to a topic.

PMSRTO

Any information filled into the MDRQ and MDRM fields of the MQMD of this publication is not passed on to subscribers. If this option is used with a report option that requires a ReplyToQ, the call fails with `RC2027` .

PMRET

The publication being sent is to be retained by the queue manager. This allows a subscriber to request a copy of this publication after the time it was published, by using the MQSUBRQ call. It also allows a publication to be sent to applications which make their subscription after the time this publication was made, unless they choose not to be sent it by using the option SONEWP. If an application is sent a publication which was retained, it is indicated by the mq.IsRetained message property of that publication.

Only one publication can be retained at each node of the topic tree. That means if there already is a retained publication for this topic, published by any other application, it is replaced with this publication. It is therefore better to avoid having more than one publisher retaining messages on the same topic.

When retained publications are requested by a subscriber, the subscription used may contain a wildcard in the topic, in which case a number of retained publications might match (at various nodes in the topic tree) and several publications may be sent to the requesting application. See the description of the “MQSUBRQ - Subscription request” on page 2789 call for more details.

If this option is used and the publication cannot be retained, the message is not published and the call fails with `RC2479` .

Syncpoint options: The following options relate to the participation of the MQPUT or MQPUT1 call within a unit of work:

PMSYP

Put message with syncpoint control.

The request is to operate within the normal unit-of-work protocols. The message is not visible outside the unit of work until the unit of work is committed. If the unit of work is backed out, the message is deleted.

If this option and PMNSYP are not specified, the put request is not within a unit of work.

PMSYP must not be specified with PMNSYP.

PMNSYP

Put message without syncpoint control.

The request is to operate outside the normal unit-of-work protocols. The message is available immediately, and it cannot be deleted by backing out a unit of work.

If this option and PMSYP are not specified, the put request is not within a unit of work.

PMNSYP must not be specified with PMSYP.

Message-identifier and correlation-identifier options: The following options request the queue manager to generate a new message identifier or correlation identifier:

PMNMID
Generate a new message identifier.

This option causes the queue manager to replace the contents of the *MDMID* field in MQMD with a new message identifier. This message identifier is sent with the message, and returned to the application on output from the MQPUT or MQPUT1 call.

This option can also be specified when the message is being put to a distribution list; see the description of the *PRMID* field in the MQPMR structure for details.

Using this option relieves the application of the need to reset the *MDMID* field to MINONE before each MQPUT or MQPUT1 call.

PMNCID
Generate a new correlation identifier.

This option causes the queue manager to replace the contents of the *MDCID* field in MQMD with a new correlation identifier. This correlation identifier is sent with the message, and returned to the application on output from the MQPUT or MQPUT1 call.

This option can also be specified when the message is being put to a distribution list; see the description of the *PRCID* field in the MQPMR structure for details.

PMNCID is useful in situations where the application requires a unique correlation identifier.

Group and segment options: The following option relates to the processing of messages in groups and segments of logical messages. These definitions might be of help in understanding the option:

Physical message
This is the smallest unit of information that can be placed on or removed from a queue; it often corresponds to the information specified or retrieved on a single MQPUT, MQPUT1, or MQGET call. Every physical message has its own message descriptor (MQMD). Generally, physical messages are distinguished by differing values for the message identifier (*MDMID* field in MQMD), although this is not enforced by the queue manager.

Logical message
This is a single unit of application information. In the absence of system constraints, a logical message would be the same as a physical message. But where logical messages are large, system constraints may make it advisable or necessary to split a logical message into two or more physical messages, called *segments*.

A logical message that has been segmented consists of two or more physical messages that have the same nonnull group identifier (*MDGID* field in MQMD), and the same message sequence number (*MDSEQ* field in MQMD). The segments are distinguished by differing values for the segment offset (*MDOFF* field in MQMD), which gives the offset of the data in the physical message from the start of the data in the logical message. Because each segment is a physical message, the segments in a logical message typically have differing message identifiers.

A logical message that has not been segmented, but for which segmentation has been permitted by the sending application, also has a nonnull group identifier, although in this case there is only one physical message with that group identifier if the logical message

does not belong to a message group. Logical messages for which segmentation has been inhibited by the sending application have a null group identifier (GINONE), unless the logical message belongs to a message group.

Message group
This is a set of one or more logical messages that have the same nonnull group identifier. The logical messages in the group are distinguished by differing values for the message sequence number, which is an integer in the range 1 through n, where n is the number of logical messages in the group. If one or more of the logical messages is segmented, there are more than n physical messages in the group.

PMLOGO
Messages in groups and segments of logical messages are put in logical order.

This option tells the queue manager how the application puts messages in groups and segments of logical messages. It can be specified only on the MQPUT call; it is not valid on the MQPUT1 call.

If PMLOGO is specified, it indicates that the application uses successive MQPUT calls to:

- Put the segments in each logical message in the order of increasing segment offset, starting from 0, with no gaps.
- Put all of the segments in one logical message before putting the segments in the next logical message.
- Put the logical messages in each message group in the order of increasing message sequence number, starting from 1, with no gaps.
- Put all of the logical messages in one message group before putting logical messages in the next message group.

This order is called "logical order".

Because the application has told the queue manager how it puts messages in groups and segments of logical messages, the application does not have to maintain and update the group and segment information about each MQPUT call, as the queue manager does this. Specifically, it means that the application does not need to set the *MDGID*, *MDSEQ*, and *MDOFF* fields in MQMD, as the queue manager sets these to the appropriate values. The application need set only the *MDMFL* field in MQMD, to indicate when messages belong to groups or are segments of logical messages, and to indicate the last message in a group or last segment of a logical message.

Once a message group or logical message has been started, subsequent MQPUT calls must specify the appropriate MF* flags in *MDMFL* in MQMD. If the application tries to put a message not in a group when there is an unterminated message group, or put a message which is not a segment when there is an unterminated logical message, the call fails with reason code RC2241 or RC2242, as appropriate. However, the queue manager retains the information about the current message group or current logical message, and the application can terminate them by sending a message (possibly with no application message data) specifying MFLMIG or MFLSEG as appropriate, before reissuing the MQPUT call to put the message that is not in the group or not a segment.

Table 348 on page 3189 shows the combinations of options and flags that are valid, and the values of the *MDGID*, *MDSEQ*, and *MDOFF* fields that the queue manager uses in each case. Combinations of options and flags that are not shown in the table are not valid. The columns in the table have the following meanings:

LOG ORD
Indicates whether the PMLOGO option is specified on the call.

MIG Indicates whether the MFMIG or MFLMIG option is specified on the call.

SEG Indicates whether the MFSEG or MFLSEG option is specified on the call.

SEG OK
Indicates whether the MFSEGA option is specified on the call.

Cur grp
Indicates whether a current message group exists before the call.

Cur log msg
Indicates whether a current logical message exists before the call.

Other columns
Show the values that the queue manager uses. "Previous" denotes the value used for the field in the previous message for the queue handle.

PMRLOC
Specifies that the PMRQN in the MQPMO structure must be completed with the name of the local queue which the message actually gets put to. The ResolvedQMgrName is similarly completed with the name of the local queue manager hosting the local queue. See OORLOQ for what this means. If a user is authorized for a put to a queue then they have the required authority to specify this flag on the MQPUT call. No special authority is needed.

Table 348. MQPUT options relating to messages in groups and segments of logical messages

Options you specify				Group and log-msg status before call		Values the queue manager uses		
LOG ORD	**MIG**	**SEG**	**SEG OK**	**Cur grp**	**Cur log msg**	*MDGID*	*MDSEQ*	*MDOFF*
Yes	No	No	No	No	No	GINONE	1	0
Yes	No	No	Yes	No	No	New group id	1	0
Yes	No	Yes	Yes or No	No	No	New group id	1	0
Yes	No	Yes	Yes or No	No	Yes	Previous group id	1	Previous offset + previous segment length
Yes	Yes	Yes or No	Yes or No	No	No	New group id	1	0
Yes	Yes	Yes or No	Yes or No	Yes	No	Previous group id	Previous sequence number + 1	0
Yes	Yes	Yes	Yes or No	Yes	Yes	Previous group id	Previous sequence number	Previous offset + previous segment length
No	No	No	No	Yes or No	Yes or No	GINONE	1	0
No	No	No	Yes	Yes or No	Yes or No	New group ID if GINONE, else value in field	1	0
No	No	Yes	Yes or No	Yes or No	Yes or No	New group ID if GINONE, else value in field	1	Value in field
No	Yes	No	Yes or No	Yes or No	Yes or No	New group ID if GINONE, else value in field	Value in field	0
No	Yes	Yes	Yes or No	Yes or No	Yes or No	New group ID if GINONE, else value in field	Value in field	Value in field

Table 348. MQPUT options relating to messages in groups and segments of logical messages (continued)

Options you specify	Group and log-msg status before call	Values the queue manager uses
Note: • PMLOGO is not valid on the MQPUT1 call. • For the *MDMID* field, the queue manager generates a new message identifier if PMNMID or MINONE is specified, and uses the value in the field otherwise. • For the *MDCID* field, the queue manager generates a new correlation identifier if PMNCID is specified, and uses the value in the field otherwise.		

When PMLOGO is specified, the queue manager requires that all messages in a group and segments in a logical message be put with the same value in the *MDPER* field in MQMD, that is, all must be persistent, or all must be nonpersistent. If this condition is not satisfied, the MQPUT call fails with reason code RC2185 .

The PMLOGO option affects units of work as follows:

- If the first physical message in a group or logical message is put within a unit of work, all of the other physical messages in the group or logical message must be put within a unit of work, if the same queue handle is used. However, they need not be put within the same unit of work. This allows a message group or logical message consisting of many physical messages to be split across two or more consecutive units of work for the queue handle.
- If the first physical message in a group or logical message is not put within a unit of work, none of the other physical messages in the group or logical message can be put within a unit of work, if the same queue handle is used.

If these conditions are not satisfied, the MQPUT call fails with reason code RC2245 .

When PMLOGO is specified, the MQMD supplied on the MQPUT call must not be less than MDVER2. If this condition is not satisfied, the call fails with reason code RC2257 .

If PMLOGO is not specified, messages in groups and segments of logical messages can be put in any order, and it is not necessary to put complete message groups or complete logical messages. It is the responsibility of the application to ensure that the *MDGID*, *MDSEQ*, *MDOFF*, and *MDMFL* fields have appropriate values.

This is the technique that can be used to restart a message group or logical message in the middle, after a system failure has occurred. When the system restarts, the application can set the *MDGID*, *MDSEQ*, *MDOFF*, *MDMFL*, and *MDPER* fields to the appropriate values, and then issue the MQPUT call with PMSYP or PMNSYP set as *necessary*, but without specifying PMLOGO. If this call is successful, the queue manager retains the group and segment information, and subsequent MQPUT calls using that queue handle can specify PMLOGO as normal.

The group and segment information that the queue manager retains for the MQPUT call is separate from the group and segment information that it retains for the MQGET call.

For any given queue handle, the application is free to mix MQPUT calls that specify PMLOGO with MQPUT calls that do not, but the following points should be noted:

- If PMLOGO is not specified, each successful MQPUT call causes the queue manager to set the group and segment information for the queue handle to the values specified by the application; this replaces the existing group and segment information retained by the queue manager for the queue handle.
- If PMLOGO is not specified, the call does not fail if there is a current message group or logical message; the call might however succeed with a CCWARN completion code.

Table 349 shows the various cases that can arise. In these cases, if the completion code is not CCOK, the reason code is one of the following (as appropriate):

- RC2241
- RC2242
- RC2185
- RC2245

Note: The queue manager does not check the group and segment information for the MQPUT1 call.

Table 349. Outcome when MQPUT or MQCLOSE call is not consistent with group and segment information

Current call is	Previous call was MQPUT with PMLOGO	Previous call was MQPUT without PMLOGO
MQPUT with PMLOGO	CCFAIL	CCFAIL
MQPUT without PMLOGO	CCWARN	CCOK
MQCLOSE with an unterminated group or logical message	CCWARN	CCOK

Applications that simply want to put messages and segments in logical order are recommended to specify PMLOGO, as this is the simplest option to use. This option relieves the application of the need to manage the group and segment information, because the queue manager manages that information. However, specialized applications may need more control than provided by the PMLOGO option, and this can be achieved by not specifying that option. If this is done, the application must ensure that the *MDGID*, *MDSEQ*, *MDOFF*, and *MDMFL* fields in MQMD are set correctly, before each MQPUT or MQPUT1 call.

For example, an application that wants to forward physical messages that it receives, without regard for whether those messages are in groups or segments of logical messages, must not specify PMLOGO. There are two reasons for this:

- If the messages are retrieved and put in order, specifying PMLOGO causes a new group identifier to be assigned to the messages, and this might make it difficult or impossible for the originator of the messages to correlate any reply or report messages that result from the message group.
- In a complex network with multiple paths between sending and receiving queue managers, the physical messages might arrive out of order. By not specifying PMLOGO and the corresponding GMLOGO on the MQGET call, the forwarding application can retrieve and forward each physical message as soon as it arrives, without needing to wait for the next one in logical order to arrive.

Applications that generate report messages for messages in groups or segments of logical messages must also not specify PMLOGO when putting the report message.

PMLOGO can be specified with any of the other PM* options.

Context options: The following options control the processing of message context:

PMNOC

No context is to be associated with the message.

Both identity and origin context are set to indicate no context. This means that the context fields in MQMD are set to:

- Blanks for character fields
- Nulls for byte fields
- Zeros for numeric fields

PMDEFC

Use default context.

The message is to have default context information associated with it, for both identity and origin. The queue manager sets the context fields in the message descriptor as follows:

Field in MQMD	Value used
MDUID	Determined from the environment if possible; set to blanks otherwise.
MDACC	Determined from the environment if possible; set to ACNONE otherwise.
MDAID	Set to blanks.
MDPAT	Determined from the environment.
MDPAN	Determined from the environment if possible; set to blanks otherwise.
MDPD	Set to date when message is put.
MDPT	Set to time when message is put.
MDAOD	Set to blanks.

For more information about message context, see Message context and Controlling context information.

This is the default action if no context options are specified.

PMPASI

Pass identity context from an input queue handle.

The message is to have context information associated with it. Identity context is taken from the queue handle specified in the *PMCT* field. Origin context information is generated by the queue manager in the same way that it is for PMDEFC (see the previous table for values). For more information about message context, see Message context and Controlling context information.

For the MQPUT call, the queue must have been opened with the OOPASI option (or an option that implies it). For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the OOPASI option.

PMPASA

Pass all context from an input queue handle.

The message is to have context information associated with it. Both identity and origin context are taken from the queue handle specified in the *PMCT* field. For more information about message context, see Message context and Controlling context information.

For the MQPUT call, the queue must have been opened with the OOPASA option (or an option that implies it). For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the OOPASA option.

PMSETI

Set identity context from the application.

The message is to have context information associated with it. The application specifies the identity context in the MQMD structure. Origin context information is generated by the queue manager in the same way that it is for PMDEFC (see the previous table for values). For more information about message context, see Message context and Controlling context information.

For the MQPUT call, the queue must have been opened with the OOSETI option (or an option that implies it). For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the OOSETI option.

PMSETA

Set all context from the application.

The message is to have context information associated with it. The application specifies the identity and origin context in the MQMD structure. For more information about message context, see Message context and Controlling context information.

For the MQPUT call, the queue must have been opened with the OOSETA option. For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the OOSETA option.

Only one of the PM* context options can be specified. If none of these options is specified, PMDEFC is assumed. **Put response types.** The following options control the response returned to an MQPUT or MQPUT1 call . You can only specify one of these options. If PMARES and PMSRES are not specified, PMRASQ or PMRAST is assumed.

PMARES

The PMARES option requests that an MQPUT or MQPUT1 operation is completed without the application waiting for the queue manager to complete the call. Using this option can improve messaging performance, particularly for applications using client bindings. An application can periodically check, using the MQSTAT verb, whether an error has occurred during any previous asynchronous calls. With this option, only the following fields are guaranteed to be completed in the MQMD;

- MDAID
- MDPAT
- MDPAN
- MDAOD

Additionally, if either or both of PMNMID or PMNCID are specified as options, the MDMID and MDCID returned are also completed. (PMNMID can be implicitly specified by specifying a blank MDMID field).

Only the fields previously specified are completed. Other information that would normally be returned in the MQMD or MQPMO structure is undefined.

When requesting asynchronous put response for MQPUT or MQPUT1, a CMPCOD and REASON of CCOK and RCNONE does not necessarily mean that the message was successfully put to a queue. When developing an MQI application that uses asynchronous put response and require confirmation that messages have been put to a queue you should check both CMPCOD and REASON codes from the put operations and also use MQSTAT to query asynchronous error information.

Although the success or failure of each individual MQPUT/MQPUT1 call might not be returned immediately, the first error that occurred under an asynchronous call can be determined at a later juncture through a call to MQSTAT.

If a persistent message under syncpoint fails to be delivered using asynchronous put response, and you attempt to commit the transaction, the commit fails and the transaction is backed out with a completion code of CCFAIL and a reason of `RC2003` . The application can make a call to MQSTAT to determine the cause of a previous MQPUT or MQPUT1 failure

PMSRES

Specifying this value for a put option in the MQPMO structure ensures that the MQPUT or MQPUT1 operation is always issued synchronously. If the operation is successful, all fields in the MQMD and MQPMO are completed. It is provided to ensure a synchronous response irrespective of the default put response value defined on the queue or topic object.

PMRASQ

If this value is specified for an MQPUT call, the put response type used is taken from the

DEFPRESP value specified on the queue when it was opened by the application. If a client application is connected to a queue manager at a level earlier than Version 7.0, it behaves as if PMSRES was specified.

If this option is specified for an MQPUT1 call, the DEFPRESP value from the queue definition is not used. If the MQPUT1 call is using PMSYP it behaves as for PMARES, and if it is using PMNSYP it behaves as for PMSRES.

PMRAST
This is a synonym for PMRASQ for use with topic objects.

Other options: The following options control authorization checking, and what happens when the queue manager is quiescing:

PMALTU
Validate with specified user identifier.

This indicates that the *ODAU* field in the **OBJDSC** parameter of the MQPUT1 call contains a user identifier that is to be used to validate authority to put messages on the queue. The call can succeed only if this *ODAU* is authorized to open the queue with the specified options, regardless of whether the user identifier under which the application is running is authorized to do so. (This does not apply to the context options specified, however, which are always checked against the user identifier under which the application is running.)

This option is valid only with the MQPUT1 call.

PMFIQ
Fail if queue manager is quiescing.

This option forces the MQPUT or MQPUT1 call to fail if the queue manager is in the quiescing state.

The call returns completion code CCFAIL with reason code RC2161 .

Default option: If none of the options described previously are required, the following option can be used:

PMNONE
No options specified.

This value can be used to indicate that no other options have been specified; all options assume their default values. PMNONE is defined to aid program documentation; it is not intended that this option is used with any other, but as its value is zero, such use cannot be detected.

This is an input field. The initial value of the *PMOPT* field is PMNONE.

PMPRF (10 digit signed integer)

Flags indicating which MQPMR fields are present.

This field contains flags that must be set to indicate which MQPMR fields are present in the put message records provided by the application. *PMPRF* is used only when the message is being put to a distribution list. The field is ignored if *PMREC* is zero, or both *PMPRO* and *PMPRP* are zero.

For fields that are present, the queue manager uses for each destination the values from the fields in the corresponding put message record. For fields that are absent, the queue manager uses the values from the MQMD structure.

One or more of the following flags can be specified to indicate which fields are present in the put message records:

PFMID
Message-identifier field is present.

PFCID
Correlation-identifier field is present.

PFGID
Group-identifier field is present.

PFFB Feedback field is present.

PFACC
Accounting-token field is present.

If this flag is specified, either PMSETI or PMSETA must be specified in the *PMOPT* field; if this condition is not satisfied, the call fails with reason code RC2158 .

If no MQPMR fields are present, the following can be specified:

PFNONE
No put-message record fields are present.

If this value is specified, either *PMREC* must be zero, or both *PMPRO* and *PMPRP* must be zero.

PFNONE is defined to aid program documentation. It is not intended that this constant is used with any other, but as its value is zero, such use cannot be detected.

If *PMPRF* contains flags which are not valid, or put message records are provided but *PMPRF* has the value PFNONE, the call fails with reason code RC2158 .

This is an input field. The initial value of this field is PFNONE. This field is ignored if *PMVER* is less than PMVER2.

PMPRO (10 digit signed integer)

Offset of first put message record from start of MQPMO.

This is the offset in bytes of the first MQPMR put message record from the start of the MQPMO structure. The offset can be positive or negative. *PMPRO* is used only when the message is being put to a distribution list. The field is ignored if *PMREC* is zero.

When the message is being put to a distribution list, an array of one or more MQPMR put message records can be provided in order to specify certain properties of the message for each destination individually; these properties are:

- message identifier
- correlation identifier
- group identifier
- feedback value
- accounting token

It is not necessary to specify all of these properties, but whatever subset is chosen, the fields must be specified in the correct order. See the description of the MQPMR structure for further details.

Usually, there should be as many put message records as there are object records specified by MQOD when the distribution list is opened; each put message record supplies the message properties for the queue identified by the corresponding object record. Queues in the distribution list which fail to open must still have put message records allocated for them at the appropriate positions in the array, although the message properties are ignored in this case.

It is possible for the number of put message records to differ from the number of object records. If there are fewer put message records than object records, the message properties for the destinations which do not have put message records are taken from the corresponding fields in the message descriptor MQMD. If there are more put message records than object records, the excess are not used (although it must still be possible to access them). Put message records are optional, but if they are supplied there must be *PMREC* of them.

The put message records can be provided in a similar way to the object records in MQOD, either by specifying an offset in *PMPRO*, or by specifying an address in *PMPRP* ; for details of how to do this, see the *ODORO* field described in "MQOD (Object descriptor) on IBM i" on page 3170.

No more than one of *PMPRO* and *PMPRP* can be used; the call fails with reason code RC2159 if both are nonzero.

This is an input field. The initial value of this field is 0. This field is ignored if *PMVER* is less than PMVER2.

PMPRP (pointer)

Address of first put message record.

This is the address of the first MQPMR put message record. *PMPRP* is used only when the message is being put to a distribution list. The field is ignored if *PMREC* is zero.

Either *PMPRP* or *PMPRO* can be used to specify the put message records, but not both; see the description of the PMRRO field for details. If *PMPRP* is not used, it must be set to the null pointer or null bytes.

This is an input field. The initial value of this field is the null pointer. This field is ignored if *PMVER* is less than PMVER2.

PMREC (10 digit signed integer)

Number of put message records or response records present.

This is the number of MQPMR put message records or MQRR response records that have been provided by the application. This number can be greater than zero only if the message is being put to a distribution list. Put message records and response records are optional - the application need not provide any records, or it can choose to provide records of only one type. However, if the application provides records of both types, it must provide *PMREC* records of each type.

The value of *PMREC* need not be the same as the number of destinations in the distribution list. If too many records are provided, the excess are not used; if too few records are provided, default values are used for the message properties for those destinations that do not have put message records (see *PMPRO* later in this topic).

If *PMREC* is less than zero, or is greater than zero but the message is not being put to a distribution list, the call fails with reason code RC2154 .

This is an input field. The initial value of this field is 0. This field is ignored if *PMVER* is less than PMVER2.

PMRMN (48 byte character string)

Resolved name of destination queue manager.

This is the name of the destination queue manager after name resolution has been performed by the local queue manager. The name returned is the name of the queue manager that owns the queue identified by *PMRQN*, and can be the name of the local queue manager.

If *PMRQN* is a shared queue that is owned by the queue-sharing group to which the local queue manager belongs, *PMRMN* is the name of the queue-sharing group. If the queue is owned by some other queue-sharing group, *PMRQN* can be the name of the queue-sharing group or the name of a queue manager that is a member of the queue-sharing group (the nature of the value returned is determined by the queue definitions that exist at the local queue manager).

A nonblank value is returned only if the object is a single queue; if the object is a distribution list or topic, the value returned is undefined.

This is an output field. The length of this field is given by LNQMN. The initial value of this field is 48 blank characters.

PMRQN (48 byte character string)

Resolved name of destination queue.

This is the name of the destination queue after name resolution has been performed by the local queue manager. The name returned is the name of a queue that exists on the queue manager identified by *PMRMN*.

A nonblank value is returned only if the object is a single queue; if the object is a distribution list or topic, the value returned is undefined.

This is an output field. The length of this field is given by LNQN. The initial value of this field is 48 blank characters.

PMRRO (10 digit signed integer)

Offset of first response record from start of MQPMO.

This is the offset in bytes of the first MQRR response record from the start of the MQPMO structure. The offset can be positive or negative. *PMRRO* is used only when the message is being put to a distribution list. The field is ignored if *PMREC* is zero.

When the message is being put to a distribution list, an array of one or more MQRR response records can be provided in order to identify the queues to which the message was not sent successfully (*RRCC* field in MQRR), and the reason for each failure (*RRREA* field in MQRR). The message might not have been sent either because the queue failed to open, or because the put operation failed. The queue manager sets the response records only when the outcome of the call is mixed (that is, some messages were sent successfully while others failed, or all failed but for differing reasons); reason code RC2136 from the call indicates this case. If the same reason code applies to all queues, that reason is returned in the **REASON** parameter of the MQPUT or MQPUT1 call, and the response records are not set.

Usually, there should be as many response records as there are object records specified by MQOD when the distribution list is opened; when necessary, each response record is set to the completion code and reason code for the put to the queue identified by the corresponding object record. Queues in the distribution list which fail to open must still have response records allocated for them at the appropriate positions in the array, although they are set to the completion code and reason code resulting from the open operation, rather than the put operation.

It is possible for the number of response records to differ from the number of object records. If there are fewer response records than object records, it may not be possible for the application to identify all of the destinations for which the put operation failed, or the reasons for the failures. If there are more response records than object records, the excess are not used (although it must still be possible to access them). Response records are optional, but if they are supplied there must be *PMREC* of them.

The response records can be provided in a similar way to the object records in MQOD, either by specifying an offset in *PMRRO*, or by specifying an address in *PMRRP* ; for details of how to do this, see the *ODORO* field described in "MQOD (Object descriptor) on IBM i" on page 3170. However, no more than one of *PMRRO* and *PMRRP* can be used; the call fails with reason code RC2156 if both are nonzero.

For the MQPUT1 call, this field must be zero. This is because the response information (if requested) is returned in the response records specified by the object descriptor MQOD.

This is an input field. The initial value of this field is 0. This field is ignored if *PMVER* is less than PMVER2.

PMRRP (pointer)

Address of first response record.

This is the address of the first MQRR response record. *PMRRP* is used only when the message is being put to a distribution list. The field is ignored if *PMREC* is zero.

Either *PMRRP* or *PMRRO* can be used to specify the response records, but not both; see the description of the PMRRO field for details. If *PMRRP* is not used, it must be set to the null pointer or null bytes.

For the MQPUT1 call, this field must be the null pointer or null bytes. This is because the response information (if requested) is returned in the response records specified by the object descriptor MQOD.

This is an input field. The initial value of this field is the null pointer. This field is ignored if *PMVER* is less than PMVER2.

PMSID (4 byte character string)

Structure identifier.

The value must be:

PMSIDV
Identifier for put-message options structure.

This is always an input field. The initial value of this field is PMSIDV.

PMSL (MQLONG)

The level of subscription targeted by this publication.

Only those subscriptions with the highest *PMSL* less than or equal to this value receives this publication. This value must be in the range zero to 9; zero is the lowest level.

The initial value of this field is 9.

PMTO (10 digit signed integer)

Reserved.

This is a reserved field; its value is not significant. The initial value of this field is -1.

PMUDC (10 digit signed integer)

Number of messages sent successfully to remote queues.

This is the number of messages that the current MQPUT or MQPUT1 call has sent successfully to queues in the distribution list that resolve to remote queues. Messages that the queue manager retains temporarily in distribution-list form count as the number of individual destinations that those distribution lists contain. This field is also set when putting a message to a single queue which is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is not set if *PMVER* is less than PMVER2.

PMVER (10 digit signed integer)

Structure version number.

The value must be one of the following:

PMVER1
Version-1 put-message options structure.

PMVER2
Version-2 put-message options structure.

Fields that exist only in the more-recent version of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

PMVERC
Current version of put-message options structure.

This is always an input field. The initial value of this field is PMVER1.

Initial values

Table 350. Initial values of fields in MQPMO

Field name	Name of constant	Value of constant
PMSID	PMSIDV	'PMO¬'
PMVER	PMVER1	1
PMOPT	PMNONE	0
PMTO	None	-1
PMCT	None	0
PMKDC	None	0
PMUDC	None	0
PMIDC	None	0
PMRQN	None	Blanks
PMRMN	None	Blanks
PMREC	None	0
PMPRF	PFNONE	0
PMPRO	None	0
PMRRO	None	0
PMPRP	None	Null pointer or null bytes
PMRRP	None	Null pointer or null bytes
Note: 1. The symbol ¬ represents a single blank character.		

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQPMO Structure
D*
D* Structure identifier
D  PMSID                  1      4    INZ('PMO ')
D* Structure version number
D  PMVER                  5      8I 0 INZ(1)
D* Options that control the action of MQPUT and MQPUT1
D  PMOPT                  9     12I 0 INZ(0)
D* Reserved
D  PMTO                  13     16I 0 INZ(-1)
D* Object handle of input queue
D  PMCT                  17     20I 0 INZ(0)
D* Number of messages sent successfully to local queues
D  PMKDC                 21     24I 0 INZ(0)
D* Number of messages sent successfully to remote queues
D  PMUDC                 25     28I 0 INZ(0)
D* Number of messages that could notbe sent
D  PMIDC                 29     32I 0 INZ(0)
D* Resolved name of destination queue
D  PMRQN                 33     80    INZ
D* Resolved name of destination queue manager
D  PMRMN                 81    128    INZ
D* Number of put message records or response records present
D  PMREC                129    132I 0 INZ(0)
D* Flags indicating which MQPMR fields are present
D  PMPRF                133    136I 0 INZ(0)
D* Offset of first put message record from start of MQPMO
```

```
D  PMPRO                137    140I 0 INZ(0)
D* Offset of first response record from start of MQPMO
D  PMRRO                141    144I 0 INZ(0)
D* Address of first put message record
D  PMPRP                145    160*   INZ(*NULL)
D* Address of first response record
D  PMRRP                161    176*   INZ(*NULL)
D* Original message handle
D  PMOMH                177    184I 0
D* New message handle
D  PMNMH                185    190I 0
D* The action being performed
D  PMACT                191    194I 0
D* Reserved
D  PMRE1                195    198I 0
```

MQPMR (Put-message record) on IBM i: IBM i

The MQPMR structure is used to specify various message properties for a single destination when a message is being put to a distribution list.

Overview

Purpose: MQPMR is an input/output structure for the MQPUT and MQPUT1 calls.

Character set and encoding: Data in MQPMR must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT. However, if the application is running as an IBM MQ client, the structure must be in the character set and encoding of the client.

Usage: By providing an array of these structures on the MQPUT or MQPUT1 call, it is possible to specify different values for each destination queue in a distribution list. Some of the fields are input only, others are input/output.

Note: This structure is unusual in that it does not have a fixed layout. The fields in this structure are optional, and the presence or absence of each field is indicated by the flags in the *PMPRF* field in MQPMO. Fields that are present ***must occur in the following order*** :

- *PRMID*
- *PRCID*
- *PRGID*
- *PRFB*
- *PRACC*

Fields that are absent occupy no space in the record.

Because MQPMR does not have a fixed layout, no definition of it is provided in the COPY file. The application programmer should create a declaration containing the fields that are required by the application, and set the flags in *PMPRF* to indicate the fields that are present.

- "Fields"
- "Initial values" on page 3202
- "RPG declaration" on page 3202

Fields

The MQPMR structure contains the following fields; the fields are described in **alphabetical order**:

PRACC (32-byte bit string)

Accounting token.

This is the accounting token to be used for the message sent to the queue with a name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *MDACC* field in MQMD for a put to a single queue. See the description of *MDACC* in "MQMD (Message descriptor) on IBM i" on page 3118 for information about the content of this field.

If this field is not present, the value in MQMD is used.

This is an input field.

PRCID (24-byte bit string)

Correlation identifier.

This is the correlation identifier to be used for the message sent to the queue with the name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *MDCID* field in MQMD for a put to a single queue.

If this field is not present in the MQPMR record, or there are fewer MQPMR records than destinations, the value in MQMD is used for those destinations that do not have an MQPMR record containing a *PRCID* field.

If PMNCID is specified, a *single* new correlation identifier is generated and used for all of the destinations in the distribution list, regardless of whether they have MQPMR records. This is different from the way that PMNMID is processed (see *PRMID* field).

This is an input/output field.

PRFB (10-digit signed integer)

Feedback or reason code.

This is the feedback code to be used for the message sent to the queue with the name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *MDFB* field in MQMD for a put to a single queue.

If this field is not present, the value in MQMD is used.

This is an input field.

PRGID (24-byte bit string)

Group identifier.

This is the group identifier to be used for the message sent to the queue with the name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *MDGID* field in MQMD for a put to a single queue.

If this field is not present in the MQPMR record, or there are fewer MQPMR records than destinations, the value in MQMD is used for those destinations that do not have an MQPMR record containing a *PRGID* field. The value is processed as documented in Table 348 on page 3189, but with the following differences:

- In those cases where a new group identifier would be used, the queue manager generates a different group identifier for each destination (that is, no two destinations have the same group identifier).
- In those cases where the value in the field would be used, the call fails with reason code RC2258.

This is an input/output field.

PRMID (24-byte bit string)

Message identifier.

This is the message identifier to be used for the message sent to the queue with the name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *MDMID* field in MQMD for a put to a single queue.

If this field is not present in the MQPMR record, or there are fewer MQPMR records than destinations, the value in MQMD is used for those destinations that do not have an MQPMR record containing a *PRMID* field. If that value is MINONE, a new message identifier is generated for *each* of those destinations (that is, no two of those destinations have the same message identifier).

If PMNMID is specified, new message identifiers are generated for all of the destinations in the distribution list, regardless of whether they have MQPMR records. This is different from the way that PMNCID is processed (see *PRCID* field).

This is an input/output field.

Initial values

There are no initial values defined for this structure, as no structure declaration is provided. The following sample declaration shows how the structure should be declared by the application programmer if all of the fields are required.

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQPMR Structure
D*
D* Message identifier
D  PRMID                  1     24
D* Correlation identifier
D  PRCID                 25     48
D* Group identifier
D  PRGID                 49     72
D* Feedback or reason code
D  PRFB                  73     76I 0
D* Accounting token
D  PRACC                 77    108
```

MQRFH (Rules and formatting header) on IBM i: 

The MQRFH structure defines the layout of the rules and formatting header.

Overview

Purpose: This header can be used to send string data in the form of name-value pairs.

Format name: FMRFH.

Character set and encoding: The fields in the MQRFH structure (including *RFNVS*) are in the character set and encoding given by the *MDCSI* and *MDENC* fields in the header structure that precedes the MQRFH, or by those fields in the MQMD structure if the MQRFH is at the start of the application message data.

The character set must be one that has single-byte characters for the characters that are valid in queue names.

- "Fields" on page 3203

- "Initial values" on page 3205
- "RPG declaration" on page 3205

Fields

The MQRFH structure contains the following fields; the fields are described in **alphabetical order**:

RFCSI (10-digit signed integer)

Character set identifier of data that follows *RFNVS*.

This specifies the character set identifier of the data that follows *RFNVS* ; it does not apply to character data in the MQRFH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

CSINHT
Inherit character-set identifier of this structure.

Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value CSINHT is not returned by the MQGET call.

CSINHT cannot be used if the value of the *MDPAT* field in MQMD is ATBRKR.

The initial value of this field is CSUNDF.

Numeric encoding of data that follows *RFNVS*.

This specifies the numeric encoding of the data that follows *RFNVS* ; it does not apply to numeric data in the MQRFH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is ENNAT.

RFFLG (10-digit signed integer)

Flags.

The following can be specified:

RFNONE
No flags.

The initial value of this field is RFNONE.

RFFMT (8-byte character string)

Format name of data that follows *RFNVS*.

This specifies the format name of the data that follows *RFNVS*.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *MDFMT* field in MQMD.

The initial value of this field is FMNONE.

RFLEN (10-digit signed integer)

Total length of MQRFH including *RFNVS*.

This is the length in bytes of the MQRFH structure, including the *RFNVS* field at the end of the structure. The length does not include any user data that follows the *RFNVS* field.

To avoid problems with data conversion of the user data in some environments, consider using *RFLEN* as a multiple of four.

The following constant gives the length of the *fixed* part of the structure, that is, the length excluding the *RFNVS* field:

RFLENV
Length of fixed part of MQRFH structure.

The initial value of this field is RFLENV.

RFNVS (n-byte character string)

String containing name-value pairs.

This is a variable-length character string containing name-value pairs in the form:

```
name1 value1 name2 value2 name3 value3 ...
```

Each name or value must be separated from the adjacent name or value by one or more blank characters; these blanks are not significant. A name or value can contain significant blanks by prefixing and suffixing the name or value with the quotation mark character; all characters between the opening quotation mark and the matching closing quotation mark are treated as significant. In the following example, the name is `FAMOUS_WORDS`, and the value is `Hello World`:

```
FAMOUS_WORDS "Hello World"
```

A name or value can contain any characters other than the null character (which acts as a delimiter for *RFNVS*). However, to assist interoperability an application might prefer to restrict names to the following characters:

- First character: uppercase or lowercase alphabetic (A through Z, or a through z), or underscore.
- Subsequent characters: upper or lowercase alphabetic, decimal digit (0 through 9), underscore, hyphen, or dot.

If a name or value contains one or more quotation marks, the name or value must be enclosed in quotation marks, and each quotation mark within the string must be doubled:

```
Famous_Words "The program displayed ""Hello World"""
```

Names and values are case sensitive, that is, lowercase letters are not considered to be the same as uppercase letters. For example, `FAMOUS_WORDS` and `Famous_Words` are two different names.

The length in bytes of *RFNVS* is equal to *RFLEN* minus RFLENV. To avoid problems with data conversion of the user data in some environments, it is recommended that this length should be a multiple of four. *RFNVS* must be padded with blanks to this length, or terminated earlier by placing a null character following the last significant character in the string. The null character and the bytes following it, up to the specified length of *RFNVS*, are ignored.

Note: Because the length of this field is not fixed, the field is omitted from the declarations of the structure that are provided for the supported programming languages.

RFSID (4-byte character string)

Structure identifier.

The value must be:

RFSIDV
Identifier for rules and formatting header structure.

The initial value of this field is RFSIDV.

RFVER (10-digit signed integer)

Structure version number.

The value must be:

RFVER1
Version-1 rules and formatting header structure.

The initial value of this field is RFVER1.

Initial values

Table 351. Initial values of fields in MQRFH

Field name	Name of constant	Value of constant
RFSID	RFSIDV	'RFH¬'
RFVER	RFVER1	1
RFLEN	RFLENV	32
RFENC	ENNAT	Depends on environment
RFCSI	CSUNDF	0
RFFMT	FMNONE	Blanks
RFFLG	RFNONE	0
Notes: 1. The symbol ¬ represents a single blank character.		

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQRFH Structure
D*
D* Structure identifier
D  RFSID                  1      4    INZ('RFH ')
D* Structure version number
D  RFVER                  5      8I 0 INZ(1)
D* Total length of MQRFH includingNameValueString
D  RFLEN                  9     12I 0 INZ(32)
D* Numeric encoding of data that followsNameValueString
D  RFENC                 13     16I 0 INZ(273)
D* Character set identifier of data thatfollows NameValueString
D  RFCSI                 17     20I 0 INZ(0)
D* Format name of data that followsNameValueString
D  RFFMT                 21     28    INZ('        ')
D* Flags
D  RFFLG                 29     32I 0 INZ(0)
```

MQRFH2 (Rules and formatting header 2) on IBM i: IBM i

The MQRFH2 structure defines the format of the version-2 rules and formatting header.

Overview

Purpose: This header can be used to send data that has been encoded using an XML-like syntax. A message can contain two or more MQRFH2 structures in series, with user data optionally following the last MQRFH2 structure in the series.

Format name: FMRFH2.

Character set and encoding: Special rules apply to the character set and encoding used for the MQRFH2 structure:

- Fields other than *RF2NVD* are in the character set and encoding given by the *MDCSI* and *MDENC* fields in the header structure that precedes MQRFH2, or by those fields in the MQMD structure if the MQRFH2 is at the start of the application message data.

  The character set must be one that has single-byte characters for the characters that are valid in queue names.

  When GMCONV is specified on the MQGET call, the queue manager converts these fields to the requested character set and encoding.
- *RF2NVD* is in the character set given by the *RF2NVC* field. Only certain Unicode character sets are valid for *RF2NVC* (see the description of *RF2NVC* for details).

  Some character sets have a representation that is dependent on the encoding. If *RF2NVC* is one of these character sets, *RF2NVD* must be in the same encoding as the other fields in the MQRFH2.

  When GMCONV is specified on the MQGET call, the queue manager converts *RF2NVD* to the requested encoding, but does not change its character set.
- "Fields"
- "Initial values" on page 3210
- "RPG declaration" on page 3211

Fields

The MQRFH2 structure contains the following fields; the fields are described in **alphabetical order**:

RF2CSI (10-digit signed integer)

Character set identifier of data that follows last *RF2NVD* field.

This specifies the character set identifier of the data that follows the last *RF2NVD* field; it does not apply to character data in the MQRFH2 structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

CSINHT

Inherit character-set identifier of this structure.

Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value CSINHT is not returned by the MQGET call.

CSINHT cannot be used if the value of the *MDPAT* field in MQMD is ATBRKR.

The initial value of this field is CSINHT.

RF2ENC (10-digit signed integer)

Numeric encoding of data that follows last *RF2NVD* field.

This specifies the numeric encoding of the data that follows the last *RF2NVD* field; it does not apply to numeric data in the MQRFH2 structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is ENNAT.

RF2FLG (10-digit signed integer)

Flags.

The following value must be specified:

RFNONE
No flags.

The initial value of this field is RFNONE.

RF2FMT (8-byte character string)

Format name of data that follows last *RF2NVD* field.

This specifies the format name of the data that follows the last *RF2NVD* field.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *MDFMT* field in MQMD.

The initial value of this field is FMNONE.

RF2LEN (10-digit signed integer)

Total length of MQRFH2 including all *RF2NVL* and *RF2NVD* fields.

This is the length in bytes of the MQRFH2 structure, including the *RF2NVL* and *RF2NVD* fields at the end of the structure. It is valid for there to be multiple pairs of *RF2NVL* and *RF2NVD* fields at the end of the structure, in the sequence:

```
length1, data1, length2, data2, ...
```

RF2LEN does not include any user data that may follow the last *RF2NVD* field at the end of the structure.

To avoid problems with data conversion of the user data in some environments, consider using *RF2LEN* as a multiple of four.

The following constant gives the length of the *fixed* part of the structure, that is, the length excluding the *RF2NVL* and *RF2NVD* fields:

RFLEN2
Length of fixed part of MQRFH2 structure.

The initial value of this field is RFLEN2.

RF2NVC (10-digit signed integer)

Character set identifier of *RF2NVD*.

This specifies the coded character set identifier of the data in the *RF2NVD* field. This is different from the character set of the other strings in the MQRFH2 structure, and can be different from the character set of the data (if any) that follows the last *RF2NVD* field at the end of the structure.

RF2NVC must have one of the following values:

V 9.0.0

CCSID	Meaning
1200	UTF-16, most recent Unicode version supported
13488	UTF-16, Unicode version 2.0 subset
17584	UTF-16, Unicode version 3.0 subset (includes the Euro symbol)
1208	UTF-8, most recent Unicode version supported

For the V 9.0.0 UTF-16 character sets, the encoding (byte order) of the *RF2NVD* must be the same as the encoding of the other fields in the MQRFH2 structure. Surrogate characters (X'D800' through X'DFFF') are not supported.

Note: If *RF2NVC* does not have one of the values listed previously, and the MQRFH2 structure requires conversion on the MQGET call, the call completes with reason code RC2111 and the message is returned unconverted.

The initial value of this field is 1208.

RF2NVD (n-byte character string)

Name/value data.

This is a variable-length character string containing data encoded using an XML-like syntax. The length in bytes of this string is provided by the *RF2NVL* field that precedes the *RF2NVD* field; this length should be a multiple of four.

The *RF2NVL* and *RF2NVD* fields are optional, but if present they must occur as a pair and be adjacent. The pair of fields can be repeated as many times as required, for example:

```
length1 data1 length2 data2 length3 data3
```

Because these fields are optional, they are omitted from the declarations of the structure that are provided for the various programming languages supported.

RF2NVD is unusual because it is not converted to the character set specified on the MQGET call when the message is retrieved with the GMCONV option in effect; *RF2NVD* remains in its original character set. However, *RF2NVD* is converted to the encoding specified on the MQGET call.

Syntax of name/value data: The string consists of a single "folder" that contains zero or more properties. The folder is delimited by XML start and end tags with the same name as the the folder:

```
<folder> property1 property2 ... </folder>
```

Characters following the folder end tag, up to the length defined by *RF2NVL*, must be blank. Within the folder, each property is composed of a name and a value, and optionally a data type:

```
<name dt="datatype">value</name>
```

In these examples:

- The delimiter characters (<, =, ", /, and >) must be specified exactly as shown.
- `name` is the user-specified name of the property; see the following example for more information about names.
- `datatype` is an optional user-specified data type of the property; see the following example for valid data types.
- `value` is the user-specified value of the property; see the following paragraphs for more information about values.

- Blanks are significant between the > character which precedes a value, and the < character which follows the value, and at least one blank must precede dt=. Elsewhere blanks can be coded freely between tags, or preceding or following tags (for example, in order to improve readability); these blanks are not significant.

If properties are related to each other, they can be grouped together by enclosing them within XML start and end tags with the same name as the group:

```
<folder> <group> property1 property2 ... </group> </folder>
```

Groups can be nested within other groups, without limit, and a group can occur more than once within a folder. It is also valid for a folder to contain some properties in groups and other properties not in groups.

Names of properties, groups, and folders: Names of properties, groups, and folders must be valid XML tag names, with the exception of the colon character, which is not permitted in a property, group, or folder name. In particular:

- Names must start with a letter or an underscore. Valid letters are defined in the W3C XML specification, and consist essentially of Unicode categories Ll, Lu, Lo, Lt, and Nl.
- The remaining characters in a name can be letters, decimal digits, underscores, hyphens, or dots. These correspond to Unicode categories Ll, Lu, Lo, Lt, Nl, Mc, Mn, Lm, and Nd.
- The Unicode compatibility characters (X'F900' and above) are not permitted in any part of a name.
- Names must not start with the string XML in any mixture of upper or lowercase.

In addition:

- Names are case-sensitive. For example, ABC, abc, and Abc are three different names.
- Each folder has a separate namespace. As a result, a group or property in one folder does not conflict with a group or property of the same name in another folder.
- Groups and properties occupy the same namespace within a folder. As a result, a property cannot have the same name as a group within the folder containing that property.

Generally, programs that analyze the *RF2NVD* field should ignore properties or groups that have names that the program does not recognize, provided that those properties or groups are correctly formed.

Data types of properties: Each property can have an optional data type. If specified, the data type must be one of the following values, in upper, lower, or mixed case:

Data type	Used for
string	Any sequence of characters. Certain characters must be specified using escape sequences.
boolean	The character 0 or 1 (1 denotes TRUE).
bin.hex	Hexadecimal digits representing octets.
i1	Integer number in the range -128 through +127, expressed using only decimal digits and optional sign.
i2	Integer number in the range -32 768 through +32 767, expressed using only decimal digits and optional sign.
i4	Integer number in the range -2 147 483 648 through +2 147 483 647, expressed using only decimal digits and optional sign.
i8	Integer number in the range -9 223 372 036 854 775 808 through +9 223 372 036 854 775 807, expressed using only decimal digits and optional sign.
int	Integer number in the range -9 223 372 036 854 775 808 through +9 223 372 036 854 775 807, expressed using only decimal digits and optional sign. This can be used in place of i1, i2, i4, or i8 if the sender does not want to imply a particular precision.
r4	Floating-point number with magnitude in the range 1.175E-37 through 3.402 823 47E+38, expressed using decimal digits, optional sign, optional fractional digits, and optional exponent.

Data type	Used for
r8	Floating-point number with magnitude in the range 2.225E-307 through 1.797 693 134 862 3E+308 expressed using decimal digits, optional sign, optional fractional digits, and optional exponent.

Values of properties: The value of a property can consist of any characters, except as detailed in the following figure. Each occurrence in the value of a character marked as "mandatory" must be replaced by the corresponding escape sequence. Each occurrence in the value of a character marked as "optional" can be replaced by the corresponding escape sequence, but this is not required.

Character	Escape sequence	Usage
&	&	Mandatory
<	<	Mandatory
>	>	Optional
"	"	Optional
'	'	Optional

Note: The & character at the start of an escape sequence must not be replaced by &.

In the following example, the blanks in the value are significant; however, no escape sequences are needed:

```
<Famous_Words>The program displayed "Hello World"</Famous_Words>
```

RF2NVL (10-digit signed integer)

Length of *RF2NVD*.

This specifies the length in bytes of the data in the *RF2NVD* field. To avoid problems with data conversion of the data (if any) that *follows* the *RF2NVD* field, *RF2NVL* should be a multiple of four.

Note: The *RF2NVL* and *RF2NVD* fields are optional, but if present they must occur as a pair and be adjacent. The pair of fields can be repeated as many times as required, for example:

```
length1 data1 length2 data2 length3 data3
```

Because these fields are optional, they are omitted from the declarations of the structure that are provided for the various programming languages supported.

RF2SID (4-byte character string)

Structure identifier.

The value must be:

RFSIDV
Identifier for rules and formatting header structure.

The initial value of this field is RFSIDV.

RF2VER (10-digit signed integer)

Structure version number.

The value must be:

RFVER2
Version-2 rules and formatting header structure.

The initial value of this field is RFVER2.

Initial values

Table 352. Initial values of fields in MQRFH2

Field name	Name of constant	Value of constant
RF2SID	RFSIDV	'RFH¬'
RF2VER	RFVER2	2
RF2LEN	RFLEN2	36
RF2ENC	ENNAT	Depends on environment
RF2CSI	CSINHT	-2
RF2FMT	FMNONE	Blanks
RF2FLG	RFNONE	0
RF2NVC	None	1208
Notes: 1. The symbol ¬ represents a single blank character.		

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D*
D* MQRFH2 Structure
D*
D* Structure identifier
D  RF2SID                 1      4    INZ('RFH ')
D* Structure version number
D  RF2VER                 5      8I 0 INZ(2)
D* Total length of MQRFH2 including allNameValueLength and
D* NameValueDatafields
D  RF2LEN                 9     12I 0 INZ(36)
D* Numeric encoding of data that followslast NameValueData field
D  RF2ENC                13     16I 0 INZ(273)
D* Character set identifier of data thatfollows last NameValueData field
D  RF2CSI                17     20I 0 INZ(-2)
D* Format name of data that follows lastNameValueData field
D  RF2FMT                21     28    INZ('        ')
D* Flags
D  RF2FLG                29     32I 0 INZ(0)
D* Character set identifier ofNameValueData
D  RF2NVC                33     36I 0 INZ(1208)
```

MQRMH (Reference message header) on IBM i: IBM i

The MQRMH structure defines the format of a reference message header.

Overview

Purpose: This header is used with user-written message channel exits to send large amounts of data (called "bulk data") from one queue manager to another. The difference compared to normal messaging is that the bulk data is not stored on a queue; instead, only a *reference* to the bulk data is stored on the queue. This reduces the possibility of IBM MQ resources being exhausted by a few large messages.

Format name: FMRMH.

Character set and encoding: Character data in MQRMH, and the strings addressed by the offset fields, must be in the character set of the local queue manager; this is given by the **CodedCharSetId** queue manager attribute. Numeric data in MQRMH must be in the native machine encoding; this is given by the value of ENNAT for the C programming language.

The character set and encoding of the MQRMH must be set into the *MDCSI* and *MDENC* fields in:

- The MQMD (if the MQRMH structure is at the start of the message data), or
- The header structure that precedes the MQRMH structure (all other cases).

Usage: An application puts a message consisting of an MQRMH, but omitting the bulk data. When the message is read from the transmission queue by a message channel agent (MCA), a user-supplied message exit is invoked to process the reference message header. The exit can append to the reference message the bulk data identified by the MQRMH structure, before the MCA sends the message through the channel to the next queue manager.

At the receiving end, a message exit that waits for reference messages should exist. When a reference message is received, the exit should create the object from the bulk data that follows the MQRMH in the message, and then pass on the reference message without the bulk data. The reference message can later be retrieved by an application reading the reference message (without the bulk data) from a queue.

Normally, the MQRMH structure is all that is in the message. However, if the message is on a transmission queue, one or more additional headers will precede the MQRMH structure.

A reference message can also be sent to a distribution list. In this case, the MQDH structure and its related records precede the MQRMH structure when the message is on a transmission queue.

Note: A reference message should not be sent as a segmented message, because the message exit cannot process it correctly.

- "Data conversion"
- "Fields"
- "Initial values" on page 3216
- "RPG declaration" on page 3218

Data conversion

For data conversion purposes, conversion of the MQRMH structure includes conversion of the source environment data, source object name, destination environment data, and destination object name. Any other bytes within *RMLEN* bytes of the start of the structure are either discarded or have undefined values after data conversion. The bulk data will be converted provided that all of the following statements are true:

- The bulk data is present in the message when the data conversion is performed.
- The *RMFMT* field in MQRMH has a value other than FMNONE.
- A user-written data-conversion exit exists with the format name specified.

Be aware, however, that typically the bulk data is not present in the message when the message is on a queue, and that as a result the bulk data will not be converted by the GMCONV option.

Fields

The MQRMH structure contains the following fields; the fields are described in **alphabetical order**:

RMCSI (10-digit signed integer)

Character set identifier of bulk data.

This specifies the character set identifier of the bulk data; it does not apply to character data in the MQRMH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

CSINHT

Inherit character-set identifier of this structure.

Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value CSINHT is not returned by the MQGET call.

CSINHT cannot be used if the value of the *MDPAT* field in MQMD is ATBRKR.

The initial value of this field is CSUNDF.

RMDEL (10-digit signed integer)

Length of destination environment data.

If this field is zero, there is no destination environment data, and *RMDEO* is ignored.

RMDEO (10-digit signed integer)

Offset of destination environment data.

This field specifies the offset of the destination environment data from the start of the MQRMH structure. Destination environment data can be specified by the creator of the reference message, if that data is known to the creator. For example, the destination environment data might be the directory path of the object where the bulk data is to be stored. However, if the creator does not know the destination environment data, it is the responsibility of the user-supplied message exit to determine any environment information needed.

The length of the destination environment data is given by *RMDEL* ; if this length is zero, there is no destination environment data, and *RMDEO* is ignored. If present, the destination environment data must reside completely within *RMLEN* bytes from the start of the structure.

Applications should not assume that the destination environment data is contiguous with any of the data addressed by the *RMSEO*, *RMSNO*, and *RMDNO* fields.

The initial value of this field is 0.

RMDL (10-digit signed integer)

Length of bulk data.

The *RMDL* field specifies the length of the bulk data referenced by the MQRMH structure.

If the bulk data is present in the message, the data begins at an offset of *RMLEN* bytes from the start of the MQRMH structure. The length of the entire message minus *RMLEN* gives the length of the bulk data present.

If data is present in the message, *RMDL* specifies the amount of that data that is relevant. The normal case is for *RMDL* to have the same value as the length of data present in the message.

If the MQRMH structure represents the remaining data in the object (starting from the specified logical offset), the value zero can be used for *RMDL*, if the bulk data is not present in the message.

If no data is present, the end of MQRMH coincides with the end of the message.

The initial value of this field is 0.

RMDNL (10-digit signed integer)

Length of destination object name.

If this field is zero, there is no destination object name, and *RMDNO* is ignored.

RMDNO (10-digit signed integer)

Offset of destination object name.

This field specifies the offset of the destination object name from the start of the MQRMH structure. The destination object name can be specified by the creator of the reference message, if that data is known to the creator. However, if the creator does not know the destination object name, it is the responsibility of the user-supplied message exit to identify the object to be created or modified.

The length of the destination object name is given by *RMDNL* ; if this length is zero, there is no destination object name, and *RMDNO* is ignored. If present, the destination object name must reside completely within *RMLEN* bytes from the start of the structure.

Applications should not assume that the destination object name is contiguous with any of the data addressed by the *RMSEO*, *RMSNO*, and *RMDEO* fields.

The initial value of this field is 0.

RMDO (10-digit signed integer)

Low offset of bulk data.

This field specifies the low offset of the bulk data from the start of the object of which the bulk data forms part. The offset of the bulk data from the start of the object is called the *logical offset*. This is not the physical offset of the bulk data from the start of the MQRMH structure - that offset is given by *RMLEN*.

To allow large objects to be sent using reference messages, the logical offset is divided into two fields, and the actual logical offset is given by the sum of these two fields:

- *RMDO* represents the remainder obtained when the logical offset is divided by 1 000 000 000. It is thus a value in the range 0 through 999 999 999.
- *RMDO2* represents the result obtained when the logical offset is divided by 1 000 000 000. It is thus the number of complete multiples of 1 000 000 000 that exist in the logical offset. The number of multiples is in the range 0 through 999 999 999.

The initial value of this field is 0.

RMDO2 (10-digit signed integer)

High offset of bulk data.

This field specifies the high offset of the bulk data from the start of the object of which the bulk data forms part. It is a value in the range 0 through 999 999 999. See *RMDO* for details.

The initial value of this field is 0.

RMENC (10-digit signed integer)

Numeric encoding of bulk data.

This specifies the numeric encoding of the bulk data; it does not apply to numeric data in the MQRMH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is ENNAT.

RMFLG (10-digit signed integer)

Reference message flags.

The following flags are defined:

RMLAST

Reference message contains or represents last part of object.

This flag indicates that the reference message represents or contains the last part of the referenced object.

RMNLST

Reference message does not contain or represent last part of object.

RMNLST is defined to aid program documentation. It is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

The initial value of this field is RMNLST.

RMFMT (8-byte character string)

Format name of bulk data.

This specifies the format name of the bulk data.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *MDFMT* field in MQMD.

The initial value of this field is FMNONE.

RMLEN (10-digit signed integer)

Total length of MQRMH, including strings at end of fixed fields, but not the bulk data.

The initial value of this field is zero.

RMOII (24-byte bit string)

Object instance identifier.

This field can be used to identify a specific instance of an object. If it is not needed, it should be set to the following value:

OIINON

No object instance identifier specified.

The value is binary zero for the length of the field.

The length of this field is given by LNOIID. The initial value of this field is OIINON.

RMOT (8-byte character string)

Object type.

This is a name that can be used by the message exit to recognize types of reference message that it supports. Consider making the name conform to the same rules as the *RMFMT* field.

The initial value of this field is 8 blanks.

RMSEL (10-digit signed integer)

Length of source environment data.

If this field is zero, there is no source environment data, and *RMSEO* is ignored.

The initial value of this field is 0.

RMSEO (10-digit signed integer)

Offset of source environment data.

This field specifies the offset of the source environment data from the start of the MQRMH structure. Source environment data can be specified by the creator of the reference message, if that data is known to the creator. For example, the source environment data might be the directory path of the object containing the bulk data. However, if the creator does not know the source environment data, it is the responsibility of the user-supplied message exit to determine any environment information needed.

The length of the source environment data is given by *RMSEL* ; if this length is zero, there is no source environment data, and *RMSEO* is ignored. If present, the source environment data must reside completely within *RMLEN* bytes from the start of the structure.

Applications should not assume that the environment data starts immediately after the last fixed field in the structure or that it is contiguous with any of the data addressed by the *RMSNO*, *RMDEO*, and *RMDNO* fields.

The initial value of this field is 0.

RMSID (4-byte character string)

Structure identifier.

The value must be:

RMSIDV
Identifier for reference message header structure.

The initial value of this field is RMSIDV.

RMSNL (10-digit signed integer)

Length of source object name.

If this field is zero, there is no source object name, and *RMSNO* is ignored.

The initial value of this field is 0.

RMSNO (10-digit signed integer)

Offset of source object name.

This field specifies the offset of the source object name from the start of the MQRMH structure. The source object name can be specified by the creator of the reference message, if that data is known to the creator. However, if the creator does not know the source object name, it is the responsibility of the user-supplied message exit to identify the object to be accessed.

The length of the source object name is given by *RMSNL* ; if this length is zero, there is no source object name, and *RMSNO* is ignored. If present, the source object name must reside completely within *RMLEN* bytes from the start of the structure.

Applications should not assume that the source object name is contiguous with any of the data addressed by the *RMSEO*, *RMDEO*, and *RMDNO* fields.

The initial value of this field is 0.

RMVER (10-digit signed integer)

Structure version number.

The value must be:

RMVER1
Version-1 reference message header structure.

The following constant specifies the version number of the current version:

RMVERC
Current version of reference message header structure.

The initial value of this field is RMVER1.

Initial values

Table 353. Initial values of fields in MQRMH

Field name	Name of constant	Value of constant
RMSID	RMSIDV	'RMH¬'
RMVER	RMVER1	1
RMLEN	None	0
RMENC	ENNAT	Depends on environment
RMCSI	CSUNDF	0
RMFMT	FMNONE	Blanks
RMFLG	RMNLST	0
RMOT	None	Blanks
RMOII	OIINON	Nulls
RMSEL	None	0
RMSEO	None	0
RMSNL	None	0
RMSNO	None	0
RMDEL	None	0
RMDEO	None	0
RMDNL	None	0
RMDNO	None	0
RMDL	None	0
RMDO	None	0
RMDO2	None	0
Notes: 1. The symbol ¬ represents a single blank character.		

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D*
D* MQRMH Structure
D*
D* Structure identifier
D  RMSID                  1      4    INZ('RMH ')
D* Structure version number
D  RMVER                  5      8I 0 INZ(1)
D* Total length of MQRMH, includingstrings at end of fixed fields,but not
D* the bulk data
D  RMLEN                  9     12I 0 INZ(0)
D* Numeric encoding of bulk data
D  RMENC                 13     16I 0 INZ(273)
D* Character set identifier of bulkdata
D  RMCSI                 17     20I 0 INZ(0)
D* Format name of bulk data
D  RMFMT                 21     28    INZ('        ')
D* Reference message flags
D  RMFLG                 29     32I 0 INZ(0)
D* Object type
D  RMOT                  33     40    INZ
D* Object instance identifier
D  RMOII                 41     64    INZ(X'00000000000000-
D                                     00000000000000000000-
D                                     000000000000')
D* Length of source environmentdata
D  RMSEL                 65     68I 0 INZ(0)
D* Offset of source environmentdata
```

```
D  RMSEO                 69     72I 0 INZ(0)
D* Length of source object name
D  RMSNL                 73     76I 0 INZ(0)
D* Offset of source object name
D  RMSNO                 77     80I 0 INZ(0)
D* Length of destination environmentdata
D  RMDEL                 81     84I 0 INZ(0)
D* Offset of destination environmentdata
D  RMDEO                 85     88I 0 INZ(0)
D* Length of destination objectname
D  RMDNL                 89     92I 0 INZ(0)
D* Offset of destination objectname
D  RMDNO                 93     96I 0 INZ(0)
D* Length of bulk data
D  RMDL                  97    100I 0 INZ(0)
D* Low offset of bulk data
D  RMDO                 101    104I 0 INZ(0)
D* High offset of bulk data
D  RMDO2                105    108I 0 INZ(0)
```

RPG declaration

MQRR (Response record) on IBM i: 

The MQRR structure is used to receive the completion code and reason code resulting from the open or put operation for a single destination queue, when the destination is a distribution list.

Overview

Purpose: MQRR is an output structure for the MQOPEN, MQPUT, and MQPUT1 calls.

Character set and encoding: Data in MQRR must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT. However, if the application is running as an IBM MQ client, the structure must be in the character set and encoding of the client.

Usage: By providing an array of these structures on the MQOPEN and MQPUT calls, or on the MQPUT1 call, it is possible to determine the completion codes and reason codes for all of the queues in a distribution list when the outcome of the call is mixed, that is, when the call succeeds for some queues in the list but fails for others. Reason code RC2136 from the call indicates that the response records (if provided by the application) have been set by the queue manager.

- "Fields"
- "Initial values" on page 3219
- "RPG declaration" on page 3219

Fields

The MQRR structure contains the following fields; the fields are described in **alphabetical order**:

RRCC (10-digit signed integer)

Completion code for queue.

This is the completion code resulting from the open or put operation for the queue with the name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call.

This is always an output field. The initial value of this field is CCOK.

RRREA (10-digit signed integer)

Reason code for queue.

This is the reason code resulting from the open or put operation for the queue with the name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call.

This is always an output field. The initial value of this field is RCNONE.

Initial values

Table 354. Initial values of fields in MQRR

Field name	Name of constant	Value of constant
RRCC	CCOK	0
RRREA	RCNONE	0

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D*
D* MQRR Structure
D*
D* Completion code for queue
D  RRCC                  1      4I 0 INZ(0)
D* Reason code for queue
D  RRREA                 5      8I 0 INZ(0)
```

MQSCO (TLS configuration options) on IBM i: 

The MQSCO structure (with the TLS fields in the MQCD structure) allows an application running as an IBM MQ MQI client to specify configuration options that control the use of TLS for the client connection when the channel protocol is TCP/IP.

Overview

Purpose: The structure is an input parameter on the MQCONNX call.

If the channel protocol for the client channel is not TCP/IP, the MQSCO structure is ignored.

Character set and encoding: Data in MQSCO must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT.

- "Fields"
- "Initial values" on page 3223
- "RPG declaration" on page 3223

Fields

The MQSCO structure contains the following fields; the fields are described in **alphabetical order**:

SCAIC (10-digit signed integer)

This is the number of authentication information (MQAIR) records addressed by the *SCAIP* or *SCAIO* fields. For more information, see"MQAIR (Authentication information record) on IBM i" on page 3020. The value must be zero or greater. If the value is not valid, the call fails with reason code RC2383.

This is an input field. The initial value of this field is 0.

SCAIO (10-digit signed integer)

This is the offset in bytes of the first authentication information record from the start of the MQSCO structure. The offset can be positive or negative. The field is ignored if *SCAIC* is zero.

You can use either *SCAIO* or *SCAIP* to specify the MQAIR records, but not both; see the description of the *SCAIP* field for details.

This is an input field. The initial value of this field is 0.

SCAIP (10-digit signed integer)

This is the address of the first authentication information record. The field is ignored if *SCAIC* is zero.

You can provide the array of MQAIR records in one of two ways:

- By using the pointer field *SCAIP*

  In this case, the application can declare an array of MQAIR records that is separate from the MQSCO structure, and set *SCAIP* to the address of the array.

  Consider using *SCAIP* for programming languages that support the pointer data type in a fashion that is portable to different environments (for example, the C programming language).

- By using the offset field *SCAIO*

  In this case, the application must declare a compound structure containing an MQSCO followed by the array of MQAIR records, and set *SCAIO* to the offset of the first record in the array from the start of the MQSCO structure. Ensure that this value is correct, and has a value that can be accommodated within an MQLONG (the most restrictive programming language is COBOL, for which the valid range is -999 999 999 through +999 999 999).

  Consider using *SCAIO* for programming languages that do not support the pointer data type, or that implement the pointer data type in a fashion that is not portable to different environments (for example, the COBOL programming language).

Whatever technique you choose, only one of *SCAIP* and *SCAIO* can be used; the call fails with reason code RC2384 if both are nonzero.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

SCCERLBL (10-digit signed integer)

This field gives details of the certificate label being used.

IBM MQ initializes the value for the SCCERLBL field as blanks. Either enter the required value, or accept the default value.

`ibmwebspheremq`*`user_id`* is a valid value for this field for all versions of the product, and for MQSCO versions less than 5.0 it is the only valid value. Therefore the value of this field is interpreted at run time, and changed if necessary. If you specify an MQSCO version less than 5.0, or accept the default value of blanks for the SCCERLBL field, the system uses the value `ibmwebspheremq`*`user_id`*.

This is an input field.

SCCERTVPOL (10-digit signed integer)

This field specifies what type of certificate validation policy is used. The field can be set to one of the following values:

MQ_CERT_VAL_POLICY_ANY
: Apply each of the certificate validation policies supported by the secure sockets library. Accept the certificate chain if any of the policies considers the certificate chain valid.

MQ_CERT_VAL_POLICY_RFC5280
Apply only the RFC5280 compliant certificate validation policy. This setting provides stricter validation than the ANY setting, but rejects some older digital certificates.

The initial value of this field is MQ_CERT_VAL_POLICY_ANY

SCCH (10-digit signed integer)

This field gives configuration details for cryptographic hardware connected to the client system.

Set the field to a string in the following format, or leave it blank or null:

```
GSK_PKCS11=the PKCS #11 driver path and file name;the PKCS #11
token label;the PKCS #11 token password;symmetric cipher setting>;
```

To use cryptographic hardware which conforms to the PKCS11 interface, for example, the IBM 4960 or IBM 4963, specify the PKCS11 driver path, PKCS11 token label, and PKCS11 token password strings, each terminated by a semi-colon.

The PKCS #11 driver path is an absolute path to the shared library providing support for the PKCS #11 card. The PKCS #11 driver file name is the name of the shared library. An example of the value required for the PKCS #11 path and file name is:

```
/usr/lib/pkcs11/PKCS11_API.so
```

The PKCS #11 token label must be entirely in lowercase. If you have configured your hardware with a mixed case or uppercase token label, reconfigure it with this lowercase label.

If no cryptographic hardware configuration is required, set the field to blank or null.

If the value is shorter than the length of the field, terminate the value with a null character, or pad it with blanks to the length of the field. If the value is not valid, or leads to a failure when used to configure the cryptographic hardware, the call fails with reason code RC2382.

This is an input field. The length of this field is given by LNSSCH. The initial value of this field is blank characters.

SCEPSUITEB (10-digit signed integer)

This field Specifies whether Suite B compliant cryptography is used and what level of strength is employed. The value can be one or more of:

- SCEPSUITEB0

  Suite B compliant cryptography is not used.
- SCEPSUITEB1

  Suite B 128-bit strength security is used.
- SCEPSUITEB2

  Suite B 192-bit strength security is used.

Note: Using SCEPSUITEB0 with any other value in this field is invalid.

SCFR (10-digit signed integer)

IBM MQ can be configured with cryptographic hardware so that the cryptography modules used are those provided by the hardware product; these can be FIPS-certified to a particular level depending on the cryptographic hardware product in use.

Use this field to specify that only FIPS-certified algorithms are used if the cryptography is provided in IBM MQ-provided software.

When IBM MQ is installed an implementation of TLS cryptography is also installed which provides some FIPS-certified modules.

The values can be:

MQSSL_FIPS_NO

This is the default value. When set to this value:

- Any CipherSpec supported on a particular platform can be used.
- If run without use of cryptographic hardware, the following CipherSpecs run using FIPS 140-2 certified cryptography on the IBM MQ platforms:
  - TLS_RSA_WITH_3DES_EDE_CBC_SHA
  - TLS_RSA_WITH_AES_128_CBC_SHA
  - TLS_RSA_WITH_AES_256_CBC_SHA

MQSSL_FIPS_YES

When set to this value, unless you are using cryptographic hardware to perform the cryptography, you can be sure that

- Only FIPS-certified cryptographic algorithms can be used in the CipherSpec applying to this client connection.
- Inbound and outbound TLS channel connections only succeed if one of the following Cipher Specs are used:
  - TLS_RSA_WITH_3DES_EDE_CBC_SHA
  - TLS_RSA_WITH_AES_128_CBC_SHA
  - TLS_RSA_WITH_AES_256_CBC_SHA

Notes:

1. CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA is deprecated.
2. Where possible, if FIPS-only CipherSpecs is configured then the MQI client rejects connections which specify a non-FIPS CipherSpec withRC2393. IBM MQ does not guarantee to reject all such connections and it is your responsibility to determine whether your IBM MQ configuration is FIPS-compliant.

SCKR (10-digit signed integer)

This field is relevant only for IBM MQ MQI clients running on UNIX and Windows systems. It specifies the location of the key database file in which keys and certificates are stored. The key database file must have a file name of the form `zzz.kdb`, where `zzz` is user-selectable. The *SCKR* field contains the path to this file, along with the file name stem (all characters in the file name up to but not including the final `.kdb`). The `.kdb` file suffix is added automatically.

Each key database file has an associated *password stash file*. This holds encrypted passwords that are used to allow programmatic access to the key database. The password stash file must reside in the same directory and have the same file stem as the key database, and must end with the suffix `.sth`.

For example, if the *SCKR* field has the value `/xxx/yyy/key`, the key database file must be `/xxx/yyy/key.kdb`, and the password stash file must be `/xxx/yyy/key.sth`, where `xxx` and `yyy` represent directory names.

If the value is shorter than the length of the field, terminate the value with a null character, or pad it with blanks to the length of the field. The value is not checked; if there is an error in accessing the key repository, the call fails with reason code RC2381.

To run a TLS connection from an IBM MQ MQI client, set *SCKR* to a valid key database file name.

This is an input field. The length of this field is given by LNSSKR. The initial value of this field is a blank character.

SCSID (10-digit signed integer)

This is the structure identifier; the value must be:

SCSIDV
Identifier for TLS configuration options structure.

This is always an input field. The initial value of this field is SCSIDV.

SCVER (10-digit signed integer)

This is the structure version number; the value must be:

SCVER1
Version-1 TLS configuration options structure.

SCVER2
Version-2 TLS configuration options structure.

The following constant specifies the version number of the current version:

SCVERC
Current version of TLS configuration options structure.

This is always an input field. The initial value of this field is SCVER2

Initial values

Table 355. Initial values of fields in MQSCO

Field name	Name of constant	Value of constant
SCSID	SCSIDV	'SCO¬'
SCVER	SCVER5	1
SCKR	None	Null string or blanks
SCCH	None	Null string or blanks
SCAIC	None	0
SCAIO	None	0
SCAIP	None	Null pointer or null bytes
SCKRC	None	Null pointer or null bytes
SCFR	None	Null pointer or null bytes
SCEPSUITEB	None	Null pointer or null bytes
SCCERTVPOL	None	Null pointer or null bytes
SCCERLBL	None	Null pointer or null bytes

Notes:

1. The symbol ¬ represents a single blank character.
2. See"RPG declaration"for the SCEPSUITEB options.

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQSCO Structure
D*
D* Structure identifier
D  SCSID                  1      4    INZ('SCO ')
D* Structure version number
D  SCVER                  5      8I 0 INZ(1)
D* Location of TLS key repository
D  SCKR                   9    264    INZ
D* Cryptographic hardware configuration string
D  SCCH                 265    520    INZ
D* Number of MQAIR records present
D  SCAIC                521    524I 0 INZ(0)
```

```
D* Offset of first MQAIR record from start of MQSCO structure
D  SCAIO                525    528I 0 INZ(0)
D* Address of first MQAIR record
D SCAIP                 529    544*   INZ(*NULL)
D* Ver:1 **
D* Number of unencrypted bytes sent/received before secret key is
D* reset
D SCKRC                 545    548I 0 INZ(0)
D* Using FIPS-certified algorithms
D SCFR                  549    552I 0 INZ(0)
D* Ver:2 **
* Use only Suite B cryptographic algorithms
D SCEPSUITEB0
D SCEPSUITEB1           553    556I 0 INZ(1)
D SCEPSUITEB2           557    560I 0 INZ(0)
D SCEPSUITEB3           561    564I 0 INZ(0)
D SCEPSUITEB4           565    568I 0 INZ(0)
D SCEPSUITEB                    10I 0 DIM(4) OVERLAY(SCEPSUITEB0)
D* Ver:3 **
D* Certificate validation policy
D SCCERTVPOL            569    572I 0 INZ(0)
D* Ver:4 **
```

MQSD (Subscription descriptor) on IBM i: IBM i

The MQSD structure is used to specify details about the subscription being made.

Overview

Purpose

The structure is an input/output parameter on the MQSUB call.

Managed subscriptions

If an application has no specific need to use a particular queue as the destination for those publications that match its subscription, it can use the managed subscription feature. If an application elects to use a managed subscription, the queue manager informs the subscriber about the destination where published messages are sent, by providing an object handle as an output from the MQSUB call. For more information, see HOBJ (10-digit signed integer) - input/output.

When the subscription is removed, the queue manager also undertakes to clean up messages that have not been retrieved from the managed destination, in the following situations:

- When the subscription is removed - by use of MQCLOSE with CORMSB - and the managed Hobj is closed.
- By implicit means when the connection is lost to an application using a non-durable subscription (SONDUR)
- By expiration when a subscription is removed because it has expired and the managed Hobj is closed.

You must use managed subscriptions with non-durable subscriptions, so that the clean up can occur, and so that messages for closed non-durable subscriptions do not take up space in your queue manager. Durable subscriptions can also use managed destinations.

Character set and encoding

Data in MQSD must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT. However, if the application is running as an IBM MQ client, the structure must be in the character set and encoding of the client.

- "Fields" on page 3225
- "Initial values" on page 3237
- "RPG declaration" on page 3237

Fields

The MQSD structure contains the following fields; the fields are described in alphabetical order:

SDAID (32 byte character string)

This value is in the *MDAID* field of the Message Descriptor (MQMD) of all publication messages matching this subscription. *SDAID* is part of the identity context of the message. For more information about message context, see Message context.

For more information about *MDAID* see MDAID.

If the SOSETI option is not specified, the *MDAID* which is set in each message published for this subscription is blanks, as default context information.

If the SOSETI option is specified, the *SDAID* is being generated by the user and this field is an input field which contains the *MDAID* to be set in each publication for this subscription.

The length of this field is given by LNAIDD. The initial value of this field is 32 blank characters.

If altering an existing subscription using the SOALT option, the *SDAID* of any future publication messages can be changed.

On return from an MQSUB call using SORES, this field is set to the current *MDAID* being used for the subscription.

SDACC (32 byte character string)

This value is in the *MDACC* field of the Message Descriptor (MQMD) of all publication messages matching this subscription. *MDACC* is part of the identity context of the message. For more information about message context, see Message context.

For more information about *MDACC* see MDACC.

You can use the following special value for the *SDACC* field:

ACNONE
No accounting token is specified.

The value is binary zero for the length of the field.

If the SOSETI option is not specified, the accounting token is generated by the queue manager as default context information and this field is an output field which contains the *MDACC* which is set in each message published for this subscription.

If the SOSETI option is specified, the accounting token is being generated by the user and this field is an input field which contains the *MDACC* to be set in each publication for this subscription.

The length of this field is given by LNACCT. The initial value of this field is ACNONE.

If altering an existing subscription using the SOALT option, the value of *MDACC* in any future publication messages can be changed.

On return from an MQSUB call using SORES, this field is set to the current *MDACC* being used for the subscription.

SDASI (40 byte bit string)

This is a security identifier that is passed with the *SDAU* to the authorization service to allow appropriate authorization checks to be performed.

SDASI is used only if SOALTU is specified, and the *SDAU* field is not entirely blank up to the first null character or the end of the field.

On return from an MQSUB call using SORES, this field is unchanged.

See the description of ODASI in the MQOD data type for more information.

SDAU (12 byte character string)

If you specify SOALTU, this field contains an alternate user identifier that is used to check the authorization for the subscription and for output to the destination queue (specified in the **Hobj** parameter of the MQSUB call), in place of the user identifier that the application is currently running under.

If successful, the user identifier specified in this field is recorded as the subscription owning user identifier in place of the user identifier that the application is currently running under.

If SOALTU is specified and this field is entirely blank up to the first null character or the end of the field, the subscription can succeed only if no user authorization is needed to subscribe to this topic with the options specified or the destination queue for output.

If SOALTU is not specified, this field is ignored.

On return from an MQSUB call using SORES, this field is unchanged.

This is an input field. The length of this field is given by LNUID. The initial value of this field is 12 blank characters.

SDCID (24 byte bit string)

All publications sent to match this subscription contain this correlation identifier in the message descriptor. If multiple subscriptions use the same queue to get their publications from, using MQGET by correlation ID allows only publications for a specific subscription to be obtained. This correlation identifier can either be generated by the queue manager or by the user.

If the SOSCID option is not specified, the correlation identifier is generated by the queue manager and this field is an output field which contains the correlation identifier which is set in each message published for this subscription.

If the SOSCID option is specified, the correlation identifier is being generated by the user and this field is an input field which contains the correlation identifier to be set in each publication for this subscription. In this case, if the field contains CINONE, the correlation identifier which is set in each message published for this subscription is the correlation identifier that was created by the original put of the message.

If the SOGRP option is specified and the correlation identifier specified is the same as an existing grouped subscription using the same queue and an overlapping topic string, only the most significant subscription in the group is provided with a copy of the publication.

The length of this field is given by LNCID. The initial value of this field is CINONE.

If altering an existing subscription using the SOALT option, and this field is an input field, then the subscription correlation ID can be changed, unless the subscription has been created using the SOGRP option.

On return from an MQSUB call using SORES, this field is set to the current correlation ID for the subscription.

SDEXP (10 digit signed integer)

This is the time expressed in tenths of a second after which the subscription expires. No more publications will match this subscription after this interval has passed. This is also used as the value in the *MDEXP* field in the MQMD of the publications sent to this subscriber.

The following special value is recognized:

EIULIM
The subscription has an unlimited expiration time.

If altering an existing subscription using the SOALT option, the expiry of the subscription can be changed.

On return from an MQSUB call using the SORES option this field is set to the original expiry of the subscription and not the remaining expiry time.

SDON (48 byte character string)

This is the name of the topic object as defined on the local queue manager.

The name can contain the following characters:

- Uppercase alphabetic characters (A through Z)
- Lowercase alphabetic characters (a through z)
- Numeric digits (0 through 9)
- Period (.), forward slash (/), underscore (_), percent (%)

The name must not contain leading or embedded blanks, but can contain trailing blanks. Use a null character to indicate the end of significant data in the name; the null and any characters following it are treated as blanks. The following restrictions apply:

- On systems that use EBCDIC Katakana, lowercase characters cannot be used.
- Names containing lowercase characters, forward slash, or percent, must be enclosed in quotation marks when specified on commands. These quotation marks must not be specified for names that occur as fields in structures or as parameters on calls.

The *SDON* is used to form the Full topic name.

The full topic name can be built from two different fields: *SDON* and *SDOS*. For details of how these two fields are used, see "Using topic strings" on page 2566.

On return from an MQSUB call using the SORES option this field is unchanged.

The length of this field is given by LNTOPN. The initial value of this field is 48 blank characters.

If altering an existing subscription using the SDALT option, the name of the topic object subscribed to cannot be changed. This field and *SDOS* can be omitted. If they are provided they must resolve to the same full topic name or the call fails with RC2510 .

SDOPT (10 digit signed integer)

You must specify at least one of the following options:

- SOALT
- SORES
- SOCRT

The values can be added. Do not add the same constant more than once. The table shows how you can combine these options: Combinations that are not valid are noted; any other combinations are valid.

Access or creation options

Access and creation options control whether a subscription is created, or whether an existing subscription is returned or altered. You must specify at least one of these options. The table displays valid combinations of access or creation options.

Combination of options	Notes
SOCRT	Creates a subscription if one does not exist; fails if the subscription exists.
SORES	Resumes an existing subscription, fails if no subscription exists.
SOCRT + SORES	Creates a subscription if one does not exist and resumes a matching one, if it does exist. Useful combination if used in an application that might be run a number of times.
SORES + SOALT (see note)	Resumes an existing subscription, altering any fields to match those specified in the MQSD, fails if no subscription exists.
SOCRT + SOALT (see note)	Creates a subscription if one does not exist and resumes a matching one, if it does exist, altering any fields to match those specified in the MQSD. Useful combination if used in an application that wants to ensure that its subscription is in a certain state before proceeding.

Note:

Options specifying SOALT can also specify SORES, but this combination has no additional effect to specifying SOALT alone. SOALT implies SORES, because calling MQSUB to alter a subscription implies that the subscriptions are also resumed. The opposite is not true, however: resuming a subscription does not imply it is to be altered.

SOCRT

Create a subscription for the topic specified. If a subscription using the same *SDSN* exists, the call fails with RC2432 . This failure can be avoided by combining the SOCRT option with SORES. The *SDSN* is not always necessary. For more details, see the description of that field.

Combining SOCRT with SORES first checks whether there is an existing subscription for the specified *SDSN*, and if there is returns a handle to that preexisting subscription; but if there is no existing subscription, a new one is created using all the fields provided in the MQSD.

SOCRT can also be combined with SOALT to similar effect (see details about SOALT later in this topic).

SORES

Return a handle to a preexisting subscription which matches those specified by *SDSN*. No changes are made to the matching subscription attributes, and they are returned on output in the MQSD structure. Most of the contents of the MQSD are not used: The fields used are *SDSID*, *SDVER*, *SDOPT*, *SDAID* and *SDASI*, and *SDSN*.

The call fails with reason code RC2428 if a subscription does not exist matching the full subscription name. This failure can be avoided by combining the SOCRT option with SORES. For details about SOCRT, see SOCRT.

The user ID of the subscription is the user ID that created the subscription, or if it has been later altered by a different user ID, it is the user ID of the most recent, successful alteration. If an *SDAID* is used, and use of alternate user IDs is allowed for that user, *SDAID* is recorded as the user ID that created the subscription instead of the user ID under which the subscription was made.

The user ID that created the subscription is recorded as *SDAU* if that field is used, and the use of alternate user IDs is allowed for that user.

If a matching subscription exists which was created without the SOAUID option and the user ID of the subscription is different from that of the application requesting a handle to the subscription, the call fails with reason code RC2434 .

If a matching subscription exists and is currently in use by another application, the call fails with reason code RC2429 . If it is currently in use by the same connection, the call does not fail and a handle to the subscription is returned.

If the subscription named in SubName is not a valid subscription to resume or alter from an application, the call fails with RC2523 .

SORES is implied by SOALT and so is not required to be combined with that option, however, it is not an error if those two options are combined.

SOALT

Return a handle to a preexisting subscription with the full subscription name matching those specified in *SDSN*. Any attributes of the subscription that are different from those specified in the MQSD is altered in the subscription unless alteration is disallowed for that attribute. Details are noted in the description of each attribute and are summarized in the following table. If you try to alter an attribute that cannot be changed, the call fails with the reason code shown in the following table.

The call fails with reason code RC2428 if a subscription does not exist matching the full subscription name. This failure can be avoided by combining the SOCRT option with SOALT.

Combining SOCRT with SOALT first checks whether there is an existing subscription for the specified full subscription name, and if there is returns a handle to that preexisting subscription with alterations made as previously detailed; but if there is no existing subscription, a new one is created using all the fields provided in the MQSD.

The user ID of the subscription is the user ID that created the subscription, or if it has been later altered by a different user ID, it is the user ID of the most recent successful alteration. If *SDAU* is used (and use of alternate user IDs is allowed for that user), then the alternate user ID is recorded as the user ID that created the subscription instead of the user ID under which the subscription was made.

If a matching subscription exists that was created without the option SOAUID and the user ID of the subscription is different from that of the application requesting a handle to the subscription, the call fails with reason code RC2434 .

If a matching subscription exists and is currently in use by another application, the call fails with RC2429 . If it is currently in use by the same connection the call does not fail and a handle to the subscription is returned.

If the subscription named in SubName is not a valid subscription to resume or alter from an application, the call fails with RC2523 .

The following tables show the subscription attributes that can be altered by SOALT.

Data type descriptor or function call	Field name	Can this attribute be altered using SOALT?	Reason code
MQSD	Durability options	No	RC2509
MQSD	Destination Options	Yes	None
MQSD	Registration options	Yes (see note 1)	RC2515 if you try to alter SOGRP
MQSD	Publication options	Yes (see note 2)	None
MQSD	Wildcard options	No	RC2510
MQSD	Other options	No (see note 3)	None

Data type descriptor or function call	Field name	Can this attribute be altered using SOALT?	Reason code
MQSD	ObjectName	No	RC2510
MQSD	SDAU	No (see note 4)	None
MQSD	SDASI	No (see note 4)	None
MQSD	SDEXP	Yes	None
MQSD	SDOS	No	RC2510
MQSD	SDSN	No (see note 5)	None
MQSD	SDSUD	Yes	None
MQSD	SDCID	Yes (see note 6)	RC2515 when in a grouped subscription
MQSD	SDPRI	Yes	None
MQSD	SDACC	Yes	None
MQSD	SDAID	Yes	None
MQSD	SDSL	No	RC2512
MQSUB	Hobj	Yes (see note 6)	RC2515 when in a grouped subscription

Notes:

1. SOGRP cannot be altered.
2. SONEWP cannot be altered because it is not part of the subscription
3. These options are not part of the subscription
4. This attribute is not part of the subscription
5. This attribute is the identity of the subscription being altered
6. Alterable except when part of a grouped sub (SOGRP)

Durability options: The following options control how durable the subscription is. You can specify only one of these options. If you are altering an existing subscription using the SOALT option, you cannot change the durability of the subscription. On return from an MQSUB call using SORES, the appropriate durability option is set.

SODUR

Request that the subscription to this topic remains until it is explicitly removed using MQCLOSE with the CORMSB option. If this subscription is not explicitly removed it will remain even after this application connects to the queue manager is closed.

If a durable subscription is requested to a topic that is defined as not allowing durable subscriptions, the call fails with RC2436 .

SONDUR

Request that the subscription to this topic is removed when the application connection to the queue manager is closed, if it has not already been explicitly removed. SONDUR is the opposite of the SODUR option, and is defined to aid program documentation. It is the default if neither is specified.

Destination options: The following options control the destination that publications for a topic that has been subscribed to are sent to. If altering an existing subscription using the SOALT option, the destination used for publications for the subscription can be changed. On return from an MQSUB call using SORES, this option is set if appropriate.

SOMAN

Request that the destination that the publications are sent to is managed by the queue manager.

The object handle returned in *HOBJ* represents a queue manager managed queue, and is for use with subsequent MQGET, MQCB, MQINQ, or MQCLOSE calls.

An object handle returned from a previous MQSUB call cannot be provided in the **Hobj** parameter when SOMAN is not specified.

Registration options: The following options control the details of the registration that is made to the queue manager for this subscription. If altering an existing subscription using the SOALT option, these registration options can be changed. On return from an MQSUB call using SORES the appropriate registration options is set.

SOGRP

This subscription is grouped with other subscriptions of the same *SDSL* using the same queue and specifying the same correlation ID so that any publications to topics that would cause more than one publication message to be provided to the group of subscriptions, due to an overlapping set of topic strings being used, only causes one message to be delivered to the queue. If this option is not used, then each unique subscription (identified by *SDSN*) that matches is provided with a copy of the publication which might mean that more than one copy of the publication might be placed on the queue shared by a number of subscriptions.

Only the most significant subscription in the group is provided with a copy of the publication. The most significant subscription is based on the Full topic name up to the point where a wildcard is found. If a mixture of wildcard schemes is used within the group, only the position of the wildcard is important. You are advised not to combine different wildcard schemes within a group of subscriptions that share the same queue.

When creating a new grouped subscription it must still have a unique *SDSN*, but if it matches the full topic name of an existing subscription in the group, the call fails with RC2514 .

If the most significant subscription in group also specifies SONOLC and this is a publication from the same application, then no publication is delivered to the queue.

When altering a subscription made with this option, the fields which imply the grouping, *Hobj* on the MQSUB call (representing the queue and queue manager name), and the *SDCID* cannot be changed. Attempting to alter them causes the call to fail with RC2515 .

This option must be combined with SOSCID with a *SDCID* that is not set to CINONE, and cannot be combined with SOMAN.

SOAUID

When SOAUID is specified, the identity of the subscriber is not restricted to a single user ID. This allows any user to alter or resume the subscription when they have suitable authority. Only a single user can have the subscription at any one time. An attempt to resume use of a subscription currently in use by another application causes the call to fail with RC2429 .

To add this option to an existing subscription, the MQSUB call, using SOALT, must come from the same user ID as the original subscription itself.

If an MQSUB call references an existing subscription with SOAUID set, and the user ID differs from the original subscription, the call succeeds only if the new user ID has authority to subscribe to the topic. On successful completion, future publications to this subscriber are put to the subscriber's queue with the new user ID set in the publication message.

Do not specify both SOAUID and SOFUID. If neither is specified, the default is SOFUID.

SOFUID

When SOFUID is specified, the subscription can be altered or resumed by only the last user ID to alter the subscription. If the subscription has not been altered, it is the user ID that created the subscription.

If an MQSUB verb references an existing subscription with SOAUID set and alters the subscription using SOALT to use the SOFUID, the user ID of the subscription is now fixed at this new user ID. The call succeeds only if the new user ID has authority to subscribe to the topic.

If a user ID other than the one recorded as owning a subscription tries to resume or alter an SOFUID subscription, the call fails with RC2434 . The owning user ID of a subscription can be viewed using the **DISPLAY SBSTATUS** command.

Do not specify both SOAUID and SOFUID. If neither is specified, the default is SOFUID.

Publication options: The following options control the way publications are sent to this subscriber. If altering an existing subscription using the SOALT option, these publication options can be changed.

SONOLC

Tells the broker that the application does not want to see any of its own publications. Publications are considered to have originated from the same application if the connection handles are the same. On return from an MQSUB call using SORES this option is set if appropriate.

SONEWP

No currently retained publications are to be sent, when this subscription is created, only new publications. This option only applies when SOCRE is specified. Any subsequent changes to a subscription do not alter the flow of publications and so any publications that have been retained on a topic, has already been sent to the subscriber as new publications.

If this option is specified without SOCRE it causes the call to fail with RC2046 . On return from an MQSUB call using SORES this option is not set even if the subscription was created using this option.

If this option is not used, previously retained messages are sent to the destination queue provided. If this action fails due to an error, either RC2525 or RC2526, the creation of the subscription fails.

This option is not valid in combination with SOPUBR.

SOPUBR

Setting this option indicates that the subscriber requests information specifically when required. The queue manager does not send unsolicited messages to the subscriber. The retained publication (or possibly multiple publications if a wildcard is specified in the topic) is sent to the subscriber each time an MQSUBRQ call is made using the Hsub handle from a previous MQSUB call. No publications are sent as a result of the MQSUB call using this option. On return from an MQSUB call using SORES this option is set if appropriate.

This option is not valid in combination with SONEWP.

Wildcard options: The following options control how wildcards are interpreted in the string provided in the *SDOS* field of the MQSD. You can specify only one of these options. If altering an existing subscription using the SOALT option, these wildcard options cannot be changed. On return from an MQSUB call using SORES the appropriate wildcard option is set.

SOWCHR

Wildcards only operate on characters within the topic string. The SOWCHR field treats forward slash (/) as just another character with no special significance.

The behavior defined by SOWCHR is shown in the following table:

Special Character	Behavior
*	Wildcard, zero or more characters
?	Wildcard, one character
%	Escape character to allow the characters '*', '?', or '%' to be used in a string and not be interpreted as a special character, for example, '%*', '%?' or '%%'.

For example, publishing on the following topic:

```
/level0/level1/level2/level3/level4
```

matches subscribers using the following topics:

```
*
/*
/ level0/level1/level2/level3/*
/ level0/level1/*/level3/level4
/ level0/level1/le?el2/level3/level4
```

Note: This use of wildcards supplies exactly the meaning provided in IBM MQ V6 and WebSphere MB V6 when using MQRFH1 formatted messages for Publish/Subscribe. It is recommended that this is not used for newly written applications and is only used for applications that were previously running against that version and have not been changed to use the default wildcard behavior as described in SOWTOP.

SOWTOP

Wildcards only operate on topic elements within the topic string. This is the default behavior if none is chosen.

The behavior required by SOWTOP is shown in the following table:

Special Character	Behavior
/	Topic level separator
#	Wildcard: multiple topic level
+	Wildcard: single topic level

Note:

The '+' and '#' are not treated as wildcards if they are mixed in with other characters (including themselves) within a topic level. In the following string, the '#' and '+' characters are treated as ordinary characters.

```
level0/level1/#+/level3/level#
```

For example, publishing on the following topic:

```
/level0/level1/level2/level3/level4
```

matches subscribers using the following topics:

```
#
/#
/ level0/level1/level2/level3/#
/ level0/level1/+/level3/level4
```

Note: This use of wildcards supplies the meaning provided in WebSphere Message Broker Version 6 when using MQRFH2 formatted messages for Publish/Subscribe.

Other options: The following options control the way the API call is issued rather than the subscription. On return from an MQSUB call using SORES these options are unchanged.

SOALTU
The SDAU field contains a user identifier to use to validate this MQSUB call. The call can succeed only if this SDAU is authorized to open the object with the specified access options, regardless of whether the user identifier under which the application is running is authorized to do so.

SOSCID
The subscription is to use the correlation identifier supplied in the *SDCID* field. If this option is not specified, a correlation identifier is automatically created by the queue manager at subscription time and is returned to the application in the *SDCID* field. See SDCID (24-byte bit string)SDCID for more information.

SOSETI

The subscription is to use the accounting token and application identity data supplied in the *SDACC* and *SDAID* fields.

If this option is specified, the same authorization check is carried out as if the destination queue was accessed using an MQOPEN call with OOSETI, except in the case where the SOMAN option is also used in which case there is no authorization check on the destination queue.

If this option is not specified, the publications sent to this subscriber has default context information associated with them as follows:

Field in MQMD	Value used
MDUID	The user ID associated with the subscription at the time the subscription was made.
MDACC	Determined from the environment if possible; Set to ACNONE if not.
MDAID	Set to blanks

This option is only valid with SOCRE and SOALT. If used with SORES, the *SDACC* and *SDAID* fields are ignored, so this option has no effect.

If a subscription is altered without using this option where previously the subscription had supplied identity context information, default context information is generated for the altered subscription.

If a subscription allowing different user IDs to use it with option SOAUID, is resumed by a different user ID, default identity context is generated for the new user ID now owning the subscription and any subsequent publications are delivered containing the new identity context.

SOFIQ
The MQSUB call fails if the queue manager is in quiescing state. On z/OS, for a CICS or IMS application, this option also forces the MQSUB call to fail if the connection is in quiescing state.

SDAU (12 byte character string)

If you specify SOALTU, this field contains an alternate user identifier that is used to check the authorization for the subscription and for output to the destination queue (specified in the **Hobj** parameter of the MQSUB call), in place of the user identifier that the application is currently running under.

If successful, the user identifier specified in this field is recorded as the subscription owning user identifier in place of the user identifier that the application is currently running under.

If SOALTU is specified and this field is entirely blank up to the first null character or the end of the field, the subscription can succeed only if no user authorization is must subscribe to this topic with the options specified or the destination queue for output.

If SOALTU is not specified, this field is ignored.

On return from an MQSUB call using SORES, this field is unchanged.

This is an input field. The length of this field is given by LNUID. The initial value of this field is 12 blank characters.

SDPRI (10 digit signed integer)

This is the value that is in the *MQPRI* field of the Message Descriptor (MQMD) of all publication messages matching this subscription. For more information about the *MQPRI* field in the MQMD, see MDPRI.

The value must be greater than or equal to zero; zero is the lowest priority. The following special values can also be used:

PRQDEF

When a subscription queue is provided in the Hobj field in the MQSUB call, and is not a managed handle, then the priority for the message is taken from the **DefPriority** attribute of this queue. If the queue so identified is a cluster queue or there is more than one definition in the queue-name resolution path then the priority is determined when the publication message is put to the queue as described for MDPRI.

If the MQSUB call uses a managed handle, the priority for the message is taken from the **DefPriority** attribute of the model queue associated with the topic subscribed to.

PRPUB

The priority for the message is the priority of the original publication. This is the initial value of the field.

If altering an existing subscription using the SOALT option, the *MQPRI* of any future publication messages can be changed.

On return from an MQSUB call using SORES, this field is set to the current priority being used for the subscription.

SDRO (MQCHARV)

SDRO is the long object name after the queue manager resolves the name provided in *SDON*.

If the long object name is provided in *SDOS* and nothing is provided in *SDON*, the value returned in this field is the same as provided in *SDOS*.

If this field is omitted (that is SDRO.VSBufSize is zero), the *SDRO* is not returned, but the length is returned in SDRO.VSLength. If the length is shorter than the full *SDRO*, it is truncated and returns as many of the rightmost characters as can fit in the provided length.

If *SDRO* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code RC2520 .

SDSID (4 byte character string)

This is the structure identifier; the value must be:

SDSIDV

Identifier for Subscription Descriptor structure.

This is always an input field. The initial value of this field is SDSIDV

SDSL (10 digit signed integer)

This is the level associated with the subscription. Publications are only delivered to this subscription if it is in the set of subscriptions with the highest *SDSL* value less than or equal to the PubLevel used at publication time.

The value must be in the range zero to 9. Zero is the lowest level.

The initial value of this field is 1.

If altering an existing subscription using the SOALT option, then *SDSL* cannot be changed.

SDSN (MQCHARV)

SDSN specifies the subscription name.

This field is required only if *SDOPT* specifies the SODUR option, but if it is provided it is used by the queue manager for SONDUR as well. If specified, *SDSN* must be unique within the queue manager, because it is the field used to identify subscriptions.

The maximum length of *SDSN* is 10240.

This field serves two purposes. For a SODUR subscription, it is the means by which you identify a subscription to resume it after it has been created, if you have either closed the handle to the subscription (using the COKPSB option) or have been disconnected from the queue manager. Identifying a subscription to remove it after it has been created is done using the MQSUB call with the SORES option. The SDSN field is also displayed in the administration view of subscriptions in the *SDSN* field in DISPLAY SBSTATUS.

If *SDSN* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, or if it is omitted when it is required (that is *SDSN*. *VCHRL* is zero), or if it exceeds the maximum length, the call fails with reason code RC2440 .

This is an input field. The initial values of the fields in this structure are the same as those in the MQCHARV structure.

If altering an existing subscription using the SOALT option, the subscription name cannot be changed, because it is the field used to identify the subscription. It is not changed on output from an MQSUB call with the SORES option.

SDSS (MQCHARV)

SDSS is the string that provides the selection criteria used when subscribing for messages from a topic.

This variable length field is returned on output from an MQSUB call using the SORES option, if a buffer is provided, and if there is also a positive buffer length in VSBufSize. If no buffer is provided on the call, only the length of the selection string is returned in the VSLength field of the MQCHARV. If the buffer provided is smaller than the space required to return the field, only VSBufSize bytes are returned in the provided buffer.

If *SDSS* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code RC2519 .

SDSUD (MQCHARV)

The data provided on the subscription in this field is included as the mq.SubUserData message property of every publication sent to this subscription.

The maximum length of *SDSUD* is 10240.

If *SDSUD* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code RC2431.

This is an input field. The initial values of the fields in this structure are the same as those in the MQCHARV structure.

If altering an existing subscription using the SOALT option, the subscription user data can be changed.

This variable length field is returned on output from an MQSUB call using the SORES option, if a buffer is provided and there is a positive buffer length in *VSBufLen*. If no buffer is provided on the call, only the length of the subscription user data is returned in the *VCHRL* field of the MQCHARV. If the buffer provided is smaller than the space required to return the field, only *VSBufLen* bytes are returned in the provided buffer.

SDVER (10 digit signed integer)

This is the structure version number; the value must be:

SDVER1
Version-1 Subscription Descriptor structure.

The following constant specifies the version number of the current version:

SDVERC
Current version of Subscription Descriptor structure.

This is always an input field. The initial value of the field is SDVER1.

Initial values

Field name	Name of constant	Value of constant
SDSID	SDSIDV	'SD¬¬'
SDVER	SDVER1	1
SDOPT	SONDUR	0
SDON	None	Blanks
SDAU	None	Blanks
SDASI	SINONE	Nulls
SDEXP	EIULIM	-1
SDOS	Names and values as defined for MQCHARV	
SDSN	Names and values as defined for MQCHARV	
SDSUD	Names and values as defined for MQCHARV	
SDCID	CINONE	Nulls
SDPRI	PRQDEF	-3
SDACC	ACNONE	Nulls
SDAID	None	Blanks
SDSL	None	1
SDRO	Names and values as defined in MQCHARV	

Note:
1. The symbol ¬ represents a single blank character.

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQSD Structure
D*
D* Structure identifier
```

```
D  SDSID                  1      4
D* Structure version number
D  SDVER                  5      8I 0
D* Options associated with subscribing
D  SDOPT                  9     12I 0
D* Object name
D  SDON                  13     60
D* Alternate user identifier
D  SDAU                  61     72
D* Alternate security identifier
D  SDASI                 73    112
D* Expiry of Subscription
D  SDEXP                113    116I 0
D* Object Long name
D SDOSP                 117    132*
D SDOSO                 133    136I 0
D SDOSS                 137    140I 0
D SDOSL                 141    144I 0
D SDOSC                 145    148I 0
D* Subscription name
D SDSNP                 149    164*
D SDSNO                 165    168I 0
D SDSNS                 169    172I 0
D SDSNL                 173    176I 0
D SDSNC                 177    180I 0
D* Subscription User data
D SDSUDP                181    196*
D SDSUDO                197    200I 0
D SDSUDS                201    204I 0
D SDSUDL                205    208I 0
D SDSUDC                209    212I 0
D* Correlation Id related to this subscription
D SDCID                 213    236
D* Priority set in publications
D SDPRI                 237    240I 0
D* Accounting Token set in publications
D SDACC                 241    272
D* Appl Identity Data set in publications
D SDAID                 273    304
D* Message Selector
D SDSSP                 305    320*
D SDSSO                 321    324I 0
D SDSSS                 325    328I 0
D SDSSL                 329    332I 0
D SDSSC                 333    336
D* Subscription level
D SDSL                  337    340  0
D* Resolved Long object name
D SDROP                 341    356*
D SDROO                 357    360I 0
D SDROS                 361    364I 0
D SDROL                 365    368I 0
D SDROC                 369    372I 0
```

MQSMPO (Set message property options) on IBM i: IBM i

The **MQSMPO** structure allows applications to specify options that control how properties of messages are set.

Overview

Purpose: The structure is an input parameter on the **MQSETMP** call.

Character set and encoding: Data in **MQSMPO** must be in the character set of the application and encoding of the application (ENNAT).

Fields

The MQSMPO structure contains the following fields; the fields are described in **alphabetical order**:

SPOPT (10-digit signed integer)

Location options: The following options relate to the relative location of the property compared to the property cursor:

SPSETF
Sets the value of the first property that matches the specified name, or if it does not exist, adds a new property after all other properties with a matching hierarchy.

SPSETC
Sets the value of the property pointed to by the property cursor. The property pointed to by the property cursor is the one that was last inquired using either the IPINQF or the IPINQN option.

The property cursor is reset when the message handle is reused, or when the message handle is specified in the *HMSG* field of the MQGMO structure on an MQGET call or the MQPMO structure on an MQPUT call.

If this option is used when the property cursor has not yet been established or if the property pointed to by the property cursor has been deleted, the call fails with completion code CCFAIL and reason code RC2471.

SPSETA
Sets a new property after the property pointed to by the property cursor. The property pointed to by the property cursor is the one that was last inquired using either the IPINQF or the IPINQO option.

The property cursor is reset when the message handle is reused, or when the message handle is specified in the *HMSG* field of the MQGMO structure on an MQGET call or the MQPMO structure on an MQPUT call.

If this option is used when the property cursor has not yet been established or if the property pointed to by the property cursor has been deleted, the call fails with completion code CCFAIL and reason code RC2471.

If you need none of the options described, use the following option:

SPNONE
No options specified.

This is always an input field. The initial value of this field is SPSETF.

SPSID (10-digit signed integer)

This is the structure identifier; the value must be:

SPSIDV

Identifier for set message property options structure.

This is always an input field. The initial value of this field is **SPSIDV**.

SPVAKCSI (10-digit signed integer)

The character set of the property value to be set if the value is a character string.

This is always an input field. The initial value of this field is **CSAPL**.

SPVALENC (10-digit signed integer)

The encoding of the property value to be set if the value is numeric.

This is always an input field. The initial value of this field is **ENNAT**.

SPVER (10-digit signed integer)

This is the structure version number; the value must be:

SPVER1
Version-1 set message property options structure.

The following constant specifies the version number of the current version:

SPVERC
Current version of set message property options structure.

This is always an input field. The initial value of this field is **SPVER1**.

Initial values

Table 356. Initial values of fields in MQSMPO

Field name	Name of constant	Value of constant
SPSID	SPSIDV	'SMPO'
SPVER	SPVER1	1
SPOPT	SPNONE	0
SPVALENC	ENNAT	Depends on environment
SPVALCSI	CSAPL	-3

RPG declaration

```
D* MQSMPO Structure
D*
D*
D* Structure identifier
D  SPSID                  1      4    INZ('SMPO')
D*
D* Structure version number
D  SPVER                  5      8I 0 INZ(1)
D*
** Options that control the action of
D* MQSETMP
D  SPOPT                  9     12I 0 INZ(0)
D*
D* Encoding of Value
```

```
D  SPVALENC              13     16I 0 INZ(273)
D*
D* Character set identifier of Value
D  SPVALCSI              17     20I 0 INZ(-3)
```

MQSRO (Subscription Request Options) on IBM i: 

The MQSRO structure allows the application to specify options that control how a subscription request is made.

Overview

Purpose: The structure is an input/output parameter on the MQSUBRQ call.

Version: The current version of MQSRO is SRVER1.

- "Fields"
- "Initial values" on page 3242
- "RPG declaration" on page 3242

Fields

The MQSRO structure contains the following fields; the fields are described in **alphabetical order**:

SRNMP (10-digit signed integer)

This is an output field, returned to the application to indicate the number of publications sent to the subscription queue as a result of this call. Although this number of publications have been sent as a result of this call, there is no guarantee that this many messages will be available for the application to get, especially if they are non-persistent messages.

There may be more than one publication if the topic subscribed to, contained a wildcard. If no wildcards were present in the topic string when the subscription represented by *HSUB* was created, then at most one publication is sent as a result of this call.

SROPT (10-digit signed integer)

One of the following options must be specified. Only one option can be specified.

Other options: The following option controls what happens when the queue manager is quiescing:

SRFIQ

The MQSUBRQ call fails if the queue manager is in the quiescing state.

Default option: If the option described previously is not required, the following option must be used:

SRNONE

Use this value to indicate that no other options have been specified; all options assume their default values.

SRNONE helps program documentation. Although it is not intended that this option be used with any other, because its value is zero, this use cannot be detected.

SRSID (4-byte character string)

This is the structure identifier; the value must be:

SRSIDV

Identifier for Subscription Request SROPT structure.

This is always an input field. The initial value of this field is SRSIDV.

SRVER (10-digit signed integer)

This is the structure version number; the value must be:

SRVER1
Version-1 Subscription Request Options structure.

The following constant specifies the version number of the current version:

SRVERC
Current version of Subscription Request Options structure.

This is always an input field. The initial value of this field is SRVER1.

Initial values

Field name	Name of constant	Value of constant
SRSID	SRSIDV	'SRO¬'
SRVER	SRVER1	1
SROPT	SRNONE	0
SRNMP	None	0

Notes:

1. The symbol ¬ represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQSRO Structure
D*
D* Structure identifier
D  SRSID                  1      4
D* Structure version number
D  SRVER                  5      8I 0
D* Options that control the action of MQSUBRQ
D  SROPT                  9     12I 0
D* Number of publications sent
D  SRNMP                 13     16I 0
```

MQSTS (Status reporting structure) on IBM i: IBM i

The MQSTS structure describes the data in the status structure returned by the MQSTAT command.

Overview

Character set and encoding: Character data in MQSTS is in the character set of the local queue manager; this is given by the *CodedCharSetId* queue manager attribute. Numeric data in MQSTS is in the native machine encoding; this is given by *ENNAT*.

Usage: The MQSTAT command is used to retrieve status information. This information is returned in an MQSTS structure. For information about MQSTAT, see "MQSTAT (Retrieve status information) on IBM i" on page 3375.

Fields

The MQSTS structure contains the following fields; the fields are described in **alphabetical order**:

STSCC (10-digit signed integer)

This is the completion code resulting from the first error reported in the MQSTS structure.

This is always an output field. The initial value of this field is CCOK.

STSFC (10-digit signed integer)

This is the number of asynchronous put calls that failed.

This is an output field. The initial value of this field is 0.

STSOBJN (48-byte character string)

This is the local name of the object involved in the first failure.

This is an output field. The initial value of this field is 48 blank characters.

STSOQMGR (48-byte character string)

This is the name of the queue manager on which the *STSOBJN* object is defined. A name that is entirely blank up to the first null character or the end of the field denotes the queue manager to which the application is connected (the local queue manager).

This is an output field. The initial value of this field is 48 blank characters.

STSOO (10-digit signed integer)

The STSOO used to open the object being reported upon. Present only in Version 2 of MQSTS or higher.

The value of STSOO depends on the value of the MQSTAT **STYPE** parameter.

STATAPT

Zero.

STATREC

Zero.

STATRER

The STSOO used when the failure occurred. The reason for the failure is reported in the *STSCC* and *STSRC* fields in the MQSTS structure.

STSOO is an output field. Its initial value is zero.

STSOS (MQCHARV)

Long object name of failing object being reported on. Present only in Version 2 of MQSTS or higher.

STSOS is a MQCHARV field with a maximum length of 10240. See MQCHARV for a description of how to use the MQCHARV structure.

The interpretation of STSOS depends on the value of the MQSTAT **STYPE** parameter.

STATAPT

This is the long object name of the queue or topic used in the MQPUT operation, which failed.

STATREC

Zero length string

STATRER

This is the long object name of the object that caused the reconnection to fail.

STSOS is an output field. Its initial value is a zero length string.

STSOT (10-digit signed integer)

The type of object being named in *ObjectName*. Possible values are:

OTALSQ
Alias queue.

OTLOCQ
Local queue.

OTMODQ
Model queue.

OTQ Queue.

OTREMQ
Remote queue.

OTTOP
Topic.

This is always an output field. The initial value of this field is OTQ.

STSRC (10-digit signed integer)

This is the reason code resulting from the first error reported in the MQSTS structure

This is always an output field. The initial value of this field is RCNONE.

STSROBJN (48-byte character string)

This is the name of the destination queue named in *STSOBJN* after the local queue manager resolves the name. The name returned is the name of a queue that exists on the queue manager identified by *STSRQMGR*.

A nonblank value is returned only if the object is a single queue opened for browse, input, or output (or any combination). If the object opened is any of the following, *STSROBJN* is set to blanks:

- A topic
- A queue, but not opened for browse, input, or output

This is an output field. The initial value of this field is 48 blank characters.

STSRQMGR (48-byte character string)

This is the name of the destination queue manager after the local queue manager resolves the name. The name returned is the name of the queue manager that owns the queue identified by *STSROBJN*. *STSRQMGR* can be the name of the local queue manager.

If *STSROBJN* is a shared queue that is owned by the queue-sharing group to which the local queue manager belongs, *STSRQMGR* is the name of the queue-sharing group. If the queue is owned by some other queue-sharing group, *STSROBJN* can be the name of the queue-sharing group or the name of a queue manager that is a member of the queue-sharing group (the nature of the value returned is determined by the queue definitions that exist at the local queue manager).

A nonblank value is returned only if the object is a single queue opened for browse, input, or output (or any combination). If the object opened is any of the following, *STSRQMGR* is set to blanks:

- A topic
- A queue, but not opened for browse, input, or output
- A cluster queue with OOBNDN specified (or with OOBNDQ in effect when the **DefBind** queue attribute has the value OOBNDN)

This is an output field. The initial value of this field is 48 blank characters.

STSSC (10-digit signed integer)

This is the number of asynchronous put calls that succeeded.

This is an output field. The initial value of this field is 0.

STSSID (4-byte character string)

This is the structure identifier. The value must be:

STSSID
Identifier for status reporting structure.

The initial value of this field is STSSID.

STSSO (10 digit signed integer)

The STSSO used to open the failing subscription. Present only in Version 2 of MQSTS or higher.

The interpretation of STSSO depends on the value of the MQSTAT **STYPE** parameter.

STATAPT

Zero.

STATREC

Zero.

STATRER

The STSSO used when the failure occurred. The reason for the failure is reported in the *STSCC* and *STSRC* fields in the MQSTS structure. If the failure is not related to subscribing to a topic, the value returned is zero.

STSSO is an output field. Its initial value is zero.

STSSUN (MQCHARV)

The name of the failing subscription. Present only in Version 2 of MQSTS or higher.

STSSUN is a MQCHARV field with a maxiumum length of 10240. See MQCHARV for a description of how to use the MQCHARV structure.

The interpretation of STSSUN depends on the value of the MQSTAT **STYPE** parameter.

STATAPT

Zero length string.

STATREC

Zero length string.

STATRER

The name of the subscription that caused reconnection to fail. If no subscription name is available, or the failure is not related to a subscription, this is a zero-length string.

STSSUN is an output field. Its initial value is a zero length string.

STSVER (10-digit signed integer)

This is the structure version number. The value must be:

STSVR1
Version number for status reporting structure.

The following constant specifies the version number of the current version:

STSVRC
Current version of status reporting structure.

The initial value of this field is STSVR1.

STSWC (10-digit signed integer)

This is the number of asynchronous put calls that completed with a warning.

This is an output field. The initial value of this field is 0.

Initial values

Table 357. Initial values of fields in MQSTS

Field name	Name of constant	Value of constant
STSSID	STSID	
STSVER	STSVRC	STSVR1
STSCC	CCOK	0
STSRC	RCNONE	0
STSSC	None	0
STSWC	None	0
STSFC	None	0
STSOT	None	0
STSOBJN	None	Blanks
STSOQMGR	None	Blanks
STSROBJN	None	Blanks
STSRQMGR	None	Blanks
STSOS	Names and values as defined for MQCHARV	
STSSUN	Names and values as defined for MQCHARV	
STSOO	None	0
STSSO	None	0

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQSTS Structure
D*
D* Structure identifier
D  STSSID                 1      4
D* Structure version number
D  STSVER                 5      8I 0
D* Completion code
D  STSCC                  9     12I 0
D* Reason code
D  STSRC                 13     16I 0
D* Success count
D  STSSC                 17     20I 0
D* Warning count
D  STSWC                 21     24I 0
D* Failure count
D  STSFC                 25     28I 0
D* Object type
D  STSOT                 29     32I 0
D* Object name
D  STSOBJN               33     80
D* Object queue manager
D  STSOQMGR              81     128
D* Resolved object name
D  STSROBJN             129     176
D* Resolved object queue manager name
D  STSRQMGR             177     224
D* Ver:1 **
D* Failing object long name
D* Address of variable length string
D  STSOSCHRP            225    240*
D* Offset of variable length string
D  STSOSCHRO            241    244I 0
D* Size of buffer
D  STSOSVSBS            245    248I 0
D* Length of variable length string
D  STSOSCHRL            249    252I 0
D* CCSID of variable length string
D  STSOSCHRC            253    256I 0
D* Failing subscription name
D* Address of variable length string
D  STSSUNCHRP           257    272*
D* Offset of variable length string
D  STSSUNCHRO           273    276I 0
D* Size of buffer
D  STSSUNVSBS           277    280I 0
D* Length of variable length string
D  STSSUNCHRL           281    284I 0
D* CCSID of variable length string
D  STSSUNCHRC           285    288I 0
D* Failing open options
D  STSOO                289    292I 0
D* Failing subscription options
D  STSSO                293    296I 0
D* Ver:2 **
```

MQTM - Trigger message:

The MQTM structure describes the data in the trigger message that is sent by the queue manager to a trigger-monitor application when a trigger event occurs for a queue.

Overview

Purpose: This structure is part of the IBM MQ Trigger Monitor Interface (TMI), which is one of the IBM MQ framework interfaces.

Format name: FMTM.

Character set and encoding: Character data in MQTM is in the character set of the queue manager that generates the MQTM. Numeric data in MQTM is in the machine encoding of the queue manager that generates the MQTM.

The character set and encoding of the MQTM are given by the *MDCSI* and *MDENC* fields in:

- The MQMD (if the MQTM structure is at the start of the message data), or
- The header structure that precedes the MQTM structure (all other cases).

Usage: A trigger-monitor application may need to pass some or all of the information in the trigger message to the application which is started by the trigger-monitor application. Information which may be needed by the started application includes *TMQN*, *TMTD*, and *TMUD*. The trigger-monitor application can pass the MQTM structure directly to the started application, or pass an MQTMC2 structure instead, depending on what is permitted by the environment and convenient for the started application. For information about MQTMC2, see "MQTMC2 (Trigger message 2 - character format) on IBM i" on page 3252.

- On IBM i, the trigger-monitor application provided with IBM MQ passes an MQTMC2 structure to the started application.

For information about triggers, see Prerequisites for triggering.

MQMD for a trigger message

MQMD for a trigger message: The fields in the MQMD of a trigger message generated by the queue manager are set as follows:

Field in MQMD	Value used
MDSID	MDSIDV
MDVER	MDVER1
MDREP	RONONE
MDMT	MTDGRM
MDEXP	EIULIM
MDFB	FBNONE
MDENC	ENNAT
MDCSI	Queue manager's **CodedCharSetId** attribute
MDFMT	FMTM
MDPRI	Initiation queue's **DefPriority** attribute
MDPER	PENPER
MDMID	A unique value
MDCID	CINONE

Field in MQMD	Value used
MDBOC	0
MDRQ	Blanks
MDRM	Name of queue manager
MDUID	Blanks
MDACC	ACNONE
MDAID	Blanks
MDPAT	ATQM, or as appropriate for the message channel agent
MDPAN	First 28 bytes of the queue manager name
MDPD	Date when trigger message is sent
MDPT	Time when trigger message is sent
MDAOD	Blanks

An application that generates a trigger message is recommended to set similar values, except for the following:

- The *MDPRI* field can be set to PRQDEF (the queue manager will change this to the default priority for the initiation queue when the message is put).
- The *MDRM* field can be set to blanks (the queue manager will change this to the name of the local queue manager when the message it put).
- The context fields should be set as appropriate for the application.

Fields

The MQTM structure contains the following fields; the fields are described in **alphabetical order**:

TMAI (256-byte character string)

Application identifier.

This is a character string that identifies the application to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the **ApplId** attribute of the process object identified by the *TMPN* field; see "Attributes for process definitions on IBM i" on page 3418 for details of this attribute. The content of this data is of no significance to the queue manager.

The meaning of *TMAI* is determined by the trigger-monitor application. The trigger monitor provided by IBM MQ requires *TMAI* to be the name of an executable program.

The length of this field is given by LNPROA. The initial value of this field is 256 blank characters.

TMAT (10-digit signed integer)

Application type.

This identifies the nature of the program to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the **ApplType** attribute of the process object identified by the *TMPN* field; see "Attributes for process definitions on IBM i" on page 3418 for details of this attribute. The content of this data is of no significance to the queue manager.

TMAT can have one of the following standard values. User-defined types can also be used, but should be restricted to values in the range ATUFST through ATULST:

ATCICS
CICS transaction.

ATVSE
CICS/VSE transaction.

AT400 IBM i application.

ATUFST
Lowest value for user-defined application type.

ATULST
Highest value for user-defined application type.

The initial value of this field is 0.

TMED (128-byte character string)

Environment data.

This is a character string that contains environment-related information pertaining to the application to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the **EnvData** attribute of the process object identified by the *TMPN* field; see "Attributes for process definitions on IBM i" on page 3418 for details of this attribute. The content of this data is of no significance to the queue manager.

The length of this field is given by LNPROE. The initial value of this field is 128 blank characters.

TMPN (48-byte character string)

Name of process object.

This is the name of the queue manager process object specified for the triggered queue, and can be used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the **ProcessName** attribute of the queue identified by the *TMQN* field; see "Attributes for queues" on page 3385 for details of this attribute.

Names that are shorter than the defined length of the field are always padded to the right with blanks; they are not ended prematurely by a null character.

The length of this field is given by LNPRON. The initial value of this field is 48 blank characters.

TMQN (48-byte character string)

Name of triggered queue.

This is the name of the queue for which a trigger event occurred, and is used by the application started by the trigger-monitor application. The queue manager initializes this field with the value of the **QName** attribute of the triggered queue; see "Attributes for queues" on page 3385 for details of this attribute.

Names that are shorter than the defined length of the field are padded to the right with blanks; they are not ended prematurely by a null character.

The length of this field is given by LNQN. The initial value of this field is 48 blank characters.

TMSID (4-byte character string)

Structure identifier.

The value must be:

TMSIDV
Identifier for trigger message structure.

The initial value of this field is TMSIDV.

TMTD (64-byte character string)

Trigger data.

This is free-format data for use by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the **TriggerData** attribute of

the queue identified by the *TMQN* field; see "Attributes for queues" on page 3385 for details of this attribute. The content of this data is of no significance to the queue manager.

The length of this field is given by LNTRGD. The initial value of this field is 64 blank characters.

TMUD (128-byte character string)

User data.

This is a character string that contains user information relevant to the application to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the **UserData** attribute of the process object identified by the *TMPN* field; see "Attributes for process definitions on IBM i" on page 3418 for details of this attribute. The content of this data is of no significance to the queue manager.

The length of this field is given by LNPROU. The initial value of this field is 128 blank characters.

TMVER (10-digit signed integer)

Structure version number.

The value must be:

TMVER1
Version number for trigger message structure.

The following constant specifies the version number of the current version:

TMVERC
Current version of trigger message structure.

The initial value of this field is TMVER1.

Initial values

Table 358. Initial values of fields in MQTM

Field name	Name of constant	Value of constant
TMSID	TMSIDV	'TM¬¬'
TMVER	TMVER1	1
TMQN	None	Blanks
TMPN	None	Blanks
TMTD	None	Blanks
TMAT	None	0
TMAI	None	Blanks
TMED	None	Blanks
TMUD	None	Blanks
Notes: 1. The symbol ¬ represents a single blank character.		

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D*
D* MQTM Structure
D*
D* Structure identifier
D  TMSID                  1      4    INZ('TM  ')
D* Structure version number
D  TMVER                  5      8I 0 INZ(1)
```

```
D* Name of triggered queue
D  TMQN                   9     56    INZ
D* Name of process object
D  TMPN                  57    104    INZ
D* Trigger data
D  TMTD                 105    168    INZ
D* Application type
D  TMAT                 169    172I 0 INZ(0)
D* Application identifier
D  TMAI                 173    428    INZ
D* Environment data
D  TMED                 429    556    INZ
D* User data
D  TMUD                 557    684    INZ
```

MQTMC2 (Trigger message 2 - character format) on IBM i: IBM i

When a trigger-monitor application retrieves a trigger message (MQTM) from an initiation queue, the trigger monitor might need to pass some or all of the information in the trigger message to the application that is started by the trigger monitor.

Overview

Purpose: Information that may be needed by the started application includes *TC2QN*, *TC2TD*, and *TC2UD*. The trigger monitor application can pass the MQTM structure directly to the started application, or pass an MQTMC2 structure instead, depending on what is permitted by the environment and convenient for the started application.

This structure is part of the IBM MQ Trigger Monitor Interface (TMI), which is one of the IBM MQ framework interfaces.

Character set and encoding: Character data in MQTMC2 is in the character set of the local queue manager; this is given by the **CodedCharSetId** queue manager attribute.

Usage: The MQTMC2 structure is like the format of the MQTM structure. The difference is that the non-character fields in MQTM are changed in MQTMC2 to character fields of the same length, and the queue manager name is added at the end of the structure.

- On IBM i, the trigger monitor application provided with IBM MQ passes an MQTMC2 structure to the started application.
- "Fields"
- "Initial values" on page 3253
- "RPG declaration" on page 3254

Fields

The MQTMC2 structure contains the following fields; the fields are described in **alphabetical order**:

TC2AI (256-byte character string)

Application identifier.

See the *TMAI* field in the MQTM structure.

TC2AT (4-byte character string)

Application type.

This field always contains blanks, whatever the value in the *TMAT* field in the MQTM structure of the original trigger message.

TC2ED (128-byte character string)

Environment data.

See the *TMED* field in the MQTM structure.

TC2PN (48-byte character string)

Name of process object.

See the *TMPN* field in the MQTM structure.

TC2QMN (48-byte character string)

Queue manager name.

This is the name of the queue manager at which the trigger event occurred.

TC2QN (48-byte character string)

Name of triggered queue.

See the *TMQN* field in the MQTM structure.

TC2SID (4-byte character string)

Structure identifier.

The value must be:

TCSIDV
Identifier for trigger message (character format) structure.

TC2TD (64-byte character string)

Trigger data.

See the *TMTD* field in the MQTM structure.

TC2UD (128-byte character string)

User data.

See the *TMUD* field in the MQTM structure.

TC2VER (4-byte character string)

Structure version number.

The value must be:

TCVER2
Version 2 trigger message (character format) structure.

The following constant specifies the version number of the current version:

TCVERC
Current version of trigger message (character format) structure.

Initial values

Table 359. Initial values of fields in MQTMC2

Field name	Name of constant	Value of constant
TC2SID	TCSIDV	'TMC¬'
TC2VER	TCVER2	'¬¬¬2'
TC2QN	None	Blanks
TC2PN	None	Blanks
TC2TD	None	Blanks
TC2AT	None	Blanks
TC2AI	None	Blanks
TC2ED	None	Blanks
TC2UD	None	Blanks
TC2QMN	None	Blanks
Notes: 1. The symbol ¬ represents a single blank character.		

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQTMC2 Structure
D*
D* Structure identifier
D  TC2SID                 1      4
D* Structure version number
D  TC2VER                 5      8
D* Name of triggered queue
D  TC2QN                  9     56
D* Name of process object
D  TC2PN                 57    104
D* Trigger data
D  TC2TD                105    168
D* Application type
D  TC2AT                169    172
D* Application identifier
D  TC2AI                173    428
D* Environment data
D  TC2ED                429    556
D* User data
D  TC2UD                557    684
D* Queue manager name
D  TC2QMN               685    732
```

MQWIH (Work information header) on IBM i: IBM i

The MQWIH structure describes the information that must be present at the start of a message that is to be handled by the z/OS workload manager.

Overview

Format name: FMWIH.

Character set and encoding: The fields in the MQWIH structure are in the character set and encoding given by the *MDCSI* and *MDENC* fields in the header structure that precedes MQWIH, or by those fields in the MQMD structure if the MQWIH is at the start of the application message data.

The character set must be one that has single-byte characters for the characters that are valid in queue names.

Usage: If a message is to be processed by the z/OS workload manager, the message must begin with an MQWIH structure.

- "Fields"
- "Initial values" on page 3257
- "RPG declaration" on page 3257

Fields

The MQWIH structure contains the following fields; the fields are described in **alphabetical order**:

WICSI (10-digit signed integer)

Character-set identifier of data that follows MQWIH.

This specifies the character set identifier of the data that follows the MQWIH structure; it does not apply to character data in the MQWIH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

CSINHT

Inherit character-set identifier of this structure.

Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value CSINHT is not returned by the MQGET call.

CSINHT cannot be used if the value of the *MDPAT* field in MQMD is ATBRKR.

The initial value of this field is CSUNDF.

WIENC (10-digit signed integer)

Numeric encoding of data that follows MQWIH.

This specifies the numeric encoding of the data that follows the MQWIH structure; it does not apply to numeric data in the MQWIH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is 0.

WIFLG (10-digit signed integer)

Flags

The value must be:

WINONE
No flags.

The initial value of this field is WINONE.

WIFMT (8-byte character string)

Format name of data that follows MQWIH.

This specifies the format name of the data that follows the MQWIH structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *MDFMT* field in MQMD.

The length of this field is given by LNFMT. The initial value of this field is FMNONE.

WILEN (10-digit signed integer)

Length of MQWIH structure.

The value must be:

WILEN1
Length of version-1 work information header structure.

The following constant specifies the length of the current version:

WILENC
Length of current version of work information header structure.

The initial value of this field is WILEN1.

WIRSV (32-byte character string)

Reserved.

This is a reserved field; it must be blank.

WISID (4-byte character string)

Structure identifier.

The value must be:

WISIDV
Identifier for work information header structure.

The initial value of this field is WISIDV.

WISNM (32-byte character string)

Service name.

This is the name of the service that is to process the message.

The length of this field is given by LNSVNM. The initial value of this field is 32 blank characters.

WISST (8-byte character string)

Service step name.

This is the name of the step of *WISNM* to which the message relates.

The length of this field is given by LNSVST. The initial value of this field is 8 blank characters.

WITOK (16-byte bit string)

Message token.

This is a message token that uniquely identifies the message.

For the MQPUT and MQPUT1 calls, this field is ignored. The length of this field is given by LNMTOK. The initial value of this field is MTKNON.

WIVER (10-digit signed integer)

Structure version number.

The value must be:

WIVER1
Version-1 work information header structure.

The following constant specifies the version number of the current version:

WIVERC
Current version of work information header structure.

The initial value of this field is WIVER1.

Initial values

Table 360. Initial values of fields in MQWIH

Field name	Name of constant	Value of constant
WISID	WISIDV	'WIH¬'
WIVER	WIVER1	1
WILEN	WILEN1	120
WIENC	None	0
WICSI	CSUNDF	0
WIFMT	FMNONE	Blanks
WIFLG	WINONE	0
WISNM	None	Blanks
WISST	None	Blanks
WITOK	MTKNON	Nulls
WIRSV	None	Blanks
Notes: 1. The symbol ¬ represents a single blank character.		

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D*
D* MQWIH Structure
D*
D* Structure identifier
D  WISID                  1      4    INZ('WIH ')
D* Structure version number
D  WIVER                  5      8I 0 INZ(1)
D* Length of MQWIH structure
D  WILEN                  9     12I 0 INZ(120)
D* Numeric encoding of data that followsMQWIH
D  WIENC                 13     16I 0 INZ(0)
D* Character-set identifier of data thatfollows MQWIH
D  WICSI                 17     20I 0 INZ(0)
D* Format name of data that followsMQWIH
D  WIFMT                 21     28    INZ('        ')
```

```
D* Flags
D  WIFLG                 29     32I 0 INZ(0)
D* Service name
D  WISNM                 33     64    INZ
D* Service step name
D  WISST                 65     72    INZ
D* Message token
D  WITOK                 73     88    INZ(X'00000000000000-
D                                     00000000000000000')
D* Reserved
D  WIRSV                 89    120    INZ
```

MQXQH (Transmission-queue header) on IBM i: IBM i

The MQXQH structure describes the information that is prefixed to the application message data of messages when they are on transmission queues.

Overview

Purpose: A transmission queue is a special type of local queue that temporarily holds messages destined for remote queues (that is, destined for queues that do not belong to the local queue manager). A transmission queue is denoted by the **Usage** queue attribute having the value USTRAN.

Format name: FMXQH.

Character set and encoding: Data in MQXQH must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT for the C programming language.

The character set and encoding of the MQXQH must be set into the *MDCSI* and *MDENC* fields in:

- The separate MQMD (if the MQXQH structure is at the start of the message data), or
- The header structure that precedes the MQXQH structure (all other cases).

Usage: A message that is on a transmission queue has *two* message descriptors:

- One message descriptor is stored separately from the message data; this is called the *separate message descriptor*, and is generated by the queue manager when the message is placed on the transmission queue. Some of the fields in the separate message descriptor are copied from the message descriptor provided by the application on the MQPUT or MQPUT1 call.

  The separate message descriptor is the one that is returned to the application in the **MSGDSC** parameter of the MQGET call when the message is removed from the transmission queue.
- A second message descriptor is stored within the MQXQH structure as part of the message data; this is called the *embedded message descriptor*, and is a copy of the message descriptor that was provided by the application on the MQPUT or MQPUT1 call (with minor variations).

  The embedded message descriptor is always a version-1 MQMD. If the message put by the application has nondefault values for one or more of the version-2 fields in the MQMD, an MQMDE structure follows the MQXQH, and is in turn followed by the application message data (if any). The MQMDE is either:

  – Generated by the queue manager (if the application uses a version-2 MQMD to put the message), or
  – Already present at the start of the application message data (if the application uses a version-1 MQMD to put the message).

  The embedded message descriptor is the one that is returned to the application in the **MSGDSC** parameter of the MQGET call when the message is removed from the final destination queue.

- "Fields in the separate message descriptor" on page 3259
- "Fields in the embedded message descriptor" on page 3259

Fields in the separate message descriptor

The fields in the separate message descriptor are set by the queue manager as shown in the following list. If the queue manager does not support the version-2 MQMD, a version-1 MQMD is used without loss of function.

Field in separate MQMD	Value used
MDSID	MDSIDV
MDVER	MDVER2
MDREP	Copied from the embedded message descriptor, but with the bits identified by ROAUXM set to zero. (This prevents a COA or COD report message being generated when a message is placed on or removed from a transmission queue.)
MDMT	Copied from the embedded message descriptor.
MDEXP	Copied from the embedded message descriptor.
MDFB	Copied from the embedded message descriptor.
MDENC	ENNAT
MDCSI	Queue manager's **CodedCharSetId** attribute.
MDFMT	FMXQH
MDPRI	Copied from the embedded message descriptor.
MDPER	Copied from the embedded message descriptor.
MDMID	A new value is generated by the queue manager. This message identifier is different from the *MDMID* that the queue manager may have generated for the embedded message descriptor (see described previously).
MDCID	The *MDMID* from the embedded message descriptor.
MDBOC	0
MDRQ	Copied from the embedded message descriptor.
MDRM	Copied from the embedded message descriptor.
MDUID	Copied from the embedded message descriptor.
MDACC	Copied from the embedded message descriptor.
MDAID	Copied from the embedded message descriptor.
MDPAT	ATQM
MDPAN	First 28 bytes of the queue manager name.
MDPD	Date when message was put on transmission queue.
MDPT	Time when message was put on transmission queue.
MDAOD	Blanks
MDGID	GINONE
MDSEQ	1
MDOFF	0
MDMFL	MFNONE
MDOLN	OLUNDF

Fields in the embedded message descriptor

The fields in the embedded message descriptor have the same values as those in the **MSGDSC** parameter of the MQPUT or MQPUT1 call, except for the following:

- The *MDVER* field always has the value MDVER1.

- If the *MDPRI* field has the value PRQDEF, it is replaced by the value of the queue's **DefPriority** attribute.
- If the *MDPER* field has the value PEQDEF, it is replaced by the value of the queue's **DefPersistence** attribute.
- If the *MDMID* field has the value MINONE, or the PMNMID option was specified, or the message is a distribution-list message, *MDMID* is replaced by a new message identifier generated by the queue manager.

  When a distribution-list message is split into smaller distribution-list messages placed on different transmission queues, the *MDMID* field in each of the new embedded message descriptors is the same as that in the original distribution-list message.
- If the PMNCID option was specified, *MDCID* is replaced by a new correlation identifier generated by the queue manager.
- The context fields are set as indicated by the PM* options specified in the **PMO** parameter; the context fields are:
  - *MDACC*
  - *MDAID*
  - *MDAOD*
  - *MDPAN*
  - *MDPAT*
  - *MDPD*
  - *MDPT*
  - *MDUID*
- The version-2 fields (if they were present) are removed from the MQMD, and moved into an MQMDE structure, if one or more of the version-2 fields has a nondefault value.

Putting messages on remote queues

: When an application puts a message on a remote queue (either by specifying the name of the remote queue directly, or by using a local definition of the remote queue), the local queue manager:
- Creates an MQXQH structure containing the embedded message descriptor
- Appends an MQMDE if one is needed and is not already present
- Appends the application message data
- Places the message on an appropriate transmission queue

Putting messages directly on transmission queues

It is also possible for an application to put a message directly on a transmission queue. In this case the application must prefix the application message data with an MQXQH structure, and initialize the fields with appropriate values. In addition, the *MDFMT* field in the **MSGDSC** parameter of the MQPUT or MQPUT1 call must have the value FMXQH.

Character data in the MQXQH structure created by the application must be in the character set of the local queue manager (defined by the **CodedCharSetId** queue manager attribute), and integer data must be in the native machine encoding. In addition, character data in the MQXQH structure must be padded with blanks to the defined length of the field; the data must not be ended prematurely by using a null character, because the queue manager does not convert the null and subsequent characters to blanks in the MQXQH structure.

Note however that the queue manager does not check that an MQXQH structure is present, or that valid values have been specified for the fields.

Getting messages from transmission queues

Applications that get messages from a transmission queue must process the information in the MQXQH structure in an appropriate fashion. The presence of the MQXQH structure at the beginning of the application message data is indicated by the value FMXQH being returned in the *MDFMT* field in the **MSGDSC** parameter of the MQGET call. The values returned in the *MDCSI* and *MDENC* fields in the **MSGDSC** parameter, indicates the character set and encoding of the character and integer data in the MQXQH structure. The character set and encoding of the application message data are defined by the *MDCSI* and *MDENC* fields in the embedded message descriptor.

Fields

The MQXQH structure contains the following fields; the fields are described in **alphabetical order**:

XQMD (MQMD1)

Original message descriptor.

This is the embedded message descriptor, and is a close copy of the message descriptor MQMD that was specified as the **MSGDSC** parameter on the MQPUT or MQPUT1 call when the message was originally put to the remote queue.

Note: This is a version-1 MQMD.

The initial values of the fields in this structure are the same as those in the MQMD structure.

XQRQ (48-byte character string)

Name of destination queue.

This is the name of the message queue that is the apparent eventual destination for the message (this may prove not to be the actual eventual destination if, for example, this queue is defined at *XQRQM* to be a local definition of another remote queue).

If the message is a distribution-list message (that is, the *MDFMT* field in the embedded message descriptor is FMDH), *XQRQ* is blank.

The length of this field is given by LNQN. The initial value of this field is 48 blank characters.

XQRQM (48-byte character string)

Name of destination queue manager.

This is the name of the queue manager or queue-sharing group that owns the queue that is the apparent eventual destination for the message.

If the message is a distribution-list message, *XQRQM* is blank.

The length of this field is given by LNQMN. The initial value of this field is 48 blank characters.

XQSID (4-byte character string)

Structure identifier.

The value must be:

XQSIDV
Identifier for transmission-queue header structure.

The initial value of this field is XQSIDV.

XQVER (10-digit signed integer)

Structure version number.

The value must be:

XQVER1
Version number for transmission-queue header structure.

The following constant specifies the version number of the current version:

XQVERC
Current version of transmission-queue header structure.

The initial value of this field is XQVER1.

Initial values

Table 361. Initial values of fields in MQXQH

Field name	Name of constant	Value of constant
XQSID	XQSIDV	'XQH¬'
XQVER	XQVER1	1
XQRQ	None	Blanks
XQRQM	None	Blanks
XQMD	Same names and values as MQMD; see Table 341 on page 3161	-
Notes: 1. The symbol ¬ represents a single blank character.		

RPG declaration

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D*
D* MQXQH Structure
D*
D* Structure identifier
D  XQSID                  1      4    INZ('XQH ')
D* Structure version number
D  XQVER                  5      8I 0 INZ(1)
D* Name of destination queue
D  XQRQ                   9     56    INZ
D* Name of destination queue manager
D  XQRQM                 57    104    INZ
D* Original message descriptor
D  XQ1SID               105    108    INZ('MD  ')
D  XQ1VER               109    112I 0 INZ(1)
D  XQ1REP               113    116I 0 INZ(0)
D  XQ1MT                117    120I 0 INZ(8)
D  XQ1EXP               121    124I 0 INZ(-1)
D  XQ1FB                125    128I 0 INZ(0)
D  XQ1ENC               129    132I 0 INZ(273)
D  XQ1CSI               133    136I 0 INZ(0)
D  XQ1FMT               137    144    INZ('        ')
D  XQ1PRI               145    148I 0 INZ(-1)
D  XQ1PER               149    152I 0 INZ(2)
D  XQ1MID               153    176    INZ(X'00000000000000-
D                                     00000000000000000000-
D                                     000000000000')
D  XQ1CID               177    200    INZ(X'00000000000000-
D                                     00000000000000000000-
D                                     000000000000')
D  XQ1BOC               201    204I 0 INZ(0)
D  XQ1RQ                205    252    INZ
D  XQ1RM                253    300    INZ
D  XQ1UID               301    312    INZ
D  XQ1ACC               313    344    INZ(X'00000000000000-
D                                     00000000000000000000-
```

```
D                                       00000000000000000000-
D                                       000000')
D  XQ1AID               345    376     INZ
D  XQ1PAT               377    380I 0  INZ(0)
D  XQ1PAN               381    408     INZ
D  XQ1PD                409    416     INZ
D  XQ1PT                417    424     INZ
D  XQ1AOD               425    428     INZ
```

Function calls on IBM i

Use this information to learn about the function calls available in IBM i programming.

Conventions used in the call descriptions on IBM i

For each call, this collection of topics gives a description of the parameters and usage of the call. This is followed by typical invocations of the call, and typical declarations of its parameters, in the RPG programming language.

Important: When coding IBM MQ API calls you must ensure that all relevant parameters (as described in the following sections) are provided. Failure to do so can produce unpredictable results.

The description of each call contains the following sections:

Call name
The call name, followed by a brief description of the purpose of the call.

Parameters
For each parameter, the name is followed by its data type in parentheses ( ) and its direction; for example:

CMPCOD (9-digit decimal integer) - output

There is more information about the structure data types in “Elementary data types” on page 3000.

The direction of the parameter can be:

Input You (the programmer) must provide this parameter.

Output
The call returns this parameter.

Input/output
You must provide this parameter, but it is modified by the call.

There is also a brief description of the purpose of the parameter, together with a list of any values that the parameter can take.

The last two parameters in each call are a completion code and a reason code. The completion code indicates whether the call completed successfully, partially, or not at all. Further information about the partial success or the failure of the call is given in the reason code.

Usage notes
Additional information about the call, describing how to use it and any restrictions on its use.

RPG invocation
Typical invocation of the call, and declaration of its parameters, in RPG.

Other notational conventions are:

Constants

Names of constants are shown in uppercase; for example, OOOUT.

Arrays

In some calls, parameters are arrays of character strings with a size that is not fixed. In the descriptions of these parameters, a lowercase *n* represents a numeric constant. When you code the declaration for that parameter, replace the *n* with the numeric value you require.

MQBACK (Back out changes) on IBM i: IBM i

The MQBACK call indicates to the queue manager that all of the message gets and puts that have occurred since the last syncpoint are to be backed out. Messages put as part of a unit of work are deleted; messages retrieved as part of a unit of work are reinstated on the queue.

Syntax

MQBACK *(Hconn, CompCode, Reason)*

Usage notes

Consider these usage notes when using MQBACK.

1. This call can be used only when the queue manager itself coordinates the unit of work. This is a local unit of work, where the changes affect only IBM MQ resources.
2. In environments where the queue manager does not coordinate the unit of work, the appropriate back-out call must be used instead of MQBACK. The environment may also support an implicit back out caused by the application terminating abnormally.
   - On IBM i, this call can be used for local units of work coordinated by the queue manager. This means that a commitment definition must not exist at job level, that is, the `STRCMTCTL` command with the **`CMTSCOPE(*JOB)`** parameter must not have been issued for the job.
3. If an application ends with uncommitted changes in a unit of work, the disposition of those changes depends on whether the application ends normally or abnormally. See the usage notes in "MQDISC (Disconnect queue manager) on IBM i" on page 3305 for further details.
4. When an application puts or gets messages in groups or segments of logical messages, the queue manager retains information relating to the message group and logical message for the last successful MQPUT and MQGET calls. This information is associated with the queue handle, and includes such things as:
   - The values of the *MDGID*, *MDSEQ*, *MDOFF*, and *MDMFL* fields in MQMD.
   - Whether the message is part of a unit of work.
   - For the MQPUT call: whether the message is persistent or nonpersistent.

   The queue manager keeps *three* sets of group and segment information, one set for each of the following:
   - The last successful MQPUT call (this can be part of a unit of work).
   - The last successful MQGET call that removed a message from the queue (this can be part of a unit of work).
   - The last successful MQGET call that browsed a message on the queue (this cannot be part of a unit of work).

If the application puts or gets the messages as part of a unit of work, and the application then decides to back out the unit of work, the group and segment information is restored to the value that it had previously:

- The information associated with the MQPUT call is restored to the value that it had before the first successful MQPUT call for that queue handle in the current unit of work.
- The information associated with the MQGET call is restored to the value that it had before the first successful MQGET call for that queue handle in the current unit of work.

Queues which were updated by the application after the unit of work had started, but outside the scope of the unit of work, do not have their group and segment information restored if the unit of work is backed out.

Restoring the group and segment information to its previous value when a unit of work is backed out allows the application to spread a large message group or large logical message consisting of many segments across several units of work, and to restart at the correct point in the message group or logical message if one of the units of work fails. Using several units of work might be advantageous if the local queue manager has only limited queue storage. However, the application must maintain sufficient information to be able to restart putting or getting messages at the correct point if a system failure occurs. For details of how to restart at the correct point after a system failure, see the PMLOGO option described in "MQPMO (Put-message options) on IBM i" on page 3185, and the GMLOGO option described in "MQGMO (Get-message options) on IBM i" on page 3084.

The remaining usage notes apply only when the queue manager coordinates the units of work:

1. A unit of work has the same scope as a connection handle. This means that all IBM MQ calls which affect a particular unit of work must be performed using the same connection handle. Calls issued using a different connection handle (for example, calls issued by another application) affect a different unit of work. See the **HCONN** parameter described in "MQCONN (Connect queue manager) on IBM i" on page 3290 for information about the scope of connection handles.
2. Only messages that were put or retrieved as part of the current unit of work are affected by this call.
3. A long-running application that issues MQGET, MQPUT, or MQPUT1 calls within a unit of work, but which never issues a commit or backout call, can cause queues to fill up with messages that are not available to other applications. To guard against this possibility, the administrator should set the **MaxUncommittedMsgs** queue manager attribute to a value that is low enough to prevent runaway applications filling the queues, but high enough to allow the expected messaging applications to work correctly.

Parameters

The MQBACK call has the following parameters:

HCONN (10-digit signed integer) - input

Connection handle.

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN or MQCONNX call.

CMPCOD (10-digit signed integer) - output

Completion code.

It is one of the following:

CCOK
Successful completion.

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

Reason code qualifying *COMCOD*.

If *COMCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *COMCOD* is CCFAIL:

RC2219
(2219, X'8AB') MQI call reentered before previous call complete.

RC2009
(2009, X'7D9') Connection to queue manager lost.

RC2018
(2018, X'7E2') Connection handle not valid.

RC2101
(2101, X'835') Object damaged.

RC2123
(2123, X'84B') Result of commit or back-out operation is mixed.

RC2162
(2162, X'872') Queue manager shutting down.

RC2102
(2102, X'836') Insufficient system resources available.

RC2071
(2071, X'817') Insufficient storage available.

RC2195
(2195, X'893') Unexpected error occurred.

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQBACK(HCONN : COMCOD : REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQBACK           PR                  EXTPROC('MQBACK')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Completion code
D COMCOD                        10I 0
D* Reason code qualifying COMCOD
D REASON                        10I 0
```

IBM i

AIX, HP-UX, IBM i, Solaris, Windows S/390

MQBEGIN (Begin unit of work) on IBM i: IBM i

The MQBEGIN call begins a unit of work that is coordinated by the queue manager, and that may involve external resource managers.

- This call is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Windows.
- "Syntax"
- "Usage notes"
- "Parameters" on page 3268
- "RPG Declaration" on page 3269

Syntax

MQBEGIN *(HCONN, BEGOP, CMPCOD, REASON)*

Usage notes

1. The MQBEGIN call can be used to start a unit of work that is coordinated by the queue manager and that might involve changes to resources owned by other resource managers. The queue manager supports three types of unit-of-work:

   Queue manager-coordinated local unit of work
   This is a unit of work in which the queue manager is the only resource manager participating, and so the queue manager acts as the unit-of-work coordinator.
   - To start this type of unit of work, the PMSYP or GMSYP option should be specified on the first MQPUT, MQPUT1, or MQGET call in the unit of work.

     It is not necessary for the application to issue the MQBEGIN call to start the unit of work, but if MQBEGIN is used, the call completes with CCWARN and reason code RC2121.
   - To commit or back out this type of unit of work, the MQCMIT or MQBACK call must be used.

   Queue manager-coordinated global unit of work
   This is a unit of work in which the queue manager acts as the unit-of-work coordinator, both for IBM MQ resources *and* for resources belonging to other resource managers. Those resource managers cooperate with the queue manager to ensure that all changes to resources in the unit of work are committed or backed out together.
   - To start this type of unit of work, the MQBEGIN call must be used.
   - To commit or back out this type of unit of work, the MQCMIT and MQBACK calls must be used.

   Externally-coordinated global unit of work
   This is a unit of work in which the queue manager is a participant, but the queue manager does not act as the unit-of-work coordinator. Instead, there is an external unit-of-work coordinator with whom the queue manager cooperates.
   - To start this type of unit of work, the relevant call provided by the external unit-of-work coordinator must be used.

     If the MQBEGIN call is used to try to start the unit of work, the call fails with reason code RC2012.
   - To commit or back out this type of unit of work, the commit and back-out calls provided by the external unit-of-work coordinator must be used.

     If the MQCMIT or MQBACK call is used to try to commit or back out the unit of work, the call fails with reason code RC2012.
2. If the application ends with uncommitted changes in a unit of work, the disposition of those changes depends on whether the application ends normally or abnormally. See the usage notes in "MQDISC (Disconnect queue manager) on IBM i" on page 3305 for further details.

3. An application can participate in only one unit of work at a time. The MQBEGIN call fails with reason code RC2128 if there is already a unit of work in existence for the application, regardless of which type of unit of work it is.
4. The MQBEGIN call is not valid in an IBM MQ client environment. An attempt to use the call fails with reason code RC2012.
5. When the queue manager is acting as the unit-of-work coordinator for global units of work, the resource managers that can participate in the unit of work are defined in the queue manager's configuration file.
6. On IBM i, the three types of unit of work are supported as follows:
   - **Queue manager-coordinated local units of work** can be used only when a commitment definition does not exist at the job level, that is, the `STRCMTCTL` command with the **`CMTSCOPE(*JOB)`** parameter must not have been issued for the job.
   - **Queue manager-coordinated global units of work** are not supported.
   - **Externally-coordinated global units of work** can be used only when a commitment definition exists at job level, that is, the `STRCMTCTL` command with the **`CMTSCOPE(*JOB)`** parameter must have been issued for the job. If this has been done, the IBM i `COMMIT` and `ROLLBACK` operations apply to IBM MQ resources as well as to resources belonging to other participating resource managers.

Parameters

The MQBEGIN call has the following parameters:

HCONN (10-digit signed integer) - input

Connection handle.

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN or MQCONNX call.

BEGOP (MQBO) - input/output

Options that control the action of MQBEGIN.

See "MQBO (Begin options) on IBM i" on page 3024 for details.

If no options are required, programs written in C or S/390 assembler can specify a null parameter address, instead of specifying the address of an MQBO structure.

CMPCOD (10-digit signed integer) - output

Completion code.

It is one of the following:

CCOK
Successful completion.

CCWARN
Warning (partial completion).

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

Reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCWARN:

RC2121
(2121, X'849') No participating resource managers registered.

RC2122
(2122, X'84A') Participating resource manager not available.

If *CMPCOD* is CCFAIL:

RC2134
(2134, X'856') Begin-options structure not valid.

RC2219
(2219, X'8AB') MQI call reentered before previous call complete.

RC2009
(2009, X'7D9') Connection to queue manager lost.

RC2012
(2012, X'7DC') Call not valid in environment.

RC2018
(2018, X'7E2') Connection handle not valid.

RC2046
(2046, X'7FE') Options not valid or not consistent.

RC2162
(2162, X'872') Queue manager shutting down.

RC2102
(2102, X'836') Insufficient system resources available.

RC2071
(2071, X'817') Insufficient storage available.

RC2195
(2195, X'893') Unexpected error occurred.

RC2128
(2128, X'850') Unit of work already started.

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQBEGIN(HCONN : BEGOP : CMPCOD :
C                                       REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQBEGIN          PR                  EXTPROC('MQBEGIN')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Options that control the action of MQBEGIN
D BEGOP                         12A
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
```

MQBUFMH (Convert buffer into message handle) on IBM i: IBM i

The MQBUFMH function call converts a buffer into a message handle and is the inverse of the MQMHBUF call.

This call takes a message descriptor and MQRFH2 properties in the buffer and makes them available through a message handle. The MQRFH2 properties in the message data are, optionally, removed. The *Encoding*, *CodedCharSetId*, and *Format* fields of the message descriptor are updated, if necessary, to correctly describe the contents of the buffer after the properties have been removed.

- "Syntax"
- "Usage notes"
- "Parameters"
- "RPG Declaration" on page 3272

Syntax

MQBUFMH *(Hconn, Hmsg, BufMsgHOpts, MsgDesc, Buffer, BufferLength, DataLength, CompCode, Reason)*

Usage notes

MQBUFMH calls cannot be intercepted by API exits - a buffer is converted into a message handle in the application space; the call does not reach the queue manager.

Parameters

The MQBUFMH call has the following parameters:

HCONN (10-digit signed integer) - input

This handle represents the connection to the queue manager. The value of *HCONN* must match the connection handle that was used to create the message handle specified in the **Hmsg** parameter.

If the message handle was created by using HCUNAS, a valid connection must be established on the thread converting a buffer into a message handle. If a valid connection is not established, the call fails with RC2009.

HMSG (20-digit signed integer) - input

This handle is the message handle for which a buffer is required. The value was returned by a previous MQCRTMH call.

BMHOPT (MQBMHO) - input

The MQBMHO structure allows applications to specify options that control how message handles are produced from buffers.

See "MQBMHO (Buffer to message handle options) on IBM i" on page 3023 for details.

MSGDSC (MQMD) - input/output

The *MSGDSC* structure contains the message descriptor properties and describes the contents of the buffer area.

On output from the call, the properties are optionally removed from the buffer area and, in this case, the message descriptor is updated to correctly describe the buffer area.

Data in this structure must be in the character set and encoding of the application.

BUFLEN (10-digit signed integer) - input

BUFLEN is the length of the Buffer area, in bytes.

A *BUFLEN* of zero bytes is valid, and indicates that the buffer area contains no data.

BUFFER (1-byte bit string x BUFLEN) - input/output

BUFFER defines the area containing the message buffer. For most data, you must align the buffer on a 4-byte boundary.

If *BUFFER* contains character or numeric data, set the *CodedCharSetId* and *Encoding* fields in the **MSGDSC** parameter to the values appropriate to the data; this enables the data to be converted, if necessary.

If properties are found in the message buffer they are optionally removed; they later become available from the message handle on return from the call.

In the C programming language, the parameter is declared as a pointer-to-void, which means the address of any type of data can be specified as the parameter.

If the **BUFLEN** parameter is zero, *BUFFER* is not referred to. In this case, the parameter address passed by programs written in C or System/390 assembler can be null.

DATLEN (10-digit signed integer) - output

DATLEN is the length, in bytes, of the buffer which might have the properties removed.

CMPCOD (10-digit signed integer) - output

CCOK
Successful completion.

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

The reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCFAIL:

RC2204
(2204, X'089C') Adapter not available.

RC2130
(2130, X'852') Unable to load adapter service module.

RC2157
(2157, X'86D') Primary and home ASIDs differ.

RC2489
(2489, X'09B9') Buffer to message handle options structure not valid.

RC2004
(2004, X'07D4') Buffer parameter not valid.

RC2005
(2005, X'07D5') Buffer length parameter not valid.

RC2219
(2219, X'08AB') MQI call entered before previous call completed.

RC2009
(2009, X'07D9') Connection to queue manager lost.

RC2460
(2460, X'099C') Message handle not valid.

RC2026
(2026, X'07EA') Message descriptor not valid.

RC2499
(2499, X'09C3') Message handle already in use.

RC2046
(2046, X'07FE') Options not valid or not consistent.

RC2334
(2334, X'091E') MQRFH2 structure not valid.

RC2421
(2421, X'0975') An MQRFH2 folder containing properties could not be parsed.

RC2195
(2195, X'893') Unexpected error occurred.

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQBUFMH(HCONN : HMSG : BMHOPT :
                                        MSGDSC : BUFLEN : BUFFER :
                                        DATLEN : CMPCOD : REASON)
```

The prototype definition for the call is:

```
DMQBUFMH          PR                  EXTPROC('MQBUFMH')
D* Connection handle
D HCONN                           10I 0
D* Message handle
D HMSG                            10I 0
D* Options that control the action of MQBUFMH
D BMHOPT                          12A   VALUE
D* Message descriptor
D MSGDSC                         364A
D* Length in bytes of the Buffer area
D BUFLEN                          10I 0
D* Area to contain the message buffer
D BUFFER                            *   VALUE
D* Length of the output buffer
D DATLEN                          10I 0
D* Completion code
D CMPCOD                          10I 0
D* Reason code qualifying CompCode
D REASON                          10I 0
```

MQCB (Manage callback) on IBM i: 

The MQCB call reregisters a callback for the specified object handle and controls activation and changes to the callback.

A callback is a piece of code (specified as either the name of a function that can be dynamically linked or as a function pointer) that is called by IBM MQ when certain events occur.

To use MQCB and MQCTL on a V7 client you must be connected to a V7 server and the **SHARECNV** parameter of the channel must have a non-zero value.

For information about Global units of work see: Global units of work.

The types of callback that can be defined are:

Message consumer
A message consumer callback function is called when a message, meeting the selection criteria specified, is available on an object handle.

Only one callback function can be registered against each object handle. If a single queue is to be read with multiple selection criteria then the queue must be opened multiple times and a consumer function registered on each handle.

Event handler
The event handler is called for conditions that affect the whole callback environment.

The function is called when an event condition occurs, for example, a queue manager or connection stopping or quiescing.

The function is not called for conditions that are specific to a single message consumer, for example RC2016; it is called however if a callback function does not end normally.

Syntax

MQCB *(HCONN, OPERATN, HOBJ, CBDSC, MSGDSC, GMO, CMPCOD, REASON)*

Usage notes for MQCB

1. MQCB is used to define the action to be invoked for each message, matching the specified criteria, available on the queue. When the action is processed, either the message is removed from the queue and passed to the defined message consumer, or a message token is provided, which is used to retrieve the message.
2. MQCB can be used to define callback routines before starting consumption with MQCTL or it can be used from within a callback routine.
3. To use MQCB from outside of a callback routine, you must first suspend message consumption by using MQCTL and resume consumption afterward.

Message consumer callback sequence

You can configure a consumer to invoke callback at key points during the lifecycle of the consumer. For example:

- when the consumer is first registered,
- when the connection is started,
- when the connection is stopped and

- when the consumer is deregistered, either explicitly, or implicitly by an MQCLOSE.

Table 362. MQCTL verb definitions

Verb	Meaning
MQCTL(START)	MQCTL call by using the CTLSR Operation
MQCTL(STOP)	MQCTL call by using the CTLSP Operation
MQCTL(WAIT)	MQCTL call by using the CTLSW Operation

Allows the consumer to maintain state associated with the consumer. When a callback is requested by an application, the rules for consumer invocation are as follows:

REGISTER
Is always the first type of invocation of the callback.

Is always called on the same thread as the MQCB(CBREG) call.

START
Is always called synchronously with the MQCTL(START) verb.

- All START callbacks are completed before the MQCTL(START) verb returns.

Is on the same thread as the message delivery if CTLTHR is requested.

The call with start is not guaranteed if, for example, a previous callback issues MQCTL(STOP) during the MQCTL(START).

STOP No further messages or events are delivered after this call until the connection is restarted.

A STOP is guaranteed if the application was previously called for START, or a message, or an event.

DEREGISTER
Is always the last type of invocation of the callback.

Ensure that your application performs thread-based initialization and cleanup in the START and STOP callbacks. You can do non thread-based initialization and cleanup with REGISTER and DEREGISTER callbacks.

Do not make any assumptions about the life and availability of the thread other than what is stated. For example, do not rely on a thread staying alive beyond the last call to DEREGISTER. Similarly, when you have chosen not to use CTLTHR, do not assume that the thread exists whenever the connection is started.

If your application has particular requirements for thread characteristics, it can always create a thread accordingly, then use MQCTL(WAIT). This step *donates* the thread to IBM MQ for asynchronous message delivery.

Message consumer connection usage

Normally, when an application issues another MQI call while one is outstanding, the call fails with reason code RC2219.

There are special cases, however, when the application must issue a further MQI call before the previous call has completed. For example, the consumer can be invoked during an MQCB call with CBRE.

In such an instance, when as a result of the application issuing either an MQCB or MQCTL verb, the application is called back, the application is allowed to issue a further MQI call. This instance means you can issue, for example, an MQOPEN call, in the consumer function when called with a CBCCALLT type of CBCTRC. Any MQI call, except for MQDISC, is allowed.

Parameters for MQCB

The MQCB call has the following parameters:

HCONN (10-digit signed integer) - input

Manage callback function - HCONN parameter.

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN or MQCONNX call.

OPERATN (10-digit signed integer) - input

Manage callback function - OPERATN parameter.

The operation being processed on the callback defined for the specified object handle. You must specify one of the following options; if more than one option is required, the values can be added (do not add the same constant more than once) or combined by using the bitwise OR operation (if the programming language supports bit operations).

Combinations that are not valid are noted; all other combinations are valid.

CBREG
Define the callback function for the specified object handle. This operation defines the function to be called and the selection criteria to be used.

If a callback function is already defined for the object handle the definition is replaced. If an error is detected while replacing the callback, the function is deregistered.

If a callback is registered in the same callback function in which it was previously deregistered, this is treated as a replace operation; any initial or final calls are not invoked.

You can use CBREG with CTLSU or CTLRE.

CBUNR
Stop the consuming of messages for the object handle and removes the handle from those eligible for a callback.

A callback is automatically deregistered if the associated handle is closed.

If CBUNR is called from within a consumer, and the callback has a stop call defined, it is invoked upon return from the consumer.

If this operation is issued against an *Hobj* with no registered consumer, the call returns with RC2448.

CTLSU
Suspends the consuming of messages for the object handle.

If this operation is applied to an event handler, the event handler does not get events while suspended, and any events missed while in the suspended state are not provided to the operation when it is resumed.

While suspended, the consumer function continues to get the control type callbacks.

CTLRE
Resume the consuming of messages for the object handle.

If this operation is applied to an event handler, the event handler does not get events while suspended, and any events missed while in the suspended state are not provided to the operation when it is resumed.

CBDSC (MQCBD) - input

Manage callback function - CBDSC parameter.

This is a structure that identifies the callback function that is being registered by the application and the options used when registering it.

See "MQCBD - Callback descriptor" on page 2243 for details of the structure.

Callback descriptor is required only for the CBREG option; if the descriptor is not required, the parameter address passed can be null.

HOBJ (10-digit signed integer) - input

Manage callback function - HOBJ parameter.

This handle represents the access that has been established to the object from which a message is to be consumed. This is a handle that has been returned from a previous MQOPEN or MQSUB call (in the **HOBJ** parameter).

HOBJ is not required when defining an event handler routine (CBTEH) and must be specified as HONONE.

If this *Hobj* has been returned from an MQOPEN call, the queue must have been opened with one or more of the following options:

- OOINPS
- OOINPX
- OOINPQ
- OOBRW

MSGDSC (MQMD) - input

Manage callback function -MSGDSC parameter.

This structure describes the attributes of the message required, and the attributes of the message retrieved.

The **MsgDesc** parameter defines the attributes of the messages required by the consumer, and the version of the MQMD to be passed to the message consumer.

The *MsgId*, *CorrelId*, *GroupId*, *MsgSeqNumber*, and *Offset* in the MQMD are used for message selection, depending on the options specified in the **GetMsgOpts** parameter.

The *Encoding* and *CodedCharSetId* are used for message conversion if you specify the GMCONV option.

See MQMD for details.

MsgDesc is used only for CBREG and, if you require values other than the default for any fields. *MsgDesc* is not used for an event handler.

If the descriptor is not required the parameter address passed can be null.

Note, that if multiple consumers are registered against the same queue with overlapping selectors, the chosen consumer for each message is undefined.

GMO (MQGMO) - input

Manage callback function - GMO parameter.

Options that control how the message consumer gets messages.

All options have the meaning as described in "MQGMO (Get-message options) on IBM i" on page 3084, when used on an MQGET call, except:

GMSSIG
This option is not permitted.

GMBRWF, GMBRWN, GMMBH, GMMBC

The order of messages delivered to a browsing consumer is dictated by the combinations of these options. Significant combinations are:

GMBRWF

The first message on the queue is delivered repeatedly to the consumer. This is useful when the consumer destructively consumes the message in the callback. Use this option with care.

GMBRWN

The consumer is given each message on the queue, from the current cursor position until the end of the queue is reached.

GMBRWF + GMBRWN

The cursor is reset to the start of the queue. The consumer is then given each message until the cursor reaches the end of the queue.

GMBRWF + GMMBH or GMMBC

Starting at the beginning of the queue, the consumer is given the first nonmarked message on the queue, which is then marked for this consumer. This combination ensures that the consumer can receive new messages added behind the current cursor point.

GMBRWN + GMMBH or GMMBC

Starting at the cursor position the consumer is given the next nonmarked message on the queue, which is then marked for this consumer. Use this combination with care because messages can be added to the queue behind the current cursor position.

GMBRWF + GMBRWN + GMMBH or GMMBC

This combination is not permitted, if used the call returns RC2046.

GMNWT, GMWT and GMWI

These options control how the consumer is invoked.

GMNWT

The consumer is never called with RC2033. The consumer is only invoked for messages and events

GMWT with a zero GMWI

The RC2033 code is only passed to the consumer when there are no messages and

- the consumer has been started
- the consumer has been delivered at least one message since the last no messages reason code.

This prevents the consumer from polling in a busy loop when a zero wait interval is specified.

GMWT and a positive GMWI

The user is invoked after the specified wait interval with reason code RC2033. This call is made regardless of whether any messages have been delivered to the consumer. This allows the user to perform heartbeat or batch type processing.

GMWT and GMWI of WIULIM

This specifies an infinite wait before returning RC2033. The consumer is never called with RC2033.

GMO is used only for CBREG and, if you require values other than the default for any fields. *GMO* is not used for an event handler.

If the options are not required the parameter address passed can be null.

If a message properties handle is provided in the MQGMO structure, a copy is provided in the MQGMO structure that is passed into the consumer callback. On return from the MQCB call, the application can delete the message properties handle.

CMPCOD (10-digit signed integer) - output

Manage callback function - CMPCOD parameter.

The completion code; it is one of the following:

CCOK
Successful completion.

CCWARN
Warning (partial completion).

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

Manage callback function - REASON parameter.

The following reason codes are the codes that the queue manager can return for the **REASON** parameter.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CompCode* is CCFAIL:

RC2204
(2204, X'89C') Adapter not available.

RC2133
(2133, X'855') Unable to load data conversion services modules.

RC2130
(2130, X'852') Unable to load adapter service module.

RC2374
(2374, X'946') API exit failed.

RC2183
(2183, X'887') Unable to load API exit.

RC2157
(2157, X'86D') Primary and home ASIDs differ.

RC2005
(2005, X'7D5') Buffer length parameter not valid.

RC2219
(2219, X'8AB') MQI call entered before previous call complete.

RC2487
(2487, X'9B7') Incorrect callback type field.

RC2448
(2448, X'990') Unable to deregister, suspend, or resume because there is no registered callback.

RC2486
(2486, X'9B6') Either *CallbackFunction* or *CallbackName* must be specified but not both.

RC2483
(2483, X'9B3') Incorrect callback type field.

RC2484
(2484, X'9B4') Incorrect MQCBD options field.

RC2140
(2140, X'85C') Wait request rejected by CICS.

RC2009
(2009, X'7D9') Connection to queue manager lost.

RC2217
(2217, X'8A9') Not authorized for connection.

RC2202
(2202, X'89A') Connection quiescing.

RC2203
(2203, X'89B') Connection shutting down.

RC2207
(2207, X'89F') Correlation-identifier error.

RC2010
(2010, X'7DA') Data length parameter not valid.

RC2016
(2016, X'7E0') Gets inhibited for the queue.

RC2351
(2351, X'92F') Global units of work conflict.

RC2186
(2186, X'88A') Get-message options structure not valid.

RC2353
(2353, X'931') Handle in use for global unit of work.

RC2018
(2018, X'7E2') Connection handle not valid.

RC2019
(2019, X'7E3') Object handle not valid.

RC2259
(2259, X'8D3') Inconsistent browse specification.

RC2245
(2245, X'8C5') Inconsistent unit-of-work specification.

RC2246
(2246, X'8C6') Message under cursor not valid for retrieval.

RC2352
(2352, X'930') Global unit of work conflicts with local unit of work.

RC2247
(2247, X'8C7') Match options not valid.

RC2485
(2485, X'9B4') Incorrect *MaxMsgLength* field.

RC2026
(2026, X'7EA') Message descriptor not valid.

RC2497
(2497, X'9C1') The specified function entry point could not be found in the module.

RC2496
(2496, X'9C0') Module found, however it is of the wrong type; not 32 bit, 64 bit, or a valid dynamic link library.

RC2495
(2495, X'9BF') Module not found in the search path or not authorized to load.

RC2250
(2250, X'8CA') Message sequence number not valid.

RC2331
(2331, X'91B') Use of message token not valid.

RC2033
(2033, X'7F1') No message available.

RC2034
(2034, X'7F2') Browse cursor not positioned on message.

RC2036
(2036, X'7F4') Queue not open for browse.

RC2037
(2037, X'7F5') Queue not open for input.

RC2041
(2041, X'7F9') Object definition changed since opened.

RC2101
(2101, X'835') Object damaged.

RC2206
(2206, X'89E') Incorrect operation code on API Call.

RC2046
(2046, X'7FE') Options not valid or not consistent.

RC2193
(2193, X'891') Error accessing page-set data set.

RC2052
(2052, X'804') Queue has been deleted.

RC2394
(2394, X'95A') Queue has wrong index type.

RC2058
(2058, X'80A') Queue manager name not valid or not known.

RC2059
(2059, X'80B') Queue manager not available for connection.

RC2161
(2161, X'871') Queue manager quiescing.

RC2162
(2162, X'872') Queue manager shutting down.

RC2102
(2102, X'836') Insufficient system resources available.

RC2069
(2069, X'815') Signal outstanding for this handle.

RC2071
(2071, X'817') Insufficient storage available.

RC2109
(2109, X'83D') Call suppressed by exit program.

RC2024
(2024, X'7E8') No more messages can be handled within current unit of work.

RC2072
(2072, X'818') Syncpoint support not available.

RC2195
(2195, X'893') Unexpected error occurred.

RC2354
(2354, X'932') Enlistment in global unit of work failed.

RC2355
(2355, X'933') Mixture of unit-of-work calls not supported.

RC2255
(2255, X'8CF') Unit of work not available for the queue manager to use.

RC2090
(2090, X'82A') Wait interval in MQGMO not valid.

RC2256
(2256, X'8D0') Wrong version of MQGMO supplied.

RC2257
(2257, X'8D1') Wrong version of MQMD supplied.

RC2298
(2298, X'8FA') The function requested is not available in the current environment.

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQCB(HCONN : OPERATN : CBDSC :
                                HOBJ : MSGDSC : GMO :
                                DATLEN : CMPCOD : REASON)
```

The prototype definition for the call is:

```
DMQCB             PR                  EXTPROC('MQCB')
D* Connection handle
D HCONN                          10I 0 VALUE
D* Operation
D OPERATN                        10I 0 VALUE
D* Callback descriptor
D CBDSC                         180A
D* Object handle
D HOBJ                           10I 0 VALUE
D* Message Descriptor
D MSGDSC                        364A
D* Get options
D GMO                           112A
D* Completion code
D CMPCOD                         10I 0
* Reason code qualifying CompCode
D REASON                         10I 0
```

MQCLOSE (Close object) on IBM i: IBM i

The MQCLOSE call relinquishes access to an object, and is the inverse of the MQOPEN call.

Syntax

MQCLOSE *(HCONN, HOBJ, OPTS, CMPCOD, REASON)*

Usage notes

1. When an application issues the MQDISC call, or ends either normally or abnormally, any objects that were opened by the application and are still open are closed automatically with the CONONE option.
2. The following points apply if the object being closed is a *queue*:
   - If operations on the queue are performed as part of a unit of work, the queue can be closed before or after the syncpoint occurs without affecting the outcome of the syncpoint.
   - If the queue was opened with the OOBRW option, the browse cursor is destroyed. If the queue is later reopened with the OOBRW option, a new browse cursor is created (see the OOBRW option described in MQOPEN).
   - If a message is currently locked for this handle at the time of the MQCLOSE call, the lock is released (see the GMLK option described in "MQGMO (Get-message options) on IBM i" on page 3084).
3. The following points apply if the object being closed is a *dynamic queue* (either permanent or temporary):
   - For a dynamic queue, the options CODEL or COPURG can be specified regardless of the options specified on the corresponding MQOPEN call.
   - When a dynamic queue is deleted, all MQGET calls with the GMWT option that are outstanding against the queue are canceled and reason code RC2052 is returned. See the GMWT option described in "MQGMO (Get-message options) on IBM i" on page 3084.

     After a dynamic queue has been deleted, any call (other than MQCLOSE) that attempts to reference the queue using a previously acquired *HOBJ* handle fails with reason code RC2052.

     Be aware that although a deleted queue cannot be accessed by applications, the queue is not removed from the system, and associated resources are not freed, until all handles that reference the queue have been closed, and all units of work that affect the queue have been either committed or backed out.
   - When a permanent dynamic queue is deleted, if the *HOBJ* handle specified on the MQCLOSE call is not the one that was returned by the MQOPEN call that created the queue, a check is made that the user identifier which was used to validate the MQOPEN call is authorized to delete the queue. If the OOALTU option was specified on the MQOPEN call, the user identifier checked is the *ODAU*.

     This check is not performed if:

     – The handle specified is the one returned by the MQOPEN call that created the queue.

     – The queue being deleted is a temporary dynamic queue.
   - When a temporary dynamic queue is closed, if the *HOBJ* handle specified on the MQCLOSE call is the one that was returned by the MQOPEN call that created the queue, the queue is deleted. This occurs regardless of the close options specified on the MQCLOSE call. If there are messages on the queue, they are discarded; no report messages are generated.

If there are uncommitted units of work that affect the queue, the queue and its messages are still deleted, but this does not cause the units of work to fail. However, as described previously, the resources associated with the units of work are not freed until each of the units of work has been either committed or backed out.

4. The following points apply if the object being closed is a *distribution list*:
   - The only valid close option for a distribution list is CONONE; the call fails with reason code RC2046 or RC2045 if any other options are specified.
   - When a distribution list is closed, individual completion codes and reason codes are not returned for the queues in the list - only the **CMPCOD** and **REASON** parameters of the call are available for diagnostic purposes.

     If a failure occurs closing one of the queues, the queue manager continues processing and attempts to close the remaining queues in the distribution list. The **CMPCOD** and **REASON** parameters of the call are then set to return information describing the failure. Thus it is possible for the completion code to be CCFAIL, even though most of the queues were closed successfully. The queue that encountered the error is not identified.

     If there is a failure on more than one queue, it is not defined which failure is reported in the **CMPCOD** and **REASON** parameters.

Parameters

The MQCLOSE call has the following parameters:

HCONN (10-digit signed integer) - input

Connection handle.

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN or MQCONNX call.

HOBJ (10-digit signed integer) - input/output

Object handle.

This handle represents the object that is being closed. The object can be of any type. The value of *HOBJ* was returned by a previous MQOPEN call.

On successful completion of the call, the queue manager sets this parameter to a value that is not a valid handle for the environment. This value is:

HOUNUH
Unusable object handle.

OPTS (10-digit signed integer) - input

Options that control the action of MQCLOSE.

The **OPTS** parameter controls how the object is closed. Only permanent dynamic queues and subscriptions can be closed in more than one way. Permanent dynamic queues can either be retained or deleted; these are queues with a **DefinitionType** attribute that has the value QDPERM (see the **DefinitionType** attribute described in "Attributes for queues" on page 3385). The close options are summarized in a table later in this topic.

Durable subscriptions can either be kept or removed; these are created using the MQSUB call with the SODUR option.

When closing the handle to a managed destination (that is the **Hobj** parameter returned on an MQSUB call which used the SOMAN option) the queue manager will clean up any unretrieved publications when the associated subscription has also been removed. That is done using the CORMSB option on the **Hsub** parameter returned on an MQSUB call. Note that CORMSB is the default behavior on MQCLOSE for a non-durable subscription.

When closing a handle to a non-managed destination you are responsible for cleaning up the queue where publications are sent. You are recommended to close the subscription using CORMSB first and then process messages off the queue until there are none remaining.

One (and only one) of the following must be specified:

Dynamic queue closure options
These options control how permanent dynamic queues are closed:

CODEL
Delete the queue.

The queue is deleted if either of the following is true:

- It is a permanent dynamic queue, created by a previous MQOPEN call, and there are no messages on the queue and no uncommitted get or put requests outstanding for the queue (either for the current task or any other task).
- It is the temporary dynamic queue that was created by the MQOPEN call that returned *HOBJ*. In this case, all the messages on the queue are purged.

In all other cases, including the case where the *Hobj* was returned on an MQSUB call, the call fails with reason code RC2045, and the object is not deleted.

COPURG
Delete the queue, purging any messages on it.

The queue is deleted if either of the following is true:

- It is a permanent dynamic queue, created by a previous MQOPEN call, and there are no uncommitted get or put requests outstanding for the queue (either for the current task or any other task).
- It is the temporary dynamic queue that was created by the MQOPEN call that returned *HOBJ*.

In all other cases, including the case where the *Hobj* was returned on an MQSUB call, the call fails with reason code RC2045, and the object is not deleted.

The next table shows which close options are valid, and whether the object is retained or deleted.

Table 363. Valid close options for use with retained or deleted objects

Type of object or queue	CONONE	CODEL	COPURG
Object other than a queue	Retained	Not valid	Not valid
Predefined queue	Retained	Not valid	Not valid
Permanent dynamic queue	Retained	Deleted if empty and no pending updates	Messages deleted; queue deleted if no pending updates
Temporary dynamic queue (call issued by creator of queue)	Deleted	Deleted	Deleted
Temporary dynamic queue (call not issued by creator of queue)	Retained	Not valid	Not valid
Distribution list	Retained	Not valid	Not valid
Managed subscription destination	Retained	Not valid	Not valid
Distribution list (subscription has been removed)	Messages deleted; queue deleted	Not valid	Not valid

Subscription closure options
These options control whether durable subscriptions are removed when the handle is closed, and whether publications still waiting to be read by the application are cleaned up. These options are only valid for use with an object handle returned in the **HSUB** parameter of an MQSUB call.

COKPSB
The handle to the subscription is closed but the subscription made is kept. Publications will continue to be sent to the destination specified in the subscription. This option is only valid if the subscription was made with the option SODUR. COKPSB is the default if the subscription is durable

CORMSB
The subscription is removed and the handle to the subscription is closed.

The **Hobj** parameter of the MQSUB call is not invalidated by closure of the **Hsub** parameter and may continue to be used for MQGET or MQCB to receive the remaining publications. When the **Hobj** parameter of the MQSUB call is also closed, if it was a managed destination any unretrieved publications will be removed.

CORMSB is the default if the subscription is non-durable.

These subscription closure options are summarized in the following tables:
To close a durable subscription handle but leave the subscription around, use the following subscription closure options:

Task	Subscription closure option
Keep publications on an MQOPENed handle	COKPSB
Remove publications on an MQOPENed handle	Action not allowed
Keep publications on a handle with SOMAN	COKPSB
Remove publications on a handle with SOMAN	Action not allowed

To unsubscribe, either by closing a durable subscription handle and unsubscribing it or closing a non-durable subscription handle, use the following subscription closure options:

Task	Subscription closure option
Keep publications on an MQOPENed handle	CORMSB
Remove publications on an MQOPENed handle	Action not allowed
Keep publications on a handle with SOMAN	CORMSB
Remove publications on a handle with SOMAN	COPGSB

Read ahead options
The following options control what happens to non-persistent messages which have been sent to the client before an application requested them and have not yet been consumed by the application. These messages are stored in the client read ahead buffer waiting to be requested by the application and can either be discarded or consumed from the queue before the MQCLOSE is completed.

COIMM
The object is closed immediately and any messages which have been sent to the client before an application requested them are discarded and are not available to be consumed by any application. This is the default value.

COQSC
A request to close the object is made, but if any messages which have been sent to the client before an application requested them, still reside in the client read ahead buffer, the MQCLOSE call will return with a warning code of RC2458, and the object handle will remain valid.

The application can then continue to use the object handle to retrieve messages until no more are available, and then close the object again. No more messages will be sent to the client ahead of an application requesting then, read ahead is now turned off.

Applications are advised to use COQSC rather than trying to reach a point where there are no more messages in the client read ahead buffer, since a message could arrive between the last MQGET call and the following MQCLOSE which would be discarded if COIMM was used.

If an MQCLOSE with COQSC is issued from within an asynchronous callback function, the same behavior of reading ahead messages applies. If the warning code RC2458 is returned, then the callback function will be called at least one more time. When the last remaining message that was read ahead has been passed to the callback function the CBCFLG field is set to CBCFBE.

Default option
If you require none of the options described previously, you can use the following option:

CONONE
No optional close processing required.

This must be specified for:

- Objects other than queues
- Predefined queues
- Temporary dynamic queues (but only in those cases where *HOBJ* is not the handle returned by the MQOPEN call that created the queue).
- Distribution lists

In all of the previous cases, the object is retained and not deleted.

If this option is specified for a temporary dynamic queue:

- The queue is deleted, if it was created by the MQOPEN call that returned *HOBJ* ; any messages that are on the queue are purged.
- In all other cases the queue (and any messages on it) are retained.

If this option is specified for a permanent dynamic queue, the queue is retained and not deleted.

CMPCOD (10-digit signed integer) - output

Completion code.

It is one of the following:

CCOK
Successful completion.

CCWARN
Warning (partial completion).

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

Reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCWARN:

RC2241
(2241, X'8C1') Message group not complete.

RC2242
(2242, X'8C2') Logical message not complete.

If *CMPCOD* is CCFAIL:

RC2219
(2219, X'8AB') MQI call reentered before previous call complete.

RC2009
(2009, X'7D9') Connection to queue manager lost.

RC2018
(2018, X'7E2') Connection handle not valid.

RC2019
(2019, X'7E3') Object handle not valid.

RC2035
(2035, X'7F3') Not authorized for access.

RC2101
(2101, X'835') Object damaged.

RC2045
(2045, X'7FD') Option not valid for object type.

RC2046
(2046, X'7FE') Options not valid or not consistent.

RC2058
(2058, X'80A') Queue manager name not valid or not known.

RC2059
(2059, X'80B') Queue manager not available for connection.

RC2162
(2162, X'872') Queue manager shutting down.

RC2055
(2055, X'807') Queue contains one or more messages or uncommitted put or get requests.

RC2102
(2102, X'836') Insufficient system resources available.

RC2063
(2063, X'80F') Security error occurred.

RC2071
(2071, X'817') Insufficient storage available.

RC2195
(2195, X'893') Unexpected error occurred.

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQCLOSE(HCONN : HOBJ : OPTS :
C                                       CMPCOD : REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQCLOSE          PR                  EXTPROC('MQCLOSE')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Object handle
D HOBJ                          10I 0
```

```
D* Options that control the action of MQCLOSE
D OPTS                          10I 0 VALUE
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
```

MQCMIT (Commit changes) on IBM i: IBM i

The MQCMIT call indicates to the queue manager that the application has reached a syncpoint, and that all of the message gets and puts that have occurred since the last syncpoint are to be made permanent. Messages put as part of a unit of work are made available to other applications; messages retrieved as part of a unit of work are deleted.

Syntax

MQCMIT *(HCONN, COMCOD, REASON)*

Usage notes

Consider these usage notes when using MQCMIT.

1. This call can be used only when the queue manager itself coordinates the unit of work. This is a local unit of work, where the changes affect only IBM MQ resources.
2. In environments where the queue manager does not coordinate the unit of work, the appropriate commit call must be used instead of MQCMIT. The environment may also support an implicit commit caused by the application terminating normally.
   - On IBM i, this call can be used for local units of work coordinated by the queue manager. This means that a commitment definition must not exist at job level, that is, the `STRCMTCTL` command with the **`CMTSCOPE(*JOB)`** parameter must not have been issued for the job.
3. If an application ends with uncommitted changes in a unit of work, the disposition of those changes depends on whether the application ends normally or abnormally. See the usage notes in "MQDISC (Disconnect queue manager) on IBM i" on page 3305 for further details.
4. When an application puts or gets messages in groups or segments of logical messages, the queue manager retains information relating to the message group and logical message for the last successful MQPUT and MQGET calls. This information is associated with the queue handle, and includes such things as:
   - The values of the *MDGID*, *MDSEQ*, *MDOFF*, and *MDMFL* fields in MQMD.
   - Whether the message is part of a unit of work.
   - For the MQPUT call: whether the message is persistent or nonpersistent.

   When a unit of work is committed, the queue manager retains the group and segment information, and the application can continue putting or getting messages in the current message group or logical message.

   Retaining the group and segment information when a unit of work is committed allows the application to spread a large message group or large logical message consisting of many segments across several units of work. Using several units of work might be advantageous if the local queue manager has only limited queue storage. However, the application must maintain sufficient information to be able to restart putting or getting messages at the correct point if a system failure occurs. For details of how to restart at the correct point after a system failure, see the PMLOGO

option described in "MQPMO (Put-message options) on IBM i" on page 3185, and the GMLOGO option described in "MQGMO (Get-message options) on IBM i" on page 3084.

The remaining usage notes apply only when the queue manager coordinates the units of work:

1. A unit of work has the same scope as a connection handle. This means that all IBM MQ calls which affect a particular unit of work must be performed using the same connection handle. Calls issued using a different connection handle (for example, calls issued by another application) affect a different unit of work. See the **HCONN** parameter described in MQCONN for information about the scope of connection handles.
2. Only messages that were put or retrieved as part of the current unit of work are affected by this call.
3. A long-running application that issues MQGET, MQPUT, or MQPUT1 calls within a unit of work, but which never issues a commit or back-out call, can cause queues to fill up with messages that are not available to other applications. To guard against this possibility, the administrator should set the **MaxUncommittedMsgs** queue manager attribute to a value that is low enough to prevent runaway applications filling the queues, but high enough to allow the expected messaging applications to work correctly.

Parameters

The MQCMIT call has the following parameters:

HCONN (10-digit signed integer) - input

Connection handle.

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN or MQCONNX call.

COMCOD (10-digit signed integer) - output

Completion code.

It is one of the following:

CCOK
Successful completion.

CCWARN
Warning (partial completion).

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

Reason code qualifying *COMCOD*.

If *COMCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *COMCOD* is CCWARN:

RC2003
(2003, X'7D3') Unit of work backed out.

RC2124
(2124, X'84C') Result of commit operation is pending.

If *COMCOD* is CCFAIL:

RC2219
(2219, X'8AB') MQI call reentered before previous call complete.

RC2009
(2009, X'7D9') Connection to queue manager lost.

RC2018
(2018, X'7E2') Connection handle not valid.

RC2101
(2101, X'835') Object damaged.

RC2123
(2123, X'84B') Result of commit or back-out operation is mixed.

RC2162
(2162, X'872') Queue manager shutting down.

RC2102
(2102, X'836') Insufficient system resources available.

RC2071
(2071, X'817') Insufficient storage available.

RC2195
(2195, X'893') Unexpected error occurred.

RPG Declaration

```
C*..1...:....2...:....3...:....4...:....5...:....6...:....7..
C                   CALLP     MQCMIT(HCONN : COMCOD : REASON)
```

The prototype definition for the call is:

```
D*..1...:....2...:....3...:....4...:....5...:....6...:....7..
DMQCMIT           PR                  EXTPROC('MQCMIT')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Completion code
D COMCOD                        10I 0
D* Reason code qualifying COMCOD
D REASON                        10I 0
```

MQCONN (Connect queue manager) on IBM i: 

The MQCONN call connects an application program to a queue manager. It provides a queue manager connection handle, which is used by the application on subsequent message queuing calls.

- Applications must use the MQCONN or MQCONNX call to connect to the queue manager, and the MQDISC call to disconnect from the queue manager.

On IBM MQ for Windows, UNIX, and IBM i, each thread in an application can connect to different queue managers. On other systems, all concurrent connections within a process must be to the same queue manager.

- "Syntax"
- "Usage notes" on page 3291
- "Parameters" on page 3291
- "RPG Declaration" on page 3294

Syntax

MQCONN *(QMNAME, HCONN, CMPCOD, REASON)*

Usage notes

1. The queue manager to which connection is made using the MQCONN call is called the *local queue manager*.
2. Queues that are owned by the local queue manager appear to the application as local queues. It is possible to put messages on and get messages from these queues.

   Shared queues that are owned by the queue-sharing group to which the local queue manager belongs appear to the application as local queues. It is possible to put messages on and get messages from these queues.

   Queues that are owned by remote queue managers appear as remote queues. It is possible to put messages on these queues, but not possible to get messages from these queues.
3. If the queue manager fails while an application is running, the application must issue the MQCONN call again in order to obtain a new connection handle to use on subsequent IBM MQ calls. The application can issue the MQCONN call periodically until the call succeeds.

   If an application is not sure whether it is connected to the queue manager, the application can safely issue an MQCONN call in order to obtain a connection handle. If the application is already connected, the handle returned is the same as that returned by the previous MQCONN call, but with completion code CCWARN and reason code RC2002.
4. When the application has finished using IBM MQ calls, the application should use the MQDISC call to disconnect from the queue manager.
5. On IBM i, programs that end abnormally are not automatically disconnected from the queue manager. Therefore applications should be written to allow for the possibility of the MQCONN or MQCONNX call returning completion code CCWARN and reason code RC2002. The connection handle returned in this situation can be used as normal.

Parameters

The MQCONN call has the following parameters:

QMNAME (48-byte character string) - input

Name of queue manager.

This is the name of the queue manager to which the application wants to connect. The name can contain the following characters:

- Uppercase alphabetic characters (A through Z)
- Lowercase alphabetic characters (a through z)
- Numeric digits (0 through 9)
- Period (.), forward slash (/), underscore (_), percent (%)

The name must not contain leading or embedded blanks, but might contain trailing blanks. A null character can be used to indicate the end of significant data in the name; the null and any characters following it are treated as blanks. The following restrictions apply in the environments indicated:

- On IBM i, names containing lowercase characters, forward slash, or percent must be enclosed in quotation marks when specified on commands. These quotation marks must not be specified in the **QMNAME** parameter.

If the name consists entirely of blanks, the name of the *default* queue manager is used.

The name specified for *QMNAME* must be the name of a *connectable* queue manager.

Queue-sharing groups: On systems where several queue managers exist and are configured to form a queue-sharing group, the name of the queue-sharing group can be specified for *QMNAME* in place of the name of a queue manager. This allows the application to connect to *any* queue

manager that is available in the queue-sharing group. The system can also be configured so that a blank *QMNAME* causes connection to the queue-sharing group instead of to the default queue manager.

If *QMNAME* specifies the name of the queue-sharing group, but there is also a queue manager with that name on the system, connection is made to the latter in preference to the former. Only if that connection fails is connection to one of the queue managers in the queue-sharing group attempted.

If the connection is successful, the handle returned by the MQCONN or MQCONNX call can be used to access *all* of the resources (both shared and nonshared) that belong to the particular queue manager to which connection has been made. Access to these resources is subject to the typical authorization controls.

If the application issues two MQCONN or MQCONNX calls in order to establish concurrent connections, and one or both calls specifies the name of the queue-sharing group, the second call may return completion code CCWARN and reason code RC2002. This occurs when the second call connects to the same queue manager as the first call.

Queue-sharing groups are supported only on z/OS. Connection to a queue-sharing group is supported only in the batch, RRS batch, and TSO environments.

IBM MQ client applications: For IBM MQ MQI client applications, a connection is attempted for each client-connection channel definition with the specified queue manager name, until one is successful. The queue manager, however, must have the same name as the specified name. If an all-blank name is specified, each client-connection channel with an all-blank queue manager name is tried until one is successful; in this case there is no check against the actual name of the queue manager.

IBM MQ client queue manager groups: If the specified name starts with an asterisk (*), the actual queue manager to which connection is made may have a name that is different from that specified by the application. The specified name (without the asterisk) defines a *group* of queue managers that are eligible for connection. The implementation selects one from the group by trying each one in turn, in alphabetic order, until one is found to which a connection can be made. If none of the queue managers in the group is available for connection, the call fails. Each queue manager is tried once only. If an asterisk alone is specified for the name, an implementation-defined default queue manager group is used.

Queue manager groups are supported only for applications running in an MQ-client environment; the call fails if a non-client application specifies a queue manager name beginning with an asterisk. A group is defined by providing several client connection channel definitions with the same queue manager name (the specified name without the asterisk), to communicate with each of the queue managers in the group. The default group is defined by providing one or more client connection channel definitions, each with a blank queue manager name (specifying an all-blank name therefore has the same effect as specifying a single asterisk for the name for a client application).

After connecting to one queue manager of a group, an application can specify blanks in the typical way in the queue manager name fields in the message and object descriptors to mean the name of the queue manager to which the application has actually connected (the *local queue manager*). If the application needs to know this name, the MQINQ call can be issued to inquire the **QMgrName** queue manager attribute.

Prefixing an asterisk to the connection name implies that the application is not dependent on connecting to a particular queue manager in the group. Suitable applications would be:

- Applications that put messages but do not get messages.
- Applications that put request messages and then get the reply messages from a *temporary dynamic* queue.

Unsuitable applications would be those that need to get messages from a particular queue at a particular queue manager; such applications should not prefix the name with an asterisk.

Note that if an asterisk is specified, the maximum length of the remainder of the name is 47 characters.

The length of this parameter is given by LNQMN.

HCONN (10-digit signed integer) - output

Connection handle.

This handle represents the connection to the queue manager. It must be specified on all subsequent message queuing calls issued by the application. It ceases to be valid when the MQDISC call is issued, or when the unit of processing that defines the scope of the handle terminates.

The scope of the handle is restricted to the smallest unit of parallel processing supported by the platform on which the application is running; the handle is not valid outside the unit of parallel processing from which the MQCONN call was issued.

- On IBM i, the scope of the handle is the job issuing the call.

CMPCOD (10-digit signed integer) - output

Completion code.

It is one of the following:

CCOK
Successful completion.

CCWARN
Warning (partial completion).

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

Reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCWARN:

RC2002
(2002, X'7D2') Application already connected.

If *CMPCOD* is CCFAIL:

RC2219
(2219, X'8AB') MQI call reentered before previous call complete.

RC2267
(2267, X'8DB') Unable to load cluster workload exit.

RC2009
(2009, X'7D9') Connection to queue manager lost.

RC2018
(2018, X'7E2') Connection handle not valid.

RC2035
(2035, X'7F3') Not authorized for access.

RC2137
(2137, X'859') Object not opened successfully.

RC2058
(2058, X'80A') Queue manager name not valid or not known.

RC2059
(2059, X'80B') Queue manager not available for connection.

RC2161
(2161, X'871') Queue manager quiescing.

RC2162
(2162, X'872') Queue manager shutting down.

RC2102
(2102, X'836') Insufficient system resources available.

RC2063
(2063, X'80F') Security error occurred.

RC2071
(2071, X'817') Insufficient storage available.

RC2195
(2195, X'893') Unexpected error occurred.

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQCONN(QMNAME : HCONN : CMPCOD :
C                                      REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQCONN           PR                  EXTPROC('MQCONN')
D* Name of queue manager
D QMNAME                        48A
D* Connection handle
D HCONN                         10I 0
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
```

MQCONNX (Connect queue manager (extended)) on IBM i: 

The MQCONNX call connects an application program to a queue manager. It provides a queue manager connection handle, which is used by the application on subsequent IBM MQ calls.

The MQCONNX call is like the MQCONN call, except that MQCONNX allows options to be specified to control the way that the call works.

On IBM MQ for Windows, UNIX, and IBM i, each thread in an application can connect to different queue managers. On other systems, all concurrent connections within a process must be to the same queue manager.

- "Syntax" on page 3295
- "Parameters" on page 3295
- "RPG Declaration" on page 3295

Syntax

MQCONNX *(QMNAME, CNOPT, HCONN, CMPCOD, REASON)*

Parameters

The MQCONNX call has the following parameters:

QMNAME (48-byte character string) - input

Name of queue manager.

See the **QMNAME** parameter described in "MQCONN (Connect queue manager) on IBM i" on page 3290 for details.

CNOPT (MQCNO) - input/output

Options that control the action of MQCONNX.

See "MQCNO (Connect options) on IBM i" on page 3054 for details.

HCONN (10-digit signed integer) - output

Connection handle.

See the **HCONN** parameter described in "MQCONN (Connect queue manager) on IBM i" on page 3290 for details.

CMPCOD (10-digit signed integer) - output

Completion code.

See the **CMPCOD** parameter described in "MQCONN (Connect queue manager) on IBM i" on page 3290 for details.

REASON (10-digit signed integer) - output

Reason code qualifying *CMPCOD*.

See the **REASON** parameter described in "MQCONN (Connect queue manager) on IBM i" on page 3290 for details of possible reason codes.

The following additional reason codes can be returned by the MQCONNX call:

If *CMPCOD* is CCFAIL:

RC2278
(2278, X'8E6') Client connection fields not valid.

RC2139
(2139, X'85B') Connect-options structure not valid.

RC2046
(2046, X'7FE') Options not valid or not consistent.

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQCONN(QMNAME : HCONN : CMPCOD :
C                                      REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQCONN           PR                  EXTPROC('MQCONN')
D* Name of queue manager
D QMNAME                         48A
D* Options that control the action of MQCONNX
D HCONN                         224A
```

```
D* Connection handle
D HCONN                         10I 0
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
```

MQCRTMH (Create message handle) on IBM i: 

The MQCRTMH call returns a message handle.

An application can use it on subsequent message queuing calls:

- Use the MQSETMP call to set a property of the message handle.
- Use the MQINQMP call to inquire on the value of a property of the message handle.
- Use the MQDLTMP call to delete a property of the message handle.

The message handle can be used on the MQPUT and MQPUT1 calls to associate the properties of the message handle with the properties of the message being put. Similarly, by specifying a message handle on the MQGET call, the properties of the message being retrieved can be accessed by using the message handle when the MQGET call completes.

Use MQDLTMH to delete the message handle.

- "Syntax"
- "Parameters"
- "RPG Declaration" on page 3298

Syntax

MQCRTMH *(Hconn, CrtMsgHOpts, Hmsg, CompCode, Reason)*

Parameters

The MQCRTMH call has the following parameters:

HCONN (10-digit signed integer) - input

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN or MQCONNX call. If the connection to the queue manager ceases to be valid and no IBM MQ call is operating on the message handle, MQDLTMH is implicitly called to delete the message.

Alternatively, you can specify the following value:

HCUNAS

The connection handle does not represent a connection to any particular queue manager.

When this value is used, the message handle must be deleted with an explicit call to MQDLTMH in order to release any storage allocated to it; IBM MQ never implicitly deletes the message handle.

There must be at least one valid connection to a queue manager established on the thread creating the message handle, otherwise the call fails with RC2018.

CRTOPT (MQCMHO) - input

The options that control the action of MQCRTMH. See MQCMHO for details.

HMSG (20-digit signed integer) - output

On output a message handle is returned that can be used to set, inquire, and delete properties of the message handle. Initially the message handle contains no properties.

A message handle also has an associated message descriptor. Initially this message descriptor contains the default values. The values of the associated message descriptor fields can be set and inquired by using the MQSETMP and MQINQMP calls. The MQDLTMP call resets a field of the message descriptor back to its default value.

If the *HCONN* parameter is specified as the value HCUNAS then the returned message handle can be used on MQGET, MQPUT, or MQPUT1 calls with any connection within the unit of processing, but can be in use by only one IBM MQ call at a time. If the handle is in use when a second IBM MQ call attempts to use the same message handle, the second IBM MQ call fails with reason code RC2499.

If the *HCONN* parameter is not HCUNAS then the returned message handle can be used only on the specified connection.

The same *HCONN* parameter value must be used on the subsequent MQI calls where this message handle is used:

- MQDLTMH
- MQSETMP
- MQINQMP
- MQDLTMP
- MQMHBUF
- MQBUFMH

The returned message handle ceases to be valid when the MQDLTMH call is issued for the message handle, or when the unit of processing that defines the scope of the handle terminates. MQDLTMH is called implicitly if a specific connection is supplied when the message handle is created and the connection to the queue manager ceases to be valid, for example, if MQDBC is called.

CMPCOD (10-digit signed integer) - output

The completion code; it is one of the following:

CCOK
Successful completion.

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

The reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCFAIL:

RC2204
(2204, X'089C') Adapter not available.

RC2130
(2130, X'852') Unable to load adapter service module.

RC2157
(2157, X'86D') Primary and home ASIDs differ.

RC2219
(2219, X'08AB') MQI call entered before previous call completed.

RC2461
(2461, X'099D') Create message handle options structure not valid.

RC2273
(2273, X'7D9') Connection to queue manager lost.

RC2017
(2017, X'07E1') No more handles available.

RC2018
(2018, X'7E2') Connection handle not valid.

RC2460
(2460, X'099C') Message handle pointer not valid.

RC2046
(2046, X'07FE') Options not valid or not consistent.

RC2071
(2071, X'817') Insufficient storage available.

RC2195
(2195, X'893') Unexpected error occurred.

See “Return codes for IBM i (ILE RPG)” on page 3452 for more details.

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQCRTMH(HCONN : CRTOPT : HMSG :
                                        CMPCOD : REASON)
```

The prototype definition for the call is:

```
DMQCRTMH          PR                  EXTPROC('MQCRTMH')
D* Connection handle
D HCONN                           10I 0 VALUE
D* Options that control the action of MQCRTMH
D CRTOPT                          12A
D* Message handle
D HMSG                            20I 0
D* Completion code
D CMPCOD                          10I 0
D* Reason code qualifying CompCode
D REASON                          10I 0
```

MQCTL (Control callback) on IBM i: IBM i

The MQCTL call performs controlling actions on the object handles opened for a connection.

- "Syntax"
- "Usage notes"
- "Parameters"
- "RPG Declaration" on page 3304

Syntax

MQCTL *(Hconn, Operation, ControlOpts, CompCode, Reason)*

Usage notes

1. Callback routines must check the responses from all services they invoke, and if the routine detects a condition that cannot be resolved, it must issue an MQCB(CBREG) command to prevent repeated calls to the callback routine.

Parameters

The MQCTL call has the following parameters:

HCONN (10-digit signed integer) - input

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN or MQCONNX call.

OPERATN (10-digit signed integer) - input

The operation being processed on the callback defined for the specified object handle. You must specify one, and one only, of the following options:

CTLSR

Start the consuming of messages for all defined message consumer functions for the specified connection handle.

Callbacks run on a thread started by the system, which is different from any of the application threads.

This operation gives control of the provided connection handle to system. The only MQI calls which can be issued by a thread other than the consumer thread are:

- MQCTL with Operation CTLSP
- MQCTL with Operation CTLSU
- MQDISC - This performs MQCTL with Operation CTLSP before disconnection the HConn.

RC2500 is returned if an IBM MQ API call is issued while the connection handle is started, and the call does not originate from a message consumer function.

If a connection fails, this stops the conversation as soon as possible. It is possible, therefore, for an IBM MQ API call being issued on the main thread to receive the return code RC2500 for a while, followed by the return code RC2009 when the connection reverts to the stopped state.

This can be issued in a consumer function. For the same connection as the callback routine, its only purpose is to cancel a previously issued CTLSP operation.

This option is not supported if the application is bound with a nonthreaded IBM MQ library.

CTLSW

Start the consuming of messages for all defined message consumer functions for the specified connection handle.

Message consumers run on the same thread and control is not returned to the caller of MQCTL until:

- Released by the use of the MQCTL CTLSP or CTLSU operations, or
- All consumer routines have been deregistered or suspended.

If all consumers are deregistered or suspended, an implicit CTLSP operation is issued.

This option cannot be used from within a callback routine, either for the current connection handle or any other connection handle. If the call is attempted it returns with RC2012.

If, at any time during a CTLSW operation there are no registered, non-suspended consumers the call fails with a reason code of RC2446.

If, during a CTLSW operation, the connection is suspended, the MQCTL call returns a warning reason code of RC2521; the connection remains 'started'.

The application can choose to issue CTLSP or CTLRE. In this instance, the CTLRE operation blocks.

This option is not supported in a single threaded client.

CTLSP

Stop the consuming of messages, and wait for all consumers to complete their operations before this option completes. This operation releases the connection handle.

If issued from within a callback routine, this option does not take effect until the routine exits. No more message consumer routines are called after the consumer routines for messages already read have completed, and after stop calls (if requested) to callback routines have been made.

If issued outside a callback routine, control does not return to the caller until the consumer routines for messages already read have completed, and after stop calls (if requested) to callbacks have been made. The callbacks themselves, however, remain registered.

This function has no effect on read ahead messages. You must ensure that consumers run MQCLOSE(COQSC), from within the callback function, to determine whether there are any further messages available to be delivered.

CTLSU

Pause the consuming of messages. This operation releases the connection handle.

This does not affect the reading ahead of messages for the application. If you intend to stop consuming messages for a long period, consider closing the queue and reopening it when consumption must continue.

If issued from within a callback routine, it does not take effect until the routine exits. No more message consumer routines will be called after the current routine exits.

If issued outside a callback, control does not return to the caller until the current consumer routine has completed and no more are called.

CTLRE

Resume the consuming of messages.

This option is normally issued from the main application thread, but it can also be used from within a callback routine to cancel an earlier suspension request issued in the same routine.

If CTLRE is used to resume a CTLSW, then the operation blocks.

PCTLOP (MQCTLO) - input

Options that control the action of MQCTL

See MQCTLO for details of the structure.

CMPCOD (10-digit signed integer) - output

The completion code; it is one of the following:

CCOK
Successful completion.

CCWARN
Warning (partial completion).

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

The following reason codes are the ones that the queue manager can return for the **Reason** parameter.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCFAIL:

RC2133
(2133, X'855') Unable to load data conversion services modules.

RC2204
(2204, X'89C') Adapter not available.

RC2130
(2130, X'852') Unable to load adapter service module.

RC2374
(2374, X'946') API exit failed.

RC2183
(2183, X'887') Unable to load API exit.

RC2157
(2157, X'86D') Primary and home ASIDs differ.

RC2005
(2005, X'7D5') Buffer length parameter not valid.

RC2487
(2487, X'9B7') Unable to call the callback routine

RC2448
(2448, X'990') Unable to Deregister, Suspend, or Resume because there is no registered callback

RC2486
(2486, X'9B6') Either, both CallbackFunction and CallbackName have been specified on a CBREG call, or either one of CallbackFunction or CallbackName has been specified but does not match the currently registered callback function.

RC2483
(2483, X'9B3') Incorrect CallBackType field.

RC2219
(2219, X'8AB') MQI call entered before previous call complete.

RC2444
(2444, X'98C') Option block is incorrect.

RC2484
(2484, X'9B4') Incorrect MQCBD options field.

RC2140
(2140, X'85C') Wait request rejected by CICS.

RC2009
(2009, X'7D9') Connection to queue manager lost.

RC2217
(2217, X'8A9') Not authorized for connection.

RC2202
(2202, X'89A') Connection quiescing.

RC2203
(2203, X'89B') Connection shutting down.

RC2207
(2207, X'89F') Correlation-identifier error.

RC2016
(2016, X'7E0') Gets inhibited for the queue.

RC2351
(2351, X'92F') Global units of work conflict.

RC2186
(2186, X'88A') Get-message options structure not valid.

RC2353
(2353, X'931') Handle in use for global unit of work.

RC2018
(2018, X'7E2') Connection handle not valid.

RC2019
(2019, X'7E3') Object handle not valid.

RC2259
(2259, X'8D3') Inconsistent browse specification.

RC2245
(2245, X'8C5') Inconsistent unit-of-work specification.

RC2246
(2246, X'8C6') Message under cursor not valid for retrieval.

RC2352
(2352, X'930') Global unit of work conflicts with local unit of work.

RC2247
(2247, X'8C7') Match options not valid.

RC2485
(2485, X'9B5') Incorrect MaxMsgLength field

RC2026
(2026, X'7EA') Message descriptor not valid.

RC2497
(2497, X'9C1')The specified function entry point was not be found in the module.

RC2496
(2496, X'9C0') Module is found but is of the wrong type (32-bit or 64-bit) or is not a valid dll.

RC2495
(2495, X'9BF') Module not found in the search path or not authorized to load.

RC2206
(2206, X'89E') Message-identifier error.

RC2250
(2250, X'8CA') Message sequence number not valid.

RC2331
(2331, X'91B') Use of message token not valid.

RC2036
(2036, X'7F4') Queue not open for browse.

RC2037
(2037, X'7F5') Queue not open for input.

RC2041
(2041, X'7F9') Object definition changed since opened.

RC2101
(2101, X'835') Object damaged.

RC2488
(2488, X'9B8') Incorrect Operation code on API Call

RC2046
(2046, X'7FE') Options not valid or not consistent.

RC2193
(2193, X'891') Error accessing page-set data set.

RC2052
(2052, X'804') Queue has been deleted.

RC2394
(2394, X'95A') Queue has wrong index type.

RC2058
(2058, X'80A') Queue manager name not valid or not known.

RC2059
(2059, X'80B') Queue manager not available for connection.

RC2161
(2161, X'871') Queue manager quiescing.

RC2162
(2162, X'872') Queue manager shutting down.

RC2102
(2102, X'836') Insufficient system resources available.

RC2069
(2069, X'815') Signal outstanding for this handle.

RC2071
(2071, X'817') Insufficient storage available.

RC2109
(2109, X'83D') Call suppressed by exit program.

RC2072
(2072, X'818') Syncpoint support not available.

RC2195
(2195, X'893') Unexpected error occurred.

RC2354
(2354, X'932') Enlistment in global unit of work failed.

RC2355
(2355, X'933') Mixture of unit-of-work calls not supported.

RC2255
(2255, X'8CF') Unit of work not available for the queue manager to use.

RC2090
(2090, X'82A') Wait interval in MQGMO not valid.

RC2256
(2256, X'8D0') Wrong version of MQGMO supplied.

RC2257
(2257, X'8D1') Wrong version of MQMD supplied.

RC2298
(2298, X'8FA') The function requested is not available in the current environment.

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQCTL(HCONN : OPERATN : PCTLOP :
                                      CMPCOD : REASON)
```

The prototype definition for the call is:

```
DMQCTL            PR                  EXTPROC('MQCTL')
D* Connection handle
D HCONN                           10I 0 VALUE
D* Operation
D OPERATN                         10I 0 VALUE
D* Control options
D PCTLOP                          32A
D* Completion code
D CMPCOD                          10I 0
D* Reason code qualifying CompCode
D REASON                          10I 0
```

MQDISC (Disconnect queue manager) on IBM i: IBM i

The MQDISC call breaks the connection between the queue manager and the application program, and is the inverse of the MQCONN or MQCONNX call.

- "Syntax"
- "Usage notes"
- "Parameters"
- "RPG Declaration" on page 3306

Syntax

MQDISC *(HCONN, CMPCOD, REASON)*

Usage notes

1. If an MQDISC call is issued when the application still has objects open, those objects are closed by the queue manager, with the close options set to CONONE.
2. If the application ends with uncommitted changes in a unit of work, the disposition of those changes depends on how the application ends:
   a. If the application issues the MQDISC call before ending:
      - For a queue manager coordinated unit of work, the queue manager issues the MQCMIT call on behalf of the application. The unit of work is committed if possible, and backed out if not.
      - For an externally coordinated unit of work, there is no change in the status of the unit of work; however, the queue manager will indicate that the unit of work should be committed, when asked by the unit-of-work coordinator.
   b. If the application ends normally but without issuing the MQDISC call, the unit of work is backed out.
   c. If the application ends *abnormally* without issuing the MQDISC call, the unit of work is backed out.

Parameters

The MQDISC call has the following parameters:

HCONN (10-digit signed integer) - input/output

Connection handle.

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN or MQCONNX call.

On successful completion of the call, the queue manager sets *HCONN* to a value that is not a valid handle for the environment. This value is:

HCUNUH
Unusable connection handle.

CMPCOD (10-digit signed integer) - output

Completion code.

It is one of the following:

CCOK
Successful completion.

CCWARN
Warning (partial completion).

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

Reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCFAIL:

RC2219
(2219, X'8AB') MQI call reentered before previous call complete.

RC2009
(2009, X'7D9') Connection to queue manager lost.

RC2018
(2018, X'7E2') Connection handle not valid.

RC2058
(2058, X'80A') Queue manager name not valid or not known.

RC2059
(2059, X'80B') Queue manager not available for connection.

RC2162
(2162, X'872') Queue manager shutting down.

RC2102
(2102, X'836') Insufficient system resources available.

RC2071
(2071, X'817') Insufficient storage available.

RC2195
(2195, X'893') Unexpected error occurred.

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQDISC(HCONN : CMPCOD : REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQDISC           PR                  EXTPROC('MQDISC')
D* Connection handle
D HCONN                         10I 0
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
```

MQDLTMH (Delete message handle) on IBM i: IBM i

The MQDLTMH call deletes a message handle and is the inverse of the MQCRTMH call.

Syntax

MQDLTMH *((Hconn, Hmsg, DltMsgHOpts, CompCode, Reason)*

Usage notes

1. You can use this call only when the queue manager itself coordinates the unit of work. This can be:
   - A local unit of work, where the changes affect only IBM MQ resources.
   - A global unit of work, where the changes can affect resources belonging to other resource managers, as well as affecting IBM MQ resources.

   For further details about local and global units of work, see "MQBEGIN (Begin unit of work) on IBM i" on page 3267.
2. In environments where the queue manager does not coordinate the unit of work, use the appropriate back-out call instead of MQBACK. The environment might also support an implicit back out caused by the application terminating abnormally.
   - On z/OS, use the following calls:
     - Batch programs (including IMS batch DL/I programs) can use the MQBACK call if the unit of work affects only IBM MQ resources. However, if the unit of work affects both IBM MQ resources and resources belonging to other resource managers (for example, Db2), use the SRRBACK call provided by the z/OS Recoverable Resource Service (RRS). The SRRBACK call backs out changes to resources belonging to the resource managers that have been enabled for RRS coordination.
     - CICS applications must use the `EXEC CICS SYNCPOINT ROLLBACK` command to back out the unit of work. Do not use the MQBACK call for CICS applications.
     - IMS applications (other than batch DL/I programs) must use IMS calls such as `ROLB` to back out the unit of work. Do not use the MQBACK call for IMS applications (other than batch DL/I programs).
   - On IBM i, use this call for local units of work coordinated by the queue manager. This means that a commitment definition must not exist at job level, that is, the `STRCMTCTL` command with the **`CMTSCOPE(*JOB)`** parameter must not have been issued for the job.
3. If an application ends with uncommitted changes in a unit of work, the disposition of those changes depends on whether the application ends normally or abnormally. See the usage notes in "MQDISC (Disconnect queue manager) on IBM i" on page 3305 for further details.
4. When an application puts or gets messages in groups or segments of logical messages, the queue manager retains information relating to the message group and logical message for the last successful MQPUT and MQGET calls. This information is associated with the queue handle, and includes such things as:
   - The values of the *GroupId*, *MsgSeqNumber*, *Offset*, and *MsgFlags* fields in MQMD.
   - Whether the message is part of a unit of work.
   - For the MQPUT call: whether the message is persistent or nonpersistent.

   The queue manager keeps three sets of group and segment information, one set for each of the following:
   - The last successful MQPUT call (this can be part of a unit of work).

- The last successful MQGET call that removed a message from the queue (this can be part of a unit of work).
- The last successful MQGET call that browsed a message on the queue (this cannot be part of a unit of work).

If the application puts or gets the messages as part of a unit of work, and the application then backs out the unit of work, the group and segment information is restored to the value that it had previously:

- The information associated with the MQPUT call is restored to the value that it had before the first successful MQPUT call for that queue handle in the current unit of work.
- The information associated with the MQGET call is restored to the value that it had before the first successful MQGET call for that queue handle in the current unit of work.

Queues that were updated by the application after the unit of work started, but outside the scope of the unit of work, do not have their group and segment information restored if the unit of work is backed out.

Restoring the group and segment information to its previous value when a unit of work is backed out allows the application to spread a large message group or large logical message consisting of many segments across several units of work, and to restart at the correct point in the message group or logical message if one of the units of work fails. Using several units of work might be advantageous if the local queue manager has only limited queue storage. However, the application must maintain sufficient information to be able to restart putting or getting messages at the correct point if that a system failure occurs.

For details of how to restart at the correct point after a system failure, see the PMLOGO option described in PMOPT (10 digit signed integer), and the GMLOGO option described in GMOPT (10 digit signed integer).

The remaining usage notes apply only when the queue manager coordinates the units of work:

5. A unit of work has the same scope as a connection handle. All IBM MQ calls that affect a particular unit of work must be performed using the same connection handle. Calls issued using a different connection handle (for example, calls issued by another application) affect a different unit of work. See HCONN (10 digit signed integer) - output for information about the scope of connection handles.
6. Only messages that were put or retrieved as part of the current unit of work are affected by this call.
7. A long-running application that issues MQGET, MQPUT, or MQPUT1 calls within a unit of work, but that never issues a commit or backout call, can fill queues with messages that are not available to other applications. To guard against this possibility, the administrator must set the **MaxUncommittedMsgs** queue manager attribute to a value that is low enough to prevent runaway applications filling the queues, but high enough to allow the expected messaging applications to work correctly.

Parameters

The MQDLTMH call has the following parameters:

HCONN (10-digit signed integer) - input

This handle represents the connection to the queue manager.

The value must match the connection handle that was used to create the message handle specified in the **HMSG** parameter.

If the message handle was created using HCUNAS then a valid connection must be established on the thread deleting the message handle, otherwise the call fails with `RC2009` .

HMSG (20-digit signed integer) - input/output

This is the message handle to be deleted. The value was returned by a previous MQCRTMH call.

On successful completion of the call, the handle is set to an invalid value for the environment. This value is:

HMUNUH
Unusable message handle.

The message handle cannot be deleted if another IBM MQ call is in progress that was passed the same message handle.

DLTOPT (MQDMHO) - input

See MQDMHO for details.

CMPCOD (10-digit signed integer) - output

The completion code; it is one of the following:

CCOK
Successful completion.

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

The reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCFAIL:

RC2204
(2204, X'089C') Adapter not available.

RC2130
(2130, X'852') Unable to load adapter service module.

RC2157
(2157, X'86D') Primary and home ASIDs differ.

RC2219
(2219, X'08AB') MQI call entered before previous call completed.

RC2009
(2009, X'07D9') Connection to queue manager lost.

RC2462
(2462, X'099E') Delete message handle options structure not valid.

RC2460
(2460, X'099C') Message handle pointer not valid.

RC2499
(2499, X'09C3') Message handle already in use.

RC2046
(2046, X'07FE') Options not valid or not consistent.

RC2071
(2071, X'817') Insufficient storage available.

RC2195
(2195, X'893') Unexpected error occurred.

See “Return codes for IBM i (ILE RPG)” on page 3452 for more details.

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQDLTMH(HCONN : HMSG : DLTOPT :
                                   CMPCOD : REASON)
```

The prototype definition for the call is:

```
DMQDLTMH          PR                  EXTPROC('MQDLTMH')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Message handle
D HMSG                          20I 0
D* Options that control the action of MQDLTMH
D DLTOPT                        12A
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CompCode
D REASON                        10I 0
```

MQDLTMP - Delete message property:

The MQDLTMP call deletes a property from a message handle and is the inverse of the MQSETMP call.

- "Syntax"
- "Parameters"
- "RPG Declaration" on page 3311

Syntax

MQDLTMP *(Hconn, Hmsg, DltPropOpts, Name, CompCode, Reason)*

Parameters

The MQDLTMP call has the following parameters:

HCONN (10-digit signed integer) - Input

This handle represents the connection to the queue manager. The value must match the connection handle that was used to create the message handle specified in the **HMSG** parameter.

If the message handle was created using HCUNAS then a valid connection must be established on the thread deleting the message handle otherwise the call fails with RC2009.

HMSG (20-digit signed integer) - input

This is the message handle containing the property to be deleted. The value was returned by a previous MQCRTMH call.

DLTOPT (MQDMPO) - Input

See the MQDMPO data type for details.

PRNAME (MQCHARV) - input

The name of the property to delete. See Property names for further information about property names.

Wildcards are not allowed in the property name.

CMPCOD (10-digit signed integer) - output

The completion code; it is one of the following:

CCOK
Successful completion.

CCWARN
Warning (partial completion).

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

The reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCWARN:

RC2471
(2471, X'09A7') Property not available.

RC2421
(2421, X'0975') An MQRFH2 folder containing properties could not be parsed.

If *CMPCOD* is CCFAIL:

RC2204
(2204, X'089C') Adapter not available.

RC2130
(2130, X'0852') Unable to load adapter service module.

RC2157
(2157, X'086D') Primary and home ASIDs differ.

RC2219
(2219, X'08AB') MQI call entered before previous call completed.

RC2009
(2009, X'07D9') Connection to queue manager lost.

RC2481
(2481, X'09B1') Delete message property options structure not valid.

RC2460
(2460, X'099C') Message handle not valid.

RC2499
(2499, X'09C3') Message handle already in use.

RC2046
(2046, X'07FE') Options not valid or not consistent.

RC2442
(2442, X'098A') Invalid property name.

RC2111
(2111, X'083F') Property name coded character set identifier not valid.

RC2195
(2195, X'0893') Unexpected error occurred.

For more information about these codes, see API reason codes.

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQDLTMP(HCONN : HMSG : DLTOPT :
                                    PRNAME : CMPCOD : REASON)
```

The prototype definition for the call is:

```
DMQDLTMP          PR                  EXTPROC('MQDLTMP')
D* Connection handle
D HCONN                           10I 0 VALUE
D* Message handle
D HMSG                            20I 0 VALUE
D* Options that control the action of MQDLTMP
D DLTOPT                          12A
D* Property name
D PRNAME                          32A
D* Completion code
D CMPCOD                          10I 0
D* Reason code qualifying CompCode
D REASON                          10I 0
```

MQGET (Get message) on IBM i: IBM i

The MQGET call retrieves a message from a local queue that has been opened by using the MQOPEN call.

- "Syntax"
- "Usage notes"
- "Parameters" on page 3315
- "RPG Declaration" on page 3320

Syntax

MQGET *(HCONN, HOBJ, MSGDSC, GMO, BUFLEN, BUFFER, DATLEN, CMPCOD, REASON)*

Usage notes

1. The message retrieved is normally deleted from the queue. This deletion can occur as part of the MQGET call itself, or as part of a syncpoint. Message deletion does not occur if a GMBRWF or GMBRWN option is specified on the **GMO** parameter (see the *GMOPT* field described in "MQGMO (Get-message options) on IBM i" on page 3084).
2. If the GMLK option is specified with one of the browse options, the browsed message is locked so that it is visible only to this handle.

   If the GMUNLK option is specified, a previously locked message is unlocked. No message is retrieved in this case, and the **MSGDSC**, **BUFLEN**, **BUFFER** and **DATLEN** parameters are not checked or altered.
3. If the application issuing the MQGET call is running as an IBM MQ MQI client, it is possible for the message retrieved to be lost if during the processing of the MQGET call the IBM MQ MQI client terminates abnormally or the client connection is severed. This arises because the surrogate that is running on the platform of the queue manager and which issues the MQGET call on the behalf of the client cannot detect the loss of the client until the surrogate is about to return the message to the client; this is after the message has been removed from the queue. This can occur for both persistent messages and nonpersistent messages.

   The risk of losing messages in this way can be eliminated by always retrieving messages within units of work (that is, by specifying the GMSYP option on the MQGET call, and using the MQCMIT or MQBACK calls to commit or back out the unit of work when processing of the message is complete). If GMSYP is specified, and the client terminates abnormally or the connection is severed, the surrogate backs out the unit of work on the queue manager and the message is reinstated on the queue.

   In principle, the same situation can arise with applications that are running on the platform of the queue manager, but in this case the window during which a message can be lost is small. However, as with IBM MQ MQI clients the risk can be eliminated by retrieving the message within a unit of work.

4. If an application puts a sequence of messages on a particular queue within a single unit of work, and then commits that unit of work successfully, the messages become available for retrieval as follows:
   - If the queue is a *nonshared* queue (that is, a local queue), all messages within the unit of work become available at the same time.
   - If the queue is a *shared* queue, messages within the unit of work become available in the order in which they were put, but not all at the same time. When the system is heavily laden, it is possible for the first message in the unit of work to be retrieved successfully, but for the MQGET call for the second or subsequent message in the unit of work to fail with RC2033. If this occurs, the application must wait a short while and then try the operation again.
5. If an application puts a sequence of messages on the same queue without using message groups, the order of those messages is preserved if certain conditions are satisfied. See the usage notes in the description of the MQPUT call for details. If the conditions are satisfied, the messages are presented to the receiving application in the order in which they were sent, if:
   - Only one receiver is getting messages from the queue.

     If there are two or more applications getting messages from the queue, they must agree with the sender the mechanism to be used to identify messages that belong to a sequence. For example, the sender might set all of the *MDCID* fields in the messages in a sequence to a value that was unique to that sequence of messages.
   - The receiver does not deliberately change the order of retrieval, for example by specifying a particular *MDMID* or *MDCID*.

   If the sending application put the messages as a message group, the messages are presented to the receiving application in the correct order if the receiving application specifies the GMLOGO option on the MQGET call. For more information about message groups, see:
   - *MDMFL* field in MQMD
   - PMLOGO option in MQPMO
   - GMLOGO option in MQGMO
6. Applications test for the feedback code FBQUIT in the *MDFB* field of the **MSGDSC** parameter. If this value is found, the application ends. See the *MDFB* field described in "MQMD (Message descriptor) on IBM i" on page 3118 for more information.
7. If the queue identified by *HOBJ* was opened with the OOSAVA option, and the completion code from the MQGET call is CCOK or CCWARN, the context associated with the queue handle *HOBJ* is set to the context of the message that has been retrieved (unless the GMBRWF or GMBRWN option is set, in which case the context is marked as not available). This context can be used on a subsequent MQPUT or MQPUT1 call by specifying the PMPASI or PMPASA options. This enables the context of the message received to be transferred in whole or in part to another message (for example, when the message is forwarded to another queue). For more information about message context, see Message context and Controlling context information.
8. If the GMCONV option is included in the **GMO** parameter, the application message data is converted to the representation requested by the receiving application, before the data is placed in the **BUFFER** parameter:
   - The *MDFMT* field in the control information in the message identifies the structure of the application data, and the *MDCSI* and *MDENC* fields in the control information in the message specify its character-set identifier and encoding.
   - The application issuing the MQGET call specifies in the *MDCSI* and *MDENC* fields in the **MSGDSC** parameter the character-set identifier and encoding to which the application message data must be converted.

   When conversion of the message data is necessary, the conversion is performed either by the queue manager itself or by a user-written exit, depending on the value of the *MDFMT* field in the control information in the message:
   - The following formats are converted automatically by the queue manager; these formats are called "built-in" formats:

FMADMN
FMCICS
FMCMD1
FMCMD2
FMDLH
FMDH
FMEVNT
FMIMS
FMIMVS
FMMDE
FMPCF
FMRMH
FMRFH
FMRFH2
FMSTR
FMTM
FMXQH

- The format name FMNONE is a special value that indicates that the nature of the data in the message is undefined. As a consequence, the queue manager does not attempt conversion when the message is retrieved from the queue.

  Note: If GMCONV is specified on the MQGET call for a message that has a format name of FMNONE, and the character set or encoding of the message differs from that specified in the **MSGDSC** parameter, the message is still returned in the **BUFFER** parameter (assuming no other errors), but the call completes with completion code CCWARN and reason code RC2110.

  FMNONE can be used either when the nature of the message data means that it does not require conversion, or when the sending and receiving applications have agreed between themselves the form in which the message data should be sent.
- All other format names cause the message to be passed to a user-written exit for conversion. The exit has the same name as the format, apart from environment-specific additions. User-specified format names must not begin with the letters "MQ", as such names might conflict with format names supported in the future.

User data in the message can be converted between any supported character sets and encodings. However, be aware that if the message contains one or more IBM MQ header structures, the message cannot be converted from or to a character set that has double-byte or multi-byte characters for any of the characters that are valid in queue names. Reason code RC2111 or RC2115 results if this is attempted, and the message is returned unconverted. Unicode character set V 9.0.0 UTF-16 is an example of such a character set.

On return from MQGET, the following reason code indicates that the message was converted successfully:

- RCNONE

The following reason code indicates that the message might have been converted successfully; the application must check the *MDCSI* and *MDENC* fields in the **MSGDSC** parameter to find out:

- RC2079

All other reason codes indicate that the message was not converted.

Note: The interpretation of the reason code described in this example is true for conversions performed by user-written exits only if the exit conforms to the processing guidelines.

9. For the built-in formats listed previously, the queue manager might perform *default conversion* of character strings in the message when the GMCONV option is specified. Default conversion allows the queue manager to use an installation-specified default character set that approximates the actual character set, when converting string data. As a result, the MQGET call can succeed with completion code CCOK, instead of completing with CCWARN and reason code RC2111 or RC2115.

   Note: The result of using an approximate character set to convert string data is that some characters might be converted incorrectly. This can be avoided by using in the string only characters which are common to both the actual character set and the default character set.

   Default conversion applies both to the application message data and to character fields in the MQMD and MQMDE structures:

- Default conversion of the application message data occurs only when *all* of the following statements are true:
  - The application specifies GMCONV.
  - The message contains data that must be converted either from or to a character set which is not supported.
  - Default conversion was enabled when the queue manager was installed or restarted.
- Default conversion of the character fields in the MQMD and MQMDE structures occurs as necessary, if default conversion is enabled for the queue manager. The conversion is performed even if the GMCONV option is not specified by the application on the MQGET call.

10. The **BUFFER** parameter shown in the RPG programming example is declared as a string; this restricts the maximum length of the parameter to 256 bytes. If a larger buffer is required, the parameter must be declared instead as a structure, or as a field in a physical file.

    Declaring the parameter as a structure increases the maximum length possible to 9999 bytes, while declaring the parameter as a field in a physical file increases the maximum length possible to approximately 32 KB.

Parameters

The MQGET call has the following parameters:

HCONN (10-digit signed integer) - input

Connection handle.

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN or MQCONNX call.

HOBJ (10-digit signed integer) - input

Object handle.

This handle represents the queue from which a message is to be retrieved. The value of *HOBJ* was returned by a previous MQOPEN call. The queue must have been opened with one or more of the following options (see "MQOPEN (Open object) on IBM i" on page 3337 for details):

- OOINPS
- OOINPX
- OOINPQ
- OOBRW

MSGDSC (MQMD) - input/output

Message descriptor.

This structure describes the attributes of the message required, and the attributes of the message retrieved. See "MQMD (Message descriptor) on IBM i" on page 3118 for details.

If *BUFLEN* is less than the message length, *MSGDSC* is still entered by the queue manager, whether GMATM is specified on the **GMO** parameter (see the *GMOPT* field described in "MQGMO (Get-message options) on IBM i" on page 3084).

If the application provides a version-1 MQMD, the message returned has an MQMDE prefixed to the application message data, but only if one or more of the fields in the MQMDE has a nondefault value. If all of the fields in the MQMDE have default values, the MQMDE is omitted. A format name of FMMDE in the *MDFMT* field in MQMD indicates that an MQMDE is present.

GMO (MQGMO) - input/output

Options that control the action of MQGET.

See "MQGMO (Get-message options) on IBM i" on page 3084 for details.

BUFLEN (10-digit signed integer) - input

Length in bytes of the *BUFFER* area.

Zero can be specified for messages that have no data, or if the message is to be removed from the queue and the data discarded (GMATM must be specified in this case).

Note: The length of the longest message that it is possible to read from the queue is given by the **MaxMsgLength** queue attribute; see "Attributes for queues" on page 3385.

BUFFER (1-byte bit string x BUFLEN) - output

Area to contain the message data.

The buffer must be aligned on a boundary appropriate to the nature of the data in the message. 4-byte alignment must be suitable for most messages (including messages containing IBM MQ header structures), but some messages might require more stringent alignment. For example, a message containing a 64-bit binary integer might require 8-byte alignment.

If *BUFLEN* is less than the message length, as much of the message as possible is moved into *BUFFER* ; this happens whether GMATM is specified on the **GMO** parameter (see the *GMOPT* field described in "MQGMO (Get-message options) on IBM i" on page 3084 for more information).

The character set and encoding of the data in **BUFFER** are given by the *MDCSI* and *MDENC* fields returned in the **MSGDSC** parameter. If these values are different from the values required by the receiver, the receiver must convert the application message data to the character set and encoding required. The GMCONV option can be used with a user-written exit to perform the conversion of the message data (see "MQGMO (Get-message options) on IBM i" on page 3084 for details of this option).

Note: All of the other parameters on the MQGET call are in the character set and encoding of the local queue manager (given by the **CodedCharSetId** queue manager attribute and ENNAT).

If the call fails, the contents of the buffer might still have changed.

DATLEN (10-digit signed integer) - output

Length of the message.

This is the length in bytes of the application data in the message. If this message length is greater than *BUFLEN*, only *BUFLEN* bytes are returned in the **BUFFER** parameter (that is, the message is truncated). If the value is zero, it means that the message contains no application data.

If *BUFLEN* is less than the message length, *DATLEN* is still entered by the queue manager, whether GMATM is specified on the **GMO** parameter (see the *GMOPT* field described in "MQGMO (Get-message options) on IBM i" on page 3084 for more information). This allows the application to determine the size of the buffer required to accommodate the message data, and then reissue the call with a buffer of the appropriate size.

However, if the GMCONV option is specified, and the converted message data is too long to fit in *BUFFER*, the value returned for *DATLEN* is:

- The length of the *unconverted* data, for queue manager defined formats.

  In this case, if the nature of the data causes it to expand during conversion, the application must allocate a buffer bigger than the value returned by the queue manager for *DATLEN*.
- The value returned by the data-conversion exit, for application-defined formats.

CMPCOD (10-digit signed integer) - output

Completion code.

It is one of the following:

CCOK
: Successful completion.

CCWARN
Warning (partial completion).

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

Reason code qualifying *CMPCOD*.

The following reason codes are the ones that the queue manager can return for the **REASON** parameter. If the application specifies the GMCONV option, and a user-written exit is invoked to convert some or all of the message data, it is the exit that decides what value is returned for the **REASON** parameter. As a result, values other than the values documented later in this section are possible.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCWARN:

RC2120
(2120, X'848') Converted data too large for buffer.

RC2190
(2190, X'88E') Converted string too large for field.

RC2150
(2150, X'866') DBCS string not valid.

RC2110
(2110, X'83E') Message format not valid.

RC2243
(2243, X'8C3') Message segments have differing CCSIDs.

RC2244
(2244, X'8C4') Message segments have differing encodings.

RC2209
(2209, X'8A1') No message locked.

RC2119
(2119, X'847') Message data not converted.

RC2272
(2272, X'8E0') Message data partially converted.

RC2145
(2145, X'861') Source buffer parameter not valid.

RC2111
(2111, X'83F') Source coded character set identifier not valid.

RC2113
(2113, X'841') Packed-decimal encoding in message not recognized.

RC2114
(2114, X'842') Floating-point encoding in message not recognized.

RC2112
(2112, X'840') Source integer encoding not recognized.

RC2143
(2143, X'85F') Source length parameter not valid.

RC2146
(2146, X'862') Target buffer parameter not valid.

RC2115
(2115, X'843') Target coded character set identifier not valid.

RC2117
(2117, X'845') Packed-decimal encoding specified by receiver not recognized.

RC2118
(2118, X'846') Floating-point encoding specified by receiver not recognized.

RC2116
(2116, X'844') Target integer encoding not recognized.

RC2079
(2079, X'81F') Truncated message returned (processing completed).

RC2080
(2080, X'820') Truncated message returned (processing not completed).

If *CMPCOD* is CCFAIL:

RC2004
(2004, X'7D4') Buffer parameter not valid.

RC2005
(2005, X'7D5') Buffer length parameter not valid.

RC2219
(2219, X'8AB') MQI call reentered before previous call complete.

RC2009
(2009, X'7D9') Connection to queue manager lost.

RC2010
(2010, X'7DA') Data length parameter not valid.

RC2016
(2016, X'7E0') Gets inhibited for the queue.

RC2186
(2186, X'88A') Get-message options structure not valid.

RC2018
(2018, X'7E2') Connection handle not valid.

RC2019
(2019, X'7E3') Object handle not valid.

RC2241
(2241, X'8C1') Message group not complete.

RC2242
(2242, X'8C2') Logical message not complete.

RC2259
(2259, X'8D3') Inconsistent browse specification.

RC2245
(2245, X'8C5') Inconsistent unit-of-work specification.

RC2246
(2246, X'8C6') Message under cursor not valid for retrieval.

RC2247
(2247, X'8C7') Match options not valid.

RC2026
(2026, X'7EA') Message descriptor not valid.

RC2250
(2250, X'8CA') Message sequence number not valid.

RC2033
(2033, X'7F1') No message available.

RC2034
(2034, X'7F2') Browse cursor not positioned on message.

RC2036
(2036, X'7F4') Queue not open for browse.

RC2037
(2037, X'7F5') Queue not open for input.

RC2041
(2041, X'7F9') Object definition changed since opened.

RC2101
(2101, X'835') Object damaged.

RC2046
(2046, X'7FE') Options not valid or not consistent.

RC2052
(2052, X'804') Queue has been deleted.

RC2058
(2058, X'80A') Queue manager name not valid or not known.

RC2059
(2059, X'80B') Queue manager not available for connection.

RC2161
(2161, X'871') Queue manager quiescing.

RC2162
(2162, X'872') Queue manager shutting down.

RC2102
(2102, X'836') Insufficient system resources available.

RC2071
(2071, X'817') Insufficient storage available.

RC2024
(2024, X'7E8') No more messages can be handled within current unit of work.

RC2072
(2072, X'818') Syncpoint support not available.

RC2195
(2195, X'893') Unexpected error occurred.

RC2255
(2255, X'8CF') Unit of work not available for the queue manager to use.

RC2090
(2090, X'82A') Wait interval in MQGMO not valid.

RC2256
(2256, X'8D0') Wrong version of MQGMO supplied.

RC2257

(2257, X'8D1') Wrong version of MQMD supplied.

RPG Declaration

```
C*..1...:....2...:....3...:....4...:....5...:....6...:....7..
C                     CALLP     MQGET(HCONN : HOBJ : MSGDSC : GMO :
C                                     BUFLEN : BUFFER : DATLEN :
C                                     CMPCOD : REASON)
```

The prototype definition for the call is:

```
D*..1...:....2...:....3...:....4...:....5...:....6...:....7..
DMQGET            PR                  EXTPROC('MQGET')
D* Connection handle
D HCONN                          10I 0 VALUE
D* Object handle
D HOBJ                           10I 0 VALUE
D* Message descriptor
D MSGDSC                        364A
D* Options that control the action of MQGET
D GMO                           112A
D* Length in bytes of the Buffer area
D BUFLEN                         10I 0 VALUE
D* Area to contain the message data
D BUFFER                           *   VALUE
D* Length of the message
D DATLEN                         10I 0
D* Completion code
D CMPCOD                         10I 0
D* Reason code qualifying CMPCOD
D REASON                         10I 0
```

MQINQ (Inquire about object attributes) on IBM i: 

The MQINQ call returns an array of integers and a set of character strings containing the attributes of an object.

The following types of object are valid:

- Queue
- Namelist
- Process definition
- Queue manager
- "Syntax"
- "Usage notes"
- "Parameters" on page 3322
- "RPG Declaration" on page 3328

Syntax

MQINQ *(HCONN, HOBJ, SELCNT, SELS, IACNT, INTATR, CALEN, CHRATR, CMPCOD, REASON)*

Usage notes

1. The values returned are a snapshot of the selected attributes. There is no guarantee that the attributes are not changed before the application can act upon the returned values.
2. When you open a model queue, a dynamic local queue is created. This is true even if you open the model queue to inquire about its attributes.

The attributes of the dynamic queue (with certain exceptions) are the same as those of the model queue at the time the dynamic queue is created. If you then use the MQINQ call on this queue, the queue manager returns the attributes of the dynamic queue, and not those of the model queue. See Table 1 for details of which attributes of the model queue are inherited by the dynamic queue.

3. If the object being inquired is an alias queue, the attribute values returned by the MQINQ call are those of the alias queue, and not those of the base queue to which the alias resolves.
4. If the object being inquired is a cluster queue, the attributes that can be inquired depend on how the queue is opened:
   - If the cluster queue is opened for inquire plus one or more of input, browse, or set, there must be a local instance of the cluster queue in order for the open to succeed. In this case the attributes that can be inquired are those valid for local queues.
   - If the cluster queue is opened for inquire alone, or inquire and output, only the following attributes can be inquired; the **QType** attribute has the value QTCLUS in this case:
     – CAQD
     – CAQN
     – IADBND
     – IADPER
     – IADPRI
     – IAIPUT
     – IAQTYP

     If the cluster queue is opened with no fixed binding (that is, OOBNDN specified on the MQOPEN call, or OOBNDQ specified when the **DefBind** attribute has the value BNDNOT), successive MQINQ calls for the queue might inquire different instances of the cluster queue, although typically all of the instances have the same attribute values.

   For more information about cluster queues, see Configuring a queue manager cluster.
5. If a number of attributes are to be inquired, and then some of them are to be set using the MQSET call, it might be convenient to position at the beginning of the selector arrays the attributes that are to be set, so that the same arrays (with reduced counts) can be used for MQSET.
6. If more than one of the warning situations arise (see the **CMPCOD** parameter), the reason code returned is the *first* one in the following list that applies:
   a. `RC2068`
   b. `RC2022`
   c. `RC2008`
7. For more information about object attributes, see:
   - "Attributes for queues" on page 3385
   - "Attributes for namelists" on page 3417
   - "Attributes for process definitions on IBM i" on page 3418
   - "Attributes for the queue manager on IBM i" on page 3420
8. A new local queue SYSTEM.ADMIN.COMMAND.EVENT is used for queuing messages that are generated whenever commands are issued. Messages are put onto this queue for most commands, depending on how the CMDEV queue manager attribute is set:
   - ENABLED - command event messages are generated and put onto the queue for all successful commands.
   - NODISPLAY - command event messages are generated and put onto the queue for all successful commands other than the DISPLAY (MQSC) command, and the Inquire (PCF) command.
   - DISABLED - command event messages are not generated (this is the queue manager's initial default value).

Parameters

The MQINQ call has the following parameters:

HCONN (10 digit signed integer) - input

Connection handle.

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN or MQCONNX call.

HOBJ (10 digit signed integer) - input

Object handle.

This handle represents the object (of any type) with attributes that are required. The handle must have been returned by a previous MQOPEN call that specified the OOINQ option.

SELCNT (10 digit signed integer) - input

Count of selectors.

This is the count of selectors that are supplied in the *SELS* array. It is the number of attributes that are to be returned. Zero is a valid value. The maximum number allowed is 256.

SELS (10 digit signed integer x SELCNT) - input

Array of attribute selectors.

This is an array of **SELCNT** attribute selectors; each selector identifies an attribute (integer or character) with a value that is required.

Each selector must be valid for the type of object that *HOBJ* represents, otherwise the call fails with completion code CCFAIL and reason code RC2067.

In the special case of queues:

- If the selector is not valid for queues of *any* type, the call fails with completion code CCFAIL and reason code RC2067.
- If the selector is applicable only to queues of type or types other than that of the object, the call succeeds with completion code CCWARN and reason code RC2068.
- If the queue being inquired is a cluster queue, the selectors that are valid depend on how the queue was resolved; see usage note 4 for further details.

Selectors can be specified in any order. Attribute values that correspond to integer attribute selectors (IA* selectors) are returned in *INTATR* in the same order in which these selectors occur in *SELS*. Attribute values that correspond to character attribute selectors (CA* selectors) are returned in *CHRATR* in the same order in which those selectors occur. IA* selectors can be interleaved with the CA* selectors; only the relative order within each type is important.

Note:

1. The integer and character attribute selectors are allocated within two different ranges; the IA* selectors reside within the range IAFRST through IALAST, and the CA* selectors within the range CAFRST through CALAST.

   For each range, the constants IALSTU and CALSTU define the highest value that the queue manager accepts.
2. If all the IA* selectors occur first, the same element numbers can be used to address corresponding elements in the *SELS* and *INTATR* arrays.

The attributes that can be inquired are listed in the following tables. For the CA* selectors, the constant that defines the length in bytes of the resulting string in *CHRATR* is given in parentheses.

Table 364. MQINQ attribute selectors for queues.

See the bottom of the table for an explanation of the notes.

Selector	Description	Note
CAALTD	Date of most recent alteration (LNDATE).	1
CAALTT	Time of most recent alteration (LNTIME).	1
CABRQN	Excessive backout-requeue name (LNQN).	5
CABASQ	Name of queue that alias resolves to (LNQN).	
CACFSN	Coupling-facility structure name (LNCFSN).	3
CACLN	Cluster name (LNCLUN).	1
CACLNL	Cluster namelist (LNNLN).	1
CACRTD	Queue creation date (LNCRTD).	
CACRTT	Queue creation time (LNCRTT).	
CAINIQ	Initiation queue name (LNQN).	
CAPRON	Name of process definition (LNPRON).	
CAQD	Queue description (LNQD).	
CAQN	Queue name (LNQN).	
CARQMN	Name of remote queue manager (LNQMN).	
CARQN	Name of remote queue as known on remote queue manager (LNQN).	
CATRGD	Trigger data (LNTRGD).	5
CAXQN	Transmission queue name (LNQN).	
IABTHR	Backout threshold.	5
IACDEP	Number of messages on queue.	
IADBND	Default binding.	1
IADINP	Default open-for-input option.	5
IADPER	Default message persistence.	
IADPRI	Default message priority.	5
IADEFT	Queue definition type.	
IADIST	Distribution list support.	2
IAHGB	Whether to harden backout count.	5
IAIGET	Whether get operations are allowed.	
IAIPUT	Whether put operations are allowed.	
IAMLEN	Maximum message length.	
IAMDEP	Maximum number of messages allowed on queue.	
IAMDS	Whether message priority is relevant.	5
IAOIC	Number of MQOPEN calls that have the queue open for input.	
IAOOC	Number of MQOPEN calls that have the queue open for output.	
IAQDHE	Control attribute for queue depth high events.	4, 5
IAQDHL	High limit for queue depth.	4, 5
IAQDLE	Control attribute for queue depth low events.	4, 5
IAQDLL	Low limit for queue depth.	4, 5
IAQDME	Control attribute for queue depth max events.	4, 5

Table 364. MQINQ attribute selectors for queues (continued).

See the bottom of the table for an explanation of the notes.

Selector	Description	Note
IAQSI	Limit for queue service interval.	4, 5
IAQSIE	Control attribute for queue service interval events.	4, 5
IAQTYP	Queue type.	
IAQSGD	Queue-sharing group disposition.	3
IARINT	Queue retention interval.	5
IASCOP	Queue definition scope.	4, 5
IASHAR	Whether queue can be shared for input.	
IATRGC	Trigger control.	
IATRGD	Trigger depth.	5
IATRGP	Threshold message priority for triggers.	5
IATRGT	Trigger type.	
IAUSAG	Usage.	
CLWLUSEQ	Use remote queues.	

Note:

1. Supported on AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ MQI clients connected to these systems.
2. Supported on AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.
3. Supported on z/OS.
4. Not supported on z/OS.
5. Not supported on VSE/ESA.

Table 365. MQINQ attribute selectors for namelists.

See notes for an explanation of the notes.

Selector	Description	Note
CAALTD	Date of most recent alteration (LNDATE)	1
CAALTT	Time of most recent alteration (LNTIME)	1
CALSTD	Namelist description (LNNLD)	1
CALSTN	Name of namelist object (LNNLN)	1
CANAMS	Names in the namelist (LNQN x *Number of names in the list)*	1
IANAMC	Number of names in the namelist	1
IAQSGD	Queue-sharing group disposition	3

Table 366. MQINQ attribute selectors for process definitions.

See notes for an explanation of the notes.

Selector	Description	Note
CAALTD	Date of most recent alteration (LNDATE)	1
CAALTT	Time of most recent alteration (LNTIME)	1
CAAPPI	Application identifier (LNPROA)	5
CAENVD	Environment data (LNPROE)	5
CAPROD	Description of process definition (LNPROD)	5
CAPRON	Name of process definition (LNPRON)	5
CAUSRD	User data (LNPROU)	5
IAAPPT	Application type	5
IAQSGD	Queue-sharing group disposition	3

Table 367. MQINQ attribute selectors for the queue manager.

See notes for an explanation of the notes.

Selector	Description	Note
CAALTD	Date of most recent alteration (LNDATE)	1
CAALTT	Time of most recent alteration (LNTIME)	1
CACADX	Automatic channel definition exit name (LNEXN)	1
CACLWD	Data passed to cluster workload exit (LNEXDA)	1
CACLWX	Name of cluster workload exit (LNEXN)	1
CACMDQ	System command input queue name (LNQN)	5
CADLQ	Name of dead-letter queue (LNQN)	5
CADXQN	Default transmission queue name (LNQN)	5
CAQMD	Queue manager description (LNQMD)	5
CAQMID	Queue manager identifier (LNQMID)	1
CAQMN	Name of local queue manager (LNQMN)	5
CAQSGN	Queue-sharing group name (LNQSGN)	3
CARPN	Name of cluster for which queue manager provides repository services (LNQMN)	1
CARPNL	Name of namelist object containing names of clusters for which queue manager provides repository services (LNNLN)	1
CMDEV	Control attribute that determines whether messages generated when commands are issued, are put onto a queue	8
IAAUTE	Control attribute for authority events	4, 5
IACAD	Control attribute for automatic channel definition	2
IACADE	Control attribute for automatic channel definition events	2
IACLXQ	Default cluster transmission queue type	4
IACLWL	Cluster workload length	1
IACCSI	Coded character set identifier	5
IACMDL	Command level supported by queue manager	5
IACFGE	Control attribute for configuration events	3

Table 367. MQINQ attribute selectors for the queue manager (continued).

See notes for an explanation of the notes.

Selector	Description	Note
IADIST	Distribution list support	2
IAINHE	Control attribute for inhibit events	4, 5
IALCLE	Control attribute for local events	4, 5
IAMHND	Maximum number of handles	5
IAMLEN	Maximum message length	5
IAMPRI	Maximum priority	5
IAMUNC	Maximum number of uncommitted messages within a unit of work	5
IAPFME	Control attribute for performance events	4, 5
IAPLAT	Platform on which the queue manager resides	5
IARMTE	Control attribute for remote events	4, 5
IASSE	Control attribute for start stop events	4, 5
IASYNC	Sync point availability	5
IATRLFT	Lifetime of unused non-administrative topics	
IATRGI	Trigger interval	5

IACNT (10 digit signed integer) - input

Count of integer attributes.

This is the number of elements in the *INTATR* array. Zero is a valid value.

If this is at least the number of IA* selectors in the **SELS** parameter, all integer attributes requested are returned.

INTATR (10 digit signed integer x IACNT) - output

Array of integer attributes.

This is an array of *IACNT* integer attribute values.

Integer attribute values are returned in the same order as the IA* selectors in the **SELS** parameter. If the array contains more elements than the number of IA* selectors, the excess elements are unchanged.

If *HOBJ* represents a queue, but an attribute selector is not applicable to that type of queue, the specific value IAVNA is returned for the corresponding element in the *INTATR* array.

CALEN (10 digit signed integer) - input

Length of character attributes buffer.

This is the length in bytes of the **CHRATR** parameter.

This must be at least the sum of the lengths of the requested character attributes (see *SELS*). Zero is a valid value.

CHRATR (1 byte character string x CALEN) - output

Character attributes.

This is the buffer in which the character attributes are returned, concatenated together. The length of the buffer is given by the **CALEN** parameter.

Character attributes are returned in the same order as the CA* selectors in the **SELS** parameter. The length of each attribute string is fixed for each attribute (see *SELS*), and the value in it is

padded to the right with blanks if necessary. If the buffer is larger than that needed to contain all of the requested character attributes (including padding), the bytes beyond the last attribute value returned are unchanged.

If *HOBJ* represents a queue, but an attribute selector is not applicable to that type of queue, a character string consisting entirely of asterisks (*) is returned as the value of that attribute in *CHRATR*.

CMPCOD (10 digit signed integer) - output

Completion code.

It is one of the following:

CCOK
Successful completion.

CCWARN
Warning (partial completion).

CCFAIL
Call failed.

REASON (10 digit signed integer) - output

Reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCWARN:

RC2008
(2008, X'7D8') Not enough space allowed for character attributes.

RC2022
(2022, X'7E6') Not enough space allowed for integer attributes.

RC2068
(2068, X'814') Selector not applicable to queue type.

If *CMPCOD* is CCFAIL:

RC2219
(2219, X'8AB') MQI call reentered before previous call complete.

RC2006
(2006, X'7D6') Length of character attributes not valid.

RC2007
(2007, X'7D7') Character attributes string not valid.

RC2009
(2009, X'7D9') Connection to queue manager lost.

RC2018
(2018, X'7E2') Connection handle not valid.

RC2019
(2019, X'7E3') Object handle not valid.

RC2021
(2021, X'7E5') Count of integer attributes not valid.

RC2023
(2023, X'7E7') Integer attributes array not valid.

RC2038
(2038, X'7F6') Queue not open for inquire.

RC2041
(2041, X'7F9') Object definition changed since opened.

RC2101
(2101, X'835') Object damaged.

RC2052
(2052, X'804') Queue has been deleted.

RC2058
(2058, X'80A') Queue manager name not valid or not known.

RC2059
(2059, X'80B') Queue manager not available for connection.

RC2162
(2162, X'872') Queue manager shutting down.

RC2102
(2102, X'836') Insufficient system resources available.

RC2065
(2065, X'811') Count of selectors not valid.

RC2067
(2067, X'813') Attribute selector not valid.

RC2066
(2066, X'812') Count of selectors too large.

RC2071
(2071, X'817') Insufficient storage available.

RC2195
(2195, X'893') Unexpected error occurred.

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQINQ(HCONN : HOBJ : SELCNT :
C                                     SELS(1) : IACNT : INTATR(1) :
C                                     CALEN : CHRATR : CMPCOD :
C                                     REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQINQ            PR                  EXTPROC('MQINQ')
D* Connection handle
D HCONN                           10I 0 VALUE
D* Object handle
D HOBJ                            10I 0 VALUE
D* Count of selectors
D SELCNT                          10I 0 VALUE
D* Array of attribute selectors
D SELS                            10I 0
D* Count of integer attributes
D IACNT                           10I 0 VALUE
D* Array of integer attributes
D INTATR                          10I 0
D* Length of character attributes buffer
D CALEN                           10I 0 VALUE
D* Character attributes
D CHRATR                            *   VALUE
```

```
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
```

MQINQMP (Inquire message property) on IBM i: 

The MQINQMP call returns the value of a property of a message.

- "Syntax"
- "Parameters"
- "RPG Declaration" on page 3333

Syntax

MQINQMP *(Hconn, Hmsg, InqPropOpts, Name, PropDesc, Type, ValueLength, Value, DataLength, CompCode, Reason)*

Parameters

The MQINQMP call has the following parameters:

HCONN (10-digit signed integer) - input

This handle represents the connection to the queue manager. The value of *Hconn* must match the connection handle that was used to create the message handle specified in the **Hmsg** parameter.

If the message handle was created using HCUNAS then a valid connection must be established on the thread inquiring a property of the message handle, otherwise the call fails with RC2009.

HMSG (20-digit signed integer) - input

This is the message handle to be inquired. The value was returned by a previous **MQCRTMH** call.

INQOPT (MQIMPO) - input

See the MQIMPO data type for details.

PRNAME (MQCHARV) - input

This describes the name of the property to inquire.

If no property with this name can be found, the call fails with reason RC2471.

You can use the percent sign (%) character at the end of the property name. The wildcard matches zero or more characters, including the period (.) character. This allows an application to inquire the value of many properties. Call MQINQMP with option IPINQF to get the first matching property and again with the option IPINQN to get the next matching property. When no more matching properties are available, the call fails with RC2471. If the *ReturnedName* field of the InqPropOpts structure is initialized with an address or offset for the returned name of the property, this is completed on return from MQINQMP with the name of the property that has been matched. If the *VSBufSize* field of the *ReturnedName* in the InqPropOpts structure is less than the length of the returned property name the completion code is set CCFAIL with reason RC2465.

Properties that have known synonyms are returned as follows:

1. Properties with the prefix "mqps." are returned with the IBM MQ property name. For example, "MQTopicString" is the returned name rather than "mqps.Top".
2. Properties with the prefix "jms." or "mcd." are returned as the JMS header field name. For example, "JMSExpiration" is the returned name rather than "jms.Exp".
3. Properties with the prefix "usr." are returned without that prefix. For example, "Color" is returned rather than "usr.Color".

Properties with synonyms are only returned once.

In the RPG programming language, the following macro variables are defined for inquiring on all properties and all properties that begin "usr.":

INQALL
Inquire on all properties of the message.

INQUSR
Inquire on all properties of the message that start "usr.". The returned name is returned without the "usr." prefix.

If IPINQN is specified but Name has changed since the previous call or this is the first call, then IPINQF is implied.

See Property names and Property name restrictions for further information about the use of property names.

PRPDSC (MQPD) - output

This structure is used to define the attributes of a property, including what happens if the property is not supported, what message context the property belongs to, and what messages the property should be copied into. See MQPD for details of this structure.

TYPE (10-digit signed integer) - input/output

On return from the MQINQMP call this parameter is set to the data type of *Value*. The data type can be any of the following:

TYPBOL
A boolean.

TYPBST
a byte string.

TYPI8 An 8-bit signed integer.

TYPI16
A 16-bit signed integer.

TYPI32
A 32-bit signed integer.

TYPI64
A 64-bit signed integer.

TYPF32
A 32-bit floating-point number.

TYPF64
A 64-bit floating-point number.

TYPSTR
A character string.

TYPNUL
The property exists but has a null value.

If the data type of the property value is not recognized then TYPSTR is returned and a string representation of the value is placed into the *Value* area. A string representation of the data type can be found in the *IPTYP* field of the *IPOPT* parameter. A warning completion code is returned with reason RC2467.

Additionally, if the option IPCTYP is specified, conversion of the property value is requested. Use *Type* as an input to specify the data type that you want the property to be returned as. See the description of the IPCTYP option of the "MQIMPO (Inquire message property options) on IBM i" on page 3111 for details of data type conversion.

If you do not request type conversion, you can use the following value on input:

TYPAST
The value of the property is returned without converting its data type.

VALLEN (10-digit signed integer) - input

The length in bytes of the Value area.

Specify zero for properties that you do not require the value returned for. These could be properties which are designed by an application to have a null value or an empty string. Also specify zero if the IPQLEN option has been specified; in this case no value is returned.

VALUE (1-byte bit stringxVALLEN) - output

This is the area to contain the inquired property value. The buffer should be aligned on a boundary appropriate for the value being returned. Failure to do so might result in an error when the value is later accessed.

If *VALLEN* is less than the length of the property value, as much of the property value as possible is moved into *VALUE* and the call fails with completion code CCFAIL and reason RC2469.

The character set of the data in *VALUE* is given by the IPRETCSI field in the INQOPT parameter. The encoding of the data in *VALUE* is given by the IPRETENC field in the INQOPT parameter.

If the *VALLEN* parameter is zero, VALUE is not referred to.

DATLEN (10-digit signed integer) - output

This is the length in bytes of the actual property value as returned in the *Value* area.

If *DataLength* is less than the property value length, *DataLength* is still entered on return from the MQINQMP call. This allows the application to determine the size of the buffer required to accommodate the property value, and then reissue the call with a buffer of the appropriate size.

The following values may also be returned.

If the *Type* parameter is set to TYPSTR or TYPBST:

VLEMP
The property exists but contains no characters or bytes.

CMPCOD (10-digit signed integer) - output

The completion code; it is one of the following:

CCOK
Successful completion.

CCWARN
Warning (partial completion).

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

The reason code qualifying *CompCode*.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CompCode* is CCWARN:

RC2492
(2492, X'09BC') Returned property name not converted.

RC2466
(2466, X'09A2') Property value not converted.

RC2467
(2467, X'09A3') Property data type is not supported.

RC2421
(2421, X'0975') An MQRFH2 folder containing properties could not be parsed.

If *CMPCOD* is CCFAIL:

RC2204
(2204, X'089C') Adapter not available.

RC2130
(2130, X'0852') Unable to load adapter service module.

RC2157
(2157, X'086D') Primary and home ASIDs differ.

RC2004
(2004, X'07D4') Value parameter not valid.

RC2005
(2005, X'07D5') Value length parameter not valid.

RC2219
(2219, X'08AB') MQI call entered before previous call completed.

RC2009
(2009, X'07D9') Connection to queue manager lost.

RC2010
(2010, X'07DA') Data length parameter not valid.

RC2464
(2464, X'09A0') Inquire message property options structure not valid.

RC2460
(2460, X'099C') Message handle not valid.

RC2499
(2499, X'09C3') Message handle already in use.

RC2064
(2046, X'07F8') Options not valid or not consistent.

RC2482
(2482, X'09B2') Property descriptor structure not valid.

RC2470
(2470, X'09A6') Conversion from the actual to requested data type not supported.

RC2442
(2442, X'098A') Invalid property name.

RC2465
(2465, X'09A1') Property name too large for returned name buffer.

RC2471
(2471, X'09A7') Property not available.

RC2469
(2469, X'09A5') Property value too large for the Value area.

RC2472
(2472, X'09A8') Number format error encountered in value data.

RC2473
(2473, X'09A9') Invalid requested property type.

RC2111
(2111, X'083F') Property name coded character set identifier not valid.

RC2071
(2071, X'0871') Insufficient storage available.

RC2195
(2195, X'0893') Unexpected error occurred.

For detailed information about these codes, see:
- IBM MQ for z/OS messages, completion, and reason codes for IBM MQ for z/OS
- Reason codes for all other IBM MQ platforms

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQINQMP(HCONN : HMSG : INQOPT :
                                        PRNAME : PRPDSC : TYPE :
                                        VALLEN : VALUE : DATLEN :
                                        CMPCOD : REASON)
```

The prototype definition for the call is:

```
DMQINQMP          PR                  EXTPROC('MQINQMP')
D* Connection handle
D HCONN                           10I 0 VALUE
D* Message handle
D HMSG                            20I 0 VALUE
D* Options that control the action of MQINQMP
D INQOPT                          72A
D* Property name
D PRNAME                          32A
D* Property descriptor
D PRPDSC                          24A
D* Property data type
D TYPE                            10I 0
D* Length in bytes of the Value area
D VALLEN                          10I 0 VALUE
D* Property value
D VALUE                             *   VALUE
D* Length of the property value
D DATLEN                          10I 0
D* Completion code
D CMPCOD                          10I 0
D* Reason code qualifying CompCode
D REASON                          10I 0
```

MQMHBUF (Convert message handle into buffer) on IBM i: IBM i

The MQMHBUF converts a message handle into a buffer and is the inverse of the MQBUFMH call.

- "Syntax"
- "Usage notes"
- "Parameters"
- "RPG Declaration" on page 3336

Syntax

MQMHBUF *(Hconn, Hmsg, MsgHBufOpts, Name, MsgDesc, BufferLength, Buffer, DataLength, CompCode, Reason)*

Usage notes

MQMHBUF converts a message handle into a buffer.

You can use it with an MQGET API exit to access certain properties, by using the message property APIs, and then pass these properties in a buffer back to an application designed to use MQRFH2 headers rather than message handles.

This call is the inverse of the MQBUFMH call, which you can use to parse message properties from a buffer into a message handle.

Parameters

The MQMHBUF call has the following parameters:

HCONN (10-digit signed integer) - input

This handle represents the connection to the queue manager.

The value of *HCONN* must match the connection handle that was used to create the message handle specified in the **HMSG** parameter.

If the message handle was created by using HCUNAS, a valid connection must be established on the thread deleting the message handle. If a valid connection is not established, the call fails with RC2009.

HMSG (20-digit signed integer) - input

This handle is the message handle for which a buffer is required.

The value was returned by a previous MQCRTMH call.

MHBOPT (MQMHBO) - input

The MQMHBO structure allows applications to specify options that control how buffers are produced from message handles.

See "MQBMHO (Buffer to message handle options) on IBM i" on page 3023 for details.

PRNAME (MQCHARV) - input

The name of the property or properties to put into the buffer.

If no property matching the name can be found, the call fails with RC2471.

Wildcards

You can use a wildcard to put more than one property into the buffer. To do so, use the percent sign (%) at the end of the property name. This wildcard matches zero or more characters, including the period (.) character.

See Property names and Property name restrictions for further information about the use of property names.

MSGDSC (MQMD) - input/output

The *MSGDSC* structure describes the contents of the buffer area.

On output, the *Encoding*, *CodedCharSetId* and *Format* fields are set to correctly describe the encoding, character set identifier, and format of the data in the buffer area as written by the call.

Data in this structure is in the character set and encoding of the application.

BUFLEN (10-digit signed integer) - input

BUFFLEN is the length of the Buffer area, in bytes.

BUFFER (1-byte bit string x BUFLEN) - input/output

BUFFER defines the area containing the message buffer. For most data, you must align the buffer on a 4-byte boundary.

If *BUFFER* contains character or numeric data, set the *CodedCharSetId* and *Encoding* fields in the **MSGDSC** parameter to the values appropriate to the data; this enables the data to be converted, if necessary.

If properties are found in the message buffer they are optionally removed; they later become available from the message handle on return from the call.

In the C programming language, the parameter is declared as a pointer-to-void, which means the address of any type of data can be specified as the parameter.

If the **BUFLEN** parameter is zero, *BUFFER* is not referred to. In this case, the parameter address passed by programs written in C or System/390 assembler can be null.

DATLEN (10-digit signed integer) - output

DATLEN is the length, in bytes, of the returned properties in the buffer. If the value is zero, no properties matched the value given in *PRNAME* and the call fails with reason code RC2471.

If *BUFLEN* is less than the length required to store the properties in the buffer, the MQMHBUF call fails with RC2469, but a value is still entered into *DATLEN*. This allows the application to determine the size of the buffer required to accommodate the properties, and then reissue the call with the required *BUFLEN*.

CMPCOD (10-digit signed integer) - output

The completion code; it is one of the following:

CCOK
Successful completion.

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

The reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCFAIL:

RC2204
(2204, X'089C') Adapter not available.

RC2130
(2130, X'852') Unable to load adapter service module.

RC2157
(2157, X'86D') Primary and home ASIDs differ.

RC2501
(2501, X'095C') Message handle to buffer options structure not valid.

RC2004
(2004, X'07D4') Buffer parameter not valid.

RC2005
(2005, X'07D5') Buffer length parameter not valid.

RC2219
(2219, X'08AB') MQI call entered before previous call completed.

RC2009
(2009, X'07D9') Connection to queue manager lost.

RC2010
(2010, X'07DA') Data length parameter not valid.

RC2460
(2460, X'099C') Message handle not valid.

RC2026
(2026, X'07EA') Message descriptor not valid.

RC2499
(2499, X'09C3') Message handle already in use.

RC2046
(2046, X'07FE') Options not valid or not consistent.

RC2442
(2442, X'098A') Property name is not valid.

RC2471
(2471, X'09A7') Property not available.

RC2469
(2469, X'09A5') BufferLength value is too small to contain specified properties.

RC2195
(2195, X'893') Unexpected error occurred.

RPG Declaration

```
C*..1...:....2...:....3...:....4...:....5...:....6...:....7..
C                     CALLP     MQMHBUF(HCONN : HMSG : MHBOPT :
                                        PRNAME : MSGDSC : BUFLEN :
                                        BUFFER : DATLEN :
                                        CMPCOD : REASON)
```

The prototype definition for the call is:

```
DMQMHBUF          PR                  EXTPROC('MQMHBUF')
D* Connection handle
D HCONN                           10I 0 VALUE
D* Message handle
D HMSG                            20I 0 VALUE
D* Options that control the action of MQMHBUF
```

```
D MHBOPT                        12A
D* Property name
D PRNAME                        32A
D* Message descriptor
D MSGDSC                       364A
D* Length in bytes of the Buffer area
D BUFLEN                        10I 0 VALUE
D* Area to contain the properties
D BUFFER                          *   VALUE
D* Length of the properties
D DATLEN                        10I 0
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CompCode
D REASON                        10I 0
```

MQOPEN (Open object) on IBM i: 

The MQOPEN call establishes access to an object.

The following types of object are valid:
- Queue (including distribution lists)
- Namelist
- Process definition
- Queue manager
- Topic
- "Syntax"
- "Usage notes"
- "Parameters" on page 3341
- "RPG Declaration" on page 3348

Syntax

MQOPEN *(HCONN, OBJDSC, OPTS, HOBJ, CMPCOD, REASON)*

Usage notes

1. The object opened is one of the following:
   - A queue, in order to:
     – Get or browse messages (using the MQGET call)
     – Put messages (using the MQPUT call)
     – Inquire about the attributes of the queue (using the MQINQ call)
     – Set the attributes of the queue (using the MQSET call)

     If the queue named is a model queue, a dynamic local queue is created.

     A distribution list is a special type of queue object that contains a list of queues. It can be opened to put messages, but not to get or browse messages, or to inquire or set attributes. See usage note 8 for further details.

     A queue that has `QSGDISP(GROUP)` is a special type of queue definition that cannot be used with the MQOPEN or MQPUT1 calls.
   - A namelist, in order to:
     – Inquire about the names of the queues in the list (using the MQINQ call).
   - A process definition, in order to:
     – Inquire about the process attributes (using the MQINQ call).

- The queue manager, in order to:
  - Inquire about the attributes of the local queue manager (using the MQINQ call).

2. It is valid for an application to open the same object more than once. A different object handle is returned for each open. Each handle that is returned can be used for the functions for which the corresponding open was performed.
3. If the object being opened is a queue but not a cluster queue, all name resolution within the local queue manager takes place at the time of the MQOPEN call. This might include one or more of the following for a particular MQOPEN call:
   - Alias resolution to the name of a base queue
   - Resolution of the name of a local definition of a remote queue to the name of the remote queue manager, and the name by which the queue is known at the remote queue manager
   - Resolution of the remote queue manager name to the name of a local transmission queue

   However, be aware that subsequent MQINQ or MQSET calls for the handle relate solely to the name that has been opened, and not to the object resulting after name resolution has occurred. For example, if the object opened is an alias, the attributes returned by the MQINQ call are the attributes of the alias, not the attributes of the base queue to which the alias resolves. Name resolution checking is still carried out, however, regardless of what is specified for the **OPTS** parameter on the corresponding MQOPEN.

   If the object being opened is a cluster queue, name resolution can occur at the time of the MQOPEN call, or be deferred until later. The point at which resolution occurs is controlled by the OOBND* options specified on the MQOPEN call:
   - OOBNDO
   - OOBNDN
   - OOBNDQ

   See Name resolution for more information about name resolution for cluster queues.
4. The attributes of an object can change while an application has the object open. In many cases, the application does not notice this, but for certain attributes the queue manager marks the handle as no longer valid. These are:
   - Any attribute that affects the name resolution of the object. This applies regardless of the open options used, and includes the following:
     - A change to the **BaseQName** attribute of an alias queue that is open.
     - A change to the **RemoteQName** or **RemoteQMgrName** queue attributes, for any handle that is open for this queue, or for a queue which resolves through this definition as a queue manager alias.
     - Any change that causes a currently open handle for a remote queue to resolve to a different *transmission* queue, or to fail to resolve to one at all. For example, this can include:
       - A change to the **XmitQName** attribute of the local definition of a remote queue, whether the definition is being used for a queue, or for a queue manager alias.

       There is one exception to this, namely the creation of a new transmission queue. A handle that would have resolved to this queue had it been present when the handle was opened, but instead resolved to the default transmission queue, is not made invalid.
     - A change to the **DefXmitQName** queue manager attribute. In this case all open handles that resolved to the previously named queue (that resolved to it only because it was the default transmission queue) are marked as invalid. Handles that resolved to this queue for other reasons are not affected.
   - The **Shareability** queue attribute, if there are two or more handles that are currently providing OOINPS access for this queue, or for a queue that resolves to this queue. If so, *all* handles that are open for this queue, or for a queue that resolves to this queue, are marked as invalid, regardless of the open options.
   - The **Usage** queue attribute, for all handles that are open for this queue, or for a queue that resolves to this queue, regardless of the open options.

When a handle is marked as invalid, all subsequent calls (other than MQCLOSE) using this handle fail with reason code RC2041; the application should issue an MQCLOSE call (using the original handle) and then reopen the queue. Any uncommitted updates against the old handle from previous successful calls can still be committed or backed out, as required by the application logic.

If changing an attribute will cause this to happen, a special "force" version of the command must be used.

5. The queue manager performs security checks when an MQOPEN call is issued, to verify that the user identifier under which the application is running has the appropriate level of authority before access is permitted. The authority check is made on the name of the object being opened, and not on the name, or names, resulting after a name has been resolved.

   If the object being opened is a model queue, the queue manager performs a full security check against both the name of the model queue and the name of the dynamic queue that is created. If the resulting dynamic queue is then opened explicitly, a further resource security check is performed against the name of the dynamic queue.

6. A remote queue can be specified in one of two ways in the **OBJDSC** parameter of this call (see the *ODON* and *ODMN* fields described in "MQOD (Object descriptor) on IBM i" on page 3170):

   - By specifying for *ODON* the name of a local definition of the remote queue. In this case, *ODMN* refers to the local queue manager, and can be specified as blanks.

     The security validation performed by the local queue manager verifies that the user is authorized to open the local definition of the remote queue.
   - By specifying for *ODON* the name of the remote queue as known to the remote queue manager. In this case, *ODMN* is the name of the remote queue manager.

     The security validation performed by the local queue manager verifies that the user is authorized to send messages to the transmission queue resulting from the name resolution process.

   In either case:

   - No messages are sent by the local queue manager to the remote queue manager in order to check that the user is authorized to put messages on the queue.
   - When a message arrives at the remote queue manager, the remote queue manager might reject it because the user originating the message is not authorized.

7. An MQOPEN call with the OOBRW option establishes a browse cursor, for use with MQGET calls that specify the object handle and one of the browse options. This allows the queue to be scanned without altering its contents. A message that has been found by browsing can later be removed from the queue by using the GMMUC option.

   Multiple browse cursors can be active for a single application by issuing several MQOPEN requests for the same queue.

8. The following notes apply to the use of distribution lists.

   - Fields in the MQOD structure must be set as follows when opening a distribution list:
     - *ODVER* must be ODVER2 or greater.
     - *ODOT* must be OTQ.
     - *ODON* must be blank or the null string.
     - *ODMN* must be blank or the null string.
     - *ODREC* must be greater than zero.
     - One of *ODORO* and *ODORP* must be zero and the other nonzero.
     - No more than one of *ODRRO* and *ODRRP* can be nonzero.
     - There must be *ODREC* object records, addressed by either *ODORO* or *ODORP*. The object records must be set to the names of the destination queues to be opened.
     - If one of *ODRRO* and *ODRRP* is nonzero, there must be *ODREC* response records present. They are set by the queue manager if the call completes with reason code RC2136.

A version-2 MQOD can also be used to open a single queue that is not in a distribution list, by ensuring that *ODREC* is zero.

- Only the following open options are valid in the **OPTS** parameter:
  - OOOUT
  - OOPAS*
  - OOSET*
  - OOALTU
  - OOFIQ
- The destination queues in the distribution list can be local, alias, or remote queues, but they cannot be model queues. If a model queue is specified, that queue fails to open, with reason code RC2057. However, this does not prevent other queues in the list being opened successfully.
- The completion code and reason code parameters are set as follows:
  - If the open operations for the queues in the distribution list all succeed or fail in the same way, the completion code and reason code parameters are set to describe the common result. The MQRR response records (if provided by the application) are not set in this case.

    For example, if every open succeeds, the completion code is set to CCOK and the reason code is RCNONE; if every open fails because none of the queues exists, the parameters are set to CCFAIL and RC2085.
  - If the open operations for the queues in the distribution list do not all succeed or fail in the same way:
    - The completion code parameter is set to CCWARN if at least one open succeeded, and to CCFAIL if all failed.
    - The reason code parameter is set to RC2136.
    - The response records (if provided by the application) are set to the individual completion codes and reason codes for the queues in the distribution list.
- When a distribution list has been opened successfully, the handle *HOBJ* returned by the call can be used on subsequent MQPUT calls to put messages to queues in the distribution list, and on an MQCLOSE call to relinquish access to the distribution list. The only valid close option for a distribution list is CONONE.

  The MQPUT1 call can also be used to put a message to a distribution list; the MQOD structure defining the queues in the list is specified as a parameter on that call.
- Each successfully opened destination in the distribution list counts as a *separate* handle when checking whether the application has exceeded the permitted maximum number of handles (see the **MaxHandles** queue manager attribute). This is true even when two or more of the destinations in the distribution list actually resolve to the same physical queue. If the MQOPEN or MQPUT1 call for a distribution list would cause the number of handles in use by the application to exceed *MaxHandles*, the call fails with reason code RC2017.
- Each destination that is opened successfully has the value of its **OpenOutputCount** attribute incremented by one. If two or more of the destinations in the distribution list actually resolve to the same physical queue, that queue has its **OpenOutputCount** attribute incremented by the number of destinations in the distribution list that resolve to that queue.
- Any change to the queue definitions that would have caused a handle to become invalid had the queues been opened individually (for example, a change in the resolution path), does not cause the distribution-list handle to become invalid. However, it does result in a failure for that particular queue when the distribution-list handle is used on a subsequent MQPUT call.
- It is valid for a distribution list to contain only one destination.

9. The following notes apply to the use of cluster queues.
   - When a cluster queue is opened for the first time, and the local queue manager is not a full repository queue manager, the local queue manager obtains information about the cluster queue from a full repository queue manager. When the network is busy, it may take several seconds for

the local queue manager to receive the needed information from the repository queue manager. As a result, the application issuing the MQOPEN call might have to wait for up to 10 seconds before control returns from the MQOPEN call. If the local queue manager does not receive the needed information about the cluster queue within this time, the call fails with reason code RC2189.

- When a cluster queue is opened and there are multiple instances of the queue in the cluster, the instance actually opened depends on the options specified on the MQOPEN call:
  - If the options specified include any of the following:
    - OOBRW
    - OOINPQ
    - OOINPX
    - OOINPS
    - OOSET

    the instance of the cluster queue opened is required to be the local instance. If there is no local instance of the queue, the MQOPEN call fails.
  - If the options specified include none of the above, but do include one or both of the following:
    - OOINQ
    - OOOUT

    the instance opened is the local instance if there is one, and a remote instance otherwise. The instance chosen by the queue manager can, however, be altered by a cluster workload exit (if there is one).

For more information about cluster queues, see Cluster queues.

10. Applications started by a trigger monitor are passed the name of the queue that is associated with the application when the application is started. This queue name can be specified in the **OBJDSC** parameter to open the queue. See the description of the MQTMC structure for further details.
11. When using the OORLOQ option, the local queue is already returned when either a local, alias, or model queue is opened, but this is not the case when, for example, a remote queue or a non-local cluster queue is opened; the ResolvedQName and ResolvedQMgrName are entered with the RemoteQName and RemoteQMgrName found in the remote queue definition, or similarly with the chosen remote cluster queue. If OORLOQ is specified when opening, for example, a remote queue, ResolvedQName will now be the transmission queue which messages will be put to. The ResolvedQMgrName will be entered with the name of the local queue manager hosting the transmission queue. If a user is authorized for browse, input or output on a queue, they have the required authority to specify this flag on the MQOPEN call. No special authority is needed.

Parameters

The MQOPEN call has the following parameters:

HCONN (10-digit signed integer) - input

Connection handle.

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN or MQCONNX call.

OBJDSC (MQOD) - input/output

Object descriptor.

This is a structure that identifies the object to be opened; see “MQOD (Object descriptor) on IBM i” on page 3170 for details.

If the *ODON* field in the **OBJDSC** parameter is the name of a model queue, a dynamic local queue is created with the attributes of the model queue; this happens irrespective of the open options specified by the **OPTS** parameter. Subsequent operations using the *HOBJ* returned by the MQOPEN

call are performed on the new dynamic queue, and not on the model queue. This is true even for the MQINQ and MQSET calls. The name of the model queue in the **OBJDSC** parameter is replaced with the name of the dynamic queue created. The type of the dynamic queue is determined by the value of the **DefinitionType** attribute of the model queue (see "Attributes for queues" on page 3385). For information about the close options applicable to dynamic queues, see the description of the MQCLOSE call.

OPTS (10-digit signed integer) - input

Options that control the action of MQOPEN.

At least one of the following options must be specified:

- OOBRW
- OOINP* (only one of these)
- OOINQ
- OOOUT
- OOSET
- OORLQ

Other options can be specified as required. If more than one option is required, the values can be added (do not add the same constant more than once). Combinations that are not valid are noted; all other combinations are valid. Only options that are applicable to the type of object specified by *OBJDSC* are allowed (see Valid MQOPEN options for each queue type).

Access options: The following options control the type of operations that can be performed on the object:

OOINPQ
Open queue to get messages using queue-defined default.

The queue is opened for use with subsequent MQGET calls. The type of access is either shared or exclusive, depending on the value of the **DefInputOpenOption** queue attribute; see "Attributes for queues" on page 3385 for details.

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues.

OOINPS
Open queue to get messages with shared access.

The queue is opened for use with subsequent MQGET calls. The call can succeed if the queue is currently open by this or another application with OOINPS, but fails with reason code RC2042 if the queue is currently open with OOINPX.

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues.

OOINPX
Open queue to get messages with exclusive access.

The queue is opened for use with subsequent MQGET calls. The call fails with reason code RC2042 if the queue is currently open by this or another application for input of any type (OOINPS or OOINPX).

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues.

The following notes apply to these options:

- Only one of these options can be specified.

- An MQOPEN call with one of these options can succeed even if the **InhibitGet** queue attribute is set to QAGETI (although subsequent MQGET calls will fail while the attribute is set to this value).
- If the queue is defined as not being shareable (that is, the **Shareability** queue attribute has the value QANSHR), attempts to open the queue for shared access are treated as attempts to open the queue with exclusive access.
- If an alias queue is opened with one of these options, the test for exclusive use (or for whether another application has exclusive use) is against the base queue to which the alias resolves.
- These options are not valid if *ODMN* is the name of a queue manager alias; this is true even if the value of the **RemoteQMgrName** attribute in the local definition of a remote queue used for queue manager aliasing is the name of the local queue manager.

OOBRW

Open queue to browse messages.

The queue is opened for use with subsequent MQGET calls with one of the following options:

- GMBRWF
- GMBRWN
- GMBRWC

This is allowed even if the queue is currently open for OOINPX. An MQOPEN call with the OOBRW option establishes a browse cursor, and positions it logically before the first message on the queue; see the *GMOPT* field described in "MQGMO (Get-message options) on IBM i" on page 3084 for further information.

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects which are not queues. It is also not valid if *ODMN* is the name of a queue manager alias; this is true even if the value of the **RemoteQMgrName** attribute in the local definition of a remote queue used for queue manager aliasing is the name of the local queue manager.

OOOUT

Open queue to put messages, or a topic or topic string to publish messages.

The queue is opened for use with subsequent MQPUT calls.

An MQOPEN call with this option can succeed even if the **InhibitPut** queue attribute is set to QAPUTI (although subsequent MQPUT calls will fail while the attribute is set to this value).

This option is valid for all types of queue, including distribution lists and topics.

OOINQ

Open object to inquire attributes.

The queue, namelist, process definition, or queue manager is opened for use with subsequent MQINQ calls.

This option is valid for all types of object other than distribution lists. It is not valid if *ODMN* is the name of a queue manager alias; this is true even if the value of the **RemoteQMgrName** attribute in the local definition of a remote queue used for queue manager aliasing is the name of the local queue manager.

OOSET

Open queue to set attributes.

The queue is opened for use with subsequent MQSET calls.

This option is valid for all types of queue other than distribution lists. It is not valid if *ODMN* is the name of a local definition of a remote queue; this is true even if the value of

the **RemoteQMgrName** attribute in the local definition of a remote queue used for queue manager aliasing is the name of the local queue manager.

Binding options: The following options apply when the object being opened is a cluster queue; these options control the binding of the queue handle to an instance of the cluster queue:

OOBNDO
Bind handle to destination when queue is opened.

This causes the local queue manager to bind the queue handle to an instance of the destination queue when the queue is opened. As a result, all messages put using this handle are sent to the same instance of the destination queue, and by the same route.

This option is valid only for queues, and affects only cluster queues. If specified for a queue that is not a cluster queue, the option is ignored.

OOBNDN
Do not bind to a specific destination.

This stops the local queue manager binding the queue handle to an instance of the destination queue. As a result, successive MQPUT calls using this handle may result in the messages being sent to *different* instances of the destination queue, or being sent to the same instance but by different routes. It also allows the instance selected to be changed later by the local queue manager, by a remote queue manager, or by a message channel agent (MCA), according to network conditions.

Note: Client and server applications which need to exchange a *series* of messages in order to complete a transaction should not use OOBNDN (or OOBNDQ when *DefBind* has the value BNDNOT), because successive messages in the series may be sent to different instances of the server application.

If OOBRW or one of the OOINP* options is specified for a cluster queue, the queue manager is forced to select the local instance of the cluster queue. As a result, the binding of the queue handle is fixed, even if OOBNDN is specified.

If OOINQ is specified with OOBNDN, successive MQINQ calls using that handle may inquire different instances of the cluster queue, although typically all of the instances have the same attribute values.

OOBNDN is valid only for queues, and affects only cluster queues. If specified for a queue that is not a cluster queue, the option is ignored.

OOBNDQ
Use default binding for queue.

This causes the local queue manager to bind the queue handle in the way defined by the **DefBind** queue attribute. The value of this attribute is either BNDOPN or BNDNOT.

OOBNDQ is the default if OOBNDO and OOBNDN are not specified.

OOBNDQ is defined to aid program documentation. It is not intended that this option is used with either of the other two bind options, but because its value is zero such use cannot be detected.

Context options: The following options control the processing of message context:

OOSAVA
Save context when message retrieved.

Context information is associated with this queue handle. This information is set from the context of any message retrieved using this handle. For more information about message context, see Message context and Controlling context information.

This context information can be passed to a message that is later put on a queue using the MQPUT or MQPUT1 calls. See the PMPASI and PMPASA options described in "MQPMO (Put-message options) on IBM i" on page 3185.

Until a message has been successfully retrieved, context cannot be passed to a message being put on a queue.

A message retrieved using one of the GMBRW* browse options does not have its context information saved (although the context fields in the **MSGDSC** parameter are set after a browse).

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects which are not queues. One of the OOINP* options must be specified.

OOPASI
Allow identity context to be passed.

This allows the PMPASI option to be specified in the **PMO** parameter when a message is put on a queue; this gives the message the identity context information from an input queue that was opened with the OOSAVA option. For more information about message context, see Message context and Controlling context information.

The OOOUT option must be specified.

This option is valid for all types of queue, including distribution lists.

OOPASA
Allow all context to be passed.

This allows the PMPASA option to be specified in the **PMO** parameter when a message is put on a queue; this gives the message the identity and origin context information from an input queue that was opened with the OOSAVA option. For more information about message context, see Message context and Controlling context information.

This option implies OOPASI, which need not therefore be specified. The OOOUT option must be specified.

This option is valid for all types of queue, including distribution lists.

OOSETI
Allow identity context to be set.

This allows the PMSETI option to be specified in the **PMO** parameter when a message is put on a queue; this gives the message the identity context information contained in the **MSGDSC** parameter specified on the MQPUT or MQPUT1 call. For more information about message context, see Message context and Controlling context information.

This option implies OOPASI, which need not therefore be specified. The OOOUT option must be specified.

This option is valid for all types of queue, including distribution lists.

OOSETA
Allow all context to be set.

This allows the PMSETA option to be specified in the **PMO** parameter when a message is put on a queue; this gives the message the identity and origin context information contained in the **MSGDSC** parameter specified on the MQPUT or MQPUT1 call. For more information about message context, see Message context and Controlling context information.

This option implies the following options, which need not therefore be specified:

- OOPASI

- OOPASA
- OOSETI

The OOOUT option must be specified.

This option is valid for all types of queue, including distribution lists.

Other options: The following options control authorization checking, and what happens when the queue manager is quiescing:

OOALTU
Validate with specified user identifier.

This indicates that the *ODAU* field in the **OBJDSC** parameter contains a user identifier that is to be used to validate this MQOPEN call. The call can succeed only if this *ODAU* is authorized to open the object with the specified access options, regardless of whether the user identifier under which the application is running is authorized to do so. This does not apply to any context options specified, however, which are always checked against the user identifier under which the application is running.

This option is valid for all types of object.

OOFIQ
Fail if queue manager is quiescing.

This option forces the MQOPEN call to fail if the queue manager is in quiescing state.

This option is valid for all types of object.

OORLQ
Enter the name of local queue that was opened.

This option specifies that the ResolvedQName in the MQOD structure (if available) should be entered with the name of the local queue which was opened. The ResolvedQMgrName will similarly be entered with the name of the local queue manager hosting the local queue.

Valid MQOPEN options for each queue type

Option	Alias (1 on page 3347)	Local and Model	Remote	Nonlocal Cluster	Distribution list	Topic
OOINPQ	✓	✓	-	-	-	-
OOINPS	✓	✓	-	-	-	-
OOINPX	✓	✓	-	-	-	-
OOBRW	✓	✓	-	-	-	-
OOOUT	✓	✓	✓	✓	✓	✓
OOINQ	✓	✓	2 on page 3347	✓	-	-
OOSET	✓	✓	2 on page 3347	-	-	-
OOBNDO (3 on page 3347)	✓	✓	✓	✓	✓	-
OOBNDN (3 on page 3347)	✓	✓	✓	✓	✓	-

Option	Alias (1)	Local and Model	Remote	Nonlocal Cluster	Distribution list	Topic
OOBNDQ (3)	✓	✓	✓	✓	✓	-
OOSAVA	✓	✓	-	-	-	-
OOPASI	✓	✓	✓	✓	✓	5
OOPASA	✓	✓	✓	✓	✓	5
OOSETI	✓	✓	✓	✓	✓	5
OOSETA	✓	✓	✓	✓	✓	5
OOALTU	✓	✓	✓	✓	✓	✓
OOFIQ	✓	✓	✓	✓	✓	✓
OORLQ	✓	✓	✓	✓	-	-

Notes:

1. The validity of options for aliases depends on the validity of the option for the queue to which the alias resolves.
2. This option is valid only for the local definition of a remote queue.
3. This option can be specified for any queue type, but is ignored if the queue is not a cluster queue.
4. This attribute is ignored for a topic.
5. These attributes can be used with a topic, but only affect the context set for the retained message, not the context fields sent to any subscriber.

HOBJ (10-digit signed integer) - output

Object handle.

This handle represents the access that has been established to the object. It must be specified on subsequent message queuing calls that operate on the object. It ceases to be valid when the MQCLOSE call is issued, or when the unit of processing that defines the scope of the handle terminates.

The scope of the handle is restricted to the smallest unit of parallel processing supported by the platform on which the application is running; the handle is not valid outside the unit of parallel processing from which the MQOPEN call was issued:

- On IBM i, the scope of the handle is the job issuing the call.

CMPCOD (10-digit signed integer) - output

Completion code.

It is one of the following:

CCOK
Successful completion.

CCWARN
Warning (partial completion).

CCFAIL
Call failed.

RPG Declaration

```
C*..1...:....2...:....3...:....4...:....5...:....6...:....7..
C                     CALLP     MQOPEN(HCONN : OBJDSC : OPTS :
C                                      HOBJ : CMPCOD : REASON)
```

The prototype definition for the call is:

```
D*..1...:....2...:....3...:....4...:....5...:....6...:....7..
DMQOPEN           PR                  EXTPROC('MQOPEN')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Object descriptor
D OBJDSC                       468A
D* Options that control the action of MQOPEN
D OPTS                          10I 0 VALUE
D* Object handle
D HOBJ                          10I 0
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
```

MQPUT (Put message) on IBM i: IBM i

The MQPUT call puts a message on a queue, distribution list or to a topic. The queue, distribution list, or topic must already be open.

Syntax

MQPUT *(HCONN, HOBJ, MSGDSC, PMO, BUFLEN, BUFFER, CMPCOD, REASON)*

Usage notes

Topics

The following notes apply to the use of topics:

1. When using MQPUT to publish messages on a topic, where one or more subscribers to that topic cannot be given the publication due to a problem with their subscriber queue (for example it is full), the Reason code returned to the MQPUT call and the delivery behavior is dependent on the setting of the PMSGDLV or NPMSGDLV attributes on the TOPIC. Note that delivery of a publication to the dead letter queue when RODLQ is specified, or discarding the message when RODISC is specified, is considered a successful delivery of the message. If none of the publications are delivered, the MQPUT will return with RC2502. This can happen in the following cases:

- A message is published to a TOPIC with PMSGDLV or NPMSGDLV (depending on the persistence of the message) set to ALL and any subscription (durable or not) has a queue which cannot receive the publication.
- A message is published to a TOPIC with PMSGDLV or NPMSGDLV (depending on the persistence of the message) set to ALLDUR and a durable subscription has a queue which cannot receive the publication.

The MQPUT can return with RCNONE even though publications could not be delivered to some subscribers in the following cases:

- A message is published to a TOPIC with PMSGDLV or NPMSGDLV (depending on the persistence of the message) set to ALLAVAIL and any subscription, durable or not, has a queue which cannot receive the publication.
- A message is published to a TOPIC with PMSGDLV or NPMSGDLV (depending on the persistence of the message) set to ALLDUR and a non-durable subscription has a queue which cannot receive the publication.

2. If there are no subscribers to the topic being used, the message published is not sent to any queue and is discarded. It does not make any difference whether this message is persistent or non-persistent, or whether it has unlimited expiry or some small expiry time, it is still discarded if there are no subscribers. The exception to this is if the message is to be retained, in which case, although it is not sent to any subscribers' queues, it is stored against the topic to be delivered to any new subscriptions or to any subscribers that ask for retained publications using MQSUBRQ.

MQPUT and MQPUT1

Both the MQPUT and MQPUT1 calls can be used to put messages on a queue; which call to use depends on the circumstances

- The MQPUT call should be used when multiple messages are to be placed on the *same* queue.

  An MQOPEN call specifying the OOOUT option is issued first, followed by one or more MQPUT requests to add messages to the queue; finally the queue is closed with an MQCLOSE call. This gives better performance than repeated use of the MQPUT1 call.
- The MQPUT1 call should be used when only *one* message is to be put on a queue.

  This call encapsulates the MQOPEN, MQPUT, and MQCLOSE calls into a single call, minimizing the number of calls that must be issued.

Destination queues

If an application puts a sequence of messages on the same queue without using message groups, the order of those messages is preserved if the following conditions are satisfied. Some conditions apply to both local and remote destination queues; other conditions apply only to remote destination queues.

Conditions for local and remote destination queues

- All of the MQPUT calls are within the same unit of work, or none of them is within a unit of work.

  When messages are put onto a particular queue within a single unit of work, messages from other applications might be interspersed with the sequence of messages on the queue.
- All of the MQPUT calls are made using the same object handle *HOBJ*.

  In some environments, message sequence is also preserved when different object handles are used, provided the calls are made from the same application. The meaning of "same application" is determined by the environment:
  - On IBM i, the application is the job.
- The messages all have the same priority.

Additional conditions for remote destination queues

- There is only one path from the sending queue manager to the destination queue manager.

If there is a possibility that some messages in the sequence may go on a different path (for example, because of reconfiguration, traffic balancing, or path selection based on message size), the order of the messages at the destination queue manager cannot be guaranteed.

- Messages are not placed temporarily on dead-letter queues at the sending, intermediate, or destination queue managers.

  If one or more of the messages is put temporarily on a dead-letter queue (for example, because a transmission queue or the destination queue is temporarily full), the messages can arrive on the destination queue out of sequence.

- The messages are either all persistent or all nonpersistent.

  If a channel on the route between the sending and destination queue managers has its **CDNPM** attribute set to NPFAST, nonpersistent messages can jump ahead of persistent messages, resulting in the order of persistent messages relative to nonpersistent messages not being preserved. However, the order of persistent messages relative to each other, and of nonpersistent messages relative to each other, is preserved.

If these conditions are not satisfied, message groups can be used to preserve message order, but note that this requires both the sending and receiving applications to use the message-grouping support. For more information about message groups, see:

- *MDMFL* field in MQMD
- PMLOGO option in MQPMO
- GMLOGO option in MQGMO

Distribution lists

The following notes apply to the use of distribution lists.

1. Messages can be put to a distribution list using either a version-1 or a version-2 MQPMO. If a version-1 MQPMO is used (or a version-2 MQPMO with *PMREC* equal to zero), no put message records or response records can be provided by the application. This means that it will not be possible to identify the queues which encounter errors, if the message is sent successfully to some queues in the distribution list and not others.

   If put message records or response records are provided by the application, the *PMVER* field must be set to PMVER2.

   A version-2 MQPMO can also be used to send messages to a single queue that is not in a distribution list, by ensuring that *PMREC* is zero.

2. The completion code and reason code parameters are set as follows:
   - If the puts to the queues in the distribution list all succeed or fail in the same way, the completion code and reason code parameters are set to describe the common result. The MQRR response records (if provided by the application) are not set in this case.

     For example, if every put succeeds, the completion code is set to CCOK and the reason code is RCNONE; if every put fails because all of the queues are inhibited for puts, the parameters are set to CCFAIL and RC2051.

   - If the puts to the queues in the distribution list do not all succeed or fail in the same way:
     - The completion code parameter is set to CCWARN if at least one put succeeded, and to CCFAIL if all failed.
     - The reason code parameter is set to RC2136.
     - The response records (if provided by the application) are set to the individual completion codes and reason codes for the queues in the distribution list.

     If the put to a destination fails because the open for that destination failed, the fields in the response record are set to CCFAIL and RC2137; that destination is included in *PMIDC*.

3. If a destination in the distribution list resolves to a local queue, the message is placed on that queue in normal form (that is, not as a distribution-list message). If more than one destination resolves to the same local queue, one message is placed on the queue for each such destination.

   If a destination in the distribution list resolves to a remote queue, a message is placed on the appropriate transmission queue. Where several destinations resolve to the same transmission queue, a single distribution-list message containing those destinations may be placed on the transmission queue, even if those destinations were not adjacent in the list of destinations provided by the application. However, this can be done only if the transmission queue supports distribution-list messages (see the **DistLists** queue attribute described in "Attributes for queues" on page 3385).

   If the transmission queue does not support distribution lists, one copy of the message in normal form is placed on the transmission queue for each destination that uses that transmission queue.

   If a distribution list with the application message data is too large for a transmission queue, the distribution list message is split up into smaller distribution-list messages, each containing fewer destinations. If the application message data only just fits on the queue, distribution-list messages cannot be used at all, and the queue manager generates one copy of the message in normal form for each destination that uses that transmission queue.

   If different destinations have different message priority or message persistence (this can occur when the application specifies PRQDEF or PEQDEF), the messages are not held in the same distribution-list message. Instead, the queue manager generates as many distribution-list messages as are necessary to accommodate the differing priority and persistence values.
4. A put to a distribution list might result in:
   - A single distribution-list message, or
   - A number of smaller distribution-list messages, or
   - A mixture of distribution list messages and normal messages, or
   - Normal messages only.

   Which of the previous occurs depends on whether:
   - The destinations in the list are local, remote, or a mixture.
   - The destinations have the same message priority and message persistence.
   - The transmission queues can hold distribution-list messages.
   - The transmission queues' maximum message lengths are large enough to accommodate the message in distribution-list form.

   However, regardless of which of the above occurs, each *physical* message resulting (that is, each normal message or distribution-list message resulting from the put) counts as only *one* message when:
   - Checking whether the application has exceeded the permitted maximum number of messages in a unit of work (see the **MaxUncommittedMsgs** queue manager attribute).
   - Checking whether the triggering conditions are satisfied.
   - Incrementing queue depths and checking whether the queues' maximum queue depth would be exceeded.
5. Any change to the queue definitions that would have caused a handle to become invalid had the queues been opened individually (for example, a change in the resolution path), does not cause the distribution-list handle to become invalid. However, it does result in a failure for that particular queue when the distribution-list handle is used on a subsequent MQPUT call.

Headers

If a message is put with one or more IBM MQ header structures at the beginning of the application message data, the queue manager performs certain checks on the header structures to verify that they are valid. If the queue manager detects an error, the call fails with an appropriate reason code. The checks performed vary according to the particular structures that are present. In addition, the checks are

performed only if a version-2 or later MQMD is used on the MQPUT or MQPUT1 call; the checks are not performed if a version-1 MQMD is used, even if an MQMDE is present at the start of the application message data.

The following IBM MQ header structures are validated completely by the queue manager: MQDH, MQMDE.

For other IBM MQ header structures, the queue manager performs some validation, but does not check every field. Structures that are not supported by the local queue manager, and structures following the first MQDLH in the message, are not validated.

In addition to general checks on the fields in IBM MQ structures, the following conditions must be satisfied:

- An IBM MQ structure must not be split over two or more segments - the structure must be entirely contained within one segment.
- The sum of the lengths of the structures in a PCF message must equal the length specified by the **BUFLEN** parameter on the MQPUT or MQPUT1 call. A PCF message is a message that has one of the following format names:
  - FMADMN
  - FMEVNT
  - FMPCF
- IBM MQ structures must not be truncated, except in the following situations where truncated structures are permitted:
  - Messages which are report messages.
  - PCF messages.
  - Messages containing an MQDLH structure. (Structures *following* the first MQDLH can be truncated; structures preceding the MQDLH cannot.)

Buffer

The **BUFFER** parameter shown in the RPG programming example is declared as a string; this restricts the maximum length of the parameter to 256 bytes. If a larger buffer is required, the parameter should be declared instead as a structure, or as a field in a physical file. This will increase the maximum length possible to approximately 32 KB.

Parameters

The MQPUT call has the following parameters:

HCONN (10-digit signed integer) - input

Connection handle.

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN or MQCONNX call.

HOBJ (10-digit signed integer) - input

Object handle.

This handle represents the queue to which the message is added, or the topic to which the message is published. The value of *HOBJ* was returned by a previous MQOPEN call that specified the OOOUT option.

MSGDSC (MQMD) - input/output

Message descriptor.

This structure describes the attributes of the message being sent, and receives information about the message after the put request is complete. See "MQMD (Message descriptor) on IBM i" on page 3118 for details.

If the application provides a version-1 MQMD, the message data can be prefixed with an MQMDE structure in order to specify values for the fields that exist in the version-2 MQMD but not the version-1. The *MDFMT* field in the MQMD must be set to FMMDE to indicate that an MQMDE is present. See "MQMDE (Message descriptor extension) on IBM i" on page 3163 for more details.

PMO (MQPMO) - input/output

Options that control the action of MQPUT.

See "MQPMO (Put-message options) on IBM i" on page 3185 for details.

BUFLEN (10-digit signed integer) - input

Length of the message in *BUFFER*.

Zero is valid, and indicates that the message contains no application data. The upper limit for *BUFLEN* depends on various factors:

- If the destination queue is a shared queue, the upper limit is 63 KB (64 512 bytes).
- If the destination is a local queue or resolves to a local queue (but is not a shared queue), the upper limit depends on whether:
  - The local queue manager supports segmentation.
  - The sending application specifies the flag that allows the queue manager to segment the message. This flag is MFSEGA, and can be specified either in a version-2 MQMD, or in an MQMDE used with a version-1 MQMD.

  If both of these conditions are satisfied, *BUFLEN* cannot exceed 999 999 999 minus the value of the *MDOFF* field in MQMD. The longest logical message that can be put is therefore 999 999 999 bytes (when *MDOFF* is zero). However, resource constraints imposed by the operating system or environment in which the application is running may result in a lower limit.

  If one or both of the previously described conditions are not satisfied, *BUFLEN* cannot exceed the smaller of the queue's **MaxMsgLength** attribute and queue manager's **MaxMsgLength** attribute.
- If the destination is a remote queue or resolves to a remote queue, the conditions for local queues apply, *but at each queue manager through which the message must pass in order to reach the destination queue* ; in particular:
  1. The local transmission queue used to store the message temporarily at the local queue manager
  2. Intermediate transmission queues (if any) used to store the message at queue managers on the route between the local and destination queue managers
  3. The destination queue at the destination queue manager

  The longest message that can be put is therefore governed by the most restrictive of these queues and queue managers.

  When a message is on a transmission queue, additional information resides with the message data, and this reduces the amount of application data that can be carried. In this situation it is recommended that LNMHD bytes be subtracted from the *MaxMsgLength* values of the transmission queues when determining the limit for *BUFLEN*.

  Note: Only failure to comply with condition 1 can be diagnosed synchronously (with reason code RC2030 or RC2031) when the message is put. If conditions 2 or 3 are not satisfied, the message is redirected to a dead-letter (undelivered-message) queue, either at an intermediate queue manager or at the destination queue manager. If this happens, a report message is generated if one was requested by the sender.

BUFFER (1-byte bit string x BUFLEN) - input

Message data.

This is a buffer containing the application data to be sent. The buffer should be aligned on a boundary appropriate to the nature of the data in the message. 4-byte alignment should be suitable for most messages (including messages containing MQ header structures), but some messages may require more stringent alignment. For example, a message containing a 64-bit binary integer might require 8-byte alignment.

If *BUFFER* contains character data, numeric data, or both, the *MDCSI* and *MDENC* fields in the **MSGDSC** parameter should be set to the values appropriate to the data; this will enable the receiver of the message to convert the data (if necessary) to the character set and encoding used by the receiver.

Note: All of the other parameters on the MQPUT call must be in the character set given by the **CodedCharSetId** queue manager attribute, and encoding of the local queue manager given by the ENNAT.

CMPCOD (10-digit signed integer) - output

Completion code.

It is one of the following:

CCOK
Successful completion.

CCWARN
Warning (partial completion).

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

Reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCWARN:

RC2104
(2104, X'838') Report option in message descriptor not recognized.

RC2136
(2136, X'858') Multiple reason codes returned.

If *CMPCOD* is CCFAIL:

RC2004
(2004, X'7D4') Buffer parameter not valid.

RC2005
(2005, X'7D5') Buffer length parameter not valid.

RC2009
(2009, X'7D9') Connection to queue manager lost.

RC2013
(2013, X'7DD') Expiry time not valid.

RC2014
(2014, X'7DE') Feedback code not valid.

RC2018
(2018, X'7E2') Connection handle not valid.

RC2019
(2019, X'7E3') Object handle not valid.

RC2024
(2024, X'7E8') No more messages can be handled within current unit of work.

RC2026
(2026, X'7EA') Message descriptor not valid.

RC2027
(2027, X'7EB') Missing reply-to queue.

RC2029
(2029, X'7ED') Message type in message descriptor not valid.

RC2030
(2030, X'7EE') Message length greater than maximum for queue.

RC2031
(2031, X'7EF') Message length greater than maximum for queue manager.

RC2039
(2039, X'7F7') Queue not open for output.

RC2041
(2041, X'7F9') Object definition changed since opened.

RC2046
(2046, X'7FE') Options not valid or not consistent.

RC2047
(2047, X'7FF') Persistence not valid.

RC2048
(2048, X'800') Queue does not support persistent messages.

RC2050
(2050, X'802') Message priority not valid.

RC2051
(2051, X'803') Put calls inhibited for the queue.

RC2052
(2052, X'804') Queue has been deleted.

RC2053
(2053, X'805') Queue already contains maximum number of messages.

RC2056
(2056, X'808') No space available on disk for queue.

RC2058
(2058, X'80A') Queue manager name not valid or not known.

RC2059
(2059, X'80B') Queue manager not available for connection.

RC2061
(2061, X'80D') Report options in message descriptor not valid.

RC2071
(2071, X'817') Insufficient storage available.

RC2072
(2072, X'818') Syncpoint support not available.

RC2093
(2093, X'82D') Queue not open for pass all context.

RC2094
(2094, X'82E') Queue not open for pass identity context.

RC2095
(2095, X'82F') Queue not open for set all context.

RC2096
(2096, X'830') Queue not open for set identity context.

RC2097
(2097, X'831') Queue handle referred to does not save context.

RC2098
(2098, X'832') Context not available for queue handle referred to.

RC2101
(2101, X'835') Object damaged.

RC2102
(2102, X'836') Insufficient system resources available.

RC2135
(2135, X'857') Distribution header structure not valid.

RC2136
(2136, X'858') Multiple reason codes returned.

RC2137
(2137, X'859') Object not opened successfully.

RC2149
(2149, X'865') PCF structures not valid.

RC2154
(2154, X'86A') Number of records present not valid.

RC2156
(2156, X'86C') Response records not valid.

RC2158
(2158, X'86E') Put message record flags not valid.

RC2159
(2159, X'86F') Put message records not valid.

RC2161
(2161, X'871') Queue manager quiescing.

RC2162
(2162, X'872') Queue manager shutting down.

RC2173
(2173, X'87D') Put-message options structure not valid.

RC2185
(2185, X'889') Inconsistent persistence specification.

RC2188
(2188, X'88C') Call rejected by cluster workload exit.

RC2189
(2189, X'88D') Cluster name resolution failed.

RC2195
(2195, X'893') Unexpected error occurred.

RC2219
(2219, X'8AB') MQI call reentered before previous call complete.

RC2241
(2241, X'8C1') Message group not complete.

RC2242
(2242, X'8C2') Logical message not complete.

RC2245
(2245, X'8C5') Inconsistent unit-of-work specification.

RC2248
(2248, X'8C8') Message descriptor extension not valid.

RC2249
(2249, X'8C9') Message flags not valid.

RC2250
(2250, X'8CA') Message sequence number not valid.

RC2251
(2251, X'8CB') Message segment offset not valid.

RC2252
(2252, X'8CC') Original length not valid.

RC2253
(2253, X'8CD') Length of data in message segment is zero.

RC2255
(2255, X'8CF') Unit of work not available for the queue manager to use.

RC2257
(2257, X'8D1') Wrong version of MQMD supplied.

RC2258
(2258, X'8D2') Group identifier not valid.

RC2266
(2266, X'8DA') Cluster workload exit failed.

RC2269
(2269, X'8DD') Cluster resource error.

RC2270
(2270, X'8DE') No destination queues available.

RC2420
(2420) An MQPUT call was issued, but the message data contains an MQEPH structure that is not valid.

RC2479
(2479, X'9AF') Publication could not be retained.

RC2480
(2480, X'9B0') Target type has changed: the alias queue referred to a queue but now refers to a topic.

RC2502
(2502, X'9C6') Publication failed, and publication has not been delivered to any subscribers

RC2551
(2551, X'9F7') Specified selection string is not available.

RC2554
(2554, X'9FA') Message content could not be parsed to determine whether the message should be delivered to a subscriber with an extended message selector.

RPG Declaration

```
C*..1...:....2...:....3...:....4...:....5...:....6...:....7..
C                     CALLP     MQPUT(HCONN : HOBJ : MSGDSC : PMO :
C                                     BUFLEN : BUFFER : CMPCOD :
C                                     REASON)
```

The prototype definition for the call is:

```
D*..1...:....2...:....3...:....4...:....5...:....6...:....7..
DMQPUT            PR                  EXTPROC('MQPUT')
D* Connection handle
D HCONN                          10I 0 VALUE
D* Object handle
D HOBJ                           10I 0 VALUE
D* Message descriptor
D MSGDSC                        364A
D* Options that control the action of MQPUT
D PMO                           200A
D* Length of the message in Buffer
D BUFLEN                         10I 0 VALUE
D* Message data
D BUFFER                           *   VALUE
D* Completion code
D CMPCOD                         10I 0
D* Reason code qualifying CMPCOD
D REASON                         10I 0
```

MQPUT1 (Put one message) on IBM i: IBM i

The MQPUT1 call puts one message on a queue or distribution list, or to a topic. The queue, distribution list, or topic does not need to be open.

Syntax

MQPUT1 *(HCONN, OBJDSC, MSGDSC, PMO, BUFLEN, BUFFER, CMPCOD, REASON)*

Usage notes

1. Both the MQPUT and MQPUT1 calls can be used to put messages on a queue; which call to use depends on the circumstances:
   - The MQPUT call should be used when multiple messages are to be placed on the *same* queue.

     An MQOPEN call specifying the OOOUT option is issued first, followed by one or more MQPUT requests to add messages to the queue; finally the queue is closed with an MQCLOSE call. This gives better performance than repeated use of the MQPUT1 call.
   - The MQPUT1 call should be used when only *one* message is to be put on a queue.

This call encapsulates the MQOPEN, MQPUT, and MQCLOSE calls into a single call, minimizing the number of calls that must be issued.

2. If an application puts a sequence of messages on the same queue without using message groups, the order of those messages is preserved if certain conditions are satisfied. However, in most environments the MQPUT1 call does not satisfy these conditions, and so does not preserve message order. The MQPUT call must be used instead in these environments. See the usage notes in the description of the MQPUT call for details.
3. The MQPUT1 call can be used to put messages to distribution lists. For general information about this, see the usage notes for the MQOPEN and MQPUT calls.

   The following differences apply when using the MQPUT1 call:

   a. If MQRR response records are provided by the application, they must be provided using the MQOD structure; they cannot be provided using the MQPMO structure.
   b. The reason code RC2137 is never returned by MQPUT1 in the response records; if a queue fails to open, the response record for that queue contains the actual reason code resulting from the open operation.

      If an open operation for a queue succeeds with a completion code of CCWARN, the completion code and reason code in the response record for that queue are replaced by the completion and reason codes resulting from the put operation.

      As with the MQOPEN and MQPUT calls, the queue manager sets the response records (if provided) only when the outcome of the call is not the same for all queues in the distribution list; this is indicated by the call completing with reason code RC2136.
4. If the MQPUT1 call is used to put a message on a cluster queue, the call behaves as though OOBNDN had been specified on the MQOPEN call.
5. If a message is put with one or more IBM MQ header structures at the beginning of the application message data, the queue manager performs certain checks on the header structures to verify that they are valid. For more information about this, see the usage notes for the MQPUT call.
6. If more than one of the warning situations arise (see the **CMPCOD** parameter), the reason code returned is the *first* one in the following list that applies:
   a. RC2136
   b. RC2242
   c. RC2241
   d. RC2049 or RC2104
7. The **BUFFER** parameter shown in the RPG programming example is declared as a string; this restricts the maximum length of the parameter to 256 bytes. If a larger buffer is required, the parameter should be declared instead as a structure, or as a field in a physical file. This will increase the maximum length possible to approximately 32 KB.

Parameters

The MQPUT1 call has the following parameters:

HCONN (10-digit signed integer) - input

Connection handle.

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN or MQCONNX call.

OBJDSC (MQOD) - input/output

Object descriptor.

This is a structure which identifies the queue to which the message is added. See “MQOD (Object descriptor) on IBM i” on page 3170 for details.

The user must be authorized to open the queue for output. The queue must **not** be a model queue.

MSGDSC (MQMD) - input/output

Message descriptor.

This structure describes the attributes of the message being sent, and receives feedback information after the put request is complete. See "MQMD (Message descriptor) on IBM i" on page 3118 for details.

If the application provides a version-1 MQMD, the message data can be prefixed with an MQMDE structure in order to specify values for the fields that exist in the version-2 MQMD but not the version-1. The *MDFMT* field in the MQMD must be set to FMMDE to indicate that an MQMDE is present. See "MQMDE (Message descriptor extension) on IBM i" on page 3163 for more details.

PMO (MQPMO) - input/output

Options that control the action of MQPUT1.

See "MQPMO (Put-message options) on IBM i" on page 3185 for details.

BUFLEN (10-digit signed integer) - input

Length of the message in *BUFFER*.

Zero is valid, and indicates that the message contains no application data. The upper limit depends on various factors; see the description of the **BUFLEN** parameter of the MQPUT call for further details.

BUFFER (1-byte bit string x BUFLEN) - input

Message data.

This is a buffer containing the application message data to be sent. The buffer should be aligned on a boundary appropriate to the nature of the data in the message. 4-byte alignment should be suitable for most messages (including messages containing IBM MQ header structures), but some messages may require more stringent alignment. For example, a message containing a 64-bit binary integer might require 8-byte alignment.

If *BUFFER* contains character data, numeric data, or both, the *MDCSI* and *MDENC* fields in the **MSGDSC** parameter should be set to the values appropriate to the data; this will enable the receiver of the message to convert the data (if necessary) to the character set and encoding used by the receiver.

Note: All of the other parameters on the MQPUT1 call must be in the character set given by the **CodedCharSetId** queue manager attribute and encoding of the local queue manager given by ENNAT.

CMPCOD (10-digit signed integer) - output

Completion code.

It is one of the following:

CCOK
Successful completion.

CCWARN
Warning (partial completion).

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

Reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCWARN:

RC2104
(2104, X'838') Report option in message descriptor not recognized.

RC2136
(2136, X'858') Multiple reason codes returned.

RC2049
(2049, X'801') Message Priority exceeds maximum value supported.

RC2241
(2241, X'8C1') Message group not complete.

RC2242
(2242, X'8C2') Logical message not complete.

If *CMPCOD* is CCFAIL:

RC2001
(2001, X'7D1') Alias base queue not a valid type.

RC2004
(2004, X'7D4') Buffer parameter not valid.

RC2005
(2005, X'7D5') Buffer length parameter not valid.

RC2009
(2009, X'7D9') Connection to queue manager lost.

RC2013
(2013, X'7DD') Expiry time not valid.

RC2014
(2014, X'7DE') Feedback code not valid.

RC2017
(2017, X'7E1') No more handles available.

RC2018
(2018, X'7E2') Connection handle not valid.

RC2024
(2024, X'7E8') No more messages can be handled within current unit of work.

RC2026
(2026, X'7EA') Message descriptor not valid.

RC2027
(2027, X'7EB') Missing reply-to queue.

RC2029
(2029, X'7ED') Message type in message descriptor not valid.

RC2030
(2030, X'7EE') Message length greater than maximum for queue.

RC2031
(2031, X'7EF') Message length greater than maximum for queue manager.

RC2035
(2035, X'7F3') Not authorized for access.

RC2042
(2042, X'7FA') Object already open with conflicting options.

RC2043
(2043, X'7FB') Object type not valid.

RC2044
(2044, X'7FC') Object descriptor structure not valid.

RC2046
(2046, X'7FE') Options not valid or not consistent.

RC2047
(2047, X'7FF') Persistence not valid.

RC2048
(2048, X'800') Queue does not support persistent messages.

RC2050
(2050, X'802') Message priority not valid.

RC2051
(2051, X'803') Put calls inhibited for the queue.

RC2052
(2052, X'804') Queue has been deleted.

RC2053
(2053, X'805') Queue already contains maximum number of messages.

RC2056
(2056, X'808') No space available on disk for queue.

RC2057
(2057, X'809') Queue type not valid.

RC2058
(2058, X'80A') Queue manager name not valid or not known.

RC2059
(2059, X'80B') Queue manager not available for connection.

RC2061
(2061, X'80D') Report options in message descriptor not valid.

RC2063
(2063, X'80F') Security error occurred.

RC2071
(2071, X'817') Insufficient storage available.

RC2072
(2072, X'818') Syncpoint support not available.

RC2082
(2082, X'822') Unknown alias base queue.

RC2085
(2085, X'825') Unknown object name.

RC2086
(2086, X'826') Unknown object queue manager.

RC2087
(2087, X'827') Unknown remote queue manager.

RC2091
(2091, X'82B') Transmission queue not local.

RC2092
(2092, X'82C') Transmission queue with wrong usage.

RC2097
(2097, X'831') Queue handle referred to does not save context.

RC2098
(2098, X'832') Context not available for queue handle referred to.

RC2101
(2101, X'835') Object damaged.

RC2102
(2102, X'836') Insufficient system resources available.

RC2135
(2135, X'857') Distribution header structure not valid.

RC2136
(2136, X'858') Multiple reason codes returned.

RC2149
(2149, X'865') PCF structures not valid.

RC2154
(2154, X'86A') Number of records present not valid.

RC2155
(2155, X'86B') Object records not valid.

RC2156
(2156, X'86C') Response records not valid.

RC2158
(2158, X'86E') Put message record flags not valid.

RC2159
(2159, X'86F') Put message records not valid.

RC2161
(2161, X'871') Queue manager quiescing.

RC2162
(2162, X'872') Queue manager shutting down.

RC2173
(2173, X'87D') Put-message options structure not valid.

RC2184
(2184, X'888') Remote queue name not valid.

RC2188
(2188, X'88C') Call rejected by cluster workload exit.

RC2189
(2189, X'88D') Cluster name resolution failed.

RC2195
(2195, X'893') Unexpected error occurred.

RC2196
(2196, X'894') Unknown transmission queue.

RC2197
(2197, X'895') Unknown default transmission queue.

RC2198
(2198, X'896') Default transmission queue not local.

RC2199
(2199, X'897') Default transmission queue usage error.

RC2258
(2258, X'8D2') Group identifier not valid.

RC2248
(2248, X'8C8') Message descriptor extension not valid.

RC2219
(2219, X'8AB') MQI call reentered before previous call complete.

RC2249
(2249, X'8C9') Message flags not valid.

RC2250
(2250, X'8CA') Message sequence number not valid.

RC2251
(2251, X'8CB') Message segment offset not valid.

RC2252
(2252, X'8CC') Original length not valid.

RC2253
(2253, X'8CD') Length of data in message segment is zero.

RC2255
(2255, X'8CF') Unit of work not available for the queue manager to use.

RC2257
(2257, X'8D1') Wrong version of MQMD supplied.

RC2266
(2266, X'8DA') Cluster workload exit failed.

RC2269
(2269, X'8DD') Cluster resource error.

RC2270
(2270, X'8DE') No destination queues available.

RC2420
(2420) An MQPUT1 call was issued, but the message data contains an MQEPH structure that is not valid.

RC2551
(2551, X'9F7') Specified selection string is not available.

RC2554
(2554, X'9FA') Message content could not be parsed to determine whether the message should be delivered to a subscriber with an extended message selector.

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQPUT1(HCONN : OBJDSC : MSGDSC :
C                                      PMO : BUFLEN : BUFFER :
C                                      CMPCOD : REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQPUT1           PR                  EXTPROC('MQPUT1')
D* Connection handle
D HCONN                          10I 0 VALUE
D* Object descriptor
D OBJDSC                        468A
D* Message descriptor
D MSGDSC                        364A
D* Options that control the action of MQPUT1
D PMO                           200A
D* Length of the message in BUFFER
D BUFLEN                         10I 0 VALUE
D* Message data
D BUFFER                           *   VALUE
D* Completion code
D CMPCOD                         10I 0
D* Reason code qualifying CMPCOD
D REASON                         10I 0
```

MQSET (Set object attributes) on IBM i: IBM i

The MQSET call is used to change the attributes of an object represented by a handle. The object must be a queue.

- "Syntax"
- "Usage notes"
- "Parameters" on page 3366
- "RPG Declaration" on page 3369

Syntax

MQSET *(HCONN, HOBJ, SELCNT, SELS, IACNT, INTATR, CALEN, CHRATR, CMPCOD, REASON)*

Usage notes

1. Using this call, the application can specify an array of integer attributes, or a collection of character attribute strings, or both. If no errors occur, the attributes specified are all set simultaneously. If an error occurs (for example, if a selector is not valid, or an attempt is made to set an attribute to a value that is not valid), the call fails and no attributes are set.
2. The values of attributes can be determined using the MQINQ call IBM i ; see "MQINQ (Inquire about object attributes) on IBM i" on page 3320 for details.

   Note: Not all attributes with values that can be inquired upon using the MQINQ call can have their values changed using the MQSET call. For example, no process-object or queue manager attributes can be set with this call.
3. Attribute changes are preserved across restarts of the queue manager (other than alterations to temporary dynamic queues, which do not survive restarts of the queue manager).
4. You cannot change the attributes of a model queue using the MQSET call. However, if you open a model queue using the MQOPEN call with the MQOO_SET option, you can use the MQSET call to set the attributes of the dynamic local queue that is created by the MQOPEN call.

5. If the object being set is a cluster queue, there must be a local instance of the cluster queue for the open to succeed.

IBM i For more information about object attributes, see:

- "Attributes for queues" on page 3385
- "Attributes for namelists" on page 3417
- "Attributes for process definitions on IBM i" on page 3418
- "Attributes for the queue manager on IBM i" on page 3420

Parameters

The MQSET call has the following parameters:

HCONN (10-digit signed integer) - input

Connection handle.

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN or MQCONNX call.

HOBJ (10-digit signed integer) - input

Object handle.

This handle represents the queue object with attributes that are to be set. The handle was returned by a previous MQOPEN call that specified the OOSET option.

SELCNT (10-digit signed integer) - input

Count of selectors.

This is the count of selectors that are supplied in the *SELS* array. It is the number of attributes that are to be set. Zero is a valid value. The maximum number allowed is 256.

SELS (10-digit signed integer x SELCNT) - input

Array of attribute selectors.

This is an array of **SELCNT** attribute selectors; each selector identifies an attribute (integer or character) with a value that is to be set.

Each selector must be valid for the type of queue that *HOBJ* represents. Only certain IA* and CA* values are allowed; these values are listed later in this section.

Selectors can be specified in any order. Attribute values that correspond to integer attribute selectors (IA* selectors) must be specified in *INTATR* in the same order in which these selectors occur in *SELS*. Attribute values that correspond to character attribute selectors (CA* selectors) must be specified in *CHRATR* in the same order in which those selectors occur. IA* selectors can be interleaved with the CA* selectors; only the relative order within each type is important.

It is not an error to specify the same selector more than once; if this is done, the last value specified for a particular selector is the one that takes effect.

Note:

1. The integer and character attribute selectors are allocated within two different ranges; the IA* selectors reside within the range IAFRST through IALAST, and the CA* selectors within the range CAFRST through CALAST.

   For each range, the constants IALSTU and CALSTU define the highest value that the queue manager will accept.
2. If all the IA* selectors occur first, the same element numbers can be used to address corresponding elements in the *SELS* and *INTATR* arrays.

The attributes that can be set are listed in the following table. No other attributes can be set using this call. For the CA* attribute selectors, the constant that defines the length in bytes of the string that is required in *CHRATR* is provided in parentheses.

Table 368. MQSET attribute selectors for queues

Selector	Description	Note
CATRGD	Trigger data (LNTRGD).	2
IADIST	Distribution list support.	1
IAIGET	Whether get operations are allowed.	
IAIPUT	Whether put operations are allowed.	
IATRGC	Trigger control.	2
IATRGD	Trigger depth.	2
IATRGP	Threshold message priority for triggers.	2
IATRGT	Trigger type.	2

Notes:

1. Supported only on AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.
2. Not supported on VSE/ESA.

IACNT (10-digit signed integer) - input

Count of integer attributes.

This is the number of elements in the *INTATR* array, and must be at least the number of IA* selectors in the **SELS** parameter. Zero is a valid value if there are none.

INTATR (10-digit signed integ x rxIACNT) - input

Array of integer attributes.

This is an array of *IACNT* integer attribute values. These attribute values must be in the same order as the IA* selectors in the *SELS* array.

CALEN (10-digit signed integer) - input

Length of character attributes buffer.

This is the length in bytes of the **CHRATR** parameter, and must be at least the sum of the lengths of the character attributes specified in the *SELS* array. Zero is a valid value if there are no CA* selectors in *SELS*.

CHRATR (1-byte character string x CALEN) - input

Character attributes.

This is the buffer containing the character attribute values, concatenated together. The length of the buffer is given by the **CALEN** parameter.

The characters attributes must be specified in the same order as the CA* selectors in the *SELS* array. The length of each character attribute is fixed (see *SELS*). If the value to be set for an

attribute contains fewer nonblank characters than the defined length of the attribute, the value in *CHRATR* must be padded to the right with blanks to make the attribute value match the defined length of the attribute.

CMPCOD (10-digit signed integer) - output

Completion code.

It is one of the following:

CCOK
Successful completion.

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

Reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCFAIL:

RC2219
(2219, X'8AB') MQI call reentered before previous call complete.

RC2006
(2006, X'7D6') Length of character attributes not valid.

RC2007
(2007, X'7D7') Character attributes string not valid.

RC2009
(2009, X'7D9') Connection to queue manager lost.

RC2018
(2018, X'7E2') Connection handle not valid.

RC2019
(2019, X'7E3') Object handle not valid.

RC2020
(2020, X'7E4') Value for inhibit-get or inhibit-put queue attribute not valid.

RC2021
(2021, X'7E5') Count of integer attributes not valid.

RC2023
(2023, X'7E7') Integer attributes array not valid.

RC2040
(2040, X'7F8') Queue not open for set.

RC2041
(2041, X'7F9') Object definition changed since opened.

RC2101
(2101, X'835') Object damaged.

RC2052
(2052, X'804') Queue has been deleted.

RC2058
(2058, X'80A') Queue manager name not valid or not known.

RC2059
(2059, X'80B') Queue manager not available for connection.

RC2162
(2162, X'872') Queue manager shutting down.

RC2102
(2102, X'836') Insufficient system resources available.

RC2065
(2065, X'811') Count of selectors not valid.

RC2067
(2067, X'813') Attribute selector not valid.

RC2066
(2066, X'812') Count of selectors too large.

RC2071
(2071, X'817') Insufficient storage available.

RC2075
(2075, X'81B') Value for trigger-control attribute not valid.

RC2076
(2076, X'81C') Value for trigger-depth attribute not valid.

RC2077
(2077, X'81D') Value for trigger-message-priority attribute not valid.

RC2078
(2078, X'81E') Value for trigger-type attribute not valid.

RC2195
(2195, X'893') Unexpected error occurred.

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQSET(HCONN : HOBJ : SELCNT :
C                                     SELS(1) : IACNT : INTATR(1) :
C                                     CALEN : CHRATR : CMPCOD :
C                                     REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQSET            PR                  EXTPROC('MQSET')
D* Connection handle
D HCONN                           10I 0 VALUE
D* Object handle
D HOBJ                            10I 0 VALUE
D* Count of selectors
D SELCNT                          10I 0 VALUE
D* Array of attribute selectors
D SELS                            10I 0
D* Count of integer attributes
D IACNT                           10I 0 VALUE
D* Array of integer attributes
D INTATR                          10I 0
D* Length of character attributes buffer
D CALEN                           10I 0 VALUE
D* Character attributes
D CHRATR                            *   VALUE
```

```
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
```

MQSETMP (Set message handle property) on IBM i: IBM i

The MQSETMP call sets or modifies a property of a message handle.

Syntax

MQSETMP *(Hconn, Hmsg, SetPropOpts, Name, PropDesc, Type, ValueLength, Value, CompCode, Reason)*

Usage notes

- You can use this call only when the queue manager itself coordinates the unit of work. This can be:
  - A local unit of work, where the changes affect only IBM MQ resources.
  - A global unit of work, where the changes can affect resources belonging to other resource managers, as well as affecting IBM MQ resources.

  For further details about local and global units of work, see "MQBEGIN (Begin unit of work) on IBM i" on page 3267.
- In environments where the queue manager does not coordinate the unit of work, use the appropriate back-out call instead of MQBACK. The environment might also support an implicit back out caused by the application terminating abnormally.
  - On z/OS, use the following calls:
    - Batch programs (including IMS batch DL/I programs) can use the MQBACK call if the unit of work affects only IBM MQ resources. However, if the unit of work affects both IBM MQ resources and resources belonging to other resource managers (for example, Db2), use the SRRBACK call provided by the z/OS Recoverable Resource Service (RRS). The SRRBACK call backs out changes to resources belonging to the resource managers that have been enabled for RRS coordination.
    - CICS applications must use the `EXEC CICS SYNCPOINT ROLLBACK` command to back out the unit of work. Do not use the MQBACK call for CICS applications.
    - IMS applications (other than batch DL/I programs) must use IMS calls such as `ROLB` to back out the unit of work. Do not use the MQBACK call for IMS applications (other than batch DL/I programs).
  - On IBM i, use this call for local units of work coordinated by the queue manager. This means that a commitment definition must not exist at job level, that is, the `STRCMTCTL` command with the **`CMTSCOPE(*JOB)`** parameter must not have been issued for the job.
- If an application ends with uncommitted changes in a unit of work, the disposition of those changes depends on whether the application ends normally or abnormally. See the usage notes in "MQDISC (Disconnect queue manager) on IBM i" on page 3305 for further details.
- When an application puts or gets messages in groups or segments of logical messages, the queue manager retains information relating to the message group and logical message for the last successful MQPUT and MQGET calls. This information is associated with the queue handle, and includes such things as:
  - The values of the *GroupId*, *MsgSeqNumber*, *Offset*, and *MsgFlags* fields in MQMD.
  - Whether the message is part of a unit of work.

– For the MQPUT call: whether the message is persistent or nonpersistent.

The queue manager keeps three sets of group and segment information, one set for each of the following:

– The last successful MQPUT call (this can be part of a unit of work).
– The last successful MQGET call that removed a message from the queue (this can be part of a unit of work).
– The last successful MQGET call that browsed a message on the queue (this cannot be part of a unit of work).

If the application puts or gets the messages as part of a unit of work, and the application then decides to back out the unit of work, the group and segment information is restored to the value that it had previously:

– The information associated with the MQPUT call is restored to the value that it had before the first successful MQPUT call for that queue handle in the current unit of work.
– The information associated with the MQGET call is restored to the value that it had before the first successful MQGET call for that queue handle in the current unit of work.

Queues that were updated by the application after the unit of work started, but outside the scope of the unit of work, do not have their group and segment information restored if the unit of work is backed out.

Restoring the group and segment information to its previous value when a unit of work is backed out allows the application to spread a large message group or large logical message consisting of many segments across several units of work, and to restart at the correct point in the message group or logical message if one of the units of work fails.

Using several units of work might be advantageous if the local queue manager has only limited queue storage. However, the application must maintain sufficient information to be able to restart putting or getting messages at the correct point if a system failure occurs.

For details of how to restart at the correct point after a system failure, see the PMLOGO option described in PMOPT (10 digit signed integer), and the GMLOGO option described in GMOPT (10 digit signed integer).

The remaining usage notes apply only when the queue manager coordinates the units of work:

- A unit of work has the same scope as a connection handle. All IBM MQ calls that affect a particular unit of work must be performed using the same connection handle. Calls issued using a different connection handle (for example, calls issued by another application) affect a different unit of work. See HCONN (10-digit signed integer) - output for information about the scope of connection handles.
- Only messages that were put or retrieved as part of the current unit of work are affected by this call.
- A long-running application that issues MQGET, MQPUT, or MQPUT1 calls within a unit of work, but that never issues a commit or backout call, can fill queues with messages that are not available to other applications. To guard against this possibility, the administrator must set the **MaxUncommittedMsgs** queue manager attribute to a value that is low enough to prevent runaway applications filling the queues, but high enough to allow the expected messaging applications to work correctly.

Parameters

The MQSETMP call has the following parameters:

HCONN (10-digit signed integer) - input

This handle represents the connection to the queue manager.

The value must match the connection handle that was used to create the message handle specified in the **HMSG** parameter.

If the message handle was created using HCUNAS, a valid connection must be established on the thread setting a property of the message handle, otherwise the call fails with reason code `RC2009` .

HMSG (20-digit signed integer) - input

This is the message handle to be modified. The value was returned by a previous MQCRTMH call.

SETOPT (MQSMPO) - input

Control how message properties are set.

This structure allows applications to specify options that control how message properties are set. The structure is an input parameter on the MQSETMP call. See MQSMPO for further information.

PRNAME (MQCHARV) - input

This is the name of the property to set.

See Property names and Property name restrictions for further information about the use of property names.

PRPDSC (MQPD) - input/output

This structure is used to define the attributes of a property, including:

- what happens if the property is not supported
- what message context the property belongs to
- what messages the property is copied into as it flows

See MQPD for further information about this structure.

TYPE (10 digit signed integer) - input

The data type of the property being set. It can be one of the following:

TYPBOL
A boolean. *ValueLength* must be 4.

TYPBST
A byte string. *ValueLength* must be zero or greater.

TYPI8 An 8 bit signed integer. *ValueLength* must be 1.

TYPI16
A 16 bit signed integer. *ValueLength* must be 2.

TYPI32
A 32 bit signed integer. *ValueLength* must be 4.

TYPI64
A 64 bit signed integer. *ValueLength* must be 8.

TYPF32
A 32 bit floating-point number. *ValueLength* must be 4.

TYPF64
A 64 bit floating-point number. *ValueLength* must be 8.

TYPSTR
A character string. *ValueLength* must be zero or greater, or the special value VLNULL.

TYPNUL
The property exists but has a null value. *ValueLength* must be zero.

VALLEN (10-digit signed integer) - input

The length in bytes of the property value in the *Value* parameter.

Zero is valid only for null values or for strings or byte strings. Zero indicates that the property exists but that the value contains no characters or bytes.

The value must be greater than or equal to zero or the following special value if the *Type* parameter has TYPSTR set:

VLNULL
The value is delimited by the first null encountered in the string. The null is not included as part of the string. This value is invalid if TYPSTR is not also set.

Note: The null character used to terminate a string if VLNULL is set is a null from the character set of the Value.

VALUE (1-byte bit string x VALLEN) - input

The value of the property to be set. The buffer must be aligned on a boundary appropriate to the nature of the data in the value.

In the C programming language, the parameter is declared as a pointer-to-void; the address of any type of data can be specified as the parameter.

If *ValueLength* is zero, *Value* is not referred to. In this case, the parameter address passed by programs written in C or System/390 assembler can be null.

CMPCOD (10-digit signed integer) - output

The completion code; it is one of the following:

CCOK
Successful completion.

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

The reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCWARN:

RC2421
(2421, X'0975') An MQRFH2 folder containing properties could not be parsed.

If *CMPCOD* is CCFAIL:

RC2204
(2204, X'089C') Adapter not available.

RC2130
(2130, X'852') Unable to load adapter service module.

RC2157
(2157, X'86D') Primary and home ASIDs differ.

RC2004
(2004, X'07D4') Value parameter not valid.

RC2005
(2005, X'07D5') Value length parameter not valid.

RC2219
(2219, X'08AB') MQI call entered before previous call completed.

RC2460
(2460, X'099C') Message handle pointer not valid.

RC2499
(2499, X'09C3') Message handle already in use.

RC2046
(2046, X'07FE') Options not valid or not consistent.

RC2482
(2482, X'09B2') Property descriptor structure not valid.

RC2442
(2442, X'098A') Invalid property name.

RC2473
(2473, X'09A9') Invalid property data type.

RC2472
(2472, X'09A8') Number format error encountered in value data.

RC2463
(2463, X'099F') Set message property options structure not valid.

RC2111
(2111, X'083F') Property name coded character set identifier not valid.

RC2071
(2071, X'817') Insufficient storage available.

RC2195
(2195, X'893') Unexpected error occurred.

See "Return codes for IBM i (ILE RPG)" on page 3452 for more details.

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQSETMP(HCONN : HMSG : SETOPT :
                                        PRNAME : PRPDSC :
                                        TYPE : VALLEN : VALUE :
                                        CMPCOD : REASON)
```

The prototype definition for the call is:

```
DMQSETMP          PR                  EXTPROC('MQSETMP')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Message handle
D HMSG                          10I 0 VALUE
D* Options that control the action of MQSETMP
D SETOPT                        20A
D* Property name
D PRNAME                        32A
D* Property descriptor
D PRPDSC                        24A
D* Property data type
D TYPE                          10I 0 VALUE
D* Length of the Value area
D VALLEN                        10I 0 VALUE
D* Property value
D VALUE                           *   VALUE
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CompCode
D REASON                        10I 0
```

MQSTAT (Retrieve status information) on IBM i: IBM i

Use the MQSTAT call to retrieve status information. The type of status information returned is determined by the STYPE value specified on the call.

- "Syntax"
- "Usage notes"
- "Parameters"
- "RPG Declaration" on page 3376

Syntax

MQSTAT *(HCONN, STYPE, STAT, CMPCOD, REASON)*

Usage notes

1. A call to MQSTAT specifying a type of STATAPT returns information about previous asynchronous MQPUT and MQPUT1 operations. The MQSTAT structure passed on the call is completed with the first recorded asynchronous warning or error information for that connection. If further errors or warnings follow the first, they do not normally alter these values. However, if an error occurs with a completion code of CCWARN, a subsequent failure with a completion code of CCFAIL is returned instead.
2. If no errors have occurred since the connection was established or since the last call to MQSTAT then a CMPCOD of CCOK and REASON of RCNONE are returned.
3. Counts of the number of asynchronous calls that have been processed under the connection handle are returned by using three counters; STSPSC, STSPWC, and STSPFC. These counters are incremented by the queue manager each time an asynchronous operation is processed successfully, has a warning, or fails (note that for accounting purposes a put to a distribution list counts once per destination queue rather than once per distribution list).
4. A successful call to MQSTAT results in any previous error information or counts being reset.

Parameters

The MQSTAT call has the following parameters:

Hconn (MQHCONN) - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

STYPE (10-digit signed integer) - input

Type of status information being requested. The only valid value is:

STATAPT
Return information about previous asynchronous put operations.

STS (MQSTS) - input/output

Status information structure. See "MQSTS (Status reporting structure) on IBM i" on page 3243 for details.

CMPCOD (10-digit signed integer) - output

The completion code; it is one of the following:

CCOK
Successful completion.

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

The reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCFAIL:

RC2374
(2374, X'946') API exit failed

RC2183
(2183, X'887') Unable to load API exit.

RC2219
(2219, X'8AB') MQI call entered before previous call complete.

RC2009
(2009, X'7D9') Connection to queue manager lost.

RC2203
(2203, X'89B') Connection shutting down.

RC2018
(2018, X'7E2') Connection handle not valid.

RC2162
(2162, X'872') Queue manager stopping

RC2102
(2102, X'836') Insufficient system resources available.

RC2430
(2430, X'97E') Error with MQSTAT type.

RC2071
(2071, X'817') Insufficient storage available.

RC2424
(2424, X'978') Error with MQSTS structure

RC2195
(2195, X'893') Unexpected error occurred.

RC2298
(2298, X'8FA') The function requested is not available in the current environment.

For detailed information about these codes, see:
- Reason codes

RPG Declaration

```
C*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
C                     CALLP     MQSTAT(HCONN : ETYPE : ERR :
C                                      CMPCOD : REASON)
```

The prototype definition for the call is:

```
D.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
DMQSTAT           PR                  EXTPROC('MQSTAT')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Status information type
D STYPE                         10I 0 VALUE
D* Status information
```

```
D STATUS                        296A
D* Completion code
D CMPCOD                         10I 0
D* Reason code qualifying CompCode
D REASON                         10I 0
```

MQSUB (Register Subscription) on IBM i: IBM i

The MQSUB call registers the applications subscription to a particular topic.

Syntax

MQSUB *(HCONN, SUBDSC, HOBJ, HSUB, CMPCOD, REASON)*

Usage notes

- The subscription is made to a topic, named either using the short name of a pre-defined topic object, the full name of the topic string, or it is formed by the concatenation of two parts, as described in "Using topic strings" on page 2566.
- The queue manager performs security checks when an MQSUB call is issued, to verify that the user identifier under which the application is running has the appropriate level of authority before access is permitted. The appropriate topic object is located either by a short name being provided in the call, or the nearest short name object in the topic hierarchy being found if a long name is provided. An authority check is made on this topic object to ensure authority to subscribe is set and on the destination queue to ensure that authority for output is set. If the SDMAN option is used, this means that an authority check is made on the managed queue name associated with this topic object, and if a non-managed queue is provided, this means that an authority check is made on the queue represented by the **HOBJ** parameter.
- The *HOBJ* returned on the MQSUB call when the SOMAN option is used, can be inquired in order to find out attributes such as the Backout threshold and the Excessive backout requeue name. You can also inquire the name of the managed queue, but you should not attempt to directly open this queue.
- Subscriptions can be grouped allowing only a single publication to be delivered to the group of subscriptions even where more than one of the group matched the publication. Subscriptions are grouped using the SOGRP option and in order to group subscriptions they must:
  - be using the same named queue (that is not using the SOMAN option) on the same queue manager - represented by the **HOBJ** parameter on the MQSUB call
  - share the same *SDCID*
  - be of the same *SDSL*

  These attributes define the set of subscriptions considered to be in the group, and are also the attributes that cannot be altered if a subscription is grouped. Alteration of *SDSL* results in RC2512, and alteration of any of the others (which can be changed if a subscription is not grouped) results in RC2515.
- Fields in the MQSD are completed on return from an MQSUB call which uses the SORES option. The MQSD returned can be passed directly into an MQSUB call which uses the SOALT option with any changes you need to make to the subscription applied to the MQSD. Some fields have special considerations as noted in the table.

MQSD output from MQSUB

Field name in MQSD	Special considerations
Access or creation options	None of these options are set on return from the MQSUB call. If you later reuse the MQSD in an MQSUB call the option you require must be explicitly set.
Durability options, Destination options, Registration Options & Wildcard options	These options will be set as appropriate
Publication options	These options will be set as appropriate, except for SONEWP which is only applicable to SOCRE.
Other options	These options are unchanged on return from an MQSUB call. They control how the API call is issued and are not stored with the subscription. They must be set as required on any subsequent MQSUB call reusing the MQSD.
ObjectName	This input only field is unchanged on return from an MQSUB call.
ObjectString	This input only field is unchanged on return from an MQSUB call. The Full topic name used is returned in the *SDRO* field, if a buffer is provided.
AlternateUserId and AlternateSecurityId	These input only fields are unchanged on return from an MQSUB call. They control how the API call is issued and are not stored with the subscription. They must set as required on any subsequent MQSUB call reusing the MQSD.
SubExpiry	On return from an MQSUB call using the SORES option this field will be set to the original expiry of the subscription and not the remaining expiry time. If you then reuse the MQSD in an MQSUB call using the SOALT option you will reset the expiry of the subscription to start counting down again.
SubName	This field is an input field on an MQSUB call and is not changed on output.
SubUserData and SelectionString	These variable length fields will be returned on output from an MQSUB call using the SORES option, if a buffer is provided, and also a positive buffer length in *VCHRP*. If no buffer is provided only the length will be returned in the *VCHRL* field of the MQCHARV.If the buffer provided is smaller than the space required to return the field, only *VCHRP* bytes are returned in the provided buffer. If you later reuse the MQSD in an MQSUB call using the SOALT option and a buffer is not provided but a non-zero *VCHRL* is provided, if that length matches the existing length of the field, no alteration will made to the field.
SubCorrelId and PubAccountingToken	If you do not use SOSCID, then the *SDCID* will be generated by the queue manager. If you do not use SOSETI, then the *SDACC* will be generated by the queue manager. These fields will be returned in the MQSD from an MQSUB call using the SORES option. If they are generated by the queue manager, the generated value will be returned on an MQSUB call using the SOCRE or SOALT option.

MQSD output from MQSUB

Field name in MQSD	Special considerations
PubPriority, SubLevel & PubApplIdentityData	These fields will be returned in the MQSD.
ResObjectString	This output only field will be returned in the MQSD if a buffer is provided.

Parameters

The MQSUB call has the following parameters:

HCONN (10-digit signed integer) - input

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN or MQCONNX call.

SUBDSC (MQSD) - input/output

This is a structure that identifies the object with use that is being registered by the application. See "MQSD (Subscription descriptor) on IBM i" on page 3224 for more information.

HOBJ (10-digit signed integer) - input/output

This handle represents the access that has been established to obtain the messages sent to this subscription. These messages can either be stored on a specific queue or the queue manager can be asked to manage their storage without the need for a specific queue.

Object handle.

If a specific queue is to be used it must be associated with the subscription at creation time. This can be done in two ways:

- By providing this handle when calling MQSUB with the SDCRT option. If this handle is provided as an input parameter on the call, it must be a valid object handle returned from a previous MQOPEN call of a queue using at least one of OOINP*, OOOUT (if a remote queue for example), or OOBRW option. If this is not the case, the call fails with RC2019. It cannot be an object handle to an alias queue which resolves to a topic object. If so, the call fails with RC2019
- By using the DEFINE SUB MQSC command and provided that command with the name of a queue object.

If the queue manager is to manage the storage of messages sent to this subscription, you should indicate this when the subscription is created, by using the SOMAN option and setting the parameter value to HONONE. The queue manager returns the handle as an output parameter on the call, and the handle that is returned is known as a managed handle. If HONONE is specified and SOMAN is not also specified, the call fails with RC2019.

A managed handle that is returned by the queue manager can be used on an MQGET or MQCB call, with or without browse options, on an MQINQ call, or on MQCLOSE. It cannot be used on MQPUT, MQSET, or on a subsequent MQSUB; attempting to do so fails with RC2039 for MQPUT, RC2040 for MQSET, or RC2038 for MQSUB.

If the SORES option in the *OPTS* field in the MQSD structure is used to resume this subscription, the handle can be returned to the application in this parameter if HONONE is specified. You can use this whether the subscription is using a managed handle or not. It can be useful for subscriptions created using DEFINE SUB if you want the handle to the subscription queue defined on the DEFINE SUB command. In the case where an administratively created subscription is being resumed, the queue is opened with OOINPQ and OOBRW. If other options are needed, the application must open the subscription queue explicitly and provide the object handle on the call. If there is a problem opening the queue the call will fail with RC2522. If the *HOBJ* is provided, it must be equivalent to the *HOBJ* in the original MQSUB call. This means if an

object handle returned from an MQOPEN call is being provided, the handle must be to the same queue as previously used or the call fails with RC2019.

If this subscription is being altered, by using the SOALT option in the *OPTS* field in the MQSD structure, then a different *HOBJ* can be provided. Any publications that have been delivered to the queue previously identified through this parameter remain on that queue and it is the responsibility of the application to retrieve those messages if the **HOBJ** parameter now represents a different queue.

The use of this parameter with various subscription options is summarized in the following table:

Options	Hobj	Description
SOCRT + SOMAN	Ignored on input	Creates a subscription with queue manager managed storage of messages.
SOCRT	Valid object handle	Creates a subscription providing a specific queue as the destination for messages.
SORES	HONONE	Resumes a previously created subscription (managed or not) and have the queue manager return the object handle for use by the application.
SORES	Valid, matching, object handle	Resumes a previously created subscription which uses a specific queue as the destination for messages and use an object handle with specific open options.
SOALT + SOMAN	HONONE	Alters an existing subscription which was previously using a specific queue, to now be managed.
SOALT	Valid object handle	Alters an existing subscription to use a specific queue (either from managed, or from a different specific queue).

Whether it was provided or returned, *HOBJ* must be specified on subsequent MQGET calls that you need to receive the publications.

The *HOBJ* handle ceases to be valid when the MQCLOSE call is issued on it, or when the unit of processing that defines the scope of the handle terminates. The scope of the object handle returned is the same as that of the connection handle specified on the call. See HCONN for information about handle scope. An MQCLOSE of the *HOBJ* handle has no effect on the *HSUB* handle.

HSUB (10-digit signed integer) - output

This handle represents the subscription that has been made. It can be used for two further operations:

- It can be used on a subsequent MQSUBRQ call to request that publications be sent when the SOPUBR option has been used when making the subscription.
- It can be used on a subsequent MQCLOSE call to remove the subscription that has been made. The *HSUB* handle ceases to be valid when the MQCLOSE call is issued, or when the unit of processing that defines the scope of the handle terminates. The scope of the object handle returned is the same as that of the connection handle specified on the call. An MQCLOSE of the *HSUB* handle has no effect on the *HOBJ* handle.

This handle cannot be passed to an MQGET or MQCB call. You must use the **HOBJ** parameter. Passing this handle to any other IBM MQ call results in RC2019.

CMPCOD (10-digit signed integer) - output

The completion code; it is one of the following:

CCOK
Successful completion

CCWARN
Warning (partial completion)

CCFAIL
Call failed

REASON (10-digit signed integer) - output

The reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCFAIL:

RC2019
(2019 X'07E3') Object handle not valid

RC2046
(2046 X'07FE') Options not valid or not consistent

RC2085
(2085 X'0825') Object identified cannot be found

RC2161
(2161 X'0871') Queue manager quiescing

RC2298
(2298 X'08FA') Function not supported.

RC2424
(2424 X'0978') Subscription descriptor (MQSD) not valid

RC2425
(2441 X'979') Topic string not valid

RC2428
(2428 X'097C') Subscription name specified does not match existing subscriptions

RC2429
(2429 X'097D') Subscription name exists and is in use by another application

RC2431
(2431 X'097F') SubUserData field not valid

RC2432
(2432 X'0980') Subscription exists

RC2434
(2434 X'0982') Subscription name matches existing subscription

RC2440
(2440 X'0988') SubName field not valid

RC2441
(2441 X'0989') Objectstring field not valid

RC2435
(2435 X'0983') Attribute cannot be changed using SDALT, or subscription was created with SDIMM.

RC2436
(2436 X'0984') SODUR option not valid

RC2459
(2459, X'99B') Selection string syntax error.

RC2503
(2503 X'09C7') MQSUB calls are currently inhibited for the topics subscribed to.

RC2519
(2519, X'9D7') The selection string is not as specified in the description of how to use an MQCHARV structure.

RC2551
(2551, X'9F7') Specified selection string is not available.

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQSUB(HCONN : SUBDSC : HOBJ :
C                                     HSUB : CMPCOD : REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQSUB            PR                  EXTPROC('MQSUB')
D* Connection handle
D HCONN                          10I 0 VALUE
D* Subscription descriptor
D SUBDSC                        400A
D* Object handle for queue
D HOBJ                           10I 0
D* Subscription object handle
D HSUB                           10I 0
D* Completion code
D CMPCOD                         10I 0
D* Reason code qualifying CompCode
D REASON                         10I 0
```

MQSUBRQ (Subscription Request) on IBM i: 

The MQSUBRQ call makes a request on a subscription.

- "Syntax"
- "Usage notes"
- "Parameters" on page 3383
- "RPG Declaration" on page 3384

Syntax

MQSUBRQ *(HCONN, HSUB, ACTION, SUBROPT, CMPCOD, REASON)*

Usage notes

The following usage notes apply to the use of SRAPUB:

1. If this verb completes successfully, the retained publications matching the subscription specified have been sent to the subscription and can be received by using MQGET or MQCB using the HOBJ returned on the original MQSUB verb that created the subscription.

2. If the topic subscribed to by the original MQSUB verb that created the subscription contained a wildcard, more than one retained publication might be sent. The number of publications sent as a result of this call is recorded in the *SRNMP* field in the SBROPT structure.
3. If this verb completes with a reason code of RC2437 then there were no currently retained publications for the topic specified.
4. If this verb completes with a reason code of RC2525 or RC2526 then there are currently retained publications for the topic specified but an error has occurred that that meant they were unable to be delivered.
5. The application must have a current subscription to the topic before it can make this call. If the subscription was made in a previous instance of the application and a valid handle to the subscription is not available, the application must first call MQSUB with the SORES option to obtain a handle to it for use in this call.
6. The publications are sent to the destination that is registered for use with the current subscription of this application. If the publications should be sent somewhere else, the subscription must first be altered using the MQSUB call with the SOALT option.

Parameters

The MQSUBRQ call has the following parameters:

HCONN (10-digit signed integer) - input

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications the MQCONN call can be omitted, and the following value specified for *HCONN*:

HCDEFH
Default connection handle.

HSUB (10-digit signed integer) - input

This handle represents the subscription for which an update is to be requested. The value of *HSUB* was returned from a previous MQSUB call.

ACTION (10-digit signed integer) - input

This parameter controls the particular action that is being requested on the subscription. One (and only one) of the following must be specified:

SRAPUB

This action requests that an update publication be sent for the specified topic. This is normally used if the subscriber specified the option SOPUBR on the MQSUB call when it made the subscription. If the queue manager has a retained publication for the topic, this is sent to the subscriber. If not, the call fails. If an application is sent a publication which was retained, this is indicated by the MQIsRetained message property of that publication.

Since the topic in the existing subscription represented by the **HSUB** parameter can contain wildcards, the subscriber might receive multiple retained publications.

SBROPT (MQSRO) - input/output

These options control the action of MQSUBRQ, see "MQSRO - Subscription request options" on page 2575 for details.

CMPCOD (10-digit signed integer) - output

The completion code; it is one of the following:

CCOK
Successful completion

CCWARN
Warning (partial completion)

CCFAIL
Call failed

Reason (10-digit signed integer) - output

The reason code qualifying *CMPCOD*.

If *CPMCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CPMCOD* is CCFAIL:

RC2298
2298 (X'08FA') The function requested is not available in the current environment.

RC2437
2437 (X'0985') There are no retained publications currently stored for this topic.

RC2046
2046 (X'07FE') Options parameter or field contains options that are not valid, or a combination of options that is not valid.

RC2161
2161 (X'0871') Queue manager quiescing

RC2438
2438 (X'0986') On the MQSUBRQ call, the Subscription Request Options MQSRO is not valid.

RPG Declaration

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQSUBRQ(HCONN : HSUB : ACTION :
C                                       SBROPT : CMPCOD : REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQSUBRQ          PR                  EXTPROC('MQSUBRQ')
D* Connection handle
D HCONN                           10I 0 VALUE
D* Subscription handle
D HSUB                            10I 0 VALUE
D* Action requested on the subscription
D ACTION                          10I 0 VALUE
D* Subscription Request Options
D SBROPT                          16A
D* Completion code
D CMPCOD                          10I 0
D* Reason code qualifying CompCode
D REASON                          10I 0
```

Attributes of objects on IBM i

IBM i

This collection of topics lists only those IBM MQ objects that can be the subject of an MQINQ function call, and gives details of the attributes that can be inquired on and the selectors to be used.

Attributes for queues:

Use this information to learn about the different types of queue definitions and the attributes supported by each.

Types of queue: The queue manager supports the following types of queue definition:

Local queue
This is a physical queue that stores messages. The queue exists on the local queue manager.

Applications connected to the local queue manager can place messages on and remove messages from queues of this type. The value of the **QType** queue attribute is QTLOC.

Shared queue
This is a physical queue that stores messages. The queue exists in a shared repository that is accessible to all of the queue managers that belong to the queue-sharing group that owns the shared repository.

Applications connected to any queue manager in the queue-sharing group can place messages on and remove messages from queues of this type. Such queues are effectively the same as local queues. The value of the **QType** queue attribute is QTLOC.

- Shared queues are supported only on z/OS.

Cluster queue
This is a physical queue that stores messages. The queue exists either on the local queue manager, or on one or more of the queue managers that belong to the same cluster as the local queue manager.

Applications connected to the local queue manager can place messages on queues of this type, regardless of the location of the queue. If an instance of the queue exists on the local queue manager, the queue behaves in the same way as a local queue, and applications connected to the local queue manager can remove messages from the queue. The value of the **QType** queue attribute is QTCLUS.

Alias queue
This is not a physical queue - it is an alternative name for a local queue. The name of the local queue to which the alias resolves is part of the definition of the alias queue.

Applications connected to the local queue manager can place messages on and remove messages from alias queues - the messages are placed on and removed from the local queue to which the alias resolves. The value of the **QType** queue attribute is QTALS.

Remote queue
This is not a physical queue - it is the local definition of a queue that exists on a remote queue manager. The local definition of the remote queue contains information that tells the local queue manager how to route messages to the remote queue manager.

Applications connected to the local queue manager can place messages on remote queues - the messages are placed on the local transmission queue used to route messages to the remote queue manager. Applications cannot remove messages from remote queues. The value of the **QType** queue attribute is QTREM.

A remote queue definition can also be used for:

- Reply-queue aliasing

In this case the name of the definition is the name of a reply-to queue. For more information, see Reply-to queue alias definitions.

- Queue manager aliasing

  In this case the name of the definition is an alias for a queue manager, and not the name of a queue. For more information, see Queue manager alias definitions.

Model queue

This is not a physical queue - it is a set of queue attributes from which a local queue can be created.

Messages cannot be stored on queues of this type.

Some queue attributes apply to all types of queue; other queue attributes apply only to certain types of queue. The types of queue to which an attribute applies are indicated by the ✓ symbol in Table 369 and subsequent tables.

Table 369 summarizes the attributes that are specific to queues. The attributes are described in alphabetical order.

Note: The names of the attributes shown in this section are the names used with the MQINQ and MQSET calls. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see Script (MQSC) Commands for details.

Table 369. Attributes for queues. The columns apply as follows:

- The column for local queues applies also to shared queues.
- The column for model queues indicates which attributes are inherited by the local queue created from the model queue.
- The column for cluster queues indicates the attributes that can be inquired when the cluster queue is opened for inquire alone, or for inquire and output. If the cluster queue is opened for inquire plus one or more of input, browse, or set, the column for local queues applies instead.

Attribute	Description	Local	Model	Alias	Remote	Cluster
AlterationDate	Date when definition was last changed	✓		✓	✓	
AlterationTime	Time when definition was last changed	✓		✓	✓	
BackoutRequeueQName	Excessive backout requeue queue name	✓	✓			
BackoutThreshold	Backout threshold	✓	✓			
BaseQName	Queue name to which alias resolves			✓		
ClusterChannelName	Cluster-sender channel name	✓	✓			
ClusterName	Name of cluster to which queue belongs	✓		✓	✓	
ClusterNamelist	Name of namelist object containing names of clusters to which queue belongs	✓		✓	✓	
CreationDate	Date the queue was created	✓				

Table 369. Attributes for queues (continued). The columns apply as follows:

- The column for local queues applies also to shared queues.
- The column for model queues indicates which attributes are inherited by the local queue created from the model queue.
- The column for cluster queues indicates the attributes that can be inquired when the cluster queue is opened for inquire alone, or for inquire and output. If the cluster queue is opened for inquire plus one or more of input, browse, or set, the column for local queues applies instead.

Attribute	Description	Local	Model	Alias	Remote	Cluster
CreationTime	Time the queue was created	✓				
CurrentQDepth	Current queue depth	✓				
DefBind	Default binding	✓		✓	✓	✓
DefinitionType	Queue definition type	✓	✓			
DefInputOpenOption	Default input open option	✓	✓			
DefPersistence	Default message persistence	✓	✓	✓	✓	✓
DefPriority	Default message priority	✓	✓	✓	✓	✓
DistLists	Distribution list support	✓	✓			
HardenGetBackout	Whether to maintain an accurate backout count	✓	✓			
InhibitGet	Controls whether get operations for the queue are allowed	✓	✓	✓		
InhibitPut	Controls whether put operations for the queue are allowed	✓	✓	✓	✓	✓
InitiationQName	Name of initiation queue	✓	✓			
MaxMsgLength	Maximum message length in bytes	✓	✓			
MaxQDepth	Maximum queue depth	✓	✓			
MediaLog	Identity of oldest log extent (or oldest journal receiver on IBM i) needed for media recovery of a specified queue	✓	✓			
MsgDeliverySequence	Message delivery sequence	✓	✓			
OpenInputCount	Number of opens for input	✓				

Table 369. Attributes for queues (continued). The columns apply as follows:

- The column for local queues applies also to shared queues.
- The column for model queues indicates which attributes are inherited by the local queue created from the model queue.
- The column for cluster queues indicates the attributes that can be inquired when the cluster queue is opened for inquire alone, or for inquire and output. If the cluster queue is opened for inquire plus one or more of input, browse, or set, the column for local queues applies instead.

Attribute	Description	Local	Model	Alias	Remote	Cluster
OpenOutputCount	Number of opens for output	✓				
ProcessName	Process name	✓	✓			
QDepthHighEvent	Controls whether Queue Depth High events are generated	✓	✓			
QDepthHighLimit	High limit for queue depth	✓	✓			
QDepthLowEvent	Controls whether Queue Depth Low events are generated	✓	✓			
QDepthLowLimit	Low limit for queue depth	✓	✓			
QDepthMaxEvent	Controls whether Queue Full events are generated	✓	✓			
QDesc	Queue description	✓	✓	✓	✓	✓
QName	Queue name	✓		✓	✓	✓
QServiceInterval	Target for queue service interval	✓	✓			
QServiceIntervalEvent	Controls whether Service Interval High or Service Interval OK events are generated	✓	✓			
QType	Queue type	✓		✓	✓	✓
RemoteQMgrName	Name of remote queue manager				✓	
RemoteQName	Name of remote queue				✓	
RetentionInterval	Retention interval	✓	✓			
Scope	Controls whether an entry for the queue also exists in a cell directory	✓		✓	✓	
Shareability	Queue shareability	✓	✓			

Table 369. Attributes for queues (continued). The columns apply as follows:

- The column for local queues applies also to shared queues.
- The column for model queues indicates which attributes are inherited by the local queue created from the model queue.
- The column for cluster queues indicates the attributes that can be inquired when the cluster queue is opened for inquire alone, or for inquire and output. If the cluster queue is opened for inquire plus one or more of input, browse, or set, the column for local queues applies instead.

Attribute	Description	Local	Model	Alias	Remote	Cluster
TriggerControl	Trigger control	✓	✓			
TriggerData	Trigger data	✓	✓			
TriggerDepth	Trigger depth	✓	✓			
TriggerMsgPriority	Threshold message priority for triggers	✓	✓			
TriggerType	Trigger type	✓	✓			
Usage	Queue usage	✓	✓			
XmitQName	Transmission queue name				✓	

AlterationDate (12-byte character string) on IBM i: IBM i

Date when definition was last changed.

Local	Model	Alias	Remote	Cluster
✓		✓	✓	

This is the date when the definition was last changed. The format of the date is `YYYY-MM-DD`, padded with two trailing blanks to make the length 12 bytes (for example, `1992-09-23¬¬`, where ¬¬ represents two blank characters).

The values of certain attributes (for example, *CurrentQDepth*) change as the queue manager operates. Changes to these attributes do not affect *AlterationDate*.

To determine the value of this attribute, use the CAALTD selector with the MQINQ call. The length of this attribute is given by LNDATE.

AlterationTime (8-byte character string) on IBM i: IBM i

Time when definition was last changed.

Local	Model	Alias	Remote	Cluster
✓		✓	✓	

This is the time when the definition was last changed. The format of the time is `HH.MM.SS` using the 24-hour clock, with a leading zero if the hour is less than 10 (for example `09.10.20`). The time is local time.

The values of certain attributes (for example, *CurrentQDepth*) change as the queue manager operates. Changes to these attributes do not affect *AlterationTime*.

To determine the value of this attribute, use the CAALTT selector with the MQINQ call. The length of this attribute is given by LNTIME.

BackoutRequeueQName (48-byte character string) on IBM i: IBM i

Excessive backout requeue queue name.

Local	Model	Alias	Remote	Cluster
✓	✓			

Applications running inside WebSphere Application Server and those that use the IBM MQ Application Server Facilities use this attribute to determine where messages that have been backed out should go. For all other applications, apart from allowing its value to be queried, the queue manager takes no action based on the value of the attribute.

To determine the value of this attribute, use the CABRQN selector with the MQINQ call. The length of this attribute is given by LNQN.

BackoutThreshold (10-digit signed integer) on IBM i: IBM i

Backout threshold.

Local	Model	Alias	Remote	Cluster
✓	✓			

Applications running inside WebSphere Application Server and those that use the IBM MQ Application Server Facilities use this attribute to determine if a message should be backed out. For all other applications, apart from allowing its value to be queried, the queue manager takes no action based on the value of the attribute.

To determine the value of this attribute, use the IABTHR selector with the MQINQ call.

BaseQName (48-byte character string) on IBM i: IBM i

The queue name to which the alias resolves.

Local	Model	Alias	Remote	Cluster
		✓		

This is the name of a queue that is defined to the local queue manager. (For more information about queue names, see the description of the *ODON* field in MQOD. The queue is one of the following types:

QTLOC
Local queue.

QTREM
Local definition of a remote queue.

QTCLUS
Cluster queue.

To determine the value of this attribute, use the CABASQ selector with the MQINQ call. The length of this attribute is given by LNQN.

BaseType (integer parameter structure) on IBM i: IBM i

The type of object to which the alias resolves.

Local	Model	Alias	Remote	Cluster
		✓		

This attribute can have one of the following values:

OTQ Base object type is a queue

OTTOP
Base object type is a topic

CFStrucName (12-byte character string) on IBM i: IBM i

Coupling-facility structure name.

Local	Model	Alias	Remote	Cluster
✓	✓			

This is the name of the coupling-facility structure where the messages on the queue are stored. The first character of the name is in the range A through Z, and the remaining characters are in the range A through Z, 0 through 9, or blank.

The full name of the structure in the coupling facility is obtained by suffixing the value of the **QSGName** queue manager attribute with the value of the **CFStrucName** queue attribute.

This attribute applies only to shared queues; it is ignored if *QSGDisp* does not have the value QSGDSH.

To determine the value of this attribute, use the CACFSN selector with the MQINQ call. The length of this attribute is given by LNCFSN.

z/OS This attribute is supported only on z/OS.

ClusterChannelName (20-byte character string):

`ClusterChannelName` is the generic name of the cluster-sender channels that use this queue as a transmission queue. The attribute specifies which cluster-sender channels send messages to a cluster-receiver channel from this cluster transmission queue.

Local	Model	Alias	Remote	Cluster
✓	✓			

The default queue manager configuration is for all cluster-sender channels to send messages from a single transmission queue, `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. The default configuration can be changed by modified by changing the queue manager attribute, `DefClusterXmitQueueType`. The default value of the attribute is `SCTQ`. You can change the value to `CHANNEL`. If you set the `DefClusterXmitQueueType` attribute to `CHANNEL`, each cluster-sender channel defaults to using a specific cluster transmission queue, `SYSTEM.CLUSTER.TRANSMIT.`*`ChannelName`*.

You can also set the transmission queue attribute `ClusterChannelName` attribute to a cluster-sender channel manually. Messages that are destined for the queue manager connected by the cluster-sender channel are stored in the transmission queue that identifies the cluster-sender channel. They are not stored in the default cluster transmission queue. If you set the `ClusterChannelName` attribute to blanks, the channel switches to the default cluster transmission queue when the channel restarts. The default queue is either `SYSTEM.CLUSTER.TRANSMIT.`*`ChannelName`* or `SYSTEM.CLUSTER.TRANSMIT.QUEUE`, depending on the value of the queue manager `DefClusterXmitQueueType` attribute.

By specifying asterisks, "`*`", in `ClusterChannelName`, you can associate a transmission queue with a set of cluster-sender channels. The asterisks can be at the beginning, end, or any number of places in the middle of the channel name string. `ClusterChannelName` is limited to a length of 20 characters: `MQ_CHANNEL_NAME_LENGTH`.

ClusterName (48-byte character string) on IBM i: IBM i

Name of cluster to which queue belongs.

Local	Model	Alias	Remote	Cluster
✓		✓	✓	

This is the name of the cluster to which the queue belongs. If the queue belongs to more than one cluster, *`ClusterNamelist`* specifies the name of a namelist object that identifies the clusters, and *`ClusterName`* is blank. At least one of *`ClusterName`* and *`ClusterNamelist`* must be blank.

To determine the value of this attribute, use the CACLN selector with the MQINQ call. The length of this attribute is given by LNCLUN.

ClusterNamelist (48-byte character string) on IBM i: IBM i

Name of namelist object containing names of clusters to which queue belongs.

Local	Model	Alias	Remote	Cluster
✓		✓	✓	

This is the name of a namelist object that contains the names of clusters to which this queue belongs. If the queue belongs to only one cluster, the namelist object contains only one name. Alternatively, *ClusterName* can be used to specify the name of the cluster, in which case *ClusterNamelist* is blank. At least one of *ClusterName* and *ClusterNamelist* must be blank.

To determine the value of this attribute, use the CACLNL selector with the MQINQ call. The length of this attribute is given by LNNLN.

CreationDate (12-byte character string) on IBM i: IBM i

Date when queue was created.

Local	Model	Alias	Remote	Cluster
✓				

This is the date when the queue was created. The format of the date is `YYYY-MM-DD`, padded with two trailing blanks to make the length 12 bytes (for example, `1992-09-23¬¬`, where ¬¬ represents two blank characters).

- On IBM i, the creation date of a queue might differ from that of the underlying operating system entity (file or userspace) that represents the queue.

To determine the value of this attribute, use the CACRTD selector with the MQINQ call. The length of this attribute is given by LNCRTD.

CreationTime (8-byte character string) on IBM i: IBM i

Time when queue was created.

Local	Model	Alias	Remote	Cluster
✓				

This is the time when the queue was created. The format of the time is `HH.MM.SS` using the 24-hour clock, with a leading zero if the hour is less than 10 (for example `09.10.20`). The time is local time.

- On IBM i, the creation time of a queue might differ from that of the underlying operating system entity (file or user space) that represents the queue.

To determine the value of this attribute, use the CACRTT selector with the MQINQ call. The length of this attribute is given by LNCRTT.

CurrentQDepth (10-digit signed integer) on IBM i: IBM i

Current queue depth.

Local	Model	Alias	Remote	Cluster
✓				

This is the number of messages currently on the queue. It is incremented during an MQPUT call, and during backout of an MQGET call. It is decremented during a nonbrowse MQGET call, and during backout of an MQPUT call. The effect of this is that the count includes messages that have been put on the queue within a unit of work, but which have not yet been committed, even though they are not eligible to be retrieved by the MQGET call. Similarly, it excludes messages that have been retrieved within a unit of work using the MQGET call, but which have yet to be committed.

The count also includes messages which have passed their expiry time but have not yet been discarded, although these messages are not eligible to be retrieved. See the *MDEXP* field described in "MQMD (Message descriptor) on IBM i" on page 3118.

Unit-of-work processing and the segmentation of messages can both cause *CurrentQDepth* to exceed *MaxQDepth*. However, this does not affect the retrievability of the messages - *all* messages on the queue can be retrieved using the MQGET call in the normal way.

The value of this attribute fluctuates as the queue manager operates.

To determine the value of this attribute, use the IACDEP selector with the MQINQ call.

DefBind (10-digit signed integer) on IBM i: IBM i

Default binding.

Local	Model	Alias	Remote	Cluster
✓		✓	✓	✓

This attribute is the default binding that is used when OOBNDQ is specified on the MQOPEN call and the queue is a cluster queue. DefBind can have one of the following values:

BNDOPN
Binding fixed by MQOPEN call.

BNDNOT
Binding not fixed.

BNDGRP
Binding is not fixed by the MQOPEN call, but is fixed on MQPUT for all messages in a logical group.

To determine the value of this attribute, use the IADBND selector with the MQINQ call.

DefinitionType (10-digit signed integer) on IBM i: IBM i

Queue definition type.

Local	Model	Alias	Remote	Cluster
✓	✓			

This indicates how the queue was defined. The value is one of the following:

QDPRE
Predefined permanent queue.

The queue is a permanent queue created by the system administrator; only the system administrator can delete it.

Predefined queues are created using the `DEFINE` MQSC command, and can be deleted only by using the `DELETE` MQSC command. Predefined queues cannot be created from model queues.

Commands can be issued either by an operator, or by an authorized user sending a command message to the command input queue (see the **CommandInputQName** attribute described in "Attributes for the queue manager on IBM i" on page 3420).

QDPERM
Dynamically defined permanent queue.

The queue is a permanent queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value QDPERM for the **DefinitionType** attribute.

This type of queue can be deleted using the MQCLOSE call. See "MQCLOSE (Close object) on IBM i" on page 3282 for more details.

The value of the **QSGDisp** attribute for a permanent dynamic queue is QSGDQM.

QDTEMP
Dynamically defined temporary queue.

The queue is a temporary queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value QDTEMP for the **DefinitionType** attribute.

This type of queue is deleted automatically by the MQCLOSE call when it is closed by the application that created it.

The value of the **QSGDisp** attribute for a temporary dynamic queue is QSGDQM.

QDSHAR
Dynamically defined shared queue.

The queue is a shared permanent queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value QDSHAR for the **DefinitionType** attribute.

This type of queue can be deleted using the MQCLOSE call. See "MQCLOSE (Close object) on IBM i" on page 3282 for more details.

The value of the **QSGDisp** attribute for a shared dynamic queue is QSGDSH.

This attribute in a model queue definition does not indicate how the model queue was defined, because model queues are always predefined. Instead, the value of this attribute in the model queue is used to determine the *DefinitionType* of each of the dynamic queues created from the model queue definition using the MQOPEN call.

To determine the value of this attribute, use the IADEFT selector with the MQINQ call.

DefInputOpenOption (10-digit signed integer) on IBM i: IBM i

Default input open option.

Local	Model	Alias	Remote	Cluster
✓	✓			

This is the default way in which the queue should be opened for input. It applies if the OOINPQ option is specified on the MQOPEN call when the queue is opened. This can have one of the following values:

OOINPX
Open queue to get messages with exclusive access.

The queue is opened for use with subsequent MQGET calls. The call fails with reason code RC2042 if the queue is currently open by this or another application for input of any type (OOINPS or OOINPX).

OOINPS
Open queue to get messages with shared access.

The queue is opened for use with subsequent MQGET calls. The call can succeed if the queue is currently open by this or another application with OOINPS, but fails with reason code RC2042 if the queue is currently open with OOINPX.

To determine the value of this attribute, use the IADINP selector with the MQINQ call.

DefPersistence (10-digit signed integer) on IBM i: IBM i

Default message persistence.

Local	Model	Alias	Remote	Cluster
✓	✓	✓	✓	✓

This is the default persistence of messages on the queue. It applies if PEQDEF is specified in the message descriptor when the message is put.

If there is more than one definition in the queue-name resolution path, the default persistence is taken from the value of this attribute in the *first* definition in the path at the time of the MQPUT or MQPUT1 call. This could be:

- An alias queue
- A local queue
- A local definition of a remote queue
- A queue manager alias
- A transmission queue (for example, the *DefXmitQName* queue)

This can have one of the following values:

PEPER
Message is persistent.

This means that the message survives system failures and restarts of the queue manager. Persistent messages cannot be placed on:

- Temporary dynamic queues
- Shared queues

Persistent messages can be placed on permanent dynamic queues, and predefined queues.

PENPER

Message is not persistent.

This means that the message does not normally survive system failures or restarts of the queue manager. This applies even if an intact copy of the message is found on auxiliary storage during restart of the queue manager.

In the special case of shared queues, nonpersistent messages *do* survive restarts of queue managers in the queue-sharing group, but do not survive failures of the coupling facility used to store messages on the shared queues.

Both persistent and nonpersistent messages can exist on the same queue.

To determine the value of this attribute, use the IADPER selector with the MQINQ call.

DefPriority (10-digit signed integer) on IBM i: IBM i

Default message priority.

Local	Model	Alias	Remote	Cluster
✓	✓	✓	✓	✓

This is the default priority for messages on the queue. This applies if PRQDEF is specified in the message descriptor when the message is put on the queue.

If there is more than one definition in the queue-name resolution path, the default priority for the message is taken from the value of this attribute in the *first* definition in the path at the time of the put operation. This could be:

- An alias queue
- A local queue
- A local definition of a remote queue
- A queue manager alias
- A transmission queue (for example, the *DefXmitQName* queue)

The way in which a message is placed on a queue depends on the value of the queue's **MsgDeliverySequence** attribute:

- If the **MsgDeliverySequence** attribute is MSPRIO, the logical position at which a message is placed on the queue is dependent on the value of the *MDPRI* field in the message descriptor.
- If the **MsgDeliverySequence** attribute is MSFIFO, messages are placed on the queue as though they had a priority equal to the *DefPriority* of the resolved queue, regardless of the value of the *MDPRI* field in the message descriptor. However, the *MDPRI* field retains the value specified by the application that put the message. See the **MsgDeliverySequence** attribute described in "Attributes for queues" on page 3385 for more information.

Priorities are in the range zero (lowest) through *MaxPriority* (highest); see the **MaxPriority** attribute described in "Attributes for the queue manager on IBM i" on page 3420.

To determine the value of this attribute, use the IADPRI selector with the MQINQ call.

DefReadAhead (10-digit signed integer) on IBM i: IBM i

Specifies the default read ahead behavior for non-persistent messages delivered to the client.

Local	Model	Alias	Remote	Cluster
✓	✓	✓		

DefReadAhead can be set to one of the following values:

RAHNO
Non-persistent messages are not sent ahead to the client before an application requests them. A maximum of one non-persistent message can be lost if the client ends abnormally.

RAHYES
Non-persistent messages are sent ahead to the client before an application requests them. Non-persistent messages can be lost if the client ends abnormally or if the client does not consume all the messages it is sent.

RAHDIS
Read ahead of non-persistent messages in not enabled for this queue. Messages are not sent ahead to the client regardless of whether read ahead is requested by the client application.

To determine the value of this attribute, use the IADRAH selector with the MQINQ call.

DefPResp (10-digit signed integer) on IBM i: IBM i

The default put response type (DEFPRESP) attribute defines the value used by applications when the PutResponseType within MQPMO has been set to PMRASQ. This attribute is valid for all queue types.

Local	Model	Alias	Remote	Cluster
✓	✓	✓	✓	✓

This can have one of the following values:

SYNC The put operation is issued synchronously returning a response.

ASYNC
The put operation is issued asynchronously, returning a subset of MQMD fields.

To determine the value of this attribute, use the IADPRT selector with the MQINQ call.

DistLists (10-digit signed integer) on IBM i: IBM i

Distribution list support.

Local	Model	Alias	Remote	Cluster
✓	✓			

This indicates whether distribution-list messages can be placed on the queue. The attribute is set by a message channel agent (MCA) to inform the local queue manager whether the queue manager at the other end of the channel supports distribution lists. This latter queue manager (called the "partnering queue manager") is the one which next receives the message, after it has been removed from the local transmission queue by a sending MCA.

The attribute is set by the sending MCA whenever it establishes a connection to the receiving MCA on the partnering queue manager. In this way, the sending MCA can cause the local queue manager to place on the transmission queue only messages which the partnering queue manager can process correctly.

This attribute is primarily for use with transmission queues, but the processing described is performed regardless of the usage defined for the queue (see the **Usage** attribute).

This can have one of the following values:

DLSUPP
: Distribution lists supported.

  This indicates that distribution-list messages can be stored on the queue, and transmitted to the partnering queue manager in that form. This reduces the amount of processing required to send the message to multiple destinations.

DLNSUP
: Distribution lists not supported.

  This indicates that distribution-list messages cannot be stored on the queue, because the partnering queue manager does not support distribution lists. If an application puts a distribution-list message, and that message is to be placed on this queue, the queue manager splits the distribution-list message and places the individual messages on the queue instead. This increases the amount of processing required to send the message to multiple destinations, but ensures that the messages will be processed correctly by the partnering queue manager.

To determine the value of this attribute, use the IADIST selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

HardenGetBackout (10-digit signed integer) on IBM i: IBM i

Whether to maintain an accurate backout count.

Local	Model	Alias	Remote	Cluster
✓	✓			

For each message, a count is kept of the number of times that the message is retrieved by an MQGET call within a unit of work, and that unit of work later backed out. This count is available in the *MDBOC* field in the message descriptor after the MQGET call has completed.

The message backout count survives when the queue manager restarts. However, to ensure that the count is accurate, information has to be "hardened" (recorded on disk or other permanent storage device) each time a message is retrieved by an MQGET call within a unit of work for this queue. If this is not done, and a failure of the queue manager occurs together with backout of the MQGET call, the count might not be incremented.

Hardening information for each MQGET call within a unit of work, however, imposes a performance cost, and the **HardenGetBackout** attribute should be set to QABH only if the count has to be accurate.

- On IBM i, the message backout count is always hardened, regardless of the setting of this attribute.

The following values are possible:

QABH
Backout count remembered.

Hardening is used to ensure that the backout count for messages on this queue is accurate.

QABNH
Backout count might not be remembered.

Hardening is not used to ensure that the backout count for messages on this queue is accurate. The count might therefore be lower than it should be.

To determine the value of this attribute, use the IAHGB selector with the MQINQ call.

InhibitGet (10-digit signed integer) on IBM i: IBM i

Controls whether get operations for this queue are allowed.

Local	Model	Alias	Remote	Cluster
✓	✓	✓		

If the queue is an alias queue, get operations must be allowed for both the alias and the base queue at the time of the get operation, in order for the MQGET call to succeed. The value is one of the following:

QAGETI
Get operations are inhibited.

MQGET calls fail with reason code RC2016. This includes MQGET calls that specify GMBRWF or GMBRWN.

Note: If an MQGET call operating within a unit of work completes successfully, changing the value of the **InhibitGet** attribute after to QAGETI does not prevent the unit of work being committed.

QAGETA
Get operations are allowed.

To determine the value of this attribute, use the IAIGET selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

InhibitPut (10-digit signed integer) on IBM i: IBM i

Controls whether put operations for this queue are allowed.

Local	Model	Alias	Remote	Cluster
✓	✓	✓	✓	✓

If there is more than one definition in the queue-name resolution path, put operations must be allowed for *every* definition in the path (including any queue manager alias definitions) at the time of the put operation, in order for the MQPUT or MQPUT1 call to succeed. This can have one of the following values:

QAPUTI
Put operations are inhibited.

MQPUT and MQPUT1 calls fail with reason code RC2051.

Note: If an MQPUT call operating within a unit of work completes successfully, changing the value of the **InhibitPut** attribute later to QAPUTI does not prevent the unit of work being committed.

QAPUTA
Put operations are allowed.

To determine the value of this attribute, use the IAIPUT selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

InitiationQName (48-byte character string) on IBM i: IBM i

Name of initiation queue.

Local	Model	Alias	Remote	Cluster
✓	✓			

This is the name of a queue defined on the local queue manager; the queue must be of type QTLOC. The queue manager sends a trigger message to the initiation queue when application startup is required as a result of a message arriving on the queue to which this attribute belongs. The initiation queue must be monitored by a trigger monitor application which will start the appropriate application after receipt of the trigger message.

To determine the value of this attribute, use the CAINIQ selector with the MQINQ call. The length of this attribute is given by LNQN.

MaxMsgLength (10-digit signed integer) on IBM i: IBM i

Maximum message length in bytes.

Local	Model	Alias	Remote	Cluster
✓	✓			

This is an upper limit for the length of the longest *physical* message that can be placed on the queue. However, because the **MaxMsgLength** queue attribute can be set independently of the **MaxMsgLength** queue manager attribute, the actual upper limit for the length of the longest physical message that can be placed on the queue is the lesser of those two values.

If the queue manager supports segmentation, it is possible for an application to put a *logical* message that is longer than the lesser of the two **MaxMsgLength** attributes, but only if the application specifies the MFSEGA flag in MQMD. If that flag is specified, the upper limit for the length of a logical message is 999 999 999 bytes, but typically, resource constraints imposed by the operating system or by the environment in which the application is running, results in a lower limit.

An attempt to place on the queue a message that is too long fails with reason code:

- RC2030 if the message to too large for the queue
- RC2031 if the message to too large for the queue manager, but not too large for the queue

The lower limit for the **MaxMsgLength** attribute is zero. The upper limit is determined by the environment:

- On IBM i, the maximum message length is 100 MB (104 857 600 bytes).

For more information, see the **BUFLEN** parameter described in "MQPUT (Put message) on IBM i" on page 3348.

To determine the value of this attribute, use the IAMLEN selector with the MQINQ call.

MaxQDepth (10-digit signed integer) on IBM i: IBM i

Maximum queue depth.

Local	Model	Alias	Remote	Cluster
✓	✓			

This is the defined upper limit for the number of physical messages that can exist on the queue at any one time. An attempt to put a message on a queue that already contains *MaxQDepth* messages, fails with reason code RC2053.

Unit-of-work processing and the segmentation of messages can both cause the actual number of physical messages on the queue to exceed *MaxQDepth*. However, this does not affect the retrievability of the messages - *all* messages on the queue can be retrieved using the MQGET call in the normal way.

The value of this attribute is zero or greater. The upper limit is determined by the environment.

Note: It is possible for the storage space available to the queue to be exhausted even if there are fewer than *MaxQDepth* messages on the queue.

To determine the value of this attribute, use the IAMDEP selector with the MQINQ call.

MediaLog (10-digit signed integer) on IBM i: IBM i

Identity of the log extent (or journal receiver on IBM i) needed for media recovery of a particular queue.

Local	Model	Alias	Remote	Cluster
✓	✓			

On queue managers where circular logging is in use, the value is returned as a null string.

MsgDeliverySequence (10-digit signed integer) on IBM i: IBM i

Message delivery sequence.

Local	Model	Alias	Remote	Cluster
✓	✓			

This determines the order in which messages are returned to the application by the MQGET call:

MSFIFO
Messages are returned in FIFO order (first in, first out).

This means that an MQGET call will return the *first* message that satisfies the selection criteria specified on the call, regardless of the priority of the message.

MSPRIO
Messages are returned in priority order.

This means that an MQGET call will return the *highest-priority* message that satisfies the selection criteria specified on the call. Within each priority level, messages are returned in FIFO order (first in, first out).

If the relevant attributes are changed while there are messages on the queue, the delivery sequence is as follows:

- The order in which messages are returned by the MQGET call is determined by the values of the **MsgDeliverySequence** and **DefPriority** attributes in force for the queue at the time the message arrives on the queue:
  - If *MsgDeliverySequence* is MSFIFO when the message arrives, the message is placed on the queue as though its priority were *DefPriority*. This does not affect the value of the *MDPRI* field in the message descriptor of the message; that field retains the value it had when the message was first put.
  - If *MsgDeliverySequence* is MSPRIO when the message arrives, the message is placed on the queue at the place appropriate to the priority given by the *MDPRI* field in the message descriptor.

  If the value of the **MsgDeliverySequence** attribute is changed while there are messages on the queue, the order of the messages on the queue is not changed.

  If the value of the **DefPriority** attribute is changed while there are messages on the queue, the messages will not necessarily be delivered in FIFO order, even though the **MsgDeliverySequence** attribute is set to MSFIFO; those that were placed on the queue at the higher priority are delivered first.

To determine the value of this attribute, use the IAMDS selector with the MQINQ call.

OpenInputCount (10-digit signed integer) on IBM i: IBM i

Number of opens for input.

Local	Model	Alias	Remote	Cluster
✓				

This is the number of handles that are currently valid for removing messages from the queue with the MQGET call. It is the total number of such handles known to the *local* queue manager. If the queue is a shared queue, the count does not include opens for input that were performed for the queue at other queue managers in the queue-sharing group to which the local queue manager belongs.

The count includes handles where an alias queue which resolves to this queue was opened for input. The count does not include handles where the queue was opened for actions which did not include input (for example, a queue opened only for browse).

The value of this attribute fluctuates as the queue manager operates.

To determine the value of this attribute, use the IAOIC selector with the MQINQ call.

OpenOutputCount (10-digit signed integer) on IBM i: IBM i

Number of opens for output.

Local	Model	Alias	Remote	Cluster
✓				

This is the number of handles that are currently valid for adding messages to the queue with the MQPUT call. It is the total number of such handles known to the *local* queue manager; it does not include opens for output that were performed for this queue at remote queue managers. If the queue is a shared queue, the count does not include opens for output that were performed for the queue at other queue managers in the queue-sharing group to which the local queue manager belongs.

The count includes handles where an alias queue which resolves to this queue was opened for output. The count does not include handles where the queue was opened for actions which did not include output (for example, a queue opened only for inquire).

The value of this attribute fluctuates as the queue manager operates.

To determine the value of this attribute, use the IAOOC selector with the MQINQ call.

ProcessName (48-byte character string) on IBM i: IBM i

Process name.

Local	Model	Alias	Remote	Cluster
✓	✓			

This is the name of a process object that is defined on the local queue manager. The process object identifies a program that can service the queue.

To determine the value of this attribute, use the CAPRON selector with the MQINQ call. The length of this attribute is given by LNPRON.

QDepthHighEvent (10-digit signed integer) on IBM i: IBM i

Controls whether Queue Depth High events are generated.

Local	Model	Alias	Remote	Cluster
✓	✓			

A Queue Depth High event indicates that an application has put a message on a queue, which has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold (see the **QDepthHighLimit** attribute).

Note: The value of this attribute can change dynamically.

QDepthHighEvent can have one of two values:

EVRDIS
Event reporting disabled.

EVRENA
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the IAQDHE selector with the MQINQ call.

QDepthHighLimit (10-digit signed integer) on IBM i: IBM i

High limit for queue depth.

Local	Model	Alias	Remote	Cluster
✓	✓			

This is the threshold against which the queue depth is compared to generate a Queue Depth High event. This event indicates that an application has put a message on a queue, and this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold. See the **QDepthHighEvent** attribute.

The value is expressed as a percentage of the maximum queue depth (**MaxQDepth** attribute), and is in the range zero through 100. The default value is 80.

To determine the value of this attribute, use the IAQDHL selector with the MQINQ call.

QDepthLowEvent (10-digit signed integer) on IBM i: IBM i

Controls whether Queue Depth Low events are generated.

Local	Model	Alias	Remote	Cluster
✓	✓			

A Queue Depth Low event indicates that an application has retrieved a message from a queue, which has caused the number of messages on the queue to become less than or equal to the queue depth low threshold (see the **QDepthLowLimit** attribute).

Note: The value of this attribute can change dynamically.

QDepthLowEvent can have one of the following values:

EVRDIS
Event reporting disabled.

EVRENA
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the IAQDLE selector with the MQINQ call.

QDepthLowLimit (10-digit signed integer) on IBM i: IBM i

Low limit for queue depth.

Local	Model	Alias	Remote	Cluster
✓	✓			

This is the threshold against which the queue depth is compared to generate a Queue Depth Low event. This event indicates that an application has retrieved a message from a queue, and this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold. See the **QDepthLowEvent** attribute.

The value is expressed as a percentage of the maximum queue depth (**MaxQDepth** attribute), and is in the range zero through 100. The default value is 20.

To determine the value of this attribute, use the IAQDLL selector with the MQINQ call.

QDepthMaxEvent (10-digit signed integer) on IBM i: IBM i

Controls whether Queue Full events are generated.

Local	Model	Alias	Remote	Cluster
✓	✓			

A Queue Full event indicates that a put to a queue has been rejected because the queue is full, that is, the queue depth has already reached its maximum value.

Note: The value of this attribute can change dynamically.

This can have one of the following values:

EVRDIS
Event reporting disabled.

EVRENA
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the IAQDME selector with the MQINQ call.

QDesc (64-byte character string) on IBM i: IBM i

Queue description.

Local	Model	Alias	Remote	Cluster
✓	✓	✓	✓	✓

This is a field that can be used for descriptive commentary. The content of the field is of no significance to the queue manager, but the queue manager might require that the field contains only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, the field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the queue manager's character set (as defined by the **CodedCharSetId** queue manager attribute), those characters might be translated incorrectly if this field is sent to another queue manager.

To determine the value of this attribute, use the CAQD selector with the MQINQ call. The length of this attribute is given by LNQD.

QName (48-byte character string) on IBM i: IBM i

Queue name.

Local	Model	Alias	Remote	Cluster
✓		✓	✓	✓

This is the name of a queue defined on the local queue manager. For more information about queue names, see Rules for naming IBM MQ objects. All queues defined on a queue manager share the same queue namespace. Therefore, a QTLOC queue and a QTALS queue cannot have the same name.

To determine the value of this attribute, use the CAQN selector with the MQINQ call. The length of this attribute is given by LNQN.

QServiceInterval (10-digit signed integer) on IBM i: IBM i

Target for queue service interval.

Local	Model	Alias	Remote	Cluster
✓	✓			

This is the service interval used for comparison to generate Service Interval High and Service Interval OK events. See the **QServiceIntervalEvent** attribute.

The value is in units of milliseconds, and is in the range zero through 999 999 999.

To determine the value of this attribute, use the IAQSI selector with the MQINQ call.

QServiceIntervalEvent (10-digit signed integer) on IBM i: IBM i

Controls whether Service Interval High or Service Interval OK events are generated.

Local	Model	Alias	Remote	Cluster
✓	✓			

- A Service Interval High event is generated when a check indicates that no messages have been retrieved from the queue for at least the time indicated by the **QServiceInterval** attribute.
- A Service Interval OK event is generated when a check indicates that messages have been retrieved from the queue within the time indicated by the **QServiceInterval** attribute.

Note: The value of this attribute can change dynamically.

This attribute can have one of the following values:

QSIEHI
Queue Service Interval High events enabled.
- Queue Service Interval High events are **enabled** and
- Queue Service Interval OK events are **disabled**.

QSIEOK
Queue Service Interval OK events enabled.
- Queue Service Interval High events are **disabled** and
- Queue Service Interval OK events are **enabled**.

QSIENO
No queue service interval events enabled.
- Queue Service Interval High events are **disabled** and
- Queue Service Interval OK events are also **disabled**.

For shared queues, the value of this attribute is ignored; the value QSIENO is assumed.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the IAQSIE selector with the MQINQ call.

QSGDisp (10-digit signed integer) on IBM i: IBM i

Queue-sharing group disposition.

Local	Model	Alias	Remote	Cluster
✓		✓	✓	

This specifies the disposition of the queue. The value is one of the following:

QSGDQM
Queue manager disposition.

The object has queue manager disposition. This means that the object definition is known only to the local queue manager; the definition is not known to other queue managers in the queue-sharing group.

It is possible for each queue manager in the queue-sharing group to have an object with the same name and type as the current object, but these are separate objects and there is no correlation between them. Their attributes are not constrained to be the same as each other.

QSGDCP
Copied-object disposition.

The object is a local copy of a master object definition that exists in the shared repository. Each queue manager in the queue-sharing group can have its own copy of the object. Initially, all copies have the same attributes, but by using MQSC commands each copy can be altered so that its attributes differ from those of the other copies. The attributes of the copies are resynchronized when the master definition in the shared repository is altered.

QSGDSH
Shared disposition.

The object has shared disposition. This means that there exists in the shared repository a single instance of the object that is known to all queue managers in the queue-sharing group. When a queue manager in the group accesses the object, it accesses the single shared instance of the object.

To determine the value of this attribute, use the IAQSGD selector with the MQINQ call.

z/OS This attribute is supported only on z/OS.

QType (10-digit signed integer) on IBM i: IBM i

Queue type.

Local	Model	Alias	Remote	Cluster
✓		✓	✓	✓

This attribute has one of the following values:

QTALS
Alias queue definition.

QTCLUS
Cluster queue.

QTLOC
Local queue.

QTREM
Local definition of a remote queue.

To determine the value of this attribute, use the IAQTYP selector with the MQINQ call.

RemoteQMgrName (48-byte character string) on IBM i: IBM i

Name of remote queue manager.

Local	Model	Alias	Remote	Cluster
			✓	

This is the name of the remote queue manager on which the queue *RemoteQName* is defined. If the *RemoteQName* queue has a *QSGDisp* value of QSGDCP or QSGDSH, *RemoteQMgrName* can be the name of the queue-sharing group that owns *RemoteQName*.

If an application opens the local definition of a remote queue, *RemoteQMgrName* must not be blank and must not be the name of the local queue manager. If *XmitQName* is blank, the local queue with same name as *RemoteQMgrName* is used as the transmission queue. If there is no queue with the name *RemoteQMgrName*, the queue identified by the **DefXmitQName** queue manager attribute is used.

If this definition is used for a queue manager alias, *RemoteQMgrName* is the name of the queue manager that is being aliased. It can be the name of the local queue manager. Otherwise, if *XmitQName* is blank when the open occurs, there must be a local queue with the same name as *RemoteQMgrName* ; this queue is used as the transmission queue.

If this definition is used for a reply-to alias, this name is the name of the queue manager which is to be the *MDRM*.

Note: No validation is performed on the value specified for this attribute when the queue definition is created or modified.

To determine the value of this attribute, use the CARQMN selector with the MQINQ call. The length of this attribute is given by LNQMN.

RemoteQName (48-byte character string) on IBM i: IBM i

Name of remote queue.

Local	Model	Alias	Remote	Cluster
			✓	

This is the name of the queue as it is known on the remote queue manager *RemoteQMgrName*.

If an application opens the local definition of a remote queue, when the open occurs *RemoteQName* must not be blank.

If this definition is used for a queue manager alias definition, when the open occurs *RemoteQName* must be blank.

If the definition is used for a reply-to alias, this name is the name of the queue that is to be the *MDRQ*.

Note: No validation is performed on the value specified for this attribute when the queue definition is created or modified.

To determine the value of this attribute, use the CARQN selector with the MQINQ call. The length of this attribute is given by LNQN.

RetentionInterval (10-digit signed integer) on IBM i: IBM i

Retention interval.

Local	Model	Alias	Remote	Cluster
✓	✓			

This is the time which the queue should be retained. After this time has elapsed, the queue is eligible for deletion.

The time is measured in hours, counting from the date and time when the queue was created. The creation date of the queue is recorded in the *CreationDate* and the create time of the queue is recorded in the **CreationTime** attribute.

This information is provided to enable a housekeeping application or the operator to identify and delete queues that are no longer required.

Note: The queue manager never tries to delete queues based on this attribute, or to prevent the deletion of queues with a retention interval that has not expired; it is the user's responsibility to cause any required action to be taken.

A realistic retention interval should be used to prevent the accumulation of permanent dynamic queues (see *DefinitionType*). However, this attribute can also be used with predefined queues.

To determine the value of this attribute, use the IARINT selector with the MQINQ call.

Scope (10-digit signed integer) on IBM i: IBM i

Controls whether an entry for this queue also exists in a cell directory.

Local	Model	Alias	Remote	Cluster
✓		✓	✓	

A cell directory is provided by an installable Name service. This can have one of the following values:

SCOQM
Queue manager scope.

The queue definition has queue manager scope. This means that the definition of the queue does not extend beyond the queue manager which owns it. To open the queue for output from some other queue manager, either the name of the owning queue manager must be specified, or the other queue manager must have a local definition of the queue.

SCOCEL
Cell scope.

The queue definition has cell scope. This means that the queue definition is also placed in a cell directory available to all of the queue managers in the cell. The queue can be opened for output from any of the queue managers in the cell merely by specifying the name of the queue; the name of the queue manager which owns the queue need not be specified. However, the queue definition is not available to any queue manager in the cell which also has a local definition of a queue with that name, as the local definition takes precedence.

A cell directory is provided by an installable name service such as LDAP (Lightweight Directory Access Protocol. Note that IBM MQ no longer supports the DCE (Distributed Computing Environment) name service that was formerly used for inserting queue definitions into a DCE directory (also no longer supported).

Model and dynamic queues cannot have cell scope.

This value is only valid if a name service supporting a cell directory has been configured.

To determine the value of this attribute, use the IASCOP selector with the MQINQ call.

Support for this attribute is subject to the following restrictions:

- On IBM i, the attribute is supported, but only SCOQM is valid.

Shareability (10-digit signed integer) on IBM i: IBM i

Whether queue can be shared for input.

Local	Model	Alias	Remote	Cluster
✓	✓			

This indicates whether the queue can be opened for input multiple times concurrently. This can have one of the following values:

QASHR

Queue is shareable.

Multiple opens with the OOINPS option are allowed.

QANSHR

Queue is not shareable.

An MQOPEN call with the OOINPS option is treated as OOINPX.

To determine the value of this attribute, use the IASHAR selector with the MQINQ call.

TriggerControl (10-digit signed integer) on IBM i: IBM i

Trigger control.

Local	Model	Alias	Remote	Cluster
✓	✓			

This controls whether trigger messages are written to an initiation queue, in order to cause an application to be started to service the queue. This is one of the following:

TCOFF

Trigger messages not required.

No trigger messages are to be written for this queue. The value of *TriggerType* is irrelevant in this case.

TCON

Trigger messages required.

Trigger messages are to be written for this queue, when the appropriate trigger events occur.

To determine the value of this attribute, use the IATRGC selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

TriggerData (64-byte character string) on IBM i: IBM i

Trigger data.

Local	Model	Alias	Remote	Cluster
✓	✓			

This is free-format data that the queue manager inserts into the trigger message when a message arriving on this queue causes a trigger message to be written to the initiation queue.

The content of this data is of no significance to the queue manager. It is meaningful either to the trigger-monitor application which processes the initiation queue, or to the application which is started by the trigger monitor.

The character string cannot contain any nulls. It is padded to the right with blanks if necessary.

To determine the value of this attribute, use the CATRGD selector with the MQINQ call. To change the value of this attribute, use the MQSET call. The length of this attribute is given by LNTRGD.

TriggerDepth (10-digit signed integer) on IBM i: IBM i

Trigger depth.

Local	Model	Alias	Remote	Cluster
✓	✓			

This is the number of messages of priority *TriggerMsgPriority* or greater that must be on the queue before a trigger message is written. This applies when *TriggerType* is set to TTDPTH. The value of *TriggerDepth* is one or greater. This attribute is not used otherwise.

To determine the value of this attribute, use the IATRGD selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

TriggerMsgPriority (10-digit signed integer) on IBM i: IBM i

Threshold message priority for triggers on IBM MQ for IBM i.

Local	Model	Alias	Remote	Cluster
✓	✓			

This is the message priority below which messages do not contribute to the generation of trigger messages (that is, the queue manager ignores these messages when determining whether a trigger message should be generated). *TriggerMsgPriority* can be in the range zero (lowest) through *MaxPriority* (highest; see "Attributes for the queue manager on IBM i" on page 3420); a value of zero causes all messages to contribute to the generation of trigger messages.

To determine the value of this attribute, use the IATRGP selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

TriggerType (10-digit signed integer) on IBM i: IBM i

Trigger type.

Local	Model	Alias	Remote	Cluster
✓	✓			

This controls the conditions under which trigger messages are written as a result of messages arriving on this queue. The value is one of the following:

TTNONE
No trigger messages.

No trigger messages are written as a result of messages on this queue. This has the same effect as setting *TriggerControl* to TCOFF.

TTFRST
Trigger message when queue depth goes from 0 to 1.

A trigger message is written whenever the number of messages of priority *TriggerMsgPriority* or greater on the queue changes from 0 to 1.

TTEVRY
Trigger message for every message.

A trigger message is written whenever a message of priority *TriggerMsgPriority* or greater arrives on the queue.

TTDPTH
Trigger message when depth threshold exceeded.

A trigger message is written whenever the number of messages of priority *TriggerMsgPriority* or greater on the queue equals or exceeds *TriggerDepth*. After the trigger message has been written, *TriggerControl* is set to TCOFF to prevent further triggering until it is explicitly turned on again.

To determine the value of this attribute, use the IATRGT selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

Usage (10-digit signed integer) on IBM i: IBM i

Queue usage.

Local	Model	Alias	Remote	Cluster
✓	✓			

This indicates what the queue is used for. The value is one of the following:

USNORM
Normal usage.

This is a queue that normal applications use when putting and getting messages; the queue is not a transmission queue.

USTRAN
Transmission queue.

This is a queue used to hold messages destined for remote queue managers. When a normal application sends a message to a remote queue, the local queue manager stores the message temporarily on the appropriate transmission queue in a special format. A message channel agent then reads the message from the transmission queue, and transports the message to the remote queue manager. For more information about transmission queues, see Transmission queues.

Only privileged applications can open a transmission queue for OOOUT to put messages on it directly. Only utility applications would normally be expected to do this. Care must be taken that the message data format is correct (see "MQXQH (Transmission-queue header) on IBM i" on page 3258), otherwise errors might occur during the transmission process. Context is not passed or set unless one of the PM* context options is specified.

To determine the value of this attribute, use the IAUSAG selector with the MQINQ call.

XmitQName (48-byte character string) on IBM i: IBM i

Transmission queue name.

Local	Model	Alias	Remote	Cluster
			✓	

If this attribute is nonblank when an open occurs, either for a remote queue or for a queue manager alias definition, it specifies the name of the local transmission queue to be used for forwarding the message.

If *XmitQName* is blank, the local queue with the same name as *RemoteQMgrName* is used as the transmission queue. If there is no queue with the name *RemoteQMgrName*, the queue identified by the **DefXmitQName** queue manager attribute is used.

This attribute is ignored if the definition is being used as a queue manager alias and *RemoteQMgrName* is the name of the local queue manager. It is also ignored if the definition is used as a reply-to queue alias definition.

To determine the value of this attribute, use the CAXQN selector with the MQINQ call. The length of this attribute is given by LNQN.

Attributes for namelists:

This topic summarizes the attributes that are specific to namelists. The attributes are described in alphabetical order.

Note: The names of the attributes shown are the names used with the MQINQ and MQSET calls.

Attribute descriptions

A namelist object has the following attributes:

AlterationDate (12-byte character string)

Date when definition was last changed.

This is the date when the definition was last changed. The format of the date is `YYYY-MM-DD`, padded with two trailing blanks to make the length 12 bytes.

To determine the value of this attribute, use the CAALTD selector with the MQINQ call. The length of this attribute is given by LNDATE.

AlterationTime (8-byte character string)

Time when definition was last changed.

This is the time when the definition was last changed. The format of the time is `HH.MM.SS`.

To determine the value of this attribute, use the CAALTT selector with the MQINQ call. The length of this attribute is given by LNTIME.

NameCount (10-digit signed integer)

Number of names in namelist.

This is greater than or equal to zero. The following value is defined:

NCMXNL
Maximum number of names in a namelist.

To determine the value of this attribute, use the IANAMC selector with the MQINQ call.

NamelistDesc (64-byte character string)

Namelist description.

This is a field that might be used for descriptive commentary; its value is established by the definition process. The content of the field is of no significance to the queue manager, but the queue manager might require that the field contains only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, this field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the queue manager's character set (as defined by the **CodedCharSetId** queue manager attribute), those characters might be translated incorrectly if this field is sent to another queue manager.

To determine the value of this attribute, use the CALSTD selector with the MQINQ call.

The length of this attribute is given by LNNLD.

NamelistName (48-byte character string)

Namelist name.

This is the name of a namelist that is defined on the local queue manager.

Each namelist has a name that is different from the names of other namelists belonging to the queue manager, but might duplicate the names of other queue manager objects of different types (for example, queues).

To determine the value of this attribute, use the CALSTN selector with the MQINQ call.

The length of this attribute is given by LNNLN.

Names (48-byte character string x NameCount)

A list of *NameCount* names.

Each name is the name of an object that is defined to the local queue manager. For more information about object names, see Naming IBM MQ objects.

To determine the value of this attribute, use the CANAMS selector with the MQINQ call.

The length of each name in the list is given by LNOBJN.

Attributes for process definitions on IBM i: 

This topic summarizes the attributes that are specific to process definitions. The attributes are described in alphabetical order.

Note: The names of the attributes shown are the names used with the MQINQ and MQSET calls. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see MQSC commands for details.

Attribute descriptions

A process-definition object has the following attributes:

AlterationDate (12-byte character string)

Date when definition was last changed.

This is the date when the definition was last changed. The format of the date is `YYYY-MM-DD`, padded with two trailing blanks to make the length 12 bytes.

To determine the value of this attribute, use the CAALTD selector with the MQINQ call. The length of this attribute is given by LNDATE.

AlterationTime (8-byte character string)

Time when definition was last changed.

This is the time when the definition was last changed. The format of the time is `HH.MM.SS`.

To determine the value of this attribute, use the CAALTT selector with the MQINQ call. The length of this attribute is given by LNTIME.

ApplId (256-byte character string)

Application identifier.

This is a character string that identifies the application to be started. This information is for use by a trigger-monitor application that processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message.

The meaning of *ApplId* is determined by the trigger-monitor application. The trigger monitor provided by IBM MQ requires *ApplId* to be the name of an executable program.

The character string cannot contain any nulls. It is padded to the right with blanks if necessary.

To determine the value of this attribute, use the CAAPPI selector with the MQINQ call. The length of this attribute is given by LNPROA.

ApplType (10-digit signed integer)

Application type.

This identifies the nature of the program to be started in response to the receipt of a trigger message. This information is for use by a trigger-monitor application that processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message.

ApplType can have any value. You can use the following values for standard types; user-defined application types are restricted to values in the range ATUFST through ATULST:

ATCICS
CICS transaction.

AT400 IBM i application.

ATUFST
Lowest value for user-defined application type.

ATULST
Highest value for user-defined application type.

To determine the value of this attribute, use the IAAPPT selector with the MQINQ call.

EnvData (128-byte character string)

Environment data.

This is a character string that contains environment-related information pertaining to the application to be started. This information is for use by a trigger-monitor application that processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message.

The meaning of *EnvData* is determined by the trigger-monitor application. The trigger monitor provided by IBM MQ appends *EnvData* to the parameter list passed to the started application. The parameter list consists of the MQTMC2 structure, followed by one blank, followed by *EnvData* with trailing blanks removed.

The character string cannot contain any nulls. It is padded to the right with blanks if necessary.

To determine the value of this attribute, use the CAENVD selector with the MQINQ call. The length of this attribute is given by LNPROE.

ProcessDesc (64-byte character string)

Process description.

This is a field that can be used for descriptive commentary. The content of the field is of no significance to the queue manager, but the queue manager might require that the field contain only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, the field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the queue manager's character set (as defined by the **CodedCharSetId** queue manager attribute), those characters might be translated incorrectly if this field is sent to another queue manager.

To determine the value of this attribute, use the CAPROD selector with the MQINQ call.

The length of this attribute is given by LNPROD.

ProcessName (48-byte character string)

Process name.

This is the name of a process definition that is defined on the local queue manager.

Each process definition has a name that is different from the names of other process definitions belonging to the queue manager. But the name of the process definition can be the same as the names of other queue manager objects of different types (for example, queues).

To determine the value of this attribute, use the CAPRON selector with the MQINQ call.

The length of this attribute is given by LNPRON.

UserData (128-byte character string)

User data.

This is a character string that contains user information pertaining to the application to be started. This information is for use by a trigger-monitor application that processes messages on the initiation queue, or the application which is started by the trigger monitor. The information is sent to the initiation queue as part of the trigger message.

The meaning of *UserData* is determined by the trigger-monitor application. The trigger monitor provided by IBM MQ passes *UserData* to the started application as part of the parameter list. The parameter list consists of the MQTMC2 structure (containing *UserData*), followed by one blank, followed by *EnvData* with trailing blanks removed.

The character string cannot contain any nulls. It is padded to the right with blanks if necessary.

To determine the value of this attribute, use the CAUSRD selector with the MQINQ call. The length of this attribute is given by LNPROU.

Attributes for the queue manager on IBM i: IBM i

A summary of queue manager attributes.

Some queue manager attributes are fixed for particular implementations, while others can be changed by using the MQSC command `ALTER QMGR`. The attributes can also be displayed by using the command `DISPLAY QMGR`. Most queue manager attributes can be inquired by opening a special OTQM object, and using the MQINQ call with the handle returned.

The following table summarizes the attributes that are specific to the queue manager. The attributes are described in alphabetical order.

Note: The names of the attributes shown in this section are the names used with the MQINQ and MQSET calls. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see Script (MQSC) Commands for more information.

Table 370. Attributes for the queue manager

Attribute	Description
AlterationDate	Date when definition was last changed
AlterationTime	Time when definition was last changed
AuthorityEvent	Controls whether authorization (Not Authorized) events are generated
BridgeEvent	Controls whether IMS bridge events are generated
ChannelAutoDef	Controls whether automatic channel definition is permitted
ChannelAutoDefEvent	Controls whether channel automatic-definition events are generated
ChannelAutoDefExit	Name of user exit for automatic channel definition
ChannelEvent	Controls whether channel events are generated
ClusterCacheType	Controls whether the cluster cache is fixed in size or dynamically sized
ClusterWorkloadData	User data for cluster workload exit
ClusterWorkloadExit	Name of user exit for cluster workload management

Table 370. Attributes for the queue manager (continued)

Attribute	Description
ClusterWorkloadLength	Maximum length of message data passed to cluster workload exit
CodedCharSetId	Coded character set identifier
CommandEvent	Controls whether command event messages are queued
CommandInputQName	Command input queue name
CommandLevel	Command level
ConfigurationEvent	Configuration event
DeadLetterQName	Name of dead-letter queue
DefClusterXmitQueueType	Default cluster transmission queue type
DefXmitQName	Default transmission queue name
DistLists	Distribution list support
InhibitEvent	Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated
LocalEvent	Controls whether local error events are generated
LoggerEvent	Controls whether recovery log events are generated
MaxHandles	Maximum number of handles
MaxMsgLength	Maximum message length in bytes
MaxPriority	Maximum priority
MaxUncommittedMsgs	Maximum number of uncommitted messages within a unit of work
PerformanceEvent	Controls whether performance-related events are generated
Platform	Platform on which the queue manager is running
PubSubMode	Whether the publish/subscribe engine and queued publish/subscribe interface are running
QMgrDesc	Queue manager description
QMgrIdentifier	Unique internally-generated identifier of queue manager
QMgrName	Queue manager name
RemoteEvent	Controls whether remote error events are generated
RepositoryName	Name of cluster for which this queue manager provides repository services
RepositoryNamelist	Name of namelist object containing names of clusters for which this queue manager provides repository services
SSLCRLNamelist	Name of namelist object containing names of authentication information objects (See Note 1)
SSLEvent	Controls whether TLS events are generated
SSLKeyRepository	Location of TLS key repository (See Note 1)
SSLKeyResetCount	Determines the number of non-encrypted bytes sent and received within a TLS conversation before the encryption key is renegotiated
StartStopEvent	Controls whether start and stop events are generated
SyncPoint	Syncpoint availability
TraceRouteRecording	Controls the recording of trace route information for messages
TreeLifeTime	The lifetime, in seconds, of non-administrative topics
TriggerInterval	Trigger-message interval

Table 370. Attributes for the queue manager (continued)

Attribute	Description
Notes: 1. This attribute cannot be inquired using the MQINQ call, and is not described in this section. For more information about this attribute, see Change Queue Manager.	

AlterationDate (12-byte character string) on IBM i: IBM i

Date when definition was last changed.

This is the date when the definition was last changed. The format of the date is `YYYY-MM-DD`, padded with two trailing blanks to make the length 12 bytes.

To determine the value of this attribute, use the CAALTD selector with the MQINQ call. The length of this attribute is given by LNDATE.

AlterationTime (8-byte character string) on IBM i: IBM i

Time when definition was last changed.

This is the time when the definition was last changed. The format of the time is `HH.MM.SS`.

To determine the value of this attribute, use the CAALTT selector with the MQINQ call. The length of this attribute is given by LNTIME.

AuthorityEvent (10-digit signed integer) on IBM i: IBM i

Controls whether authorization (Not Authorized) events are generated.

The AuthorityEvent attribute must be set to one of the following values:

EVRDIS
Event reporting disabled.

EVRENA
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the IAAUTE selector with the MQINQ call.

BridgeEvent (character string) on IBM i: IBM i

This attribute determines whether IMS bridge event messages are put onto the SYSTEM.ADMIN.CHANNEL.EVENT queue. It is only supported on z/OS.

ChannelAutoDef (10-digit signed integer) on IBM i: IBM i

Controls whether automatic channel definition is permitted.

This attribute controls the automatic definition of channels of type CTRCVR and CTSVCN. Note that the automatic definition of CTCLSD channels is always enabled. This can have one of the following values:

CHADDI
Channel auto-definition disabled.

CHADEN
Channel auto-definition enabled.

To determine the value of this attribute, use the IACAD selector with the MQINQ call.

ChannelAutoDefEvent (10-digit signed integer) on IBM i: IBM i

Controls whether channel automatic-definition events are generated.

This applies to channels of type CTRCVR, CTSVCN, and CTCLSD. This can have one of the following values:

EVRDIS
Event reporting disabled.

EVRENA
Event reporting enabled.

For more information about events, see Monitoring and performance.

To determine the value of this attribute, use the IACADE selector with the MQINQ call.

ChannelAutoDefExit (20-byte character string) on IBM i: IBM i

Name of user exit for automatic channel definition.

If this name is nonblank, and *ChannelAutoDef* has the value CHADEN, the exit is called each time that the queue manager is about to create a channel definition. This applies to channels of type CTRCVR, CTSVCN, and CTCLSD. The exit can then do one of the following:

- Allow the creation of the channel definition to proceed without change.
- Modify the attributes of the channel definition that is created.
- Suppress creation of the channel entirely.

To determine the value of this attribute, use the CACADX selector with the MQINQ call. The length of this attribute is given by LNEXN.

ChannelEvent (character string) on IBM i: 

Determines whether channel event messages are generated.

This attribute determines whether channel event messages are put onto the SYSTEM.ADMIN.CHANNEL.EVENT queue, and if so, what type of messages are queued (for example 'channel started', 'channel stopped', 'channel not activated'). Before the implementation of this attribute, the only way of preventing channel event messages from being queued was to delete the target queue.

This attribute also allows you to collect IMS bridge events only (because you can now switch off channel events, they do not get put onto the same queue). The same applies to TLS events which can also be collected without having to collect channel events as well.

This attribute also allows you to collect significant events only (for example when channels have errors, not when they start and stop normally).

The value for the ChannelEvent attribute can be one of the following:

- EVREXP (only the following channel events are generated: RC2279, RC2283, RC2284, RC2295, RC2296).
- EVRENA (all channel events are generated; that is, in addition to the events generated by EVREXP, the RC2282, and RC2283 events are also generated).
- EVRDIS (no channel events are generated; this is the queue manager initial default value).

To determine the value of this attribute, use the IACHNE selector with the MQINQ call.

ClusterCacheType (32-byte character string) on IBM i: 

Controls whether cluster cache is fixed size, or is dynamically sized.

This is a user-defined 32-byte character string that is passed to the cluster workload exit when it is called. If there is no data to pass to the exit, the string is blank.

To determine the value of this attribute, use the CACLWD selector with the MQINQ call.

ClusterWorkloadData (32-byte character string) on IBM i: 

User data for cluster workload exit.

This is a user-defined 32-byte character string that is passed to the cluster workload exit when it is called. If there is no data to pass to the exit, the string is blank.

To determine the value of this attribute, use the CACLWD selector with the MQINQ call.

ClusterWorkloadExit (20-byte character string) on IBM i: IBM i

Name of user exit for cluster workload management.

If this name is not blank, the exit is called each time that a message is put to a cluster queue or moved from one cluster-sender queue to another. The exit can then either accept the queue instance selected by the queue manager as the destination for the message, or select another queue instance.

To determine the value of this attribute, use the CACLWX selector with the MQINQ call. The length of this attribute is given by LNEXN.

ClusterWorkloadLength (10-digit signed integer) on IBM i: IBM i

Maximum length of message data passed to cluster workload exit.

This is the maximum length of message data that is passed to the cluster workload exit. The actual length of data passed to the exit is the minimum of the following:

- The length of the message.
- The queue manager's **MaxMsgLength** attribute.
- The **ClusterWorkloadLength** attribute.

To determine the value of this attribute, use the IACLWL selector with the MQINQ call.

CodedCharSetId (10-digit signed integer) on IBM i: IBM i

Coded character set identifier.

This defines the character set used by the queue manager for all character string fields defined in the MQI such as the names of objects, and queue creation date and time. The character set must be one that has single-byte characters for the characters that are valid in object names. It does not apply to application data carried in the message. The value depends on the environment:

- On IBM i, the value is that which is set in the environment when the queue manager is first created.

To determine the value of this attribute, use the IACCSI selector with the MQINQ call.

CommandEvent (integer) on IBM i: IBM i

Controls whether messages are put onto a local queue when commands are issued.

This controls whether messages are written to a new event queue, SYSTEM.ADMIN.COMMAND.EVENT, whenever commands are issued. This feature is useful for command tracking notification, and for problem diagnosis. To inquire about the CommandEvent queue manager attribute, use the new attribute selector iacev with one of the following values:

- EVRENA - command event messages are generated and put onto the queue for all successful commands.
- EVND - command event messages are generated and put onto the queue for all successful commands other than the DISPLAY (MQSC) command, and the Inquire (PCF) command.
- EVRDIS - command event messages are not generated or put onto the queue (this is the queue manager's initial default value).

To determine the value of this attribute, use the CMDEV selector with the MQINQ call.

CommandInputQName (48-byte character string) on IBM i: IBM i

Command input queue name.

CommandInputQName is the name of the command input queue defined on the local queue manager. It is a queue to which users can send commands, if authorized to do so. The name of the queue depends on the environment:

- On IBM i, the name of the queue is SYSTEM.ADMIN.COMMAND.QUEUE, and only PCF commands can be sent to it. However, an MQSC command can be sent to this queue if the MQSC command is enclosed within a PCF command of type CMESC. For more information about the Escape command, see Escape.

To determine the value of this attribute, use the CACMDQ selector with the MQINQ call. The length of this attribute is given by LNQN.

CommandLevel (10-digit signed integer) on IBM i: IBM i

Command Level. This indicates the level of system control commands supported by the queue manager.

The level is one of the following values:

CMLVL1
Level 1 of system control commands.

This value is returned by the following applications:

- MQSeries for OS/400
  - Version 2 Release 3
  - Version 3 Release 1
  - Version 3 Release 6

CML320
Level 320 of system control commands.

This value is returned by the following applications:

- MQSeries for OS/400
  - Version 3 Release 2
  - Version 3 Release 7

CML420
Level 420 of system control commands.

This value is returned by the following applications:

- MQSeries for AS/400
  - Version 4 Release 2.0
  - Version 4 Release 2.1

CML510
Level 510 of system control commands.

This value is returned by the following applications:

- MQSeries for AS/400 Version 5 Release 1

CML520
Level 520 of system control commands.

This value is returned by the following applications:

- MQSeries for AS/400 Version 5 Release 2

CML530
Level 530 of system control commands.

This value is returned by the following applications:

- IBM MQ for IBM i Version 5 Release 3

CML600
Level 600 of system control commands.

This value is returned by the following applications:

- IBM MQ for IBM i Version 6 Release 0

CML700
Level 700 of system control commands.

This value is returned by the following applications:

- IBM MQ for IBM i Version 7 Release 0

CML701
Level 701 of system control commands.

This value is returned by the following applications:

- IBM MQ for IBM i Version 7 Release 0 Modification 1

The set of system control commands that corresponds to a particular value of the **CommandLevel** attribute varies according to the value of the **Platform** attribute; both must be used to decide which system control commands are supported.

To determine the value of this attribute, use the IACMDL selector with the MQINQ call.

ConfigurationEvent on IBM i: 

Controls whether configuration events are generated and sent to the SYSTEM.ADMIN.CONFIG.EVENT queue default object.

The ConfigurationEvent attribute can be one of the following values:

- EVRENA
- EVRDIS

If the ConfigurationEvent attribute is set to EVRENA, and certain commands are successfully issued by runmqsc or PCF, configuration events are generated and sent to the SYSTEM.ADMIN.CONFIG.EVENT queue. Events for the following commands are issued, even if an alter command does not change the object involved. The commands for which configuration events are generated and sent are:

- DEFINE/ALTER AUTHINFO
- DEFINE/ALTER CHANNEL
- DEFINE/ALTER NAMELIST
- DEFINE/ALTER PROCESS
- DEFINE/ALTER QLOCAL (unless it is a temporary dynamic queue)
- DEFINE/ALTER QMODEL/QALIAS/QREMOTE
- DELETE AUTHINFO
- DELETE CHANNEL
- DELETE NAMELIST
- DELETE PROCESS
- DELETE QLOCAL (unless it is a temporary dynamic queue)
- DELETE QMODEL/QALIAS/QREMOTE

- ALTER QMGR (unless the CONFIGEV attribute is disabled and is not changed to enabled)
- REFRESH QMGR
- An MQSET call, other than for a temporary dynamic queue.

Events are not generated (if enabled) in the following circumstances:

- The command or MQSET call fails.
- The queue manager cannot put the event message on the event queue. The command should still complete successfully.
- Temporary dynamic queues.
- Internal attribute changes done directly or implicitly (not by MQSET or command); this affects TRIGGER, CURDEPTH, IPPROCS, OPPROCS, QDPHIEV, QDPLOEV, QDPMAXEV, QSVCIEV.
- When the configuration event queue is changed, although it an event message will be generated for that change when a Refresh is requested.
- Clustering changes by the commands REFRESH/RESET CLUSTER and RESUME/SUSPEND QMGR.
- Creating or deleting a queue manager.

DeadLetterQName (48-byte character string) on IBM i: IBM i

Name of dead-letter (undelivered-message) queue.

This is the name of a queue defined on the local queue manager. Messages are sent to this queue if they cannot be routed to their correct destination.

For example, messages are put on this queue when:

- A message arrives at a queue manager, destined for a queue that is not yet defined on that queue manager
- A message arrives at a queue manager, but the queue for which it is destined cannot receive it because, possibly:
  - The queue is full
  - Put requests are inhibited
  - The sending node does not have authority to put messages on the queue

Applications can also put messages on the dead-letter queue.

Report messages are treated in the same way as ordinary messages; if the report message cannot be delivered to its destination queue (typically the queue specified by the *MDRQ* field in the message descriptor of the original message), the report message is placed on the dead-letter (undelivered-message) queue.

Note: Messages that have passed their expiry time (see the *MDEXP* field described in "MQMD (Message descriptor) on IBM i" on page 3118) are **not** transferred to this queue when they are discarded. However, an expiration report message (ROEXP) is still generated and sent to the *MDRQ* queue, if requested by the sending application.

Messages are not put on the dead-letter (undelivered-message) queue when the application that issued the put request has been notified synchronously of the problem with the reason code returned by the MQPUT or MQPUT1 call (for example, a message put on a local queue for which put requests are inhibited).

Messages on the dead-letter (undelivered-message) queue sometimes have their application message data prefixed with an MQDLH structure. This structure contains extra information that indicates why the message was placed on the dead-letter (undelivered-message) queue. See "MQDLH (Dead-letter header) on IBM i" on page 3072 for more details of this structure.

This queue must be a local queue, with a **Usage** attribute of USNORM.

If a dead-letter (undelivered-message) queue is not supported by a queue manager, or one has not been defined, the name is all blanks. All IBM MQ queue managers support a dead-letter (undelivered-message) queue, but by default it is not defined.

If the dead-letter (undelivered-message) queue is not defined, or it is full, or unusable for some other reason, a message which would have been transferred to it by a message channel agent is retained instead on the transmission queue.

To determine the value of this attribute, use the CADLQ selector with the MQINQ call. The length of this attribute is given by LNQN.

DefClusterXmitQueueType (10-digit signed integer):

The DefClusterXmitQueueType attribute controls which transmission queue is selected by default by cluster-sender channels to get messages from, to send the messages to cluster-receiver channels.

The values of DefClusterXmitQueueType are MQCLXQ_SCTQ or MQCLXQ_CHANNEL.

MQCLXQ_SCTQ

All cluster-sender channels send messages from SYSTEM.CLUSTER.TRANSMIT.QUEUE. The correlID of messages placed on the transmission queue identifies which cluster-sender channel the message is destined for.

SCTQ is set when a queue manager is defined. This behavior is implicit in versions of IBM WebSphere MQ, earlier than Version 7.5. In earlier versions, the queue manager attribute DefClusterXmitQueueType was not present.

MQCLXQ_CHANNEL

Each cluster-sender channel sends messages from a different transmission queue. Each transmission queue is created as a permanent dynamic queue from the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE.

If the queue manager attribute, DefClusterXmitQueueType, is set to CHANNEL, the default configuration is changed to cluster-sender channels being associated with individual cluster transmission queues. The transmission queues are permanent-dynamic queues created from the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE. Each transmission queue is associated with one cluster-sender channel. As one cluster-sender channel services a cluster transmission queue, the transmission queue contains messages for only one queue manager in one cluster. You can configure clusters so that each queue manager in a cluster contains only one cluster queue. In this case, the message traffic from a queue manager to each cluster queue is transferred separately from messages to other queues.

To query the value, call MQINQ, or send an Inquire Queue Manager (MQCMD_INQUIRE_Q_MGR) PCF command, setting the MQIA_DEF_CLUSTER_XMIT_Q_TYPE selector. To change the value, send a Change Queue Manager (MQCMD_CHANGE_Q_MGR) PCF command, setting the MQIA_DEF_CLUSTER_XMIT_Q_TYPE selector.

Related reference:

"MQINQ (Inquire about object attributes) on IBM i" on page 3320
The MQINQ call returns an array of integers and a set of character strings containing the attributes of an object.

Related information:

Change Queue Manager
The Change Queue Manager (`MQCMD_CHANGE_Q_MGR`) command changes the specified attributes of the queue manager.

Inquire Queue Manager
The Inquire Queue Manager (**MQCMD_INQUIRE_Q_MGR**) command inquires about the attributes of a queue manager.

DefXmitQName (48-byte character string) on IBM i: 

Default transmission queue name.

This is the name of the transmission queue that is used for the transmission of messages to remote queue managers, if there is no other indication of which transmission queue to use.

If there is no default transmission queue, the name is entirely blank. The initial value of this attribute is blank.

To determine the value of this attribute, use the CADXQN selector with the MQINQ call. The length of this attribute is given by LNQN.

DistLists (10-digit signed integer) on IBM i: 

Distribution list support.

This indicates whether the local queue manager supports distribution lists on the MQPUT and MQPUT1 calls. This can have one of the following values:

DLSUPP
: Distribution lists supported.

DLNSUP
: Distribution lists not supported.

To determine the value of this attribute, use the IADIST selector with the MQINQ call.

InhibitEvent (10-digit signed integer) on IBM i: 

Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated.

This can have one of the following values:

EVRDIS
: Event reporting disabled.

EVRENA
: Event reporting enabled.

For more information about events, see Monitoring and performance.

To determine the value of this attribute, use the IAINHE selector with the MQINQ call.

LocalEvent (10-digit signed integer) on IBM i: 

Controls whether local error events are generated.

The value is one of the following:

EVRDIS
Event reporting disabled.

EVRENA
Event reporting enabled.

For more information about events, see Event monitoring

To determine the value of this attribute, use the IALCLE selector with the MQINQ call.

LoggerEvent (10-digit signed integer) on IBM i: 

Controls whether recovery logger events are generated.

This can have one of the following values:

ENABLED
Logger events are generated.

DISABLED
Logger events are not generated. This is the queue managers initial default value.

For more information about events, see Monitoring and performance.

MaxHandles (10-digit signed integer) on IBM i: 

Maximum number of handles.

This is the maximum number of open handles that any one task can use concurrently. Each successful MQOPEN call for a single queue (or for an object that is not a queue) uses one handle. That handle becomes available for reuse when the object is closed. However, when a distribution list is opened, each queue in the distribution list is allocated a separate handle, and so that MQOPEN call uses as many handles as there are queues in the distribution list. This must be taken into account when deciding on a suitable value for *MaxHandles*.

The MQPUT1 call performs an MQOPEN call as part of its processing; as a result, MQPUT1 uses as many handles as MQOPEN would, but the handles are used only for the duration of the MQPUT1 call itself.

The value is in the range 1 through 999 999 999. On IBM i, the default value is 256.

To determine the value of this attribute, use the IAMHND selector with the MQINQ call.

MaxMsgLength (10-digit signed integer) on IBM i: IBM i

Maximum message length in bytes.

This is the length of the longest *physical* message that can be handled by the queue manager. However, because the **MaxMsgLength** queue manager attribute can be set independently of the **MaxMsgLength** queue attribute, the longest physical message that can be placed on a queue is the lesser of those two values.

If the queue manager supports segmentation, it is possible for an application to put a *logical* message that is longer than the lesser of the two **MaxMsgLength** attributes, but only if the application specifies the MFSEGA flag in MQMD. If that flag is specified, the upper limit for the length of a logical message is 999 999 999 bytes, but typically, resource constraints imposed by the operating system or by the environment in which the application is running, will result in a lower limit.

The lower limit for the **MaxMsgLength** attribute is 32 KB (32 768 bytes). On IBM i, the maximum message length is 100 MB (104 857 600 bytes).

To determine the value of this attribute, use the IAMLEN selector with the MQINQ call.

MaxPriority (10-digit signed integer) on IBM i: IBM i

Maximum priority.

This is the maximum message priority supported by the queue manager. Priorities range from zero (lowest) to *MaxPriority* (highest).

To determine the value of this attribute, use the IAMPRI selector with the MQINQ call.

MaxUncommittedMsgs (10-digit signed integer) on IBM i: IBM i

Maximum number of uncommitted messages within a unit of work.

This is the maximum number of uncommitted messages that can exist within a unit of work. The number of uncommitted messages is the sum of the following since the start of the current unit of work:

- Messages put by the application with the PMSYP option
- Messages retrieved by the application with the GMSYP option
- Trigger messages and COA report messages generated by the queue manager for messages put with the PMSYP option
- COD report messages generated by the queue manager for messages retrieved with the GMSYP option

The following messages are not counted as uncommitted:

- Messages put or retrieved by the application outside a unit of work
- Trigger messages or COA/COD report messages generated by the queue manager as a result of messages put or retrieved outside a unit of work
- Expiration report messages generated by the queue manager (even if the call causing the expiration report message specified GMSYP)
- Event messages generated by the queue manager (even if the call causing the event message specified PMSYP or GMSYP)

Note:

1. Exception report messages are generated by the Message Channel Agent (MCA), or by the application, and so are treated in the same way as ordinary messages put or retrieved by the application.

2. When a message or segment is put with the PMSYP option, the number of uncommitted messages is incremented by one regardless of how many physical messages actually result from the put. (More than one physical message might result if the queue manager needs to subdivide the message or segment.)
3. When a distribution list is put with the PMSYP option, the number of uncommitted messages is incremented by one *for each physical message that is generated*. This can be as small as one, or as great as the number of destinations in the distribution list.

The lower limit for this attribute is 1; the upper limit is 999 999 999.

To determine the value of this attribute, use the IAMUNC selector with the MQINQ call.

PerformanceEvent (10-digit signed integer) on IBM i: IBM i

Controls whether performance-related events are generated.

PerformanceEvent can have one of the following values:

EVRDIS
Event reporting disabled.

EVRENA
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the IAPFME selector with the MQINQ call.

Platform (10-digit signed integer) on IBM i: IBM i

Platform on which the queue manager is running.

This indicates the operating system on which the queue manager is running. The value is:

PL400 IBM i.

PubSubMode (10-digit signed integer) on IBM i: IBM i

Whether the publish/subscribe engine and the queued publish/subscribe interface are running, therefore allowing applications to publish/subscribe by using the application programming interface and the queues that are being monitored by the queued publish/subscribe interface.

This can have one of the following values:

PSMCP
The publish/subscribe engine is running. It is therefore possible to publish/subscribe by using the application programming interface. The queued publish/subscribe interface is not running, therefore any message that is put to the queues that are monitored by the queued publish/subscribe interface is not acted on. This setting is used for compatibility with WebSphere Message Broker V6 or earlier versions using this queue manager, because it must read the same queues from which the queued publish/subscribe interface normally reads.

PSMDS
The publish/subscribe engine and the queued publish/subscribe interface are not running. It is therefore not possible to publish/subscribe by using the application programming interface. Any publish/subscribe messages that are put to the queues that are monitored by the queued publish/subscribe interface are not acted on.

PSMEN

The publish/subscribe engine and the queued publish/subscribe interface are running. It is therefore possible to publish/subscribe by using the application programming interface and the queues that are being monitored by the queued publish/subscribe interface. This is the queue manager's initial default value.

To determine the value of this attribute, use the PSMODE selector with the MQINQ call.

QMgrDesc (64-byte character string) on IBM i: 

Queue manager description.

This is a field that can be used for descriptive commentary. The content of the field is of no significance to the queue manager, but the queue manager might require that the field contain only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, this field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the queue manager's character set (as defined by the **CodedCharSetId** queue manager attribute), those characters might be translated incorrectly if this field is sent to another queue manager.

On IBM i, the default value is blanks.

To determine the value of this attribute, use the CAQMD selector with the MQINQ call. The length of this attribute is given by LNQMD.

QMgrIdentifier (48-byte character string) on IBM i: 

Unique internally-generated identifier of queue manager.

This is an internally-generated unique name for the queue manager.

To determine the value of this attribute, use the CAQMID selector with the MQINQ call. The length of this attribute is given by LNQMID.

QMgrName (48-byte character string) on IBM i: 

Queue manager name.

This is the name of the local queue manager, that is, the name of the queue manager to which the application is connected.

The first 12 characters of the name are used to construct a unique message identifier (see the *MDMID* field described in “MQMD (Message descriptor) on IBM i” on page 3118). Queue managers that can intercommunicate must therefore have names that differ in the first 12 characters, in order for message identifiers to be unique in the queue manager network.

To determine the value of this attribute, use the CAQMN selector with the MQINQ call. The length of this attribute is given by LNQMN.

RemoteEvent (10-digit signed integer) on IBM i: 

Controls whether remote error events are generated.

The value is one of the following:

EVRDIS
Event reporting disabled.

EVRENA
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the IARMTE selector with the MQINQ call.

RepositoryName (48-byte character string) on IBM i: 

Name of cluster for which this queue manager provides repository services.

This is the name of a cluster for which this queue manager provides a repository-manager service. If the queue manager provides this service for more than one cluster, *RepositoryNamelist* specifies the name of a namelist object that identifies the clusters, and *RepositoryName* is blank. At least one of *RepositoryName* and *RepositoryNamelist* must be blank.

To determine the value of this attribute, use the CARPN selector with the MQINQ call. The length of this attribute is given by LNQMN.

RepositoryNamelist (48-byte character string) on IBM i: 

Name of namelist object containing names of clusters for which this queue manager provides repository services.

This is the name of a namelist object that contains the names of clusters for which this queue manager provides a repository-manager service. If the queue manager provides this service for only one cluster, the namelist object contains only one name. Alternatively, *RepositoryName* can be used to specify the name of the cluster, in which case *RepositoryNamelist* is blank. At least one of *RepositoryName* and *RepositoryNamelist* must be blank.

To determine the value of this attribute, use the CARPNL selector with the MQINQ call. The length of this attribute is given by LNNLN.

SSLEvent (character string) on IBM i: IBM i

Determines whether TLS events are generated.

The value is one of the following:

- EVRENA (MQINQ/PCF/config event) ENABLED (MQSC): TLS events are generated (that is, the RC2371 event is generated).
- EVRDIS (MQINQ/PCF/config event) DISABLED (MQSC): TLS events are not generated. This is the queue manager's initial default value.

To determine the value of this attribute, use the IASSLE selector with the MQINQ call.

SSLKeyResetCount (integer) on IBM i: IBM i

Determines the total number of non-encrypted bytes that are sent and received within a TLS conversation, before the secret key is renegotiated. The number of bytes includes control information sent by the message channel agent (MCA).

This value is only used by TLS channel MCAs which initiate communication from this queue manager (that is, the sender channel MCA in a sender and receiver channel pairing).

If the value of this attribute is greater than 0, and channel heartbeats are enabled for a channel, the secret key is also renegotiated before data is sent or received following a channel heartbeat. The count of bytes until the next secret key renegotiation is reset after each successful renegotiation occurs.

The value can be in the range 0 through 999 999 999. A value of 0 for this attribute indicates that the secret key is never renegotiated. If you specify a TLS secret key reset count in the range 1 byte through 32 KB, TLS channels will use a secret key reset count of 32 KB. This is to avoid the processing cost of excessive key resets which would occur for small TLS secret key reset values.

When the SSL server is an IBM MQ queue manager, and both secret key reset and channel heartbeats are enabled, renegotiation occurs immediately after each channel heartbeat.

To determine the value of this attribute, use the IASSRC selector with the MQINQ call.

StartStopEvent (10-digit signed integer) on IBM i: IBM i

Controls whether start and stop events are generated.

This attribute can have one of the following values:

EVRDIS
: Event reporting disabled.

EVRENA
: Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the IASSE selector with the MQINQ call.

SyncPoint (10-digit signed integer) on IBM i: IBM i

Syncpoint availability.

This indicates whether the local queue manager supports units of work and syncpointing with the MQGET, MQPUT, and MQPUT1 calls.

SPAVL
Units of work and syncpointing available.

SPNAVL
Units of work and syncpointing not available.

To determine the value of this attribute, use the IASYNC selector with the MQINQ call.

TraceRouteRecording (10-digit signed integer) on IBM i: IBM i

This controls whether information about messages is recorded as they flow through a queue manager.

The value is one of the following:

- RECDD: no appending to trace route messages is allowed
- RECDQ: messages are put onto a fixed named queue
- RECDM: determine using message (this is the initial default setting)

To prevent the trace route message from remaining in the system, set an expiry value on it that is greater than zero, and specify the RODISC report option. To prevent report or reply messages remaining in the system, set the report option ROPDAE. For more information, see "Report options and message flags on IBM i" on page 3460.

To determine the value of this attribute, use the IATRGI selector with the MQINQ call.

TreeLifeTime (10-digit signed integer) on IBM i: IBM i

The lifetime, in seconds, of non-administrative topics.

Non-administrative topics are those created when an application publishes to, or subscribes as, a topic string that does not exist as an administrative node. When this non-administrative node no longer has any active subscriptions, this parameter determines how long the queue manager will wait before removing that node. Only non-administrative topics that are in use by a durable subscription remain after the queue manager is recycled.

Specify a value in the range 0 through 604 000. A value of 0 means that non-administrative topics are not removed by the queue manager. The queue manager's initial default value is 1800.

To determine the value of this attribute, use the IATRLFT selector with the MQINQ call.

TriggerInterval (10-digit signed integer) on IBM i: IBM i

Trigger-message interval.

This is a time interval (in milliseconds) used to restrict the number of trigger messages. This is relevant only when the *TriggerType* is TTFRST. In this case, trigger messages are normally generated only when a suitable message arrives on the queue, and the queue was previously empty. Under certain circumstances, however, an additional trigger message can be generated with TTFRST triggering even if the queue was not empty. These additional trigger messages are not generated more often than every *TriggerInterval* milliseconds.

For more information about triggering, see Triggering channels.

The value is in the range zero through 999 999 999. The default value is 999 999 999.

To determine the value of this attribute, use the IATRGI selector with the MQINQ call.

Applications

This information describes the sample programs delivered with IBM MQ for IBM i for RPG. Also, learn how to build executable applications from the programs you write.

Building your application:

The IBM i publications describe how to build executable applications from the programs you write. This topic describes the additional tasks, and the changes to the standard tasks, you must perform when building IBM MQ for IBM i applications to run under IBM i.

In addition to coding the MQI calls in your source code, you must add the appropriate language statements to include the IBM MQ for IBM i copy files for the RPG language. You should make yourself familiar with the contents of these files; their names, and a brief description of their contents are given in the following text.

IBM MQ copy files on IBM i: IBM i

IBM MQ for IBM i provides copy files to assist you with writing your applications in the RPG programming language. They are suitable for use with the WebSphere Development toolset (5722 WDS) ILE RPG 4 Compiler.

The copy files that IBM MQ for IBM i provides to assist with the writing of channel exits are described in Channel-exit programs for messaging channels.

The names of the IBM MQ for IBM i copy files for RPG have the prefix CMQ. They have a suffix of G or H. There are separate copy files containing the named constants, and one file for each of the structures. The copy files are listed in “Language considerations” on page 3015.

Note: For ILE RPG/400, they are supplied as members of file QRPGLESRC in library QMQM.

The structure declarations do not contain `DS` statements. This allows the application to declare a data structure (or a multiple-occurrence data structure) by coding the `DS` statement and using the `/COPY` statement to copy in the remainder of the declaration:

For ILE RPG/400 the statement is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7
D* Declare an MQMD data structure
D MQMD       DS
D/COPY CMQMDG
```

Preparing your programs to run:

To create an executable IBM MQ for IBM i application, you have to compile the source code you have written.

To do this for ILE RPG/400, you can use the typical IBM i commands, CRTRPGMOD and CRTPGM.

After creating your *MODULE, you need to specify `BNDSRVPGM(QMQM/LIBMQM)` in the CRTPGM command. This includes the various IBM MQ procedures in your program.

Make sure that the library containing the copy files (QMQM) is in the library list when you perform the compilation.

For further information concerning programming considerations, including client modes, see "Language considerations" on page 3015.

Interfaces to the IBM i external syncpoint manager:

IBM MQ for IBM i uses native IBM i commitment control as an external syncpoint coordinator.

See the *IBM i Programming: Backup and Recovery Guide* for more information about the commitment control capabilities of IBM i.

To start the IBM i commitment control facilities, use the STRCMTCTL system command. To end commitment control, use the ENDCMTCTL system command.

Note: The default value of *Commitment definition scope* is *ACTGRP. This must be defined as *JOB for IBM MQ for IBM i. For example:

```
STRCMTCTL LCKLVL(*ALL) CMTSCOPE(*JOB)
```

If you call MQPUT, MQPUT1, or MQGET, specifying PMSYP or GMSYP, after starting commitment control, IBM MQ for IBM i adds itself as an API commitment resource to the commitment definition. This is typically the first such call in a job. While there are any API commitment resources registered under a particular commitment definition, you cannot end commitment control for that definition.

IBM MQ for IBM i removes its registration as an API commitment resource when you disconnect from the queue manager, provided there are no pending MQI operations in the current unit of work.

If you disconnect from the queue manager while there are pending MQPUT, MQPUT1, or MQGET operations in the current unit of work, IBM MQ for IBM i remains registered as an API commitment resource so that it is notified of the next commit or rollback. When the next syncpoint is reached, IBM MQ commits or rolls back the changes as required. It is possible for an application to disconnect and reconnect to a queue manager during an active unit of work and perform further MQGET and MQPUT operations inside the same unit of work (this is a pending disconnect).

If you attempt to issue an ENDCMTCTL system command for that commitment definition, message `CPF8355` is issued, indicating that pending changes were active. This message also appears in the job log when the job ends. To avoid this, ensure that you commit or roll back all pending IBM MQ operations, and that you disconnect from the queue manager. Thus, using COMMIT or ROLLBACK commands before ENDCMTCTL should enable end-commitment control to complete successfully.

When IBM i commitment control is used as an external syncpoint coordinator, MQCMIT, MQBACK, and MQBEGIN calls might not be issued. Calls to these functions fail with the reason code RC2012.

To commit or roll back (that is, to back out) your unit of work, use one of the programming languages that supports the commitment control. For example:

- CL commands: COMMIT and ROLLBACK
- ILE C Programming Functions: _Rcommit and _Rrollback
- RPG/400: COMMIT and ROLBK
- COBOL/400®: COMMIT and ROLLBACK

Syncpoints in CICS for IBM i applications:

IBM MQ for IBM i participates in units of work with CICS. You can use the MQI within a CICS application to put and get messages inside the current unit of work.

You can use the EXEC CICS SYNCPOINT command to establish a syncpoint that includes the IBM MQ for IBM i operations. To back out all changes up to the previous syncpoint, you can use the EXEC CICS SYNCPOINT ROLLBACK command.

If you use MQPUT, MQPUT1, or MQGET with the PMSYP, or GMSYP , option set in a CICS application, you cannot log off CICS until IBM MQ for IBM i has removed its registration as an API commitment resource. Therefore, you should commit or back out any pending put or get operations before you disconnect from the queue manager. This will allow you to log off CICS.

Sample programs on IBM i:

This topic describes the sample programs delivered with IBM MQ for IBM i for RPG. The samples demonstrate typical uses of the Message Queue Interface (MQI).

The samples are not intended to demonstrate general programming techniques, so some error checking that you may want to include in a production program has been omitted. However, these samples are suitable for use as a base for your own message queuing programs.

The source code for all the samples is provided with the product; this source includes comments that explain the message queuing techniques demonstrated in the programs.

There is one set of ILE sample programs:

1. **Programs using prototyped calls to the MQI (static bound calls)**

   The source exists in QMQMSAMP/QRPGLESRC. The members are named `AMQ3xxx4`, where `xxx` indicates the sample function. Copy members exist in QMQM/QRPGLESRC. Each member name has a suffix of `G` or `H`.

Table 371 gives a complete list of the sample programs delivered with IBM MQ for IBM i, and shows the names of the programs in each of the supported programming languages. Notice that their names all start with the prefix AMQ, the fourth character in the name indicates the programming language.

Table 371. Names of the sample programs

	RPG (ILE)
Put samples	AMQ3PUT4
Browse samples	AMQ3GBR4
Get samples	AMQ3GET4
Request samples	AMQ3REQ4
Echo samples	AMQ3ECH4
Inquire samples	AMQ3INQ4
Set samples	AMQ3SET4
Trigger Monitor sample	AMQ3TRG4

Table 371. Names of the sample programs (continued)

	RPG (ILE)
Trigger Server sample	AMQ3SRV4

In addition to these, the IBM MQ for IBM i sample option includes a sample data file, AMQSDATA, which can be used as input to certain sample programs and sample CL programs that demonstrate administration tasks. The CL samples are described in Administering IBM i . You could use the sample CL program to create queues to use with the sample programs described in this topic.

For information about how to run the sample programs, see "Preparing and running the sample programs on IBM i" on page 3442.

Features demonstrated in the sample programs on IBM i:

A table that shows the techniques demonstrated by the IBM MQ for IBM i sample programs.

Some techniques occur in more than one sample program, but only one program is listed in the table. All the samples open and close queues using the MQOPEN and MQCLOSE calls, so these techniques are not listed separately in the table.

Table 372. Sample programs demonstrating use of the MQI

Technique	**RPG (ILE)**
Using the MQCONN and MQDISC calls	AMQ3ECH4 or AMQ3INQ4
Implicitly connecting and disconnecting	AMQ3PUT4
Putting messages using the MQPUT call	AMQ3PUT4
Putting a single message using the MQPUT1 call	AMQ3ECH4 or AMQ3INQ4
Replying to a request message	AMQ3INQ4
Getting messages (no wait)	AMQ3GBR4
Getting messages (wait with a time limit)	AMQ3GET4
Getting messages (with data conversion)	AMQ3ECH4
Browsing a queue	AMQ3GBR4
Using a shared input queue	AMQ3INQ4
Using an exclusive input queue	AMQ3REQ4
Using the MQINQ call	AMQ3INQ4
Using the MQSET call	AMQ3SET4
Using a reply-to queue	AMQ3REQ4
Requesting exception messages	AMQ3REQ4
Accepting a truncated message	AMQ3GBR4
Using a resolved queue name	AMQ3GBR4
Trigger processing	AMQ3SRV4 or AMQ3TRG4

Note: All the sample programs produce a spool file that contains the results of the processing.

Preparing and running the sample programs on IBM i:

Before you can run the IBM MQ for IBM i sample programs, you must compile them as you would any other IBM MQ for IBM i applications. To do so, you can use the IBM i commands CRTRPGMOD and CRTPGM.

When you create the `AMQ3xxx4` programs, you must specify BNDSRVPGM(QMQM/LIBMQM) in the CRTPGM command. Doing so includes the various IBM MQ procedures in your program.

The sample programs are provided in library QMQMSAMP as members of QRPGLESRC. They use the copy files provided in library QMQM, so make sure that this library is in the library list when you compile them. The RPG compiler gives information messages because the samples do not use many of the variables that are declared in the copy files.

Running the sample programs

You can use your own queues when you run the samples, or you can compile and run AMQSAMP4 to create some sample queues. The source for this program is shipped in file QCLSRC in library QMQMSAMP. It can be compiled using the CRTCLPGM command.

To call one of the sample programs, use a command like:

```
CALL PGM(QMQMSAMP/AMQ3PUT4) PARM('Queue_Name','Queue_Manager_Name')
```

Where `Queue_Name` and `Queue_Manager_Name` must be 48 characters in length, which you achieve by padding the `Queue_Name` and `Queue_Manager_Name` with the required number of blanks.

For the Inquire and Set sample programs, the sample definitions created by AMQSAMP4 cause the C versions of these samples to be triggered. If you want to trigger the RPG versions, you must change the process definitions SYSTEM.SAMPLE.ECHOPROCESS and SYSTEM.SAMPLE.INQPROCESS and SYSTEM.SAMPLE.SETPROCESS. You can use the CHGMQMPRC command (described in Change MQ Process (CHGMQMPRC)) to do so, or edit and run AMQSAMP4 with the alternative definition.

The Put sample program on IBM i:

The Put sample program, AMQ3PUT4, puts messages on a queue using the MQPUT call.

To start the program, call the program and give the name of your target queue as a program parameter. The program puts a set of fixed messages on the queue; these messages are taken from the data block at the end of the program source code. A sample put program is AMQ3PUT4 in library QMQMSAMP.

Using this example program, the command is:

```
CALL PGM(QMQMSAMP/AMQ3PUT4) PARM('Queue_Name','Queue_Manager_Name')
```

Where `Queue_Name` and `Queue_Manager_Name` must be 48 characters in length, which you achieve by padding the `Queue_Name` and `Queue_Manager_Name` with the required number of blanks.

Design of the Put sample program

The program uses the MQOPEN call with the OOOUT option to open the target queue for putting messages. The results are output to a spool file. If it cannot open the queue, the program writes an error message containing the reason code returned by the MQOPEN call. To keep the program simple, on this and on subsequent MQI calls, the program uses default values for many of the options.

For each line of data contained in the source code, the program reads the text into a buffer and uses the MQPUT call to create a datagram message containing the text of that line. The program continues until either it reaches the end of the input or the MQPUT call fails. If the program reaches the end of the

input, it closes the queue using the MQCLOSE call.

The Browse sample program on IBM i:

The Browse sample program, AMQ3GBR4, browses messages on a queue using the MQGET call.

The program retrieves copies of all the messages on the queue you specify when you call the program; the messages remain on the queue. You could use the supplied queue SYSTEM.SAMPLE.LOCAL; run the Put sample program first to put some messages on the queue. You could use the queue SYSTEM.SAMPLE.ALIAS, which is an alias name for the same local queue. The program continues until it reaches the end of the queue or an MQI call fails.

An example of a command to call the RPG program is:

```
CALL PGM(QMQMSAMP/AMQ3GBR4) PARM('Queue_Name','Queue_Manager_Name')
```

Where `Queue_Name` and `Queue_Manager_Name` must be 48 characters in length, which you achieve by padding the `Queue_Name` and `Queue_Manager_Name` with the required number of blanks. Therefore, if you are using SYSTEM.SAMPLE.LOCAL as your target queue, you will need 29 blank characters.

Design of the Browse sample program

The program opens the target queue using the MQOPEN call with the OOBRW option. If it cannot open the queue, the program writes an error message to its spool file, containing the reason code returned by the MQOPEN call.

For each message on the queue, the program uses the MQGET call to copy the message from the queue, then displays the data contained in the message. The MQGET call uses these options:

GMBRWN
: After the MQOPEN call, the browse cursor is positioned logically before the first message in the queue, so this option causes the *first* message to be returned when the call is first made.

GMNWT
: The program does not wait if there are no messages on the queue.

GMATM
: The MQGET call specifies a buffer of fixed size. If a message is longer than this buffer, the program displays the truncated message, together with a warning that the message has been truncated.

The program demonstrates how you must clear the *MDMID* and *MDCID* fields of the MQMD structure after each MQGET call because the call sets these fields to the values contained in the message it retrieves. Clearing these fields means that successive MQGET calls retrieve messages in the order in which the messages are held in the queue.

The program continues to the end of the queue; here, the MQGET call returns the RC2033 (no message available) reason code and the program displays a warning message. If the MQGET call fails, the program writes an error message that contains the reason code in its spool file.

The program then closes the queue using the MQCLOSE call.

The Get sample program on IBM i:

The Get sample program, AMQ3GET4, gets messages from a queue using the MQGET call.

When the program is called, it removes messages from the specified queue. You could use the supplied queue SYSTEM.SAMPLE.LOCAL; run the Put sample program first to put some messages on the queue. You could use the SYSTEM.SAMPLE.ALIAS queue, which is an alias name for the same local queue. The program continues until the queue is empty or an MQI call fails.

An example of a command to call the RPG program is:

```
CALL PGM(QMQMSAMP/AMQ3GET4) PARM('Queue_Name','Queue_Manager_Name')
```

where `Queue_Name` and `Queue_Manager_Name` must be 48 characters in length, which you achieve by padding the `Queue_Name` and `Queue_Manager_Name` with the required number of blanks. Therefore, if you are using SYSTEM.SAMPLE.LOCAL as your target queue, you will need 29 blank characters.

Design of the Get sample program

The program opens the target queue for getting messages; it uses the MQOPEN call with the OOINPQ option. If it cannot open the queue, the program writes an error message containing the reason code returned by the MQOPEN call in its spool file.

For each message on the queue, the program uses the MQGET call to remove the message from the queue; it then displays the data contained in the message. The MQGET call uses the GMWT option, specifying a wait interval (*GMWI*) of 15 seconds, so that the program waits for this period if there is no message on the queue. If no message arrives before this interval expires, the call fails and returns the RC2033 (no message available) reason code.

The program demonstrates how you must clear the *MDMID* and *MDCID* fields of the MQMD structure after each MQGET call because the call sets these fields to the values contained in the message it retrieves. Clearing these fields means that successive MQGET calls retrieve messages in the order in which the messages are held in the queue.

The MQGET call specifies a buffer of fixed size. If a message is longer than this buffer, the call fails and the program stops.

The program continues until either the MQGET call returns the RC2033 (no message available) reason code or the MQGET call fails. If the call fails, the program displays an error message that contains the reason code.

The program then closes the queue using the MQCLOSE call.

The Request sample program on IBM i:

The Request sample program, AMQ3REQ4, demonstrates client/server processing. The sample is the client that puts request messages on a queue that is processed by a server program. It waits for the server program to put a reply message on a reply-to queue.

The Request sample puts a series of request messages on a queue using the MQPUT call. These messages specify SYSTEM.SAMPLE.REPLY as the reply-to queue. The program waits for reply messages, then displays them. Replies are sent only if the target queue (which we will call the *server queue*) is being processed by a server application, or if an application is triggered for that purpose (the Inquire and Set sample programs are designed to be triggered). The sample waits 5 minutes for the first reply to arrive (to allow time for a server application to be triggered) and 15 seconds for subsequent replies, but it can end without getting any replies.

To start the program, call the program and give the name of your target queue as a program parameter. The program puts a set of fixed messages on the queue; these messages are taken from the data block at the end of the program source code.

Design of the Request sample program

The program opens the server queue so that it can put messages. It uses the MQOPEN call with the OOOUT option. If it cannot open the queue, the program displays an error message containing the reason code returned by the MQOPEN call.

The program then opens the reply-to queue called SYSTEM.SAMPLE.REPLY so that it can get reply messages. For this, the program uses the MQOPEN call with the OOINPX option. If it cannot open the queue, the program displays an error message containing the reason code returned by the MQOPEN call.

For each line of input, the program then reads the text into a buffer and uses the MQPUT call to create a request message containing the text of that line. On this call the program uses the ROEXCD report option to request that any report messages sent about the request message will include the first 100 bytes of the message data. The program continues until either it reaches the end of the input or the MQPUT call fails.

The program then uses the MQGET call to remove reply messages from the queue, and displays the data contained in the replies. The MQGET call uses the GMWT option, specifying a wait interval (*GMWI*) of 5 minutes for the first reply (to allow time for a server application to be triggered) and 15 seconds for subsequent replies. The program waits for these periods if there is no message on the queue. If no message arrives before this interval expires, the call fails and returns the RC2033 (no message available) reason code. The call also uses the GMATM option, so messages longer than the declared buffer size are truncated.

The program demonstrates how you must clear the *MDMID* and *MDCOD* fields of the MQMD structure after each MQGET call because the call sets these fields to the values contained in the message it retrieves. Clearing these fields means that successive MQGET calls retrieve messages in the order in which the messages are held in the queue.

The program continues until either the MQGET call returns the RC2033 (no message available) reason code or the MQGET call fails. If the call fails, the program displays an error message that contains the reason code.

The program then closes both the server queue and the reply-to queue using the MQCLOSE call. Table 373 on page 3446 shows the changes to the Echo sample program that are necessary to run the Inquire and Set sample programs.

Note: The details for the Echo sample program are included as a reference.

Table 373. Client/Server sample program details

Program name	SYSTEM/SAMPLE queue	Program started
Echo	ECHO	AMQ3ECH4
Inquire	INQ	AMQ3INQ4
Set	SET	AMQ3SET4

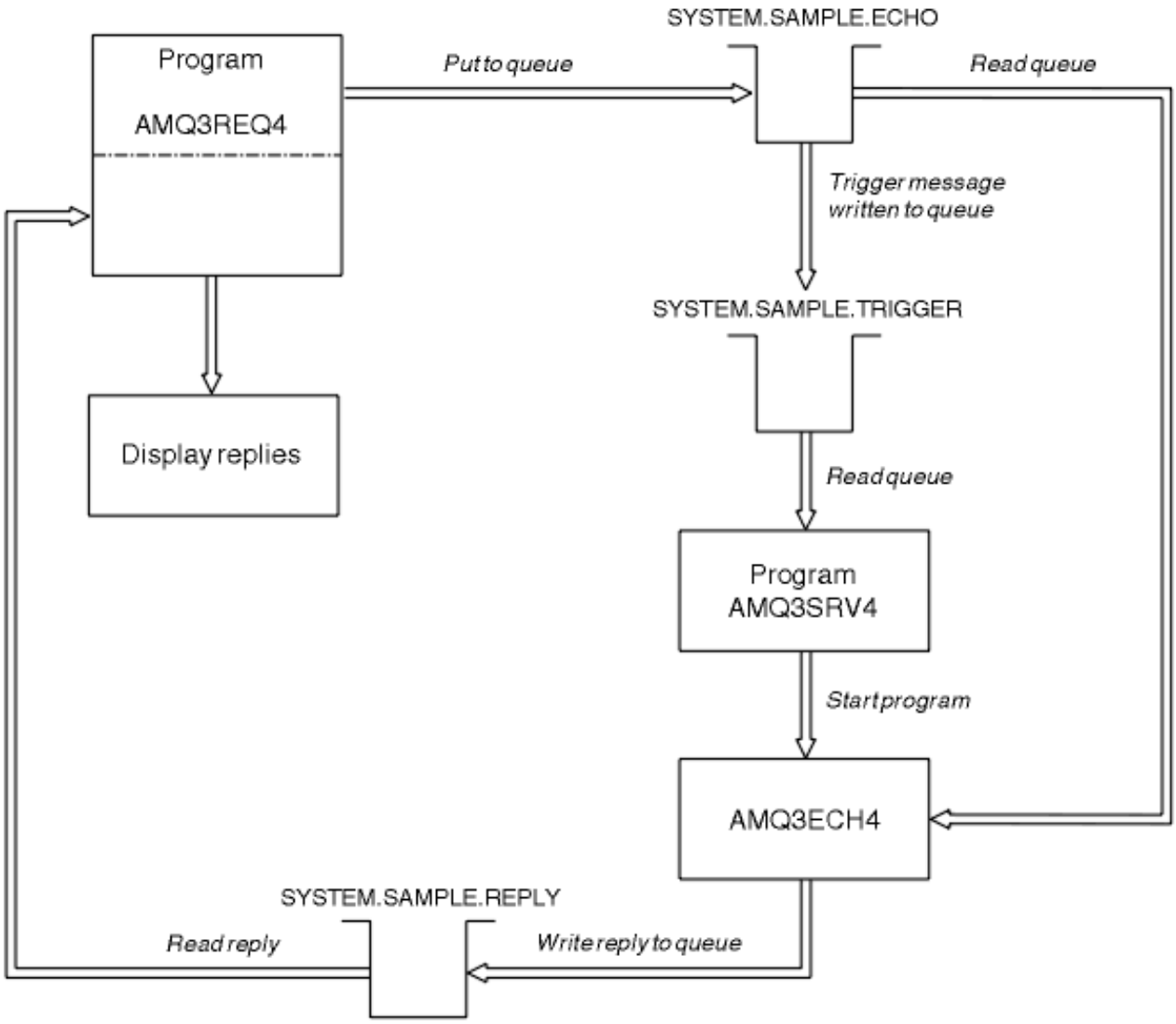


Figure 61. Sample Client/Server (Echo) program flowchart

Using triggering with the Request sample on IBM i: IBM i

To run the sample using triggering, start the trigger server program, AMQ3SRV4, against the required initiation queue in one job, then start AMQ3REQ4 in another job.

This means that the trigger server is ready when the Request sample program sends a message.

Note:

1. The samples use the SYSTEM SAMPLE TRIGGER queue as the initiation queue for SYSTEM.SAMPLE.ECHO, SYSTEM.SAMPLE.INQ, or SYSTEM.SAMPLE.SET local queues. Alternatively, you can define your own initiation queue.
2. The sample definitions created by AMQSAMP4 cause the C version of the sample to be triggered. If you want to trigger the RPG version, you must change the process definitions SYSTEM.SAMPLE.ECHOPROCESS and SYSTEM.SAMPLE.INQPROCESS and SYSTEM.SAMPLE.SETPROCESS. You can use the CHGMQMPRC command (see Change MQ Process (CHGMQMPRC) for more details) to do this, or edit and run your own version of AMQSAMP4.
3. You must compile the trigger server program from the source provided in QMQMSAMP/QRPGLESRC.

Depending on the trigger process you want to run, AMQ3REQ4 should be called with the parameter specifying request messages to be placed on one of these sample server queues:

- SYSTEM.SAMPLE.ECHO (for the Echo sample programs)
- SYSTEM.SAMPLE.INQ (for the Inquire sample programs)
- SYSTEM.SAMPLE.SET (for the Set sample programs)

A flow chart for the SYSTEM.SAMPLE.ECHO program is shown in Figure 61 on page 3446. Using the example the command to issue the RPG program request to this server is:

```
CALL PGM(QMQMSAMP/AMQ3REQ4) PARM('SYSTEM.SAMPLE.ECHO
+ 30 blank characters','Queue_Manager_Name')
```

because the queue name and queue manager name must be 48 characters in length.

Note: This sample queue has a trigger type of FIRST, so if there are already messages on the queue before you run the Request sample, server applications are not triggered by the messages you send.

If you want to attempt further examples, you can try the following variations:

- Use AMQ3TRG4 instead of AMQ3SRV4 to submit the job instead, but potential job submission delays could make it less easy to follow what is happening.
- Use the SYSTEM.SAMPLE.INQ and SYSTEM.SAMPLE.SET sample queues. Using the example data file, the commands to issue the RPG program requests to these servers are:

  ```
  CALL PGM(QMQMSAMP/AMQ3INQ4) PARM('SYSTEM.SAMPLE.INQ
  + 31 blank characters')
  CALL PGM(QMQMSAMP/AMQ3SET4) PARM('SYSTEM.SAMPLE.SET
  + 31 blank characters')
  ```

  because the queue name must be 48 characters in length.

  These sample queues also have a trigger type of FIRST.

The Echo sample program on IBM i:

The Echo sample programs return the message send to a reply queue. The program is named AMQ3ECH4

For the triggering process to work, you must ensure that the Echo sample program you want to use is triggered by messages arriving on queue SYSTEM.SAMPLE.ECHO. To do this, specify the name of the Echo sample program you want to use in the *ApplId* field of the process definition SYSTEM.SAMPLE.ECHOPROCESS. (For this, you can use the CHGMQMPRC command, described in Administering IBM i .) The sample queue has a trigger type of FIRST, so if there are already messages on the queue before you run the Request sample, the Echo sample is not triggered by the messages you send.

When you have set the definition correctly, first start AMQ3SRV4 in one job, then start AMQ3REQ4 in another. You could use AMQ3TRG4 instead of AMQ3SRV4, but potential job submission delays could make it less easy to follow what is happening.

Use the Request sample programs to send messages to queue SYSTEM.SAMPLE.ECHO. The Echo sample programs send a reply message containing the data in the request message to the reply-to queue specified in the request message.

Design of the Echo sample program

When the program is triggered, it explicitly connects to the default queue manager using the MQCONN call. Although this is not necessary on IBM i, this means you could use the same program on other platforms without changing the source code.

The program then opens the queue named in the trigger message structure it was passed when it started. (For clarity, we will call this the *request queue*.) The program uses the MQOPEN call to open this queue for shared input.

The program uses the MQGET call to remove messages from this queue. This call uses the GMATM and GMWT options, with a wait interval of 5 seconds. The program tests the descriptor of each message to see if it is a request message; if it is not, the program discards the message and displays a warning message.

For each request message removed from the request queue, the program uses the MQPUT call to put a reply message on the reply-to queue. This message contains the contents of the request message.

When there are no messages remaining on the request queue, the program closes that queue and disconnects from the queue manager.

This program can also respond to messages sent to the queue from platforms other than IBM i, although no sample is supplied for this situation. To make the ECHO program work, you:

- Write a program, correctly specifying the *Format*, *Encoding*, and *CCSID* fields, to send text request messages.

  The ECHO program requests the queue manager to perform message data conversion, if this is needed.
- Specify CONVERT(*YES) on the IBM MQ for IBM i sending channel, if the program you have written does not provide similar conversion for the reply.

The Inquire sample program on IBM i:

The Inquire sample program, AMQ3INQ4, inquires about some of the attributes of a queue using the MQINQ call.

The program is intended to run as a triggered program, so its only input is an MQTMC (trigger message) structure. This structure contains the name of a target queue with attributes that are to be inquired upon.

For the triggering process to work, you must ensure that the Inquire sample program is triggered by messages arriving on queue SYSTEM.SAMPLE.INQ. To do ao, specify the name of the Inquire sample program in the *ApplId* field of the SYSTEM.SAMPLE.INQPROCESS process definition. (For this, you can use the CHGMQMPRC command, described in Change MQ Process (CHGMQMPRC)). The sample queue has a trigger type of FIRST, so if there are already messages on the queue before you run the Request sample, the Inquire sample is not triggered by the messages you send.

When you have set the definition correctly, first start AMQ3SRV4 in one job, then start AMQ3REQ4 in another. You could use AMQ3TRG4 instead of AMQ3SRV4, but potential job submission delays might make it less easy to follow what is happening.

Use the Request sample program to send request messages, each containing just a queue name, to queue SYSTEM.SAMPLE.INQ. For each request message, the Inquire sample program sends a reply message containing information about the queue specified in the request message. The replies are sent to the reply-to queue specified in the request message.

Design of the Inquire sample program

When the program is triggered, it explicitly connects to the default queue manager using the MQCONN call. Although not necessary on IBM i, this design feature means you could use the same program on other platforms without changing the source code.

The program then opens the queue named in the trigger message structure it was passed when it started. (For clarity, we will call this the *request queue*.) The program uses the MQOPEN call to open this queue for shared input.

The program uses the MQGET call to remove messages from this queue. This call uses the GMATM and GMWT options, with a wait interval of 5 seconds. The program tests the descriptor of each message to see if it is a request message; if it is not, the program discards the message, and displays a warning message.

For each request message removed from the request queue, the program reads the name of the queue (which we will call the *target queue*) contained in the data and opens that queue using the MQOPEN call with the OOINQ option. The program then uses the MQINQ call to inquire about the values of the **InhibitGet**, **CurrentQDepth**, and **OpenInputCount** attributes of the target queue.

If the MQINQ call is successful, the program uses the MQPUT call to put a reply message on the reply-to queue. This message contains the values of the three attributes.

If the MQOPEN or MQINQ call is unsuccessful, the program uses the MQPUT call to put a *report* message on the reply-to queue. In the *MDFB* field of the message descriptor of this report message is the reason code returned by either the MQOPEN or MQINQ call, depending on which one failed.

After the MQINQ call, the program closes the target queue using the MQCLOSE call.

When there are no messages remaining on the request queue, the program closes that queue and disconnects from the queue manager.

The Set sample program on IBM i:

The Set sample program, AMQ3SET4, inhibits put operations on a queue by using the MQSET call to change the queue's **InhibitPut** attribute.

The program is intended to run as a triggered program, so its only input is an MQTMC (trigger message) structure that contains the name of a target queue with attributes that are to be inquired upon.

For the triggering process to work, you must ensure that the Set sample program is triggered by messages arriving on queue SYSTEM.SAMPLE.SET. To do this, specify the name of the Set sample program in the *ApplId* field of the process definition SYSTEM.SAMPLE.SETPROCESS. (For this, you can use the CHGMQMPRC command, described in the Administering IBM i .) The sample queue has a trigger type of FIRST, so if there are already messages on the queue before you run the Request sample, the Set sample is not triggered by the messages you send.

When you have set the definition correctly, first start AMQ3SRV4 in one job, then start AMQ3REQ4 in another. You could use AMQ3TRG4 instead of AMQ3SRV4, but potential job submission delays could make it less easy to follow what is happening.

Use the Request sample program to send request messages, each containing just a queue name, to queue SYSTEM.SAMPLE.SET. For each request message, the Set sample program sends a reply message containing a confirmation that put operations have been inhibited on the specified queue. The replies are sent to the reply-to queue specified in the request message.

Design of the Set sample program

When the program is triggered, it explicitly connects to the default queue manager using the MQCONN call. Although not necessary on IBM i, this means you could use the same program on other platforms without changing the source code.

The program then opens the queue named in the trigger message structure it was passed when it started. (For clarity, we will call this the *request queue*.) The program uses the MQOPEN call to open this queue for shared input.

The program uses the MQGET call to remove messages from this queue. This call uses the GMATM and GMWT options, with a wait interval of 5 seconds. The program tests the descriptor of each message to see if it is a request message; if it is not, the program discards the message and displays a warning message.

For each request message removed from the request queue, the program reads the name of the queue (which we will call the *target queue*) contained in the data and opens that queue using the MQOPEN call with the OOSET option. The program then uses the MQSET call to set the value of the **InhibitPut** attribute of the target queue to QAPUTI.

If the MQSET call is successful, the program uses the MQPUT call to put a reply message on the reply-to queue. This message contains the string `PUT inhibited`.

If the MQOPEN or MQSET call is unsuccessful, the program uses the MQPUT call to put a *report* message on the reply-to queue. In the *MDFB* field of the message descriptor of this report message is the reason code returned by either the MQOPEN or MQSET call, depending on which one failed.

After the MQSET call, the program closes the target queue using the MQCLOSE call.

When there are no messages remaining on the request queue, the program closes that queue and disconnects from the queue manager.

The Triggering sample programs on IBM i:

IBM MQ for IBM i supplies two Triggering sample programs that are written in ILE/RPG.

The programs are:

AMQ3TRG4
: This is a trigger monitor for the IBM i environment. It submits an IBM i job for the application to be started, but this means that there is additional processing cost associated with each trigger message.

AMQ3SRV4
: This is a trigger server for the IBM i environment. For each trigger message, this server runs the start command in its own job to start the specified application. The trigger server can call CICS transactions.

C language versions of these samples are also available as executable programs in library QMQM, called AMQSTRG4 and AMQSERV4.

The AMQ3TRG4 sample trigger monitor on IBM i:

AMQ3TRG4 is a trigger monitor. It takes one parameter: the name of the initiation queue it is to serve. AMQSAMP4 defines a sample initiation queue, SYSTEM.SAMPLE.TRIGGER, that you can use when you try the sample programs.

AMQ3TRG4 submits an IBM i job for each valid trigger message it gets from the initiation queue.

Design of the trigger monitor

The trigger monitor opens the initiation queue and gets messages from the queue, specifying an unlimited wait interval.

The trigger monitor submits an IBM i job to start the application specified in the trigger message, and passes an MQTMC (a character version of the trigger message) structure. The environment data in the trigger message is used as job submission parameters.

Finally, the program closes the initiation queue.

The AMQ3SRV4 sample trigger server:

AMQ3SRV4 is a trigger server. It takes one parameter: the name of the initiation queue it is to serve. AMQSAMP4 defines a sample initiation queue, SYSTEM.SAMPLE.TRIGGER, that you can use when you try the sample programs.

For each trigger message, AMQ3SRV4 runs a start command in its own job to start the specified application.

Using the example trigger queue the command to issue is:

```
CALL PGM(QMQM/AMQ3SRV4) PARM('Queue Name')
```

Where `Queue Name` must be 48 characters in length, which you achieve by padding the queue name with the required number of blanks. Therefore, if you are using SYSTEM.SAMPLE.TRIGGER as your target queue, you will need 28 blank characters.

Design of the trigger server

The design of the trigger server is like that of the trigger monitor, except the trigger server:

- Allows CICS as well as IBM i applications
- Does not use the environment data from the trigger message
- Calls IBM i applications in its own job (or uses STRCICSUSR to start CICS applications) rather than submitting an IBM i job
- Opens the initiation queue for shared input, so many trigger servers can run at the same time

Note: Programs started by AMQ3SRV4 must not use the MQDISC call because this will stop the trigger server. If programs started by AMQ3SRV4 use the MQCONN call, they will get the RC2002 reason code.

Ending the Triggering sample programs on IBM i:

A trigger monitor program can be ended by the sysrequest option 2 (ENDRQS) or by inhibiting gets from the trigger queue.

If the sample trigger queue is used the command is:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') GETENBL(*NO)
```

Note: To start triggering again on this queue, you must enter the command:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') GETENBL(*YES)
```

Running the samples using remote queues on IBM i:

You can demonstrate remote queuing by running the samples on connected message queue managers.

Program AMQSAMP4 provides a local definition of a remote queue (SYSTEM.SAMPLE.REMOTE) that uses a remote queue manager named `OTHER`. To use this sample definition, change `OTHER` to the name of the second message queue manager you want to use. You must also set up a message channel between your two message queue managers; for information about how to do so, see Channel-exit programs for messaging channels.

The Request sample program puts its own local queue manager name in the *MDRM* field of messages it sends. The Inquire and Set samples send reply messages to the queue and message queue manager named in the *MDRQ* and *MDRM* fields of the request messages they process.

Return codes for IBM i (ILE RPG)

This information describes the return codes associated with the MQI and MQAI.

The return codes associated with:

- Programmable Command Format (PCF) commands are listed in Programmable command formats reference.
- C++ calls are listed in Using C++.

For each call, a completion code and a reason code are returned by the queue manager or by an exit routine, to indicate the success or failure of the call.

Applications must not depend upon errors being checked for in a specific order, except where specifically noted. If more than one completion code or reason code could arise from a call, the particular error reported depends on the implementation.

Completion codes for IBM i (ILE RPG):

The completion code parameter (*CMPCOD*) allows the caller to see quickly whether the call completed successfully, completed partially, or failed.

CCOK
(MQCC_OK on other platforms)

Successful completion.

The call completed fully; all output parameters have been set. The **REASON** parameter always has the value RCNONE in this case.

CCWARN
(MQCC_WARN on other platforms)

Warning (partial completion).

The call completed partially. Some output parameters might have been set in addition to the *CMPCOD* and *REASON* output parameters. The **REASON** parameter gives additional information about the partial completion.

CCFAIL
(MQCC_FAIL on other platforms)

Call failed.

The processing of the call did not complete, and the state of the queue manager is normally unchanged; exceptions are specifically noted. The *CMPCOD* and *REASON* output parameters have been set; other parameters are unchanged, except where noted.

The reason might be a fault in the application program, or it might be a result of some situation external to the program, for example the user's authority might have been revoked. The **REASON** parameter gives additional information about the error.

Reason codes for IBM i (ILE RPG):

The reason code parameter (*REASON*) is a qualification to the completion code parameter (*CMPCOD*).

If there is no special reason to report, RCNONE is returned. A successful call returns CCOK and RCNONE.

If the completion code is either CCWARN or CCFAIL, the queue manager always reports a qualifying reason; details are given under each call description.

Where user exit routines set completion codes and reasons, they should adhere to these rules. In addition, any special reason values defined by user exits should be less than zero, to ensure that they do not conflict with values defined by the queue manager. Exits can set reasons already defined by the queue manager, where these are appropriate.

Reason codes also occur in:

- The *DLREA* field of the MQDLH structure
- The *MDFB* field of the MQMD structure

The full list of reason codes is in API completion and reason codes in .

To find your IBM i reason code in that list, remove the "RC" from the front, for example `RC2002` becomes `2002`. Also the completion codes there are shown as they are on other platforms:

Table 374.

IBM i	Other platforms
CCOK	MQCC_OK
CCWARN	MQCC_WARN
CCFAIL	MQCC_FAIL

Rules for validating MQI options for IBM i (ILE RPG)

This topic gives information about the situations that produce an RC2046 reason code from an MQOPEN, MQPUT, MQPUT1, MQGET, or MQCLOSE call.

MQOPEN call on IBM i:

For the options of the MQOPEN call:

- *At least one* of the following must be specified:
  - OOBRW
  - OOINPQ
  - OOINPX
  - OOINPS
  - OOINQ
  - OOOUT
  - OOSET
- Only *one* of the following is allowed:
  - OOINPQ
  - OOINPX
  - OOINPS
- Only *one* of the following is allowed:
  - OOBNDO
  - OOBNDN
  - OOBNDQ

  Note: The options listed previously are mutually exclusive. However, because the value of OOBNDQ is zero, specifying it with either of the other two bind options does not result in reason code RC2046. OOBNDQ is provided to aid program documentation.
- If OOSAVA is specified, one of the OOINP* options must also be specified.
- If one of the OOSET* or OOPAS* options is specified, OOOUT must also be specified.

MQPUT call on IBM i:

For the put-message options:

- The combination of PMSYP and PMNSYP is not allowed.
- Only *one* of the following is allowed:
  - PMDEFC
  - PMNOC
  - PMPASA
  - PMPASI
  - PMSETA
  - PMSETI
- PMALTU is not allowed (it is valid only on the MQPUT1 call).

MQPUT1 call on IBM i:

For the put-message options, the rules are the same as for the MQPUT call, except for the following options:

- PMALTU is allowed.
- PMLOGO is not allowed.

MQGET call on IBM i:

For the get-message options:

- Only *one* of the following options is allowed:
  - GMNSYP
  - GMSYP
  - GMPSYP
- Only *one* of the following options is allowed:
  - GMBRWF
  - GMBRWC
  - GMBRWN
  - GMMUC
- GMSYP is not allowed with any of the following options:
  - GMBRWF
  - GMBRWC
  - GMBRWN
  - GMLK
  - GMUNLK
- GMPSYP is not allowed with any of the following options:
  - GMBRWF
  - GMBRWC
  - GMBRWN
  - GMCMPM
  - GMUNLK
- If GMLK is specified, one of the following options must also be specified:
  - GMBRWF
  - GMBRWC

  - GMBRWN
- If GMUNLK is specified, only the following options are allowed:
  - GMNSYP
  - GMNWT

MQCLOSE call on IBM i:
- For the options of the MQCLOSE call. The combination of CODEL and COPURG is not allowed.
- Only one of the following is allowed:
  - COKPSB
  - CORMSB

MQSUB call on IBM i:
For the options of the MQSUB call:
- At least one of the following must be specified:
- At least one of the following must be specified:
  - SOALT
  - SORES
  - SOCRT
- Only one of the following is allowed:
  - SODUR
  - SONDUR

  Note: The options listed previously are mutually exclusive. However, as the value of SOnDUR is zero, specifying it with SODUR does not result in reason code RC2046. SONDUR is provided to aid program documentation.
- The combination of SOGRP and SOMAN is not allowed.
- SOGRP requires SOSCID to be specified.
- Only one of the following is allowed: SOAUID SOFUID
- The combination of SONEWP and SOPUBR is not allowed.
- SONEWP is only allowed in combination with SOCRT.
- Only one of the following is allowed:
  - SOWCHR
  - SOWTOP

Machine encodings on IBM i

Use this information to learn about the structure of the *MDENC* field in the message descriptor.

For more information about the message descriptor, see "MQMD (Message descriptor) on IBM i" on page 3118.

The *MDENC* field is a 32-bit integer that is divided into four separate subfields; these subfields identify:

- The encoding used for binary integers
- The encoding used for packed-decimal integers
- The encoding used for floating-point numbers
- Reserved bits

Each subfield is identified by a bit mask which has 1-bits in the positions corresponding to the subfield, and 0-bits elsewhere. The bits are numbered such that bit 0 is the most significant bit, and bit 31 the least significant bit. The following masks are defined:

ENIMSK
Mask for binary-integer encoding.

This subfield occupies bit positions 28 through 31 within the *MDENC* field.

ENDMSK
Mask for packed-decimal-integer encoding.

This subfield occupies bit positions 24 through 27 within the *MDENC* field.

ENFMSK
Mask for floating-point encoding.

This subfield occupies bit positions 20 through 23 within the *MDENC* field.

ENRMSK
Mask for reserved bits.

This subfield occupies bit positions 0 through 19 within the *MDENC* field.

Binary-integer encoding on IBM i: IBM i

Valid values for binary-integer encoding.

The following values are valid for the binary-integer encoding:

ENIUND
Undefined integer encoding.

Binary integers are represented using an encoding that is undefined.

ENINOR
Normal integer encoding.

Binary integers are represented in the conventional way:

- The least significant byte in the number has the highest address of any of the bytes in the number; the most significant byte has the lowest address.
- The least significant bit in each byte is next to the byte with the next higher address; the most significant bit in each byte is next to the byte with the next lower address.

ENIREV
Reversed integer encoding.

Binary integers are represented in the same way as ENINOR, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as ENINOR.

Packed-decimal-integer encoding on IBM i: IBM i

Valid values for packed-decimal-integer encoding

The following values are valid for the packed-decimal-integer encoding:

ENDUND
Undefined packed-decimal encoding.

Packed-decimal integers are represented using an encoding that is undefined.

ENDNOR
Normal packed-decimal encoding.

Packed-decimal integers are represented in the conventional way:

- Each decimal digit in the printable form of the number is represented in packed decimal by a single hexadecimal digit in the range X'0' through X'9'. Each hexadecimal digit occupies 4 bits, and so each byte in the packed decimal number represents two decimal digits in the printable form of the number.
- The least significant byte in the packed-decimal number is the byte which contains the least significant decimal digit. Within that byte, the most significant 4 bits contain the least significant decimal digit, and the least significant 4 bits contain the sign. The sign is either X'C' (positive), X'D' (negative), or X'F' (unsigned).
- The least significant byte in the number has the highest address of any of the bytes in the number; the most significant byte has the lowest address.
- The least significant bit in each byte is next to the byte with the next higher address; the most significant bit in each byte is next to the byte with the next lower address.

ENDREV
Reversed packed-decimal encoding.

Packed-decimal integers are represented in the same way as ENDNOR, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as ENDNOR.

Floating-point encoding on IBM i: IBM i

Valid values for floating-point encoding

The following values are valid for the floating-point encoding:

ENFUND
Undefined floating-point encoding.

Floating-point numbers are represented using an encoding that is undefined.

ENFNOR
Normal IEEE (The Institute of Electrical and Electronics Engineers) float encoding.

Floating-point numbers are represented using the standard IEEE floating-point format, with the bytes arranged as follows:

- The least significant byte in the mantissa has the highest address of any of the bytes in the number; the byte containing the exponent has the lowest address
- The least significant bit in each byte is next to the byte with the next higher address; the most significant bit in each byte is next to the byte with the next lower address

Details of the IEEE float encoding might be found in IEEE Standard 754.

ENFREV
Reversed IEEE float encoding.

Floating-point numbers are represented in the same way as ENFNOR, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as ENFNOR.

ENF390
System/390 architecture float encoding.

Floating-point numbers are represented using the standard System/390 floating-point format; this is also used by System/370.

Constructing encodings on IBM i: 

To construct a value for the *MDENC* field in MQMD, the relevant constants that describe the required encodings should be added.

Be sure to combine only one of the ENI* encodings with one of the END* encodings and one of the ENF* encodings.

Analyzing encodings on IBM i: 

The *MDENC* field contains subfields; because of this, applications that need to examine the integer, packed decimal, or float encoding should use the technique described in this topic.

Using arithmetic

The following steps should be performed using integer arithmetic:

1. Select one of the following values, according to the type of encoding required:
   - 1 for the binary integer encoding
   - 16 for the packed decimal integer encoding
   - 256 for the floating point encoding

   Call the value `A`.
2. Divide the value of the *MDENC* field by `A` ; call the result `B`.
3. Divide `B` by 16; call the result `C`.
4. Multiply `C` by 16 and subtract from `B` ; call the result `D`.
5. Multiply `D` by `A` ; call the result `E`.
6. `E` is the encoding required, and can be tested for equality with each of the values that is valid for that type of encoding.

Summary of machine architecture encodings on IBM i: IBM i

A table summarizing encodings for machine architectures.

Encodings for machine architectures are shown in Table 375.

Table 375. Summary of encodings for machine architectures

Machine architecture	Binary integer encoding	Packed-decimal integer encoding	Floating-point encoding
IBM i	normal	normal	IEEE normal
Intel x86	reversed	reversed	IEEE reversed
PowerPC	normal	normal	IEEE normal
System/390	normal	normal	System/390

Report options and message flags on IBM i

IBM i

This topic concerns the *MDREP* and *MDMFL* fields that are part of the message descriptor MQMD specified on the MQGET, MQPUT, and MQPUT1 calls.

For more information about the message descriptor, see "MQMD (Message descriptor) on IBM i" on page 3118. This information describes:

- The structure of the report field and how the queue manager processes it
- How an application should analyze the report field
- The structure of the message-flags field

Structure of the report field:

The *MDREP* field is a 32-bit integer that is divided into three separate subfields.

These subfields identify:

- Report options that are rejected if the local queue manager does not recognize them
- Report options that are always accepted, even if the local queue manager does not recognize them
- Report options that are accepted only if certain other conditions are satisfied

Each subfield is identified by a bit mask which has 1-bits in the positions corresponding to the subfield, and 0-bits elsewhere. Note that the bits in a subfield are not necessarily adjacent. The bits are numbered such that bit 0 is the most significant bit, and bit 31 the least significant bit. The following masks are defined to identify the subfields:

RORUM
Mask for unsupported report options that are rejected.

This mask identifies the bit positions within the *MDREP* field where report options which are not supported by the local queue manager will cause the MQPUT or MQPUT1 call to fail with completion code CCFAIL and reason code RC2061.

This subfield occupies bit positions 3, and 11 through 13.

ROAUM
Mask for unsupported report options that are accepted.

This mask identifies the bit positions within the *MDREP* field where report options which are not supported by the local queue manager will nevertheless be accepted on the MQPUT or MQPUT1 calls. Completion code CCWARN with reason code RC2104 are returned in this case.

This subfield occupies bit positions 0 through 2, 4 through 10, and 24 through 31.

The following report options are included in this subfield:

- ROCMTC
- RODLQ
- RODISC
- ROEXC
- ROEXCD
- ROEXCF
- ROEXP
- ROEXPD
- ROEXPF
- RONAN
- RONMI
- RONONE
- ROPAN
- ROPCI
- ROPMI

ROAUXM

Mask for unsupported report options that are accepted only in certain circumstances.

This mask identifies the bit positions within the *MDREP* field where report options which are not supported by the local queue manager will nevertheless be accepted on the MQPUT or MQPUT1 calls *provided* that both of the following conditions are satisfied:

- The message is destined for a remote queue manager.
- The application is not putting the message directly on a local transmission queue (that is, the queue identified by the *ODMN* and *ODON* fields in the object descriptor specified on the MQOPEN or MQPUT1 call is not a local transmission queue).

Completion code CCWARN with reason code RC2104 are returned if these conditions are satisfied, and CCFAIL with reason code RC2061 if not.

This subfield occupies bit positions 14 through 23.

The following report options are included in this subfield:

- ROCOA
- ROCOAD
- ROCOAF
- ROCOD
- ROCODD
- ROCODF

If there are any options specified in the *MDREP* field which the queue manager does not recognize, the queue manager checks each subfield in turn by using the bitwise AND operation to combine the *MDREP* field with the mask for that subfield. If the result of that operation is not zero, the completion code and reason codes described previously are returned.

If CCWARN is returned, it is not defined which reason code is returned if other warning conditions exist.

The ability to specify and have accepted report options which are not recognized by the local queue manager is useful when it is necessary to send a message with a report option which will be recognized and processed by a *remote* queue manager.

Analyzing the report field on IBM i: 

The MDREP field contains subfields. Because of this, some applications need to check whether the sender of the message requested a particular report. Those applications should use the technique described in this topic.

Using arithmetic

The following steps should be performed using integer arithmetic:

1. Select one of the following values, according to the type of report to be checked:
   - ROCOA for COA report
   - ROCOD for COD report
   - ROEXC for exception report
   - ROEXP for expiration report

   Call the value A.
2. Divide the *MDREP* field by A ; call the result B.
3. Divide B by 8 ; call the result C.
4. Multiply C by 8 and subtract from B ; call the result D.
5. Multiply D by A ; call the result E.
6. Test E for equality with each of the values that is possible for that type of report.

   For example, if A is ROEXC, test E for equality with each of the following to determine what was specified by the sender of the message:
   - RONONE
   - ROEXC
   - ROEXCD
   - ROEXCF

   The tests can be performed in whatever order is most convenient for the application logic.

The following pseudocode illustrates this technique for exception report messages:

```
A = ROEXC
B = Report/A
C = B/8
D = B - C*8
E = D*A
```

A similar method can be used to test for the ROPMI or ROPCI options; select as the value A whichever of these two constants is appropriate, and then proceed as described previously, but replacing the value 8 in the previous steps by the value 2.

Structure of the message-flags field on IBM i: 

The *MDMFL* field is a 32-bit integer that is divided into three separate subfields.

These subfields identify:

- Message flags that are rejected if the local queue manager does not recognize them
- Message flags that are always accepted, even if the local queue manager does not recognize them
- Message flags that are accepted only if certain other conditions are satisfied

Note: All subfields in *MDMFL* are reserved for use by the queue manager.

Each subfield is identified by a bit mask which has 1-bits in the positions corresponding to the subfield, and 0-bits elsewhere. The bits are numbered such that bit 0 is the most significant bit, and bit 31 the least significant bit. The following masks are defined to identify the subfields:

MFRUM
Mask for unsupported message flags that are rejected.

This mask identifies the bit positions within the *MDMFL* field where message flags which are not supported by the local queue manager will cause the MQPUT or MQPUT1 call to fail with completion code CCFAIL and reason code RC2249.

This subfield occupies bit positions 20 through 31.

The following message flags are included in this subfield:

- MFLMIG
- MFLSEG
- MFMIG
- MFSEG
- MFSEGA
- MFSEGI

MFAUM
Mask for unsupported message flags that are accepted.

This mask identifies the bit positions within the *MDMFL* field where message flags which are not supported by the local queue manager will nevertheless be accepted on the MQPUT or MQPUT1 calls. The completion code is CCOK.

This subfield occupies bit positions 0 through 11.

MFAUXM
Mask for unsupported message flags that are accepted only in certain circumstances.

This mask identifies the bit positions within the *MDMFL* field where message flags which are not supported by the local queue manager will nevertheless be accepted on the MQPUT or MQPUT1 calls *provided* that both of the following conditions are satisfied:

- The message is destined for a remote queue manager.
- The application is not putting the message directly on a local transmission queue (that is, the queue identified by the *ODMN* and *ODON* fields in the object descriptor specified on the MQOPEN or MQPUT1 call is not a local transmission queue).

Completion code CCOK is returned if these conditions are satisfied, and CCFAIL with reason code RC2249 if not.

This subfield occupies bit positions 12 through 19.

If there are flags specified in the *MDMFL* field that the queue manager does not recognize, the queue manager checks each subfield in turn by using the bitwise AND operation to combine the *MDMFL* field with the mask for that subfield. If the result of that operation is not zero, the completion code and reason codes described previously are returned.

Data conversion on IBM i

IBM i

This topic describes the interface to the data-conversion exit, and the processing performed by the queue manager when data conversion is required.

The data-conversion exit is invoked as part of the processing of the MQGET call. It is used to convert the application message data to the representation required by the receiving application. Conversion of the application message data is optional, and requires the GMCONV option to be specified on the MQGET call.

The following aspects of data conversion are described:

- The processing performed by the queue manager in response to the GMCONV option; see "Conversion processing on IBM i."
- Processing conventions used by the queue manager when processing a built-in format; these conventions are recommended for user-written exits too. See "Processing conventions on IBM i" on page 3466.
- Special considerations for the conversion of report messages; see "Conversion of report messages on IBM i" on page 3470.
- The parameters passed to the data-conversion exit; see "MQCONVX (Data conversion exit) on IBM i" on page 3481.
- A call that can be used from the exit in order to convert character data between different representations; see "MQXCNVC (Convert characters) on IBM i" on page 3476.
- The data-structure parameter which is specific to the exit; see "MQDXP (Data-conversion exit parameter) on IBM i" on page 3471.

Conversion processing on IBM i

IBM i

This information describes the processing performed by the queue manager in response to the GMCONV option.

The queue manager performs the following actions if the GMCONV option is specified on the MQGET call, and there is a message to be returned to the application:

1. If one or more of the following is true, no conversion is necessary:
   - The message data is already in the character set and encoding required by the application issuing the MQGET call. The application must set the *MDCSI* and *MDENC* fields in the **MSGDSC** parameter of the MQGET call to the values required, before issuing the call.
   - The length of the message data is zero.
   - The length of the **BUFFER** parameter of the MQGET call is zero.

   In these cases the message is returned without conversion to the application issuing the MQGET call; the *MDCSI* and *MDENC* values in the **MSGDSC** parameter are set to the values in the control information in the message, and the call completes with one of the following combinations of completion code and reason code:

   Completion code
   Reason code

CCOK
RCNONE

CCWARN
RC2079

CCWARN
RC2080

The following steps are performed only if the character set or encoding of the message data differs from the corresponding value in the **MSGDSC** parameter, and there is data to be converted:

1. If the *MDFMT* field in the control information in the message has the value FMNONE, the message is returned unconverted, with completion code CCWARN and reason code RC2110.

   In all other cases conversion processing continues.
2. The message is removed from the queue and placed in a temporary buffer which is the same size as the **BUFFER** parameter. For browse operations, the message is copied into the temporary buffer, instead of being removed from the queue.
3. If the message has to be truncated to fit in the buffer, the following is done:
   - If the GMATM option was not specified, the message is returned unconverted, with completion code CCWARN and reason code RC2080.
   - If the GMATM option *was* specified, the completion code is set to CCWARN, the reason code is set to RC2079, and conversion processing continues.
4. If the message can be accommodated in the buffer without truncation, or the GMATM option was specified, the following is done:
   - If the format is a built-in format, the buffer is passed to the queue manager's data-conversion service.
   - If the format is not a built-in format, the buffer is passed to a user-written exit which has the same name as the format. If the exit cannot be found, the message is returned unconverted, with completion code CCWARN and reason code RC2110.

   If no error occurs, the output from the data-conversion service or from the user-written exit is the converted message, plus the completion code and reason code to be returned to the application issuing the MQGET call.
5. If the conversion is successful, the queue manager returns the converted message to the application. In this case, the completion code and reason code returned by the MQGET call will typically be one of the following combinations:

   Completion code
   Reason code

   CCOK
   RCNONE

   CCWARN
   RC2079

   However, if the conversion is performed by a user-written exit, other reason codes can be returned, even when the conversion is successful.

   If the conversion fails (for whatever reason), the queue manager returns the unconverted message to the application, with the *MDCSI* and *MDENC* fields in the **MSGDSC** parameter set to the values in the control information in the message, and with completion code CCWARN.

Processing conventions on IBM i

IBM i

When converting a built-in format, the queue manager follows the processing conventions described in this topic.

Consider applying these conventions to user-written exits, although this is not enforced by the queue manager. The built-in formats converted by the queue manager are:

Built in formats
FMADMN
FMMDE
FMCICS
FMPCF
FMCMD1
FMRMH
FMCMD2
FMRFH
FMDLH
FMRFH2
FMDH
FMSTR
FMEVNT
FMTM
FMIMS
FMXQH
FMIMVS

1. If the message expands during conversion, and exceeds the size of the **BUFFER** parameter, the following is done:
   - If the GMATM option was not specified, the message is returned unconverted, with completion code CCWARN and reason code RC2120.
   - If the GMATM option *was* specified, the message is truncated, the completion code is set to CCWARN, the reason code is set to RC2079, and conversion processing continues.
2. If truncation occurs (either before or during conversion), it is possible for the number of valid bytes returned in the **BUFFER** parameter to be *less than* the length of the buffer.

   This can occur, for example, if a 4-byte integer or a DBCS character straddles the end of the buffer. The incomplete element of information is not converted, and so those bytes in the returned message do not contain valid information. This can also occur if a message that was truncated before conversion shrinks during conversion.

   If the number of valid bytes returned is less than the length of the buffer, the unused bytes at the end of the buffer are set to nulls.
3. If an array or string straddles the end of the buffer, as much of the data as possible is converted; only the particular array element or DBCS character which is incomplete is not converted - preceding array elements or characters are converted.
4. If truncation occurs (either before or during conversion), the length returned for the **DATLEN** parameter is the length of the *unconverted* message before truncation.
5. When strings are converted between single-byte character sets (SBCS), double-byte character sets (DBCS), or multi-byte character sets (MBCS), the strings can expand or contract.
   - In the PCF formats FMADMN, FMEVNT, and FMPCF, the strings in the MQCFST and MQCFSL structures expand or contract as necessary to accommodate the string after conversion.

For the string-list structure MQCFSL, the strings in the list might expand or contract by different amounts. If this happens, the queue manager pads the shorter strings with blanks to make them the same length as the longest string after conversion.

- In the format FMRMH, the strings addressed by the *RMSEO*, *RMSNO*, *RMDEO*, and *RMDNO* fields expand or contract as necessary to accommodate the strings after conversion.
- In the format FMRFH, the *RFNVS* field expands or contracts as necessary to accommodate the name-value pairs after conversion.
- In structures with fixed field sizes, the queue manager allows strings to expand or contract within their fixed fields, if no significant information is lost. In this regard, trailing blanks and characters following the first null character in the field are treated as insignificant.
  - If the string expands, but only insignificant characters need to be discarded to accommodate the converted string in the field, the conversion succeeds and the call completes with CCOK and reason code RCNONE (assuming no other errors).
  - If the string expands, but the converted string requires significant characters to be discarded in order to fit in the field, the message is returned unconverted and the call completes with CCWARN and reason code RC2190.

    Note: Reason code RC2190 results in this case whether the GMATM option was specified.
  - If the string contracts, the queue manager pads the string with blanks to the length of the field.

6. For messages consisting of one or more IBM MQ header structures followed by user data, it is possible for one or more of the header structures to be converted, while the remainder of the message is not. However, with two exceptions, the *MDCSI* and *MDENC* fields in each header structure always correctly indicate the character set and encoding of the data that follows the header structure.

   The two exceptions are the MQCIH and MQIIH structures, where the values in the *MDCSI* and *MDENC* fields in those structures are not significant. For those structures, the data following the structure is in the same character set and encoding as the MQCIH or MQIIH structure itself.

7. If the *MDCSI* or *MDENC* fields in the control information of the message being retrieved, or in the **MSGDSC** parameter, specify values which are undefined or not supported, the queue manager might ignore the error if the undefined or unsupported value does not need to be used in converting the message.

   For example, if the *MDENC* field in the message specifies an unsupported float encoding, but the message contains only integer data, or contains floating-point data which does not require conversion (because the source and target float encodings are identical), the error might or might not be diagnosed.

   If the error is diagnosed, the message is returned unconverted, with completion code CCWARN and one of the RC2111, RC2112, RC2113, RC2114 or RC2115, RC2116, RC2117, RC2118 reason codes (as appropriate); the *MDCSI* and *MDENC* fields in the **MSGDSC** parameter are set to the values in the control information in the message.

   If the error is not diagnosed and the conversion completes successfully, the values returned in the *MDCSI* and *MDENC* fields in the **MSGDSC** parameter, are those specified by the application issuing the MQGET call.

8. In all cases, if the message is returned to the application unconverted the completion code is set to CCWARN, and the *MDCSI* and *MDENC* fields in the **MSGDSC** parameter are set to the values appropriate to the unconverted data. This is done for FMNONE also.

   The **REASON** parameter is set to a code that indicates why the conversion could not be carried out, unless the message also had to be truncated; reason codes related to truncation take precedence over reason codes related to conversion. (To determine if a truncated message was converted, check the values returned in the *MDCSI* and *MDENC* fields in the **MSGDSC** parameter.)

   When an error is diagnosed, either a specific reason code is returned, or the general reason code RC2119. The reason code returned depends on the diagnostic capabilities of the underlying data-conversion service.

9. If completion code CCWARN is returned, and more than one reason code is relevant, the order of precedence is as follows:
   a. The following reason takes precedence over all others:
      - RC2079
   b. Next in precedence is the following reason:
      - RC2110
   c. The order of precedence within the remaining reason codes is not defined.
10. On completion of the MQGET call:
    - The following reason code indicates that the message was converted successfully:
      – RCNONE
    - The following reason code indicates that the message *may* have been converted successfully (check the *MDCSI* and *MDENC* fields in the **MSGDSC** parameter to find out):
      – RC2079
    - All other reason codes indicate that the message was not converted.

The following processing is specific to the built-in formats; it is not applicable to user-defined formats:

1. Except for the following formats:
   - FMADMN
   - FMEVNT
   - FMIMVS
   - FMPCF
   - FMSTR

   none of the built-in formats can be converted from or to character sets that do not have SBCS characters for the characters that are valid in queue names. If an attempt is made to perform such a conversion, the message is returned unconverted, with completion code CCWARN and reason code RC2111 or RC2115, as appropriate.

   The Unicode character set V 9.0.0 UTF-16 is an example of a character set that does not have SBCS characters for the characters that are valid in queue names.
2. If the message data for a built-in format is truncated, fields within the message which contain lengths of strings, or counts of elements or structures, are not adjusted to reflect the length of the data returned to the application; the values returned for such fields within the message data are the values applicable to the message before truncation.

   When processing messages such as a truncated FMADMN message, care must be taken to ensure that the application does not attempt to access data beyond the end of the data returned.
3. If the format name is FMDLH, the message data begins with an MQDLH structure, and this may be followed by zero or more bytes of application message data. The format, character set, and encoding of the application message data are defined by the *DLFMT*, *DLCSI*, and *DLENC* fields in the MQDLH structure at the start of the message. Since the MQDLH structure and application message data can have different character sets and encodings, it is possible for one, other, or both of the MQDLH structure and application message data to require conversion.

   The queue manager converts the MQDLH structure first, as necessary. If conversion is successful, or the MQDLH structure does not require conversion, the queue manager checks the *DLCSI* and *DLENC* fields in the MQDLH structure to see if conversion of the application message data is required. If conversion is required, the queue manager invokes the user-written exit with the name given by the *DLFMT* field in the MQDLH structure, or performs the conversion itself (if *DLFMT* is the name of a built-in format).

   If the MQGET call returns a completion code of CCWARN, and the reason code is one of those indicating that conversion was not successful, one of the following applies:

- The MQDLH structure could not be converted. In this case the application message data will not have been converted either.
- The MQDLH structure was converted, but the application message data was not.

The application can examine the values returned in the *MDCSI* and *MDENC* fields in the **MSGDSC** parameter, and those in the MQDLH structure, in order to determine which of the previous applies.

4. If the format name is FMXQH, the message data begins with an MQXQH structure, and this may be followed by zero or more bytes of additional data. This additional data is typically the application message data (which may be of zero length), but there can also be one or more further IBM MQ header structures present, at the start of the additional data.

   The MQXQH structure must be in the character set and encoding of the queue manager. The format, character set, and encoding of the data following the MQXQH structure are given by the *MDFMT*, *MDCSI*, and *MDENC* fields in the MQMD structure contained *within* the MQXQH. For each subsequent IBM MQ header structure present, the *MDFMT*, *MDCSI*, and *MDENC* fields in the structure describe the data that follows that structure; that data is either another IBM MQ header structure, or the application message data.

   If the GMCONV option is specified for an FMXQH message, the application message data and certain of the MQ header structures are converted, but the data in the MQXQH structure is not. On return from the MQGET call, therefore:

   - The values of the *MDFMT*, *MDCSI*, and *MDENC* fields in the **MSGDSC** parameter, describe the data in the MQXQH structure, and not the application message data; the values will therefore not be the same as those specified by the application that issued the MQGET call.

     The effect of this is that an application which repeatedly gets messages from a transmission queue with the GMCONV option specified must reset the *MDCSI* and *MDENC* fields in the **MSGDSC** parameter to the values necessary for the application message data, before each MQGET call.
   - The values of the *MDFMT*, *MDCSI*, and *MDENC* fields in the last MQ header structure present describe the application message data. If there are no other IBM MQ header structures present, the application message data is described by these fields in the MQMD structure within the MQXQH structure. If conversion is successful, the values will be the same as those specified in the **MSGDSC** parameter by the application that issued the MQGET call.

   If the message is a distribution-list message, the MQXQH structure is followed by an MQDH structure (plus its arrays of MQOR and MQPMR records), which in turn may be followed by zero or more further IBM MQ header structures and zero or more bytes of application message data. Like the MQXQH structure, the MQDH structure must be in the character set and encoding of the queue manager, and it is not converted on the MQGET call, even if the GMCONV option is specified.

   The processing of the MQXQH and MQDH structures described previously are primarily intended for use by message channel agents when they get messages from transmission queues.

Conversion of report messages on IBM i

A report message can contain varying amounts of application message data, according to the report options specified by the sender of the original message.

In particular, a report message can contain either:

1. No application message data
2. Some of the application message data from the original message

   This occurs when the sender of the original message specifies RO*D and the message is longer than 100 bytes.
3. All of the application message data from the original message

   This occurs when the sender of the original message specifies RO*F, or specifies RO*D and the message is 100 bytes or shorter.

When the queue manager or message channel agent generates a report message, it copies the format name from the original message into the *MDFMT* field in the control information in the report message. The format name in the report message might therefore imply a length of data which is different from the length present in the report message (cases 1 and 2 described previously).

If the GMCONV option is specified when the report message is retrieved:

- For case 1 described previously, the data-conversion exit will not be invoked (because the report message will have no data).
- For case 3 described previously, the format name correctly implies the length of the message data.
- But for case 2 described previously, the data-conversion exit will be invoked to convert a message which is *shorter* than the length implied by the format name.

  In addition, the reason code passed to the exit will typically be RCNONE (that is, the reason code will not indicate that the message has been truncated). This happens because the message data was truncated by the *sender* of the report message, and not by the receiver's queue manager in response to the MQGET call.

Because of these possibilities, the data-conversion exit should not use the format name to deduce the length of data passed to it; instead the exit should check the length of data provided, and be prepared to convert less data than the length implied by the format name. If the data can be converted successfully, completion code CCOK and reason code RCNONE should be returned by the exit. The length of the message data to be converted is passed to the exit as the **INLEN** parameter.

Product-sensitive programming interface

If a report message contains information about an activity that has taken place, it is known as an activity report. Examples of activities are:

- an MCA sending a message from a queue down a channel
- an MCA receiving a message from a channel and putting it onto a queue
- an MCA dead-letter queuing an undeliverable message
- an MCA getting a message off a queue and discarding it
- a dead-letter handler placing a message back on a queue
- the command server processing a PCF request - a broker processing a publish request
- a user application getting a message from a queue - a user application browsing a message on a queue

Any application, including the queue manager, can add some of the message data to the activity report following the report header. The amount of data that should be supplied if some is sent is not fixed, and is decided by the application. The information returned should be useful to the application processing the

activity report. Queue manager activity reports will return with them any standard IBM MQ header structures (beginning 'MQH') contained in the original message. This includes, for example, any MQRFH2 headers that were included in the original message. Also the queue manager will return an MQCFH header found, but not the PCF parameters associated with it. This gives monitoring applications an idea of what the message was about.

MQDXP (Data-conversion exit parameter) on IBM i

Data-conversion exit parameter block.

Overview

Purpose: The MQDXP structure is a parameter that the queue manager passes to the data-conversion exit when the exit is invoked to convert the message data as part of the processing of the MQGET call. See the description of the MQCONVX call for details of the data conversion exit.

Character set and encoding: Character data in MQDXP is in the character set of the local queue manager; this is given by the **CodedCharSetId** queue manager attribute. Numeric data in MQDXP is in the native machine encoding; this is given by ENNAT.

Usage: Only the *DXLEN*, *DXCC*, *DXREA* and *DXRES* fields in MQDXP might be changed by the exit; changes to other fields are ignored. However, the *DXLEN* field cannot be changed if the message being converted is a segment that contains only part of a logical message.

When control returns to the queue manager from the exit, the queue manager checks the values returned in MQDXP. If the values returned are not valid, the queue manager continues processing as though the exit had returned XRFAIL in *DXRES* ; however, the queue manager ignores the values of the *DXCC* and *DXREA* fields returned by the exit in this case, and uses instead the values those fields had on *input* to the exit. The following values in MQDXP cause this processing to occur:

- *DXRES* field not XROK and not XRFAIL
- *DXCC* field not CCOK and not CCWARN
- *DXLEN* field less than zero, or *DXLEN* field changed when the message being converted is a segment that contains only part of a logical message.
- "Fields"
- "RPG declaration (copy file CMQDXPH)" on page 3475

Fields

The MQDXP structure contains the following fields; the fields are described in **alphabetical order**:

DXAOP (10-digit signed integer)

Application options.

This is a copy of the *GMOPT* field of the MQGMO structure specified by the application issuing the MQGET call. The exit may need to examine these to ascertain whether the GMATM option was specified.

This is an input field to the exit.

DXCC (10-digit signed integer)

Completion code.

When the exit is invoked, this contains the completion code that will be returned to the application that issued the MQGET call, if the exit chooses to do nothing. It is always CCWARN, because either the message was truncated, or the message requires conversion and this has not yet been done.

On output from the exit, this field contains the completion code to be returned to the application in the **CMPCOD** parameter of the MQGET call; only CCOK and CCWARN are valid. See the description of the *DXREA* field for suggestions on how the exit should set this field on output.

This is an input/output field to the exit.

DXCSI (10-digit signed integer)

Character set required by application.

This is the coded character-set identifier of the character set required by the application issuing the MQGET call; see the *MDCSI* field in the MQMD structure for more details. If the application specifies the special value CSQM on the MQGET call, the queue manager changes this to the actual character-set identifier of the character set used by the queue manager, before invoking the exit.

If the conversion is successful, the exit should copy this to the *MDCSI* field in the message descriptor.

This is an input field to the exit.

DXENC (10-digit signed integer)

Numeric encoding required by application.

This is the numeric encoding required by the application issuing the MQGET call; see the *MDENC* field in the MQMD structure for more details.

If the conversion is successful, the exit should copy this to the *MDENC* field in the message descriptor.

This is an input field to the exit.

DXHCN (10-digit signed integer)

Connection handle.

This is a connection handle which can be used on the MQXCNVC call. This handle is not necessarily the same as the handle specified by the application which issued the MQGET call.

DXLEN (10-digit signed integer)

Length in bytes of message data.

When the exit is invoked, this field contains the original length of the application message data. If the message was truncated in order to fit into the buffer provided by the application, the size of the message provided to the exit will be *smaller* than the value of *DXLEN*. The size of the message provided to the exit is always given by the **INLEN** parameter of the exit, irrespective of any truncation that may have occurred.

Truncation is indicated by the *DXREA* field having the value RC2079 on input to the exit.

Most conversions will not need to change this length, but an exit can do so if necessary; the value set by the exit is returned to the application in the **DATLEN** parameter of the MQGET call. However, this length cannot be changed if the message being converted is a segment that contains only part of a logical message. This is because changing the length would cause the offsets of later segments in the logical message to be incorrect.

Note that, if the exit wants to change the length of the data, be aware that the queue manager has already decided whether the message data will fit into the application's buffer, based on the length of the *unconverted* data. This decision determines whether the message is removed from

the queue (or the browse cursor moved, for a browse request), and is not affected by any change to the data length caused by the conversion. For this reason it is recommended that conversion exits do not cause a change in the length of the application message data.

If character conversion does imply a change of length, a string can be converted into another string with the same length in bytes, truncating trailing blanks, or padding with blanks as necessary.

The exit is not invoked if the message contains no application message data; hence *DXLEN* is always greater than zero.

This is an input/output field to the exit.

DXREA (10-digit signed integer)

Reason code qualifying *DXCC*.

When the exit is invoked, this contains the reason code that will be returned to the application that issued the MQGET call, if the exit chooses to do nothing. Among possible values are RC2079, indicating that the message was truncated in order fit into the buffer provided by the application, and RC2119, indicating that the message requires conversion but that this has not yet been done.

On output from the exit, this field contains the reason to be returned to the application in the **REASON** parameter of the MQGET call; the following is recommended:

- If *DXREA* had the value RC2079 on input to the exit, the *DXREA* and *DXCC* fields should not be altered, irrespective of whether the conversion succeeds or fails.

  (If the *DXCC* field is not CCOK, the application which retrieves the message can identify a conversion failure by comparing the returned *MDENC* and *MDCSI* values in the message descriptor with the values requested; in contrast, the application cannot distinguish a truncated message from a message that just fitted the buffer. For this reason, RC2079 should be returned in preference to any of the reasons that indicate conversion failure.)
- If *DXREA* had any other value on input to the exit:
  - If the conversion succeeds, *DXCC* should be set to CCOK and *DXREA* set to RCNONE.
  - If the conversion fails, or the message expands and has to be truncated to fit in the buffer, *DXCC* should be set to CCWARN (or left unchanged), and *DXREA* set to one of the values i the following list, to indicate the nature of the failure.

    Note that, if the message after conversion is too big for the buffer, it should be truncated only if the application that issued the MQGET call specified the GMATM option:

    - If it did specify that option, reason RC2079 should be returned.
    - If it did not specify that option, the message should be returned unconverted, with reason code RC2120.

The reason codes in the following list are recommended for use by the exit to indicate the reason that conversion failed, but the exit can return other values from the set of RC* codes if deemed appropriate. In addition, the range of values RC0900 through RC0999 are allocated for use by the exit to indicate conditions that the exit wants to communicate to the application issuing the MQGET call.

Note: If the message cannot be converted successfully, the exit must return XRFAIL in the *DXRES* field, in order to cause the queue manager to return the unconverted message. This is true regardless of the reason code returned in the *DXREA* field.

RC0900
(900, X'384') Lowest value for application-defined reason code.

RC0999
(999, X'3E7') Highest value for application-defined reason code.

RC2120
(2120, X'848') Converted data too big for buffer.

RC2119
(2119, X'847') Message data not converted.

RC2111
(2111, X'83F') Source coded character set identifier not valid.

RC2113
(2113, X'841') Packed-decimal encoding in message not recognized.

RC2114
(2114, X'842') Floating-point encoding in message not recognized.

RC2112
(2112, X'840') Source integer encoding not recognized.

RC2115
(2115, X'843') Target coded character set identifier not valid.

RC2117
(2117, X'845') Packed-decimal encoding specified by receiver not recognized.

RC2118
(2118, X'846') Floating-point encoding specified by receiver not recognized.

RC2116
(2116, X'844') Target integer encoding not recognized.

RC2079
(2079, X'81F') Truncated message returned (processing completed).

This is an input/output field to the exit.

DXRES (10-digit signed integer)

Response from exit.

This is set by the exit to indicate the success or otherwise of the conversion. It must be one of the following:

XROK Conversion was successful.

If the exit specifies this value, the queue manager returns the following to the application that issued the MQGET call:

- The value of the *DXCC* field on output from the exit
- The value of the *DXREA* field on output from the exit
- The value of the *DXLEN* field on output from the exit
- The contents of the exit's output buffer *OUTBUF*. The number of bytes returned is the lesser of the exit's **OUTLEN** parameter, and the value of the *DXLEN* field on output from the exit

If the *MDENC* and *MDCSI* fields in the exit's message descriptor parameter are *both* unchanged, the queue manager returns:

- The value of the *MDENC* and *MDCSI* fields in the MQDXP structure on *input* to the exit

If one or both of the *MDENC* and *MDCSI* fields in the exit's message descriptor parameter has been changed, the queue manager returns:

- The value of the *MDENC* and *MDCSI* fields in the exit's message descriptor parameter on output from the exit
-

XRFAIL
Conversion was unsuccessful.

If the exit specifies this value, the queue manager returns the following to the application that issued the MQGET call:

- The value of the *DXCC* field on output from the exit
- The value of the *DXREA* field on output from the exit
- The value of the *DXLEN* field on *input* to the exit
- The contents of the exit's input buffer *INBUF*. The number of bytes returned is given by the **INLEN** parameter

If the exit has altered *INBUF*, the results are undefined.

DXRES is an output field from the exit.

DXSID (4-byte character string)

Structure identifier.

The value must be:

DXSIDV
Identifier for data conversion exit parameter structure.

This is an input field to the exit.

DXVER (10-digit signed integer)

Structure version number.

The value must be:

DXVER1
Version number for data-conversion exit parameter structure.

The following constant specifies the version number of the current version:

DXVERC
Current version of data-conversion exit parameter structure.

Note: When a new version of this structure is introduced, the layout of the existing part is not changed. The exit should therefore check that the *DXVER* field is equal to or greater than the lowest version which contains the fields that the exit needs to use.

This is an input field to the exit.

DXXOP (10-digit signed integer)

Reserved.

This is a reserved field; its value is 0.

RPG declaration (copy file CMQDXPH)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQDXP Structure
D*
D* Structure identifier
D  DXSID                  1      4
D* Structure version number
D  DXVER                  5      8I 0
D* Reserved
D  DXXOP                  9     12I 0
D* Application options
D  DXAOP                 13     16I 0
```

```
D* Numeric encoding required by application
D  DXENC                 17     20I 0
D* Character set required by application
D  DXCSI                 21     24I 0
D* Length in bytes of message data
D  DXLEN                 25     28I 0
D* Completion code
D  DXCC                  29     32I 0
D* Reason code qualifying DXCC
D  DXREA                 33     36I 0
D* Response from exit
D  DXRES                 37     40I 0
D* Connection handle
D  DXHCN                 41     44I 0
```

MQXCNVC (Convert characters) on IBM i

The MQXCNVC call converts characters from one character set to another.

This call is part of the IBM MQ Data Conversion Interface (DCI), which is one of the IBM MQ framework interfaces. Note: This call can be used only from a data-conversion exit.

- "Syntax"
- "Parameters"
- "RPG invocation (ILE)" on page 3480

Syntax

```
MQXCNVC (HCONN, OPTS, SRCCSI, SRCLEN, SRCBUF, TGTCSI, TGTLEN,
        TGTBUF, DATLEN, CMPCOD, REASON)
```

Parameters

The MQXCNVC call has the following parameters:

HCONN (10-digit signed integer) - input

Connection handle.

This handle represents the connection to the queue manager. It should normally be the handle passed to the data-conversion exit in the *DXHCN* field of the MQDXP structure; this handle is not necessarily the same as the handle specified by the application which issued the MQGET call.

On IBM i, the following special value can be specified for *HCONN*:

HCDEFH
Default connection handle.

OPTS (10-digit signed integer) - input

Options that control the action of MQXCNVC.

Zero or more of the options described later in this section can be specified. If more than one is required, the values can be added (do not add the same constant more than once).

Default-conversion option: The following option controls the use of default character conversion:

DCCDEF
Default conversion.

This option specifies that default character conversion can be used if one or both of the character sets specified on the call is not supported. This allows the queue manager to use an installation-specified default character set that approximates the specified character set, when converting the string.

Note: The result of using an approximate character set to convert the string is that some characters may be converted incorrectly. This can be avoided by using in the string only characters which are common to both the specified character set and the default character set.

The default character sets are defined by a configuration option when the queue manager is installed or restarted.

If DCCDEF is not specified, the queue manager uses only the specified character sets to convert the string, and the call fails if one or both of the character sets is not supported.

Padding option: The following option allows the queue manager to pad the converted string with blanks or discard insignificant trailing characters, in order to make the converted string fit the target buffer:

DCCFIL

Fill target buffer.

This option requests that conversion take place in such a way that the target buffer is filled completely:

- If the string contracts when it is converted, trailing blanks are added in order to fill the target buffer.
- If the string expands when it is converted, trailing characters that are not significant are discarded to make the converted string fit the target buffer. If this can be done successfully, the call completes with CCOK and reason code RCNONE.

  If there are too few insignificant trailing characters, as much of the string as will fit is placed in the target buffer, and the call completes with CCWARN and reason code RC2120.

  Insignificant characters are:

  – Trailing blanks
  – Characters following the first null character in the string (but excluding the first null character itself)
- If the string, *TGTCSI*, and *TGTLEN* are such that the target buffer cannot be set completely with valid characters, the call fails with CCFAIL and reason code RC2144. This can occur when *TGTCSI* is a pure DBCS character set (such as

  V 9.0.0 UTF-16), but *TGTLEN* specifies a length that is an odd number of bytes.
- *TGTLEN* can be less than or greater than *SRCLEN*. On return from MQXCNVC, *DATLEN* has the same value as *TGTLEN*.

If this option is not specified:

- The string is allowed to contract or expand within the target buffer as required. Insignificant trailing characters are not added or discarded.

  If the converted string fits in the target buffer, the call completes with CCOK and reason code RCNONE.

  If the converted string is too large for the target buffer, as much of the string as will fit is placed in the target buffer, and the call completes with CCWARN and reason code RC2120. Note that fewer than *TGTLEN* bytes can be returned in this case.
- *TGTLEN* can be less than or greater than *SRCLEN*. On return from MQXCNVC, *DATLEN* is less than or equal to *TGTLEN*.

Encoding options: The following options can be used to specify the integer encodings of the source and target strings. The relevant encoding is used only when the corresponding character set identifier indicates that the representation of the character set in main storage is dependent on the encoding used for binary integers. This affects only certain multibyte character sets (for example, V 9.0.0 UTF-16 character sets).

The encoding is ignored if the character set is a single-byte character set (SBCS), or a multibyte character set with representation in main storage that is not dependent on the integer encoding.

Only one of the DCCS* values should be specified, combined with one of the DCCT* values:

DCCSNA
Source encoding is the default for the environment and programming language.

DCCSNO
Source encoding is normal.

DCCSRE
Source encoding is reversed.

DCCSUN
Source encoding is undefined.

DCCTNA
Target encoding is the default for the environment and programming language.

DCCTNO
Target encoding is normal.

DCCTRE
Target encoding is reversed.

DCCTUN
Target encoding is undefined.

The encoding values defined previously can be added directly to the *OPTS* field. However, if the source or target encoding is obtained from the *MDENC* field in the MQMD or other structure, the following processing must be done:

1. The integer encoding must be extracted from the *MDENC* field by eliminating the float and packed-decimal encodings; see "Analyzing encodings on IBM i" on page 3459 for details of how to do this.
2. The integer encoding resulting from step 1 must be multiplied by the appropriate factor before being added to the *OPTS* field. These factors are:

   DCCSFA
   Factor for source encoding

   DCCTFA
   Factor for target encoding

If not specified, the encoding options default to undefined (DCC*UN). In most cases, this does not affect the successful completion of the MQXCNVC call. However, if the corresponding character set is a multibyte character set with representation that is dependent on the encoding (for example, a V 9.0.0 UTF-16 character set), the call fails with reason code RC2112 or RC2116 as appropriate.

Default option: If none of the options described previously is specified, the following option can be used:

DCCNON
No options specified.

DCCNON is defined to aid program documentation. It is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

SRCCSI (10-digit signed integer) - input

Coded character set identifier of string before conversion.

This is the coded character set identifier of the input string in *SRCBUF*.

SRCLEN (10-digit signed integer) - input

Length of string before conversion.

This is the length in bytes of the input string in *SRCBUF* ; it must be zero or greater.

SRCBUF (1-byte character string x SRCLEN) - input

String to be converted.

This is the buffer containing the string to be converted from one character set to another.

TGTCSI (10-digit signed integer) - input

Coded character set identifier of string after conversion.

This is the coded character set identifier of the character set to which *SRCBUF* is to be converted.

TGTLEN (10-digit signed integer) - input

Length of output buffer.

This is the length in bytes of the output buffer *TGTBUF* ; it must be zero or greater. It can be less than or greater than *SRCLEN*.

TGTBUF (1-byte character string x TGTLEN) - output

String after conversion.

This is the string after it has been converted to the character set defined by *TGTCSI*. The converted string can be shorter or longer than the unconverted string. The **DATLEN** parameter indicates the number of valid bytes returned.

DATLEN (10-digit signed integer) - output

Length of output string.

This is the length of the string returned in the output buffer *TGTBUF*. The converted string can be shorter or longer than the unconverted string.

CMPCOD (10-digit signed integer) - output

Completion code.

It is one of the following:

CCOK
Successful completion.

CCWARN
Warning (partial completion).

CCFAIL
Call failed.

REASON (10-digit signed integer) - output

Reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

RCNONE
(0, X'000') No reason to report.

If *CMPCOD* is CCWARN:

RC2120
(2120, X'848') Converted data too large for buffer.

If *CMPCOD* is CCFAIL:

RC2010
(2010, X'7DA') Data length parameter not valid.

RC2150
(2150, X'866') DBCS string not valid.

RC2018
(2018, X'7E2') Connection handle not valid.

RC2046
(2046, X'7FE') Options not valid or not consistent.

RC2102
(2102, X'836') Insufficient system resources available.

RC2145
(2145, X'861') Source buffer parameter not valid.

RC2111
(2111, X'83F') Source coded character set identifier not valid.

RC2112
(2112, X'840') Source integer encoding not recognized.

RC2143
(2143, X'85F') Source length parameter not valid.

RC2071
(2071, X'817') Insufficient storage available.

RC2146
(2146, X'862') Target buffer parameter not valid.

RC2115
(2115, X'843') Target coded character set identifier not valid.

RC2116
(2116, X'844') Target integer encoding not recognized.

RC2144
(2144, X'860') Target length parameter not valid.

RC2195
(2195, X'893') Unexpected error occurred.

For more information about these reason codes, see "Return codes for IBM i (ILE RPG)" on page 3452.

RPG invocation (ILE)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                   CALLP     MQXCNVC(HCONN : OPTS : SRCCSI :
C                                     SRCLEN : SRCBUF : TGTCSI :
C                                     TGTLEN : TGTBUF : DATLEN :
C                                     CMPCOD : REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQXCNVC          PR                  EXTPROC('MQXCNVC')
D* Connection handle
D HCONN                          10I 0 VALUE
D* Options that control the action of MQXCNVC
D OPTS                           10I 0 VALUE
D* Coded character set identifier of string before conversion
D SRCCSI                         10I 0 VALUE
D* Length of string before conversion
D SRCLEN                         10I 0 VALUE
D* String to be converted
D SRCBUF                           *   VALUE
D* Coded character set identifier of string after conversion
D TGTCSI                         10I 0 VALUE
D* Length of output buffer
D TGTLEN                         10I 0 VALUE
D* String after conversion
D TGTBUF                           *   VALUE
D* Length of output string
D DATLEN                         10I 0
D* Completion code
D CMPCOD                         10I 0
D* Reason code qualifying CMPCOD
D REASON                         10I 0
```

MQCONVX (Data conversion exit) on IBM i

IBM i

This call definition describes the parameters that are passed to the data-conversion exit.

No entry point called MQCONVX is provided by the queue manager (see usage note 11 on page 3483).

This definition is part of the IBM MQ Data Conversion Interface (DCI), which is one of the IBM MQ framework interfaces.

Syntax

MQCONVX *(MQDXP, MQMD, INLEN, INBUF, OUTLEN, OUTBUF)*

Usage notes

1. A data-conversion exit is a user-written exit which receives control during the processing of an MQGET call. The function performed by the data-conversion exit is defined by the provider of the exit; however, the exit must conform to the rules described here, and in the associated parameter structure MQDXP.

   The programming languages that can be used for a data-conversion exit are determined by the environment.
2. The exit is invoked only if *all* of the following statements are true:
   - The GMCONV option is specified on the MQGET call
   - The *MDFMT* field in the message descriptor is not FMNONE
   - The message is not already in the required representation; that is, one or both of the message's *MDCSI* and *MDENC* is different from the value specified by the application in the message descriptor supplied on the MQGET call

- The queue manager has not already done the conversion successfully
- The length of the application's buffer is greater than zero
- The length of the message data is greater than zero
- The reason code so far during the MQGET operation is RCNONE or RC2079

3. When an exit is being written, consideration should be given to coding the exit in a way that will allow it to convert messages that have been truncated. Truncated messages can arise in the following ways:
   - The receiving application provides a buffer that is smaller than the message, but specifies the GMATM option on the MQGET call.

     In this case, the *DXREA* field in the **MQDXP** parameter on input to the exit will have the value RC2079.
   - The sender of the message truncated it before sending it. This can happen with report messages, for example (see "Conversion of report messages on IBM i" on page 3470 for more details).

     In this case, the *DXREA* field in the **MQDXP** parameter on input to the exit will have the value RCNONE (if the receiving application provided a buffer that was large enough for the message).

   Thus the value of the *DXREA* field on input to the exit cannot always be used to decide whether the message has been truncated.

   The distinguishing characteristic of a truncated message is that the length provided to the exit in the **INLEN** parameter will be *less than* the length implied by the format name contained in the *MDFMT* field in the message descriptor. The exit should therefore check the value of *INLEN* before attempting to convert any of the data; the exit *should not* assume that the full amount of data implied by the format name has been provided.

   If the exit has not been written to convert truncated messages, and **INLEN** is less than the value expected, the exit should return XRFAIL in the *DXRES* field of the **MQDXP** parameter, with the *DXCC* field set to CCWARN and the *DXREA* field set to RC2110.

   If the exit *has* been written to convert truncated messages, the exit should convert as much of the data as possible (see next usage note), taking care not to attempt to examine or convert data beyond the end of *INBUF*. If the conversion completes successfully, the exit should leave the *DXREA* field in the **MQDXP** parameter unchanged. This returns RC2079 if the message was truncated by the receiver's queue manager, and RCNONE if the message was truncated by the sender of the message.

   It is also possible for a message to expand *during* conversion, to the point where it is bigger than *OUTBUF*. In this case the exit must decide whether to truncate the message; the *DXAOP* field in the **MQDXP** parameter will indicate whether the receiving application specified the GMATM option.
4. Generally it is recommended that all of the data in the message provided to the exit in *INBUF* is converted, or that none of it is. An exception to this, however, occurs if the message is truncated, either before conversion or during conversion; in this case there may be an incomplete item at the end of the buffer (for example: one byte of a double-byte character, or 3 bytes of a 4-byte integer). In this situation it is recommended that the incomplete item should be omitted, and unused bytes in *OUTBUF* set to nulls. However, complete elements or characters within an array or string *should* be converted.
5. When an exit is needed for the first time, the queue manager attempts to load an object that has the same name as the format (apart from extensions). The object loaded must contain the exit that processes messages with that format name. It is recommended that the exit name, and the name of the object that contain the exit, should be identical, although not all environments require this.
6. A new copy of the exit is loaded when an application attempts to retrieve the first message that uses that *MDFMT* since the application connected to the queue manager. A new copy might also be loaded at other times, if the queue manager has discarded a previously loaded copy. For this reason, an exit should not attempt to use static storage to communicate information from one invocation of the exit to the next - the exit may be unloaded between the two invocations.

7. If there is a user-supplied exit with the same name as one of the built-in formats supported by the queue manager, the user-supplied exit does not replace the built-in conversion routine. The only circumstances in which such an exit is invoked are:
   - If the built-in conversion routine cannot handle conversions to or from either the *MDCSI* or *MDENC* involved, or
   - If the built-in conversion routine has failed to convert the data (for example, because there is a field or character which cannot be converted).
8. The scope of the exit is environment-dependent. *MDFMT* names should be chosen so as to minimize the risk of clashes with other formats. It is recommended that they start with characters that identify the application defining the format name.
9. The data-conversion exit runs in an environment like that of the program which issued the MQGET call; environment includes address space and user profile (where applicable). The program could be a message channel agent sending messages to a destination queue manager that does not support message conversion. The exit cannot compromise the queue manager's integrity, since it does not run in the queue manager's environment.
10. The only MQI call which can be used by the exit is MQXCNVC; attempting to use other MQI calls fails with reason code RC2219, or other unpredictable errors.
11. No entry point called MQCONVX is provided by the queue manager. The name of the exit should be the same as the format name (the name contained in the *MDFMT* field in MQMD), although this is not required in all environments.

Parameters

The MQCONVX call has the following parameters:

MQDXP (MQDXP) - input/output

Data-conversion exit parameter block.

This structure contains information relating to the invocation of the exit. The exit sets information in this structure to indicate the outcome of the conversion. See "MQDXP (Data-conversion exit parameter) on IBM i" on page 3471 for details of the fields in this structure.

MQMD (MQMD) - input/output

Message descriptor.

On input to the exit, this is the message descriptor that would be returned to the application if no conversion were performed. It therefore contains the *MDFMT*, *MDENC*, and *MDCSI* of the unconverted message contained in *INBUF*.

Note: The **MQMD** parameter passed to the exit is always the most recent version of MQMD supported by the queue manager which invokes the exit. If the exit is intended to be portable between different environments, the exit should check the *MDVER* field in *MQMD* to verify that the fields that the exit needs to access are present in the structure.

On IBM i, the exit is passed a version-2 MQMD.

On output, the exit should change the *MDENC* and *MDCSI* fields to the values requested by the application, if conversion was successful; these changes will be reflected back to the application. Any other changes that the exit makes to the structure are ignored; they are not reflected back to the application.

If the exit returns XROK in the *DXRES* field of the MQDXP structure, but does not change the *MDENC* or *MDCSI* fields in the message descriptor, the queue manager returns for those fields the values that the corresponding fields in the MQDXP structure had on input to the exit.

INLEN (10-digit signed integer) - input

Length in bytes of *INBUF*.

This is the length of the input buffer *INBUF*, and specifies the number of bytes to be processed by the exit. *INLEN* is the lesser of the length of the message data before conversion, and the length of the buffer provided by the application on the MQGET call.

The value is always greater than zero.

INBUF (1-byte bit string x INLEN) - input

Buffer containing the unconverted message.

This contains the message data before conversion. If the exit is unable to convert the data, the queue manager returns the contents of this buffer to the application after the exit has completed.

Note: The exit should not alter *INBUF* ; if this parameter is altered, the results are undefined.

OUTLEN (10-digit signed integer) - input

Length in bytes of *OUTBUF*.

This is the length of the output buffer *OUTBUF*, and is the same as the length of the buffer provided by the application on the MQGET call.

The value is always greater than zero.

OUTBUF (1-byte bit string x OUTLEN) - output

Buffer containing the converted message.

On output from the exit, if the conversion was successful (as indicated by the value XROK in the *DXRES* field of the **MQDXP** parameter), **OUTBUF** contains the message data to be delivered to the application, in the requested representation. If the conversion was unsuccessful, any changes that the exit has made to this buffer are ignored.

RPG invocation (ILE)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     exitname(MQDXP : MQMD : INLEN :
C                                        INBUF : OUTLEN : OUTBUF)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
Dexitname         PR                  EXTPROC('exitname')
D* Data-conversion exit parameter block
D MQDXP                         44A
D* Message descriptor
D MQMD                         364A
D* Length in bytes of INBUF
D INLEN                         10I 0 VALUE
D* Buffer containing the unconverted message
D INBUF                          *   VALUE
D* Length in bytes of OUTBUF
D OUTLEN                        10I 0 VALUE
D* Buffer containing the converted message
D OUTBUF                         *   VALUE
```

End of product-sensitive programming interface

SOAP reference

IBM MQ transport for SOAP reference information arranged alphabetically.

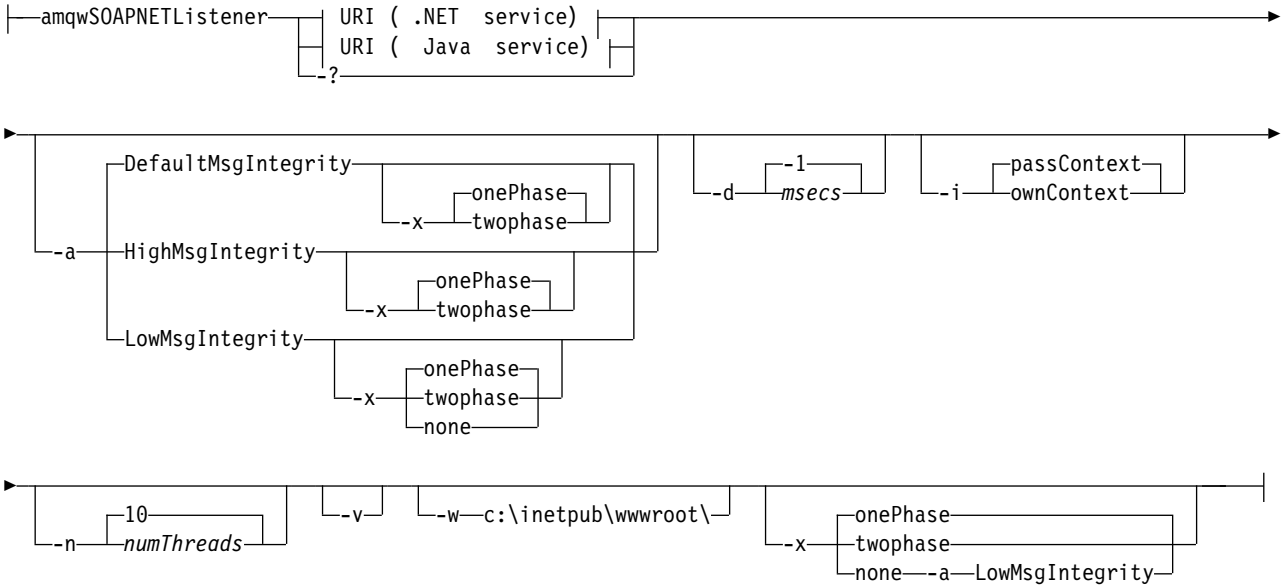
amqwSOAPNETListener: IBM MQ SOAP listener for .NET Framework 1 or 2

Syntax and parameters for the IBM MQ SOAP listener for .NET Framework 1 or 2.

Purpose

Starts the IBM MQ SOAP listener for .NET Framework 1 or 2.

.NET:



Required parameters

URI *platform*

See "URI syntax and parameters for web service deployment" on page 3518.

-? Print out help text describing how the command is used.

Optional parameters

-a *integrityOption*

integrityOption specifies the behavior of IBM MQ SOAP listeners if it is not possible to put a failed request message on the dead-letter queue. *integrityOption* can take one of the following values:

DefaultMsgIntegrity

For non-persistent messages, the listener displays a warning message and continues to execute with the original message being discarded. For persistent messages, it displays an error message, backs out the request message so it remains on the request queue and exits. DefaultMsgIntegrity applies if the -a option is omitted, or if *integrityOption* is not specified.

LowMsgIntegrity

For both persistent and non-persistent messages, the listener displays a warning and continues to execute, discarding the message.

HighMsgIntegrity

For both persistent and non-persistent messages, the listener displays an error message, backs out the request message so it remains on the request queue and exits.

The deployment utility checks for the compatibility of the -x and -a flags. If -x none is specified, then -a LowMsgIntegrity must be specified. If the flags are incompatible the deployment utility exits with an error message and with no deployment steps having been undertaken.

-d *msecs*
msecs specifies the number of milliseconds for the IBM MQ SOAP listener to stay alive if request messages have been received on any thread. If *msecs* is set to -1, the listener stays alive indefinitely.

-i *Context*
Context specifies whether the listeners pass identity context. *Context* takes the following values:

passContext
Set the identity context of the original request message into the response message. The SOAP listener checks that it has authority to save the context from the request queue and to pass it to the response queue. It makes the checks at run time when opening the request queue to save context, and the response queue to pass context. If it does not have the required authority, or the MQOPEN call fails, and the response message is not processed. The response message is put on the dead-letter queue with the dead-letter header containing the return code from the failed MQOPEN. The listener then continues to process subsequent incoming messages as normal.

ownContext
The SOAP listener does not pass context. The returned context reflects the user ID under which the listener is running rather than the user ID which created the original request message.

The fields in the origin context are set by the queue manager, and not by the SOAP listener.

-n *numThreads*
numThreads specifies the number of threads in the generated startup scripts for the IBM MQ SOAP listener. The default is 10. Consider increasing this number if you have high message throughput.

-v -v sets verbose output from external commands. Error messages are always displayed. Use -v to output commands that you can tailor to create customized deployment scripts.

-w *serviceDirectory*
serviceDirectory is the directory containing the web service.

-x *transactionality*
transactionality specifies the type of transactional control for the listener. *transactionality* can be set to one of the following values:

onePhase
IBM MQ one-phase support is used. If the system fails during processing, the request message is redelivered to the application. IBM MQ transactions ensure that the response messages are written exactly once.

twoPhase
Two-phase support is used. If the service is written appropriately the message is delivered exactly once, coordinated with other resources, within a single committed execution of the service. This option applies to server bindings connections only.

none No transactional support. If the system fails during processing, the request message can be lost, even if persistent. The service might or might not have executed, and response, report or dead-letter messages might or might not be written.

The deployment utility checks for the compatibility of the -x and -a flags. See the description of the -a flag for details.

.NET Example

```
amqwSOAPNETlistener
-u "jms:/queue?destination=myQ&connectionFactory=()
&targetService=myService&initialContextFactory=com.ibm.mq.jms.Nojndi"
-w C:/wmqsoap/demos
-n 20
```

amqswsdl: generate WSDL for .NET Framework 1 or 2 service

amqswsdl takes a web service written for .NET Framework 1 or 2, and generates the WSDL for the class, inserting the URI you provide for the IBM MQ transport for SOAP into the generated WSDL.

Purpose

Use **amqswsdl** to generate WSDL containing the URI of the service deployed to IBM MQ. Use the WSDL to generate client proxies.

Parameters

escapedUri (Input)
The URI of the service, with all "&" escaped to "&amp.". For example:

```
"jms:/queue?destination=REQUESTDOTNET
&amp.initialContextFactory=com.ibm.mq.jms.Nojndi
&amp.connectionFactory=(connectQueueManager(QM1)binding(server))
&amp.targetService=Quote.asmx"
```

className.asmx (Input)
The service class.

className.wsdl (Output)
The service WSDL.

Description

If the class is implemented using the code-behind programming model, you must build *className*.dll and store it in ./bin.

amqwclientconfig: create Axis 1.4 web services client deployment descriptor for IBM MQ transport for SOAP

amqwclientconfig creates the client-config.wsdd Axis 1.4 client deployment descriptor file.

Purpose

It adds the jms:/ transport to the descriptor, and registers java:com.ibm.mq.soap.transport.jms.WMQSender as the class to handle SOAP requests for the jms: transport.

Syntax

Description

amqwclientconfig calls **amqwsetcp** to set the CLASSPATH and runs the command:

```
java org.apache.axis.utils.Admin client "%WMQSOAP_HOME%\bin\amqwclientTransport.wsdd"
```

amqwdeployWMQService: deploy web service utility

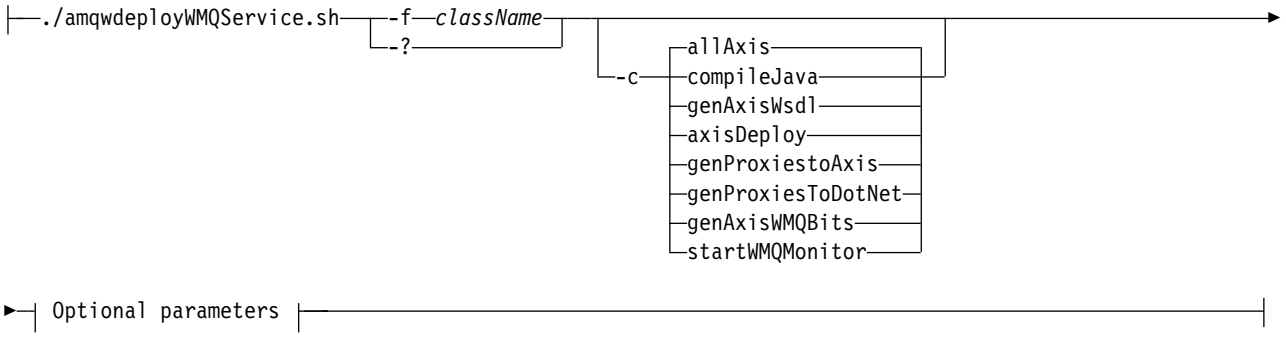
The deployment utility prepares a service class for use as a web service using IBM MQ as the transport.

Purpose

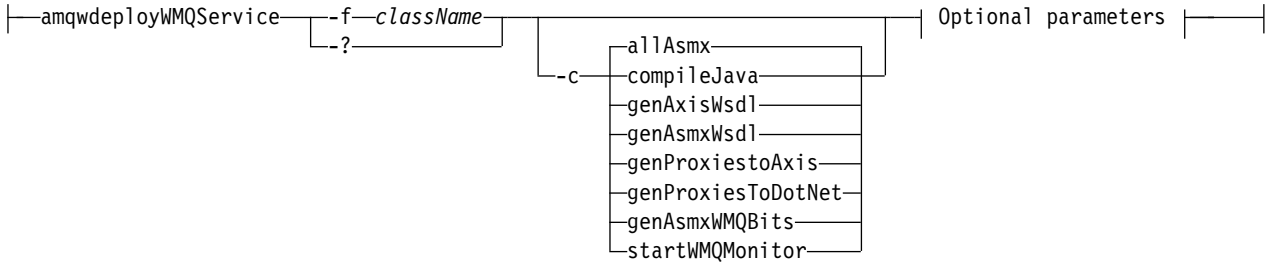
Use the deployment utility to generate the files that are needed to deploy an Axis 1.4, .NET Framework 1 or .NET Framework 2 service. Use the files to deploy a service invoked by IBM MQ. The files generated by **amqwdeployWMQService** are shown in "Output files from **amqwdeployWMQService**" on page 3492.

Syntax diagram

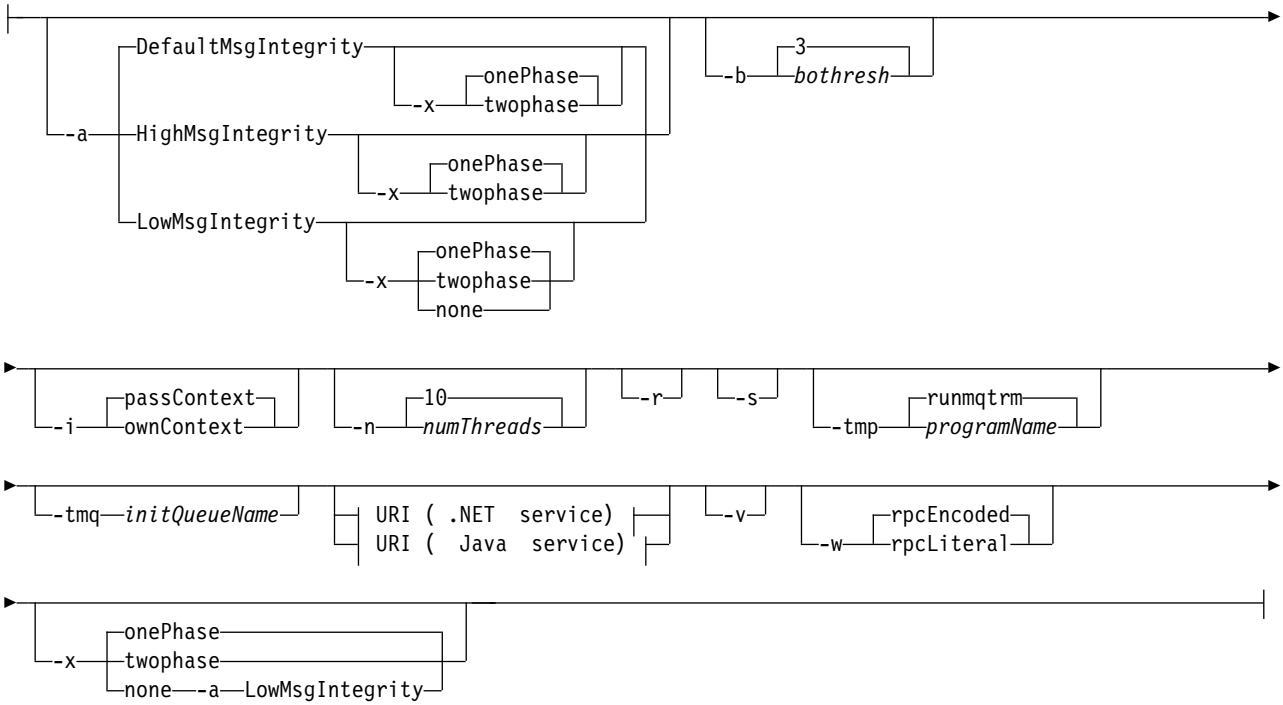
UNIX and Linux systems:



Windows:



Optional parameters:



Required parameters

-f *className*

className is the name of the class to be deployed. For Axis services *className* is the Java source file, and for .NET services, the `.asmx` file. Figure 62 illustrates the deployment of an Axis service and Figure 63 of a .NET service.

```
amqwdeployWMQService -f javaDemos/service/StockQuoteAxis.java
```

Figure 62. Example deployment of Axis service

```
amqwdeployWMQService -f StockQuoteDotNet.asmx
```

Figure 63. Example deployment of .NET service

For Java, *className* must be fully qualified by the package name. It can be specified as a path name with directory separators or as a class name with period separators. The generated class is located in `./generated/client/remote/`*path name*. For a .NET service, although the directory can be specified, generated Java proxies are always located in `./generated/client/remote/dotNetService`.

If you specify a URI with the -u option and within the URI specify *targetService*, the deployment utility checks the *className*. *className* must match *targetService*. If the class and service do not match, the deployment utility displays an error message and exits.

-? Print out help text describing how the command is used.

Optional parameters

-a *integrityOption*

integrityOption specifies the behavior of IBM MQ SOAP listeners if it is not possible to put a failed request message on the dead-letter queue. *integrityOption* can take one of the following values:

DefaultMsgIntegrity

For non-persistent messages, the listener displays a warning message and continues to execute with the original message being discarded. For persistent messages, it displays an

error message, backs out the request message so it remains on the request queue and exits. DefaultMsgIntegrity applies if the -a option is omitted, or if *integrityOption* is not specified.

LowMsgIntegrity
For both persistent and non-persistent messages, the listener displays a warning and continues to execute, discarding the message.

HighMsgIntegrity
For both persistent and non-persistent messages, the listener displays an error message, backs out the request message so it remains on the request queue and exits.

The deployment utility checks for the compatibility of the -x and -a flags. If -x none is specified, then -a LowMsgIntegrity must be specified. If the flags are incompatible the deployment utility exits with an error message and with no deployment steps having been undertaken.

-b *bothresh*
bothresh specifies the backout threshold setting for the request queue. The default is 3.

-c *operation*
operation specifies which part of the deployment process to execute. *operation* is one of the following options:

allAxis
Perform all compile and setup steps for an Axis or Java service[4] .

compileJava
Compile the Java service: `.java` to `.class`.

genAxisWsdl
Generate WSDL: `.class` to `.wsdl`.

axisDeploy
Deploy the class file: `.wsdl` to `.wsdd`, apply `.wsdd`.

genProxiestoAxis
Generate proxies: `.wsdl` to `.java` and `.class`.

genAxisWMQBits
Set up IBM MQ queues, IBM MQ SOAP listeners and triggers for an Axis service.

allAsmx
Perform all setup steps for a .NET service[5] .

genAsmxWsdl
Generate WSDL: `.asmx` to `.wsdl`.

genProxiesToDotNet
Generate proxies: `.wsdl` to `.java`, `.class`, `.cs` and `.vb`.

genAsmxWMQBits
Set up IBM MQ queues, IBM MQ SOAP listeners and triggers

startWMQMonitor
Start the trigger monitor for IBM MQ SOAP services.

Note: **runmqtrm** runs under the `mqm` user ID. If security is an issue, you must ensure that the listeners are started under appropriate user IDs.

-i *Context*
Context specifies whether the listeners pass identity context. *Context* takes the following values:

4. Default if *className* has a .java extension

5. Default if *className* has a .asmx extension.

passContext
Set the identity context of the original request message into the response message. The SOAP listener checks that it has authority to save the context from the request queue and to pass it to the response queue. It makes the checks at run time when opening the request queue to save context, and the response queue to pass context. If it does not have the required authority, or the MQOPEN call fails, and the response message is not processed. The response message is put on the dead-letter queue with the dead-letter header containing the return code from the failed MQOPEN. The listener then continues to process subsequent incoming messages as normal.

ownContext
The SOAP listener does not pass context. The returned context reflects the user ID under which the listener is running rather than the user ID which created the original request message.

The fields in the origin context are set by the queue manager, and not by the SOAP listener.

-n *numThreads*
numThreads specifies the number of threads in the generated startup scripts for the IBM MQ SOAP listener. The default is 10. Consider increasing this number if you have high message throughput.

-r *-r* specifies that any existing request or trigger monitor queue definitions are replaced. Trigger monitor queues are replaced only if -tmq is also specified. Queues are re-created with standard default attributes and existing messages on the queues are deleted. If the -r option is not used then any existing queue definitions are not altered and existing messages are not deleted. By not specifying -r, you ensure that any customized queue attributes are preserved.

-s Configure the listener to run as an IBM MQ service. If -s and -tmq are both specified, the deployment utility displays an error message and exits.

-tmp *programName*
programName specifies the name of a trigger monitor program. Use `-tmp` *programName* in a UNIX or Linux environment as an alternative to using **runmqtrm**. Programs it initiates run under `mqm` authority.

For example:

```
amqwdeployWMQService -f javaDemos/service/StockQuoteAxis.java
-tmq trigger.monitor.queue -tmp trigmon
```

-tmq *queueName*
queueName specifies a trigger monitor queue name. IBM MQ process definitions are created to configure automatic triggering of IBM MQ SOAP listeners with the associated trigger monitor queue name. If the option is not specified then no triggering configuration is defined by the deployment utility. If -s and -tmq are both specified, the deployment utility displays an error message and exits.

URI *platform*
See “URI syntax and parameters for web service deployment” on page 3518.

-v -v sets verbose output from external commands. Error messages are always displayed. Use -v to output commands that you can tailor to create customized deployment scripts.

-w -w controls the style of WSDL to generate. The default is rpcEncloded, for compatibility with previous releases of IBM MQ transport for SOAP. Use rpcLiteral to create WSDL compatible with Axis2 client proxy generation. rpcEncoded is not compatible with WS-I recommendations.

-x *transactionality*
transactionality specifies the type of transactional control for the listener. *transactionality* can be set to one of the following values:

onePhase
IBM MQ one-phase support is used. If the system fails during processing, the request message is redelivered to the application. IBM MQ transactions ensure that the response messages are written exactly once.

twoPhase
Two-phase support is used. If the service is written appropriately the message is delivered exactly once, coordinated with other resources, within a single committed execution of the service. This option applies to server bindings connections only.

none No transactional support. If the system fails during processing, the request message can be lost, even if persistent. The service might or might not have executed, and response, report or dead-letter messages might or might not be written.

The deployment utility checks for the compatibility of the -x and -a flags. See the description of the -a flag for details.

Errors

On Windows, if errors are reported from **amqswsdl**, try issuing the following command to register `.asmx` files as services.

```
%windir%/Microsoft.NET/Framework/ version number /aspnet_regiis.exe -ir
```

The problem typically occurs on systems where IIS has not been installed, or IIS has been installed after NET. The problem is encountered when **amqswsdl** generates the `.wsdl` files.

Note: The registry keys are also required to permit the listener to invoke the services. If you use your own customized deployment procedures, you might not encounter the problem until run time.

Output files from amqwdeployWMQService:

A list of the directories and files output from **amqwdeployWMQService**

Table 376. Output files from **amqwdeployWMQService**

Outputs	Description	Output directory	Filename
`.class`	Compiled Java source file	`./generated/server/`*server package*	*classname*`.class`
`.wsdl`	Service description	`./generated`	*className* `Axis_Wmq.wsdl` *className* `DotNet_Wmq.wsdl`
`.wsdd`	Axis client and service deployment files	`./`	`client-config.wsdd` `server-config.wsdd`
		`./generated/server/`*server package*	*className*`_deploy.wsdd` *className*`_undeploy.wsdd`
Client source (`.vb`, `.cs`, `.java`)	.Net client stubs to Axis service	`./generated/client`	*classname* `AxisService.cs` *classname* `AxisService.vb`
	.Net client stubs to .Net service	`./generated/client`	*classname* `DotNet.cs` *classname* `DotNet.vb`

Table 376. Output files from **amqwdeployWMQService** *(continued)*

Outputs	Description	Output directory	Filename
Client helper (.java and .class)	Java client proxies to .Net service	./generated/server/soap/client/remote/dotnetService	*className* DotNet.class *className* DotNet.java *className* DotNetLocator.class *className* DotNetLocator.java *className* DotNetSoap12Stub.class *className* DotNetSoap12Stub.java *className* DotNetSoap_BindingStub.class *className* DotNetSoap_BindingStub.java *className* DotNetSoap_PortType.class *className* DotNetSoap_PortType.java
	Java client proxies to Axis service	./generated/server/soap/client/remote/*client package*	SoapServer *className* AxisBindingSoapStub.class SoapServer *className* AxisBindingSoapStub.java *className* Axis.class *className* Axis.java *className* AxisService.class *className* AxisService.java *className* AxisServiceLocator.class *className* AxisServiceLocator.java
Scripts (.cmd and .sh)	Listener scripts	/generated/server	startWMQJListener.cmd startWMQJListener.sh startWMQNListener.cmd endWMQJListener.cmd endWMQJListener.sh endWMQNListener.cmd

Usage notes for amqwdeployWMQService:

Describes the tasks performed by **amqwdeployWMQService**.

The deployment utility performs the following actions.

1. Checks paths to the following files:
   - axis.jar.
   - *WMQSOAP_HOME*/java/lib/com.ibm.mq.soap.jar.
   - On Windows, csc.exe
2. On Windows, uses either %SystemRoot%\Microsoft.NET\Framework\v1.1.432 or, if the C# compiler is installed, the path to csc.exe as the path to the .NET Framework.

   Note: If you have Microsoft Visual Studio 2008 installed (Version 9), wsdl.exe is not in the path to csc.exe. You need to add the path to the .NET framework to your Path variable; for example:

   ```
   Set Path=C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727;%Path%
   ```
3. Creates the ./generated directory, and required subdirectories, if they do not exist.
4. For Java services, compiles the source into *className*.class.
5. Generates WSDL.
6. For Java services, creates deployment descriptor files *className*_deploy.wsdd and *className*_undeploy.wsdd
7. For Java services, creates or updates the Axis deployment descriptor file, server-config.wsdd.
8. Generates the client proxies for Java, C# and Visual Basic from the WSDL.

   Note: On Windows, the deploy utility generates proxies for Visual Basic and C# regardless of the language in which the service is written. The WSDL and the proxies generated from it include the appropriate URI to call the service:

a.

```
jms:/queue?destination=SOAPN.demos@WMQSOAP.DEMO.QM
&connectionFactory=(connectQueueManager(WMQSOAP.DEMO.QM))
&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=StockQuoteDotNet.asmx
&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE
```

Figure 64. Example URI in generated .NET client to call .NET service

b.

```
jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM
&connectionFactory=(connectQueueManager(WMQSOAP.DEMO.QM))
&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=soap.server.StockQuoteAxis.java
&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE
```

Figure 65. Example URI in generated .NET client to call Axis 1 service

9. Compiles the Java proxies.
10. Creates an IBM MQ queue, *requestQueue* to hold requests for the service. The default queue name is of the form SOAPJ. *directory*, or you can specify *requestQueue* in the -u URI option.
11. Creates command and shell script files to start the IBM MQ SOAP listeners that process the request queue.
12. If the -tmq option has been used, the deployment utility creates IBM MQ definitions to trigger IBM MQ SOAP listener processes automatically.
    - The deployment utility uses the APPLICID attribute of the **runmqsc** DEFINE PROCESS command to contain a command to start the listener. The command has the name of the deployment directory embedded in it. The APPLICID field has a maximum length of 256, which limits the maximum length of the deployment directory. The directory limit for Java services is as follows:
      – On UNIX and Linux: 218
      – On Windows: 197 minus the length of the request queue name.

      For .NET services, the directory limit is as follows:
      – On Windows: 209 minus length of the service name, minus the .asmx extension.
    - The deployment utility checks whether the limit for APPLICID is exceeded. If the limit is exceeded, the utility does not attempt to define the triggering process. It displays an error message and the deployment process fails without performing any deployment steps.

    The following examples show the configuration and start commands generated by the deployment utility to start an IBM MQ SOAP listener.

```
DEFINE PROCESS(requestQueue) APPLICID(applicIDStr) REPLACE
ALTER  QLOCAL (requestQueue) TRIGTYPE(FIRST) TRIGGER
       PROCESS(requestQueue) INITQ(initQueueName) TRIGMPRI(0)
```

Figure 66. IBM MQ configuration commands to trigger a SOAP listener.

```
applicIDStr = start "Java WMQSoapListener -requestQueue"
                    /min .\generated\server\startWMQJListener.cmd;
```

Figure 67. Starting Axis SOAP listener on Windows

```
applicIDStr = start "WMQAsmxListener -className\
                    /min .\generated\server\startWMQNListener.cmd;
```

Figure 68. Starting .NET SOAP listener on Windows

```
applicIDStr = xterm -iconic -T \"Java WMQSoapListener_requestQueue\"
                  -e ./generated/server/startWMQJListener.sh & #
```

Figure 69. Starting Axis SOAP listener on UNIX and Linux

amqwRegisterdotNet: register IBM MQ transport for SOAP to .NET

Register IBM MQ transport for SOAP to the global assembly cache on .NET.

Purpose

amqwRegisterdotNet registers the IBM MQ SOAP sender, SOAP listener, and WSDL processor with .NET Framework 1 or 2.

Syntax

Description

amqwRegisterdotNet is run automatically during installation. You do not need to run it again if the .NET Framework you are using was installed before IBM MQ transport for SOAP. You can run it as many times as you want. Use it to reregister IBM MQ transport for SOAP with different .NET Framework versions.

Note: On Windows 2003 Server, you must also run the **aspnet_regiis** utility, even if you are not deploying to Internet Information Server (IIS). The location of the `aspnet_regiis.exe` utility might vary with different versions of the Microsoft.NET Framework, but it is typically located in: `%SystemRoot%/Microsoft.NET/Framework/version number/aspnet_regiis`. If multiple versions are installed, use **aspnet_regiis** for the version of .NET Framework you are using.

Apache software license

Apache License Version 2.0, January 2004 http://www.apache.org/licenses/

http://www.apache.org/licenses/

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of

MQMD SOAP settings

The IBM MQ SOAP sender and IBM MQ SOAP listener create a message descriptor (**MQMD**). The topic describes the fields you must set in the MQMD if you create your own SOAP sender or listener.

Purpose

The values set in the **MQMD** control the exchange of messages between the IBM MQ SOAP sender, the IBM MQ SOAP listener, and the SOAP client program. If you create your own SOAP sender or listener, follow the rules in Table 377.

Description

Table 377 describes how the **MQMD** fields are set by the IBM MQ SOAP sender and IBM MQ SOAP listener. If you write your own sender or listener you must set these fields in accordance with the rules for exchanging messages. The IBM MQ SOAP listener conforms to typical IBM MQ message exchange protocols. If you write your own sender to work with the IBM MQ SOAP listeners, you can set different **MQMD** values.

In Table 377, the values in the Setting column are organized as follows:

Request, One way
Settings made by IBM MQ SOAP sender.

Response, Report
Settings made by IBM MQ SOAP listener in response to IBM MQ SOAP sender request.

ALL Settings made by both the IBM MQ SOAP sender and IBM MQ SOAP listener.

Customized sender
You can write your own sender. Typically, a customized sender overrides the standard report options.

Table 377. MQMD SOAP settings

Field name	Setting	Values
StrucId	**ALL** MQMD_STRUC_ID	'MD¬¬' [1]
Version	**ALL** MQMD_VERSION_2	2
Report	**ALL** MQRO_NONE + MQRO_NEW_MSG_ID + MQRO_COPY_MSG_ID_TO_CORREL_ID + MQRO_EXCEPTION + MQRO_EXPIRY + MQRO_DISCARD **Customized sender** See "Customized report options" on page 3501	52428800
MsgType	**Request** MQMT_REQUEST **Response** MQMT_REPLY **Report** MQMT_REPORT **One way** MQMT_DATAGRAM	**MQMT_REQUEST** 1 **MQMT_REPLY** 2 **MQMT_REPORT** 4 **MQMT_DATAGRAM** 8

Table 377. MQMD SOAP settings (continued)

Field name	Setting	Values
Expiry	**Request, One way** Specified by Expiry option in URI. Defaults to MQEI_UNLIMITED. **Response** Value of Expiry in request message **Report** MQEI_UNLIMITED	**MQEI_UNLIMITED** -1
Feedback	**Request, Response, One way** MQFB_NONE. **Report** • Generated by queue manager - value set according to normal rules. • Generated by IBM MQ SOAP Listener: **MQRC_BACKOUT_THRESHOLD_REACHED** Backout threshold for multiple attempts exceeded. **MQRCCF_MD_FORMAT_ERROR** The message is not recognized as having an MQRFH2 header. **MQRC_RFH_PARM_MISSING** A required parameter, for example, SoapAction, in **MQRFH2** is missing. **MQRC_RFH_FORMAT_ERROR** A basic integrity check of the **MQRFH2** failed, for example, internal lengths are corrupted. **MQRC_RFH_ERROR** The **MQRFH2** passed an integrity check, but the body of the message is not set to MQFMT_NONE.	**MQFB_NONE** 0 **MQRC_BACKOUT_THRESHOLD_REACHED** 2362 **MQRCCF_MD_FORMAT_ERROR** 3023 **MQRC_RFH_PARM_MISSING** 2339 **MQRC_RFH_FORMAT_ERROR** 2421 **MQRC_RFH_ERROR** 2334
Encoding	**ALL** MQENC_NATIVE	Depends on environment
CodedCharSetId	**ALL** Set to UTF-8	1208
Format	**Request, Response, One way** MQFMT_RF_HEADER_2 **Report** **Queue manager reports** Follows IBM MQ rules **IBM MQ SOAP listener reports** Format of the original request message.	**MQFMT_RF_HEADER_2** "MQRFH2 "

Table 377. MQMD SOAP settings (continued)

Field name	Setting	Values
Priority	**Request, One way** Specified by Priority option in URI. Defaults to MQPRI_PRIORITY_AS_Q_DEF. **Response, Report** Value of Priority in the request message.	**MQPRI_PRIORITY_AS_Q_DEF** -1
Persistence	**Request, One way** MQPER_PERSISTENCE_AS_Q_DEF. **Response, Report** Value of Persistence in the request message.	**MQPER_PERSISTENCE_AS_Q_DEF** 2
MsgId	**Request, One way** Generated by the queue manager. **Response, Report** The IBM MQ SOAP sender sets MQRO_NEW_MSG_ID and *MsgId* is generated	**Generated** Unique value generated by the queue manager
CorrelId	**Request, One way, Report** MQCI_NONE **Response, Report** The IBM MQ SOAP sender sets MQRO_COPY_MSG_ID_TO_CORREL_ID and the listener copies *MsgId* from the request message.	**MQCI_NONE** 0
BackoutCount	**ALL** Not used	0
ReplyToQ	**Request** Specified by replyDestination option in URI. Defaults to SYSTEM.SOAP.RESPONSE.QUEUE. **Response, One way, Report** Left blank	
ReplyToQMgr	**ALL** Field left blank	Generated by the queue manager; see Reply-to queue and queue manager.
UserIdentifier	**Request, One way, Report** Left blank **Response** Depends on the -i *passContext* option supplied to the listener, and the authority the listener is running under.	**Request, One way, Report** Generated by the queue manager; see "UserIdentifier (MQCHAR12)" on page 2436 **Response** *Variable*
AccountingToken	**ALL** MQACT_NONE	**MQACT_NONE** Null string or blanks Set by the queue manager; see "AccountingToken (MQBYTE32)" on page 2390

Table 377. MQMD SOAP settings (continued)

Field name	Setting	Values
ApplIdentityData	**ALL** None	Null string or blanks [2]
PutApplType	**ALL** MQAT_NO_CONTEXT	**MQAT_NO_CONTEXT** 0 Value generated by queue manager; see "PutApplType (MQLONG)" on page 2422.
PutApplName	**ALL** None	Value generated by queue manager; see "PutApplName (MQCHAR28)" on page 2421.
PutDate	**ALL** None	Value generated by queue manager; see "PutDate (MQCHAR8)" on page 2424.
PutTime	**ALL** None	Value generated by queue manager; see "PutTime (MQCHAR8)" on page 2424.
ApplOriginData	**ALL** None	Null string or blanks [2]
GroupId	**Request, One way, Report** MQGI_NONE **Response** Field is copied from the request message	Nulls
MsgSeqNumber	**Request, One way, Report** Not used **Response** Field is copied from the request message	Generated by the queue manager; see Physical order on a queue.
Offset	**Request, One way, Report** Not used **Response** Field is copied from the request message	0
MsgFlags	**Request, One way, Report** MQMF_NONE **Response** Field is copied from the request message	**MQMF_NONE** 0 See "MsgFlags (MQLONG)" on page 2410
OriginalLength	**Request, One way, Response** MQOL_UNDEFINED **Report** Length of original request message	**MQOL_UNDEFINED** -1

Notes:

1. The symbol ¬ represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.

Customized report options

You can write your own SOAP sender and use it with the supplied listeners. Typically you might write a sender to change the choice of report options. The IBM MQ SOAP listeners support most combinations of report options, as described in the following lists.

- Report options supported by IBM MQ SOAP listeners:
  - MQRO_EXCEPTION
  - MQRO_EXCEPTION_WITH_DATA
  - MQRO_EXCEPTION_WITH_FULL_DATA
  - MQRO_DEAD_LETTER_Q
  - MQRO_DISCARD_MSG
  - MQRO_NONE
  - MQRO_NEW_MSG_ID
  - MQRO_PASS_MSG_ID
  - MQRO_COPY_MSG_ID_TO_CORREL_ID
  - MQRO_PASS_CORREL_ID
- Report options supported by the queue manager:
  - MQRO_COA
  - MQRO_COA_WITH_DATA
  - MQRO_COA_WITH_FULL_DATA
  - MQRO_COD
  - MQRO_COD_WITH_DATA
  - MQRO_COD_WITH_FULL_DATA
  - MQRO_EXPIRATION
  - MQRO_EXPIRATION_WITH_DATA
  - MQRO_EXPIRATION_WITH_FULL_DATA
- The following report options are not supported by the IBM MQ SOAP listeners.
  - MQRO_PAN
  - MQRO_NAN

The behavior of IBM MQ SOAP listeners in response to combinations of MQRO_EXCEPTION_* and MQRO_DISCARD is described in Table 378.

The notation MQRO_EXCEPTION_* indicates the use of either MQRO_EXCEPTION, MQRO_EXCEPTION_WITH_DATA or MQRO_EXCEPTION_WITH_FULL_DATA.

Table 378. Listener behavior resulting from MQRO_EXCEPTION_* and MQRO_DISCARD settings

	MQRO_DISCARD enabled	**MQRO_DISCARD not enabled**
MQRO_EXCEPTION_* enabled	Default behavior. Report messages are automatically generated if necessary and the original request discarded. If a report message could not be returned to the response queue the report message is sent to the dead letter queue.	Report messages are automatically generated if necessary and the original message is sent to the dead letter queue. If the report message could not be returned to the response queue it is also be sent to the dead-letter queue. In this case there are therefore two dead letter queue entries for the failed request.

Table 378. Listener behavior resulting from MQRO_EXCEPTION_* and MQRO_DISCARD settings (continued)

	MQRO_DISCARD enabled	**MQRO_DISCARD not enabled**
MQRO_EXCEPTION_* not enabled	Report messages are not automatically generated when the incoming format is not recognized or the number of backout attempts is exceeded. The message is not sent to the dead letter queue. No notification is returned that the client can inspect, and the original request message is lost.	Report messages are not automatically generated when the incoming format is not recognized or the number of backout attempts is exceeded. The original request message is however written to the dead letter queue when a report would otherwise have been generated.

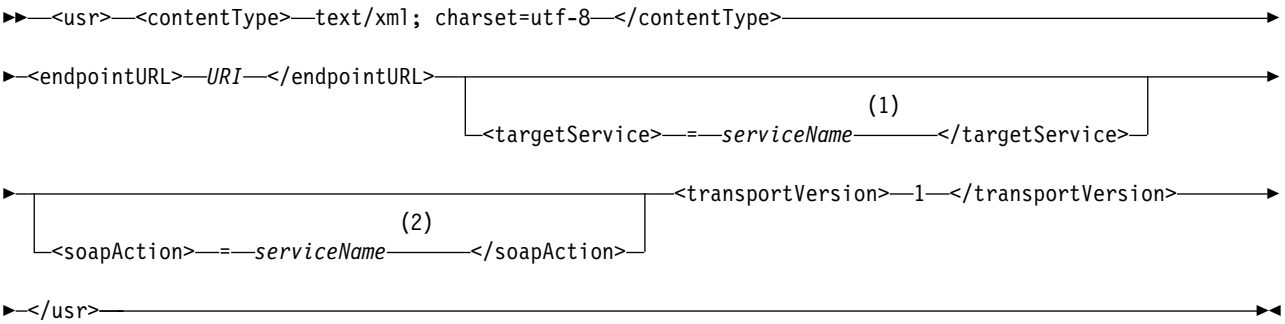
MQRFH2 SOAP settings

The IBM MQ SOAP senders and listeners create or expect to receive an MQRFH2 with the following settings.

Purpose

The IBM MQ SOAP senders add properties to the <usr> folder created by IBM MQ JMS. The properties contain information required by the SOAP container in the target environment. "Property syntax" describes the syntax of the properties when they are added to an MQRFH2. For the description an MQRFH2 header, see MQRFH2 - Rules and formatting header 2.

Property syntax



Notes:

1 targetService is required for .NET Framework 1 or 2, and not used on Axis 1.4.

2 soapAction is optional for .NET Framework 1 or 2, and not used on Axis 1.4.

Parameters

contentType
contentType always contains the string text/xml; charset=utf-8.

endpointURL
See "URI syntax and parameters for web service deployment" on page 3518.

targetService
[6]On Axis, *serviceName* is the fully qualified name of a Java service, for example: targetService=javaDemos.service.StockQuoteAxis. If targetService is not specified, a service is loaded using the default Axis mechanism.

6. Java service only

[7]On .NET, *serviceName* is the name of a .NET service located in the deployment directory, for example: `targetService=myService.asmx`. In the .NET environment, the targetService parameter makes it possible for a single IBM MQ SOAP listener to be able to process requests for multiple services. These services must be deployed from the same directory.

soapAction

transportVersion
transportVersion is always set to 1.

Example

The example shows an `MQRFH2` and the following SOAP message. The lengths of the folders are shown in decimal.

Note: `&` in the URI is encoded as `&`

```
52464820 00000002 000002B0 00000001 RFH/ 0002 1208 0001
000004B8 20202020 20202020 00000000 1208 ? ? ? ? ? ? ? 0000
000004B8              1208
32 <mcd>
<Msd>jms_bytes</Msd>
</mcd>?
208 <jms>
<Dst>queue://queue://SOAPJ.demos</Dst>
<Rto>queue://WMQSOAP.DEMO.QM/SYSTEM.SOAP.RESPONSE.QUEUE</Rto>
<Tms>1157388516465</Tms>
<Cid>ID:000000000000000000000000000000000000000000000000</Cid>
<Dlv>1</Dlv>
</jms>
400 <usr>
<contentType>text/xml; charset=utf-8</contentType>
<transportVersion>1</transportVersion>
<endpointURL>
jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM
&amp;connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)
clientConnection(localhost%25289414%2529)
clientChannel(TESTCHANNEL)
&amp;replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE
&amp;initialContextFactory=com.ibm.mq.jms.Nojndi
</endpointURL>
</usr>
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
<ns1:getQuote
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="soap.server.StockQuoteAxis_Wmq">
<in0 xsi:type="xsd:string">XXX</in0>
</ns1:getQuote>
</soapenv:Body>
</soapenv:Envelope>
```

7. .NET service only

runivt: IBM MQ transport for SOAP installation verification test

An installation verification test suite (IVT) is provided with IBM MQ transport for SOAP. **runivt** runs a number of demonstration applications and ensures that the environment is correctly set up after installation.

Purpose

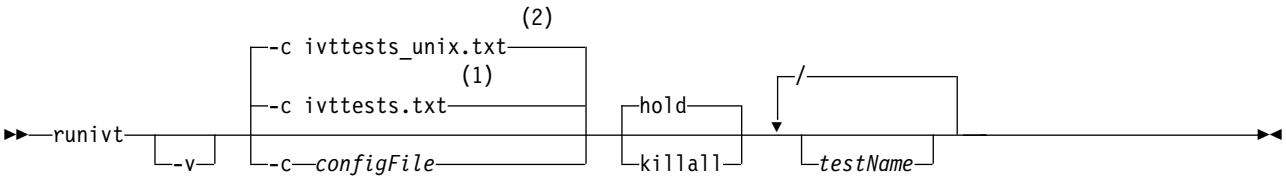
The **runivt** command uses the sample programs provided with the IBM MQ transport for SOAP to send web service requests from clients to services. It runs tests for Axis 1.4, .NET Framework 1, and .NET Framework 2. The tests are configured in a test script file. The default test script file for Windows runs a combination of tests between Java and .NET clients and services.

Description

runivt must be run from its own directory.

The command starts listeners in a different command window. For this reason, you must run the command from an X Window System session on UNIX and Linux systems.

runivt syntax



Notes:

1 Default on Windows

2 Default on UNIX and Linux systems

runivt parameters

-v Verbose mode. Write fuller error messages to the console.

-c *configFile*
A configuration file defining the tests to be run. The default configuration file supplied with Windows, UNIX or Linux systems is used by default.

hold
Leave the listener running after the tests complete

killall
End the listener when the tests complete

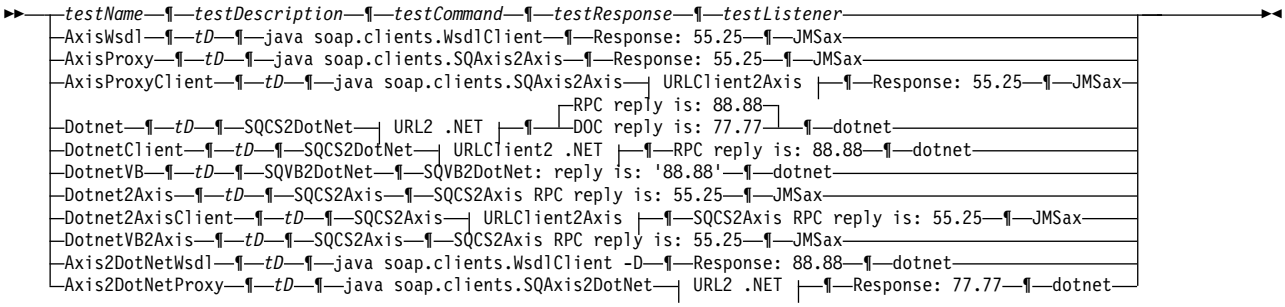
testName
A space separated list of the tests to run. The test names are selected from the configuration file. If no names are specified then all the tests in the configuration file are run.

Configuration file

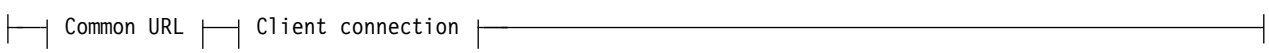
Each configuration file parameter is a separate line of the file. Leave a blank line between each group of parameters.

The parameters in the `ivttests.txt` parameter file are listed.

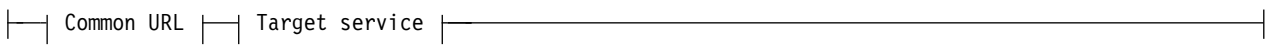
configFile syntax



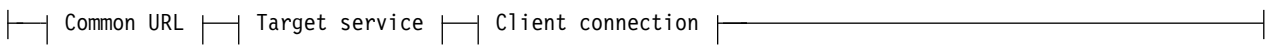
URLClient2Axis:



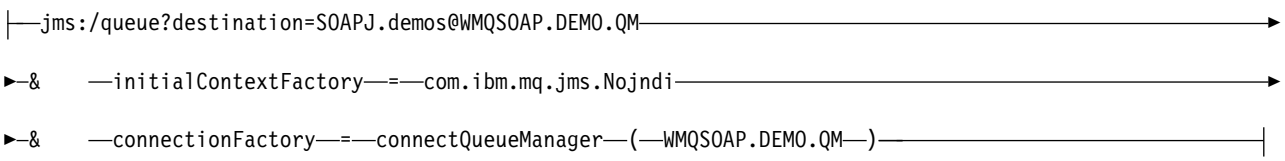
URL2 .NET:



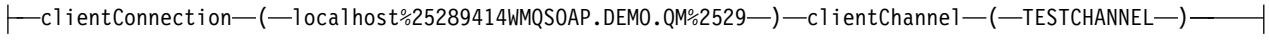
URLClient2 .NET:



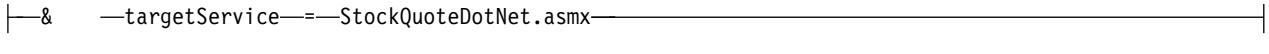
Common URL:



Client connection:



Target service:



configFile parameters

testName
The name of the test. Use *testName* in the **runivt** command

testDescription
Documentation about the test

testCommand
The command executed by the **runivt** command to make the client request.

testResponse
The exact response string returned by the client request to the console. For the test to succeed *testResponse* must match the actual response.

testListener
The name of the IBM MQ SOAP listener that is started by **runivt** to process the SOAP request. **dotnet** and **JMSax** are synonyms for the supplied listeners, **amqwSOAPNETlistener** and **SimpleJavaListener**.

Examples

```
runivt
```

Figure 70. run all the default tests

```
runivt dotnet
```

Figure 71. run a specific test from the default tests

```
runivt -c mytests.txt
```

Figure 72. run a set of custom tests

Related information:
Verifying the IBM MQ transport for SOAP

Secure web services over IBM MQ transport for SOAP

You might secure web services that use the IBM MQ transport for SOAP in one of two ways. Either create a TLS channel between the client and server, or use web services security.

TLS and the IBM MQ transport for SOAP:

The IBM MQ transport for SOAP provides several TLS options that can be specified for use with client channel configured to run in TLS mode. The options differ between the .NET and Java environments. The IBM MQ SOAP senders and listeners process only the TLS options that are applicable to their particular environment. They ignore options that are not applicable.

The presence or absence of the `sslCipherSpec` option for .NET clients and the `sslCipherSuite` option for Java clients determines whether TLS is used or not. If the option is not specified in the URI then by default TLS is not used and all other TLS options are ignored. All TLS options are optional except where indicated.

For IBM MQ clients, set the TLS attributes in the URI or channel definition table. On the server, set the attributes using the facilities of IBM MQ.

By default, the standard IBM MQ TLS option, `SSLCAUTH`, is set when enabling TLS on the channel. Clients must authenticate themselves before TLS communication can commence. If `SSLCAUTH` is not set, TLS communications are established without client authentication.

To authenticate themselves, clients must have a certificate assigned in their key repository that is acceptable to the queue manager. For additional security, IBM MQ channels can be configured to accept only certificates from a restricted list. The list is restricted by checking the distinguished name of the certificate against the peer name attribute of the channel.

If you use Java, the first TLS connection from an IBM MQ SOAP client causes the following TLS parameters to be fixed. The same values are used in subsequent connections using the same client process:

- `sslKeyStore`
- `sslKeyStorePassword`
- `sslTrustStore`
- `sslTrustStorePassword`
- `sslFipsRequired`
- `sslLDAPCRLservers`

The effect of varying these parameters on subsequent connections from this client is undefined.

If you use .NET, the first TLS connection from an IBM MQ SOAP client causes the following TLS parameters to be fixed. The same values are used in subsequent connections using the same client process:

- `sslKeyRepository`
- `sslCryptoHardware`
- `sslFipsRequired`
- `sslLDAPCRLservers`

The effect of varying these parameters on subsequent connections from this client is undefined. These parameters are reset if all TLS connections become inactive and a new TLS connection is made.

The following properties can also be specified as system properties:

- `sslKeyStore`
- `sslKeyStorePassword`
- `sslTrustStore`
- `sslTrustStorePassword`

If they are specified both as system properties and in the URI, and the values differ, the deployment utility displays a warning. The URI values take precedence.

Related information:

Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client

Federal Information Processing Standards (FIPS) for UNIX, Linux, and Windows

SSL connection factory parameters in the IBM MQ web services URI:

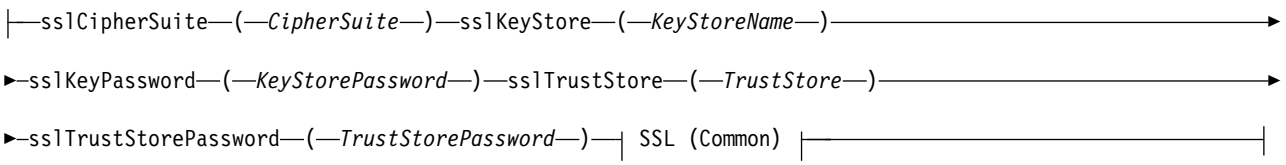
Add TLS options to the list of connection factory options in the IBM MQ web services URI.

Purpose

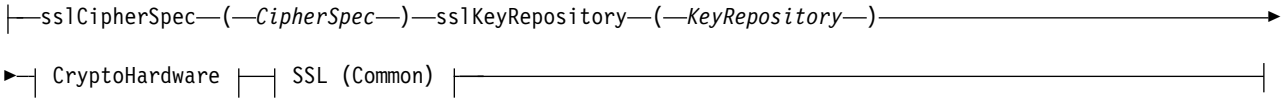
You can use a secure connection between an IBM MQ web services client and the queue manager hosting the web service. The SSL options control how TLS is configured on the IBM MQ MQI client-server channel connection.

Syntax diagram

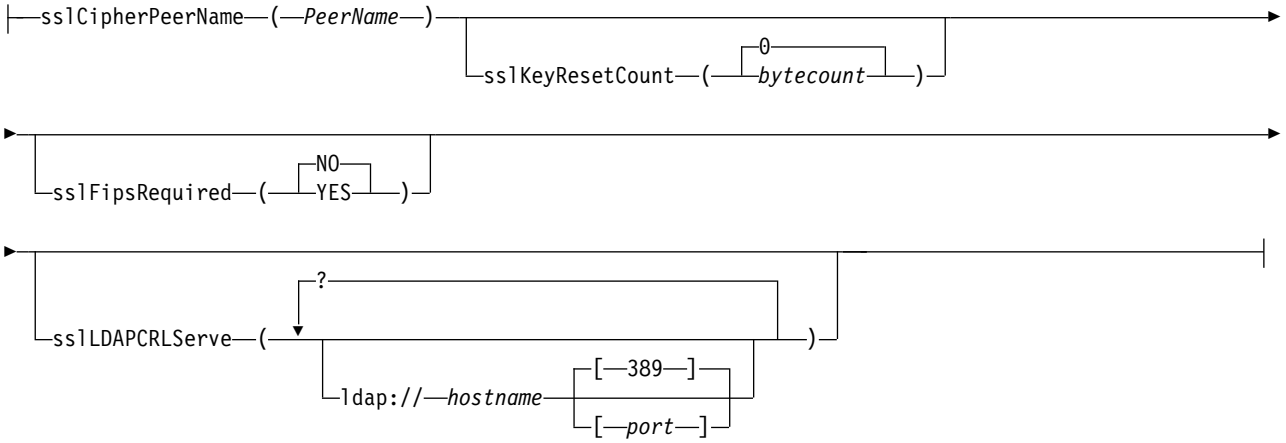
SSL (Java):



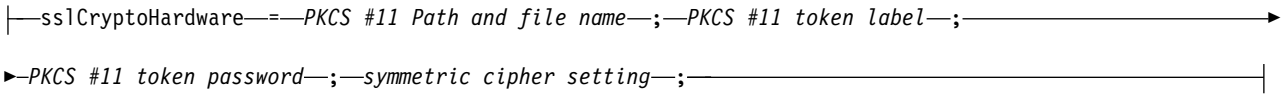
SSL (.NET):



SSL (Common):



CryptoHardware:



Required SSL parameters (Common)

sslPeerName (*peerName*)
peerName specifies the sslPeerName used on the channel.

Required SSL parameters (Java)

sslCipherSuite (*CipherSuite*)
CipherSuite specifies the sslCipherSuite used on the channel. The CipherSuite specified by the client must match the CipherSuite specified on the server connection channel.

sslKeyStore (*KeyStoreName*)
KeyStoreName specifies the sslKeyStoreName used on the channel. The keystore holds the private key

of the client used to authenticate the client to the server. The keystore is optional if the TLS connection is configured to accept anonymous client connections.

sslKeyStorePassword (*KeyStorePassword*)
KeyStorePassword specifies the sslKeyStorePassword used on the channel.

sslTrustStore (*TrustStoreName*)
TrustStoreName specifies the sslTrustStoreName used on the channel. The truststore holds the public certificate of the server, or its key chain, to authenticate the server to the client. The truststore is optional if the root certificate of a certificate authority is used to authenticate the server. In Java, root certificates are held in the JRE certificate store, `cacerts`.

sslTrustStorePassword (*TrustStorePassword*)
TrustStorePassword specifies the sslTrustStorePassword used on the channel.

Required SSL parameters (.NET)

sslCipherSpec (*CipherSpec*)
CipherSpec specifies the sslCipherSpec used on the channel. If the option is specified then TLS is used on the client channel.

sslKeyRepository (*KeyRepository*)
KeyRepository specifies the sslCipherSpec used on the channel where TLS keys and certificates are stored. *KeyRepository* is specified in stem format, that is, a full path with file name but with the file extension omitted. The effect of setting sslKeyRepository is the same as setting the `KeyRepository` field in the **MQSCO** structure on an MQCONNX call.

Optional SSL parameters (.NET)

sslCryptoHardware (*CryptoHardware*)
CryptoHardware specifies the sslCryptoHardware used on the channel. The possible values for this field, and the effect of setting it, are the same as for the `CryptoHardware` field of the **MQSCO** structure on an MQCONNX.

Optional SSL parameters (Common)

sslKeyResetCount (*bytecount*)
bytecount specifies the number of bytes passed across a TLS channel before the TLS secret key must be renegotiated. To disable the renegotiation of TLS keys omit the field or set it to zero. Zero is the only value supported in some environments, see Renegotiating the secret key in IBM MQ classes for Java . The effect of setting sslKeyResetCount is the same as setting the `KeyResetCount` field in the **MQSCO** structure on an MQCONNX call.

sslFipsRequired (*fipsCertified*)
fipsCertified specifies whether *CipherSpec* or *CipherSuite* must use FIPS-certified cryptography in IBM MQ on the channel. The effect of setting *fipsCertified* is the same as setting the `FipsRequired` field of the **MQSCO** structure on an MQCONNX call.

sslLDAPCRLServers (*LDAPServerList*)

LDAPServerList specifies a list of LDAP servers to be used for Certificate Revocation List checking.

For TLS enabled client connections, *LDAPServerList* is a list of LDAP servers to be used for Certificate Revocation List (CRL) checking. The certificate provided by the queue manager is checked against one of the listed LDAP CRL servers; if found, the connection fails. Each LDAP server is tried in turn until connectivity is established to one of them. If it is impossible to connect to any of the servers, the certificate is rejected. Once a connection has been successfully established to one of them, the certificate is accepted or rejected depending on the CRLs present on that LDAP server.

If *LDAPServerList* is blank, the certificate belonging to the queue manager is not checked against a Certificate Revocation List. An error message is displayed if the supplied list of LDAP URIs is not valid. The effect of setting this field is the same as that of including `MQAIR` records and accessing them from an **MQSCO** structure on an MQCONNX.

Related information:

Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client

Federal Information Processing Standards (FIPS) for UNIX, Linux, and Windows

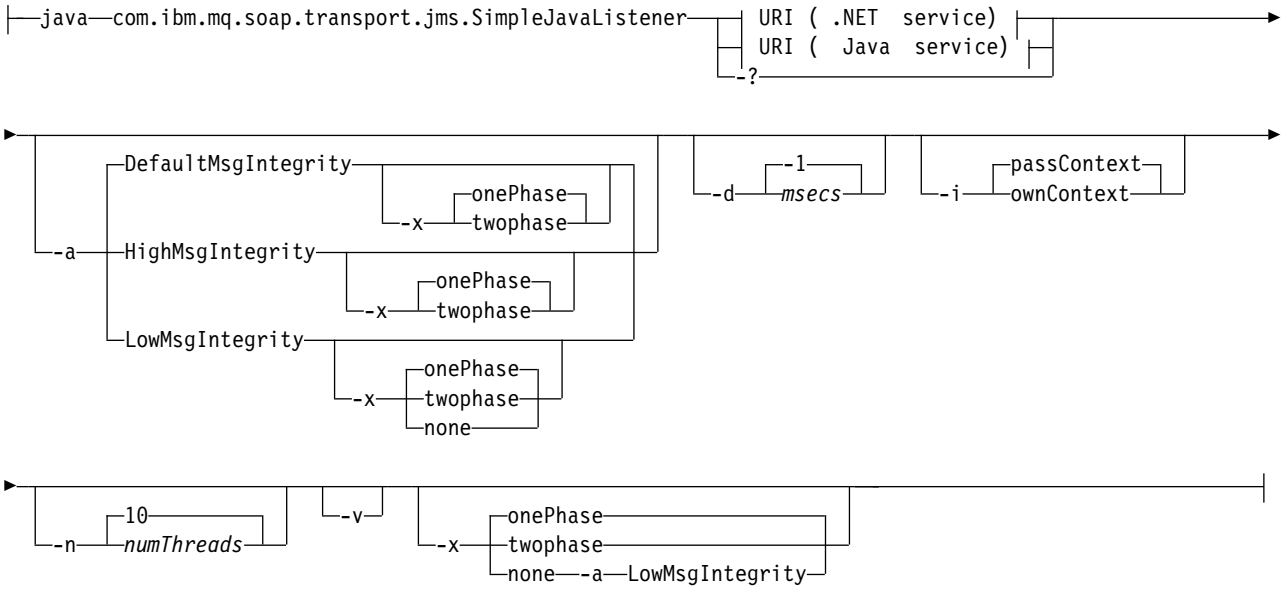
`SimpleJavaListener`: IBM MQ SOAP Listener for Axis 1.4

Syntax and parameters for the IBM MQ SOAP listener for Axis 1.4.

Purpose

Starts the IBM MQ SOAP listener for Axis 1.4.

Java:



Required parameters

URI *platform*
See "URI syntax and parameters for web service deployment" on page 3518.

-? Print out help text describing how the command is used.

Optional parameters

-a *integrityOption*
integrityOption specifies the behavior of IBM MQ SOAP listeners if it is not possible to put a failed request message on the dead-letter queue. *integrityOption* can take one of the following values:

DefaultMsgIntegrity
For non-persistent messages, the listener displays a warning message and continues to execute with the original message being discarded. For persistent messages, it displays an error message, backs out the request message so it remains on the request queue and exits. DefaultMsgIntegrity applies if the -a option is omitted, or if *integrityOption* is not specified.

LowMsgIntegrity
For both persistent and non-persistent messages, the listener displays a warning and continues to execute, discarding the message.

HighMsgIntegrity
For both persistent and non-persistent messages, the listener displays an error message, backs out the request message so it remains on the request queue and exits.

The deployment utility checks for the compatibility of the -x and -a flags. If -x none is specified, then -a LowMsgIntegrity must be specified. If the flags are incompatible the deployment utility exits with an error message and with no deployment steps having been undertaken.

-d *msecs*
msecs specifies the number of milliseconds for the IBM MQ SOAP listener to stay alive if request messages have been received on any thread. If *msecs* is set to -1, the listener stays alive indefinitely.

-i *Context*
Context specifies whether the listeners pass identity context. *Context* takes the following values:

passContext
Set the identity context of the original request message into the response message. The SOAP listener checks that it has authority to save the context from the request queue and to pass it to the response queue. It makes the checks at run time when opening the request queue to save context, and the response queue to pass context. If it does not have the required authority, or the MQOPEN call fails, and the response message is not processed. The response message is put on the dead-letter queue with the dead-letter header containing the return code from the failed MQOPEN. The listener then continues to process subsequent incoming messages as normal.

ownContext
The SOAP listener does not pass context. The returned context reflects the user ID under which the listener is running rather than the user ID which created the original request message.

The fields in the origin context are set by the queue manager, and not by the SOAP listener.

-n *numThreads*
numThreads specifies the number of threads in the generated startup scripts for the IBM MQ SOAP listener. The default is 10. Consider increasing this number if you have high message throughput.

-v -v sets verbose output from external commands. Error messages are always displayed. Use -v to output commands that you can tailor to create customized deployment scripts.

-w *serviceDirectory*
serviceDirectory is the directory containing the web service.

-x *transactionality*
transactionality specifies the type of transactional control for the listener. *transactionality* can be set to one of the following values:

onePhase
IBM MQ one-phase support is used. If the system fails during processing, the request message is redelivered to the application. IBM MQ transactions ensure that the response messages are written exactly once.

twoPhase
Two-phase support is used. If the service is written appropriately the message is delivered exactly once, coordinated with other resources, within a single committed execution of the service. This option applies to server bindings connections only.

none No transactional support. If the system fails during processing, the request message can be

lost, even if persistent. The service might or might not have executed, and response, report or dead-letter messages might or might not be written.

The deployment utility checks for the compatibility of the -x and -a flags. See the description of the -a flag for details.

Java Example

```
java com.ibm.mq.soap.transport.jms.SimpleJavaListener
-u "jms:/queue?destination=myQ&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi"
-n 20
```

IBM MQ SOAP listeners

An IBM MQ SOAP listener reads an incoming SOAP request from the queue specified as the destination in the URI. It checks the format of the request message and then invokes a web service using the web services infrastructure. An IBM MQ SOAP listener returns any response or error from a web service using the reply destination queue in the URI. It returns IBM MQ reports to the reply queue.

The term listener is used here in its standard web services sense. It is distinct from the standard IBM MQ listener invoked by the **runmqlsr** command.

Description

The Java SOAP listener is implemented as a Java class and run services using Axis 1.4. The .NET listener is a console application and runs .NET Framework 1 or .NET Framework 2 services. For .NET Framework 3 services, use the IBM MQ custom channel for Microsoft Windows Communication Foundation (WCF).

The deployment utility creates scripts to start Java or .NET SOAP listeners automatically. A SOAP Listener can be started manually using either the **amqSOAPNETListener** command, or by calling the SimpleJavaListener class. You can configure the IBM MQ SOAP listener to be started as an IBM MQ service by setting the -s option in the deployment utility. Alternatively, start listeners using triggering, or use the start and end listener scripts generated by the deployment utility. You can configure triggering manually, or use the -tmq and -tmp deployment options to configure triggering automatically. You can end a listener by setting the request queue to `GET (DISABLED)`.

Table 379. Command scripts generated by the deployment utility

Web service Infrastructure	UNIX and Linux systems	Windows Java	Windows.NET
Start listener	**startWMQJListener.sh**	**startWMQJListener.cmd**	**startWMQNListener.cmd**
Stop listener	**endWMQJListener.sh**	**endWMQJListener.cmd**	**endWMQNListener.cmd**
Define listener service	**defineWMQJListener.sh**	**defineWMQJListener.cmd**	**defineWMQNListener.cmd**

The IBM MQ SOAP listener passes the `endpointURL` and `soapAction` fields from the SOAP message to the SOAP infrastructure. The listener invokes the service through the Web Services infrastructure and waits for the response. The listener does not validate `endpointURL` and `soapAction`. The fields are set by the IBM MQ SOAP sender from the data that is provided in the URI set by a SOAP client.

The listener creates the response message and sends it to the reply destination supplied in the request message URI. In addition, the listener sets the correlation ID in the response message according to the report option in the request message. It returns the expiry, persistence, and priority settings from the request message. The listener also sends report messages back to clients in some circumstances.

If there are format errors in the SOAP request, the listener returns a report message to the client using the reply destination queue. The queue manager also returns report messages to the client using the reply destination queue, if a report has been requested. Full report messages are written to the response queue, in response to a number of events:

- An exception.
- Message expiry.
- Format of the request message is not recognized.
- Failure of the integrity check of the **MQRFH2** header.
- The format of the main message body is not MQFMT_NONE.
- The backout/retry threshold is exceeded while the IBM MQ SOAP listener is processing the request.

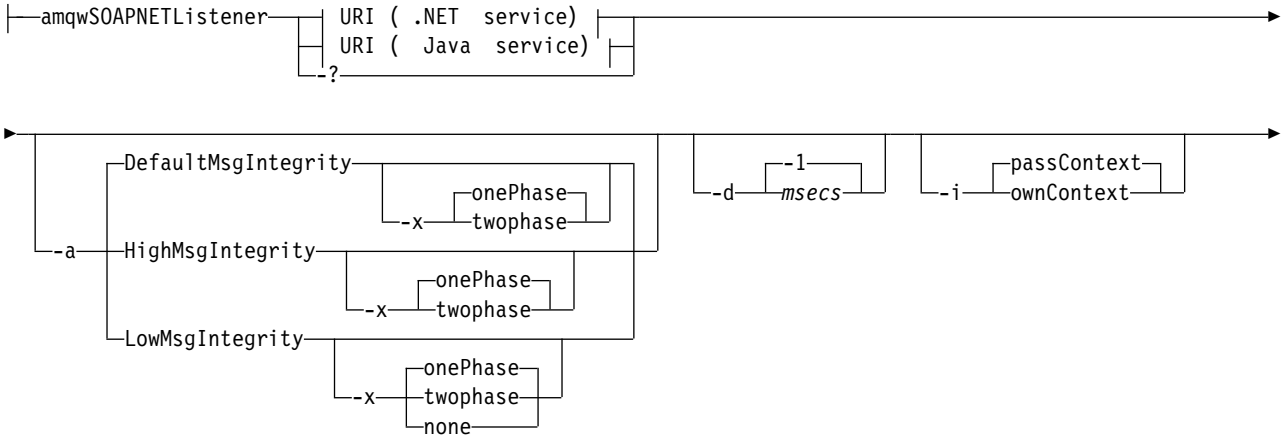
The IBM MQ SOAP sender sets MQRO_EXCEPTION_WITH_FULL_DATA and MQRO_EXPIRATION_WITH_FULL_DATA report options. As a result of the report options set by the IBM MQ SOAP sender, the report message contains the entire originating request message. The IBM MQ SOAP sender also sets the MQRO_DISCARD option, which causes the message to be discarded after a report message has been returned. If the report options do not meet your requirements, write your own senders to use different MQRO_EXCEPTION and MQRO_DISCARD report options. If the SOAP request is sent by a different sender that did not set MQRO_DISCARD, the failing message is written to the dead letter queue (DLQ).

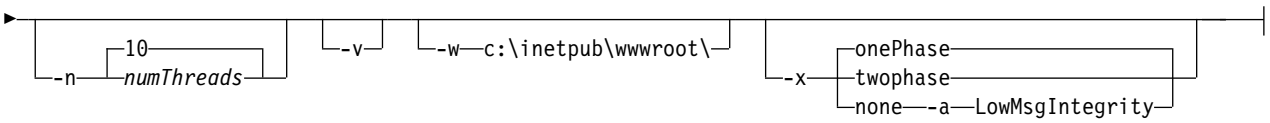
If the listener generates a report message but fails in the process of sending the report, the report message is sent to the DLQ. Ensure that your DLQ handler handles these messages correctly.

If an error occurs when attempting to write to the dead-letter queue a message is written to the IBM MQ error log. Whether the listener continues to process more messages depends on which message persistence and transactional options are selected. If the listener is running in one-phase transactional mode and is processing a non-persistent request message, the original message is discarded. The IBM MQ SOAP listener continues to execute. If the request message is persistent the request message is backed out to the request queue and the listener exits. The request queue is set to get-inhibited to prevent an accidental triggered restart.

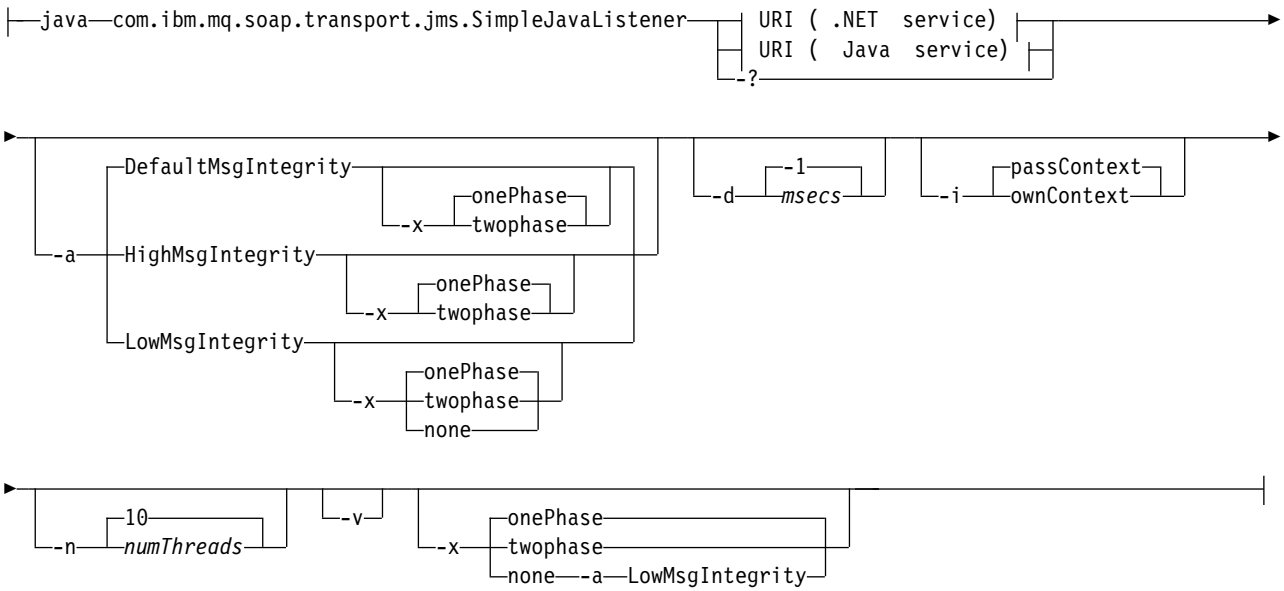
Syntax diagram

.NET:





Java:



Required parameters

URI *platform*
See "URI syntax and parameters for web service deployment" on page 3518.

-? Print out help text describing how the command is used.

Optional parameters

-a *integrityOption*
integrityOption specifies the behavior of IBM MQ SOAP listeners if it is not possible to put a failed request message on the dead-letter queue. *integrityOption* can take one of the following values:

DefaultMsgIntegrity
For non-persistent messages, the listener displays a warning message and continues to execute with the original message being discarded. For persistent messages, it displays an error message, backs out the request message so it remains on the request queue and exits. DefaultMsgIntegrity applies if the -a option is omitted, or if *integrityOption* is not specified.

LowMsgIntegrity
For both persistent and non-persistent messages, the listener displays a warning and continues to execute, discarding the message.

HighMsgIntegrity
For both persistent and non-persistent messages, the listener displays an error message, backs out the request message so it remains on the request queue and exits.

The deployment utility checks for the compatibility of the -x and -a flags. If -x none is specified, then -a LowMsgIntegrity must be specified. If the flags are incompatible the deployment utility exits with an error message and with no deployment steps having been undertaken.

-d *msecs*
msecs specifies the number of milliseconds for the IBM MQ SOAP listener to stay alive if request messages have been received on any thread. If *msecs* is set to -1, the listener stays alive indefinitely.

-i *Context*
Context specifies whether the listeners pass identity context. *Context* takes the following values:

passContext
Set the identity context of the original request message into the response message. The SOAP listener checks that it has authority to save the context from the request queue and to pass it to the response queue. It makes the checks at run time when opening the request queue to save context, and the response queue to pass context. If it does not have the required authority, or the MQOPEN call fails, and the response message is not processed. The response message is put on the dead-letter queue with the dead-letter header containing the return code from the failed MQOPEN. The listener then continues to process subsequent incoming messages as normal.

ownContext
The SOAP listener does not pass context. The returned context reflects the user ID under which the listener is running rather than the user ID which created the original request message.

The fields in the origin context are set by the queue manager, and not by the SOAP listener.

-n *numThreads*
numThreads specifies the number of threads in the generated startup scripts for the IBM MQ SOAP listener. The default is 10. Consider increasing this number if you have high message throughput.

-v -v sets verbose output from external commands. Error messages are always displayed. Use -v to output commands that you can tailor to create customized deployment scripts.

-w *serviceDirectory*
serviceDirectory is the directory containing the web service.

-x *transactionality*
transactionality specifies the type of transactional control for the listener. *transactionality* can be set to one of the following values:

onePhase
IBM MQ one-phase support is used. If the system fails during processing, the request message is redelivered to the application. IBM MQ transactions ensure that the response messages are written exactly once.

twoPhase
Two-phase support is used. If the service is written appropriately the message is delivered exactly once, coordinated with other resources, within a single committed execution of the service. This option applies to server bindings connections only.

none No transactional support. If the system fails during processing, the request message can be lost, even if persistent. The service might or might not have executed, and response, report or dead-letter messages might or might not be written.

The deployment utility checks for the compatibility of the -x and -a flags. See the description of the -a flag for details.

.NET Example

```
amqwSOAPNETlistener
-u "jms:/queue?destination=myQ&connectionFactory=()
&targetService=myService&initialContextFactory=com.ibm.mq.jms.Nojndi"
-w C:/wmqsoap/demos
-n 20
```

Java Example

```
java com.ibm.mq.soap.transport.jms.SimpleJavaListener
-u "jms:/queue?destination=myQ&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi"
-n 20
```

IBM MQ transport for SOAP sender

Sender classes are provided for Axis and .NET Framework 1 and .NET Framework 2. The sender constructs a SOAP request and puts it on a queue, it then blocks until it has read a response from the response queue. You can alter the behavior of the classes by passing different URIs from a SOAP client. For .NET Framework 3 use IBM MQ custom channel for Microsoft Windows Communication Foundation (WCF).

Purpose

The IBM MQ SOAP sender puts a SOAP request to invoke a web service onto an IBM MQ request queue. The sender sets fields in the **MQRFH2** header according to options specified in the URI, or according to defaults.

If you need to change the behavior of a sender beyond what is possible using the URI options, write your own sender. Your sender can work with the IBM MQ transport for SOAP listeners, or with other SOAP environments. Your sender must construct SOAP messages in the format defined by IBM MQ. The format is supported by the IBM MQ SOAP listener, and also SOAP listeners provided by WebSphere Application Server and CICS. Your sender must follow the rules for an IBM MQ requestor. The IBM MQ SOAP listener returns reply and report messages. See "**MQMD** SOAP settings" on page 3497 for details how to set the report options in the **MQMD**. The report options control the report messages returned by the IBM MQ SOAP listener.

Description

The IBM MQ SOAP Java sender is registered with the Axis host environment for the `jms:` URI prefix. The sender is implemented in the class `com.ibm.mq.soap.transport.jms.WMQSender`, which is derived from `org.apache.axis.handlers.BasicHandler`. If the Axis host environment detects a `jms:` URI prefix it invokes the `com.ibm.mq.soap.transport.jms.WMQSender` class. The class blocks after placing the message until it has read a response from the response queue. If no response is received within a timeout interval the sender throws an exception. If a response is received within the timeout interval the response message is returned to the client using the Axis framework. Your client application must be able to handle these response messages.

For Microsoft.NET Framework 1 and .NET Framework 2 services, the IBM MQ SOAP sender is implemented in the class `IBM.WMQSOAP.MQWebRequest`, which is derived from `System.Net.WebRequest` and `System.Net.IwebRequestCreate`. If .NET Framework 1 or .NET Framework 2 detects a `jms:` URI prefix it invokes the `IBM.WMQSOAP.MQWebRequest` class. The sender creates an `MQWebResponse` object to read the response message from the response queue and return it to the client.

`com.ibm.mq.soap.transport.jms.WMQSender` is a final class, and `IBM.WMQSOAP.MQWebRequest` is sealed. You cannot modify their behavior by creating subclasses.

Parameters

Set the URI to control the behavior of the IBM MQ SOAP sender in a web service SOAP client. The deployment utility creates web service client stubs incorporating the URI options supplied to the deployment utility.

Use a channel definition table with the IBM MQ SOAP transport for SOAP sender:

A client connection channel definition is an alternative to setting connection properties in the ConnectionFactory attribute of the web service URI. The connection properties are clientChannel, clientConnection, and SSL parameters.

Description

Create the client channel description table by defining client connections. Even if a web services client connects to different queue managers, create all the connections in the connection table on a single queue manager. The default name and location of the connection table is *queue manager directory*/@ipcc/AMQCLCHL.TAB.

Pass the location of the connection table to a Java client by setting the com.ibm.mq.soap.transport.jms.mqchlurl system property.

Pass the location of the connection table to a .NET client by setting the MQCHLLIB and MQCHLTAB environment variables.

You might provide both a channel connection table and channel connection parameters in the ConnectionFactory attribute of the web service URI. The values set in the ConnectionFactory take precedence over the values in the channel definition table.

Using a channel definition table in Java

```
java -Dcom.ibm.msg.client.config.location=file:/C:/mydir/myjms.config MyAppClass
```

Figure 73. Starting Java client using a configuration file

```
com.ibm.mq.soap.transport.jms.mqchlurl=file:/C:/ibm/wmq/qmgrs/QM1/@ipcc/AMQCLCHL.TAB
```

Figure 74. myjms.config

Transactions

Use the -x option when starting the listener, to run web services transactionally. Select the integrity of messages by setting the persistence option in the service URI.

Web services

Use the -x option when starting the listener, to run web services transactionally. On .NET Framework 1 and 2 the SOAP listener uses Microsoft Transaction Coordinator (MTS). On Axis 1.4, the SOAP listener uses queue manager coordinated transactions.

Web service clients

The SOAP senders are not transactional.

IBM MQ bindings

You can set the binding type for the SOAP sender. It can connect as an IBM MQ server application, or as a client application. You can also bind the SOAP sender as an XA-client on .NET.

Message persistence

Select the level of persistence by setting the Persistence option in the URI.

Web service transactions

You can use web service transactions, because the SOAP sender is not transactional. If you write your own SOAP sender, and intend to use web service transactions, do not create a transactional SOAP sender. You cannot send the request message and receive the reply message in the same transaction. The send and receive must not be coordinated by the web service transaction.

URI syntax and parameters for web service deployment

The syntax and parameters to deploy an IBM MQ web service are defined in a URI. The deployment utility generates a default URI based on the name of the web service. You can override the defaults by defining your own URI as a parameter to the deployment utility. The deployment utility incorporates the URI in the generated web service client stubs.

Purpose

A web service is specified using a Universal Resource Identifier (URI). The syntax diagram specifies the URI that is supported in the IBM MQ transport for SOAP. The URI controls over IBM MQ -specific SOAP parameters and options used to access target services. The URI is compatible with web services hosted by .NET, Apache Axis 1, WebSphere Application Server, CICS.

Description

The URI is incorporated into the web service client classes generated by the deployment utility. The client passes the URI to IBM MQ SOAP Sender in an IBM MQ message. The URI controls the processing performed by both the IBM MQ SOAP Sender and IBM MQ SOAP listener.

Syntax

The URI syntax is as follows:

```
jms:/queue?name=value&name=value...
```

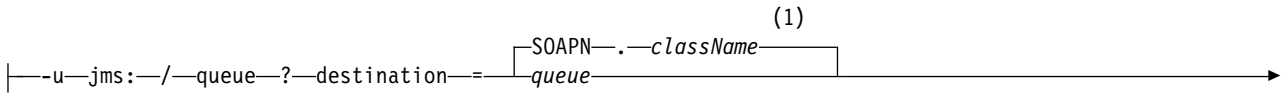
where *name* is a parameter name and *value* is an appropriate value, and the *name* = *value* element can be repeated any number of times with the second and subsequent occurrences being preceded by an ampersand (&).

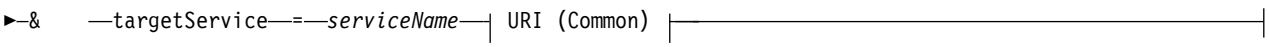
Parameter names are case-sensitive, as are names of IBM MQ objects. If any parameter is specified more than once, the final occurrence of the parameter takes effect. Client applications can override a generated parameter by appending another copy of the parameter to the URI. If any additional unrecognized parameters are included, they are ignored.

If you store a URI in an XML string, you must represent the ampersand character as `&`. Similarly, if a URI is coded in a script, take care to escape characters such as & that would otherwise be interpreted by the shell.

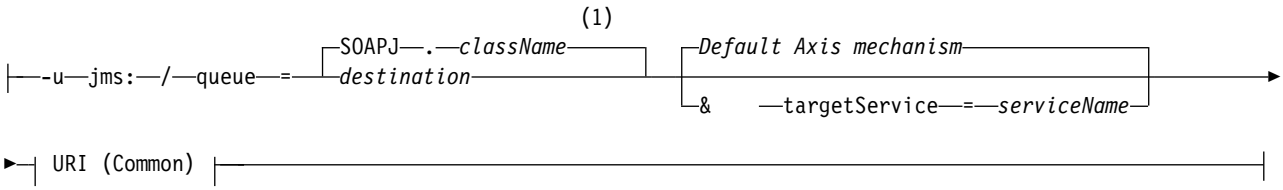
Syntax diagram

URI (.NET service):

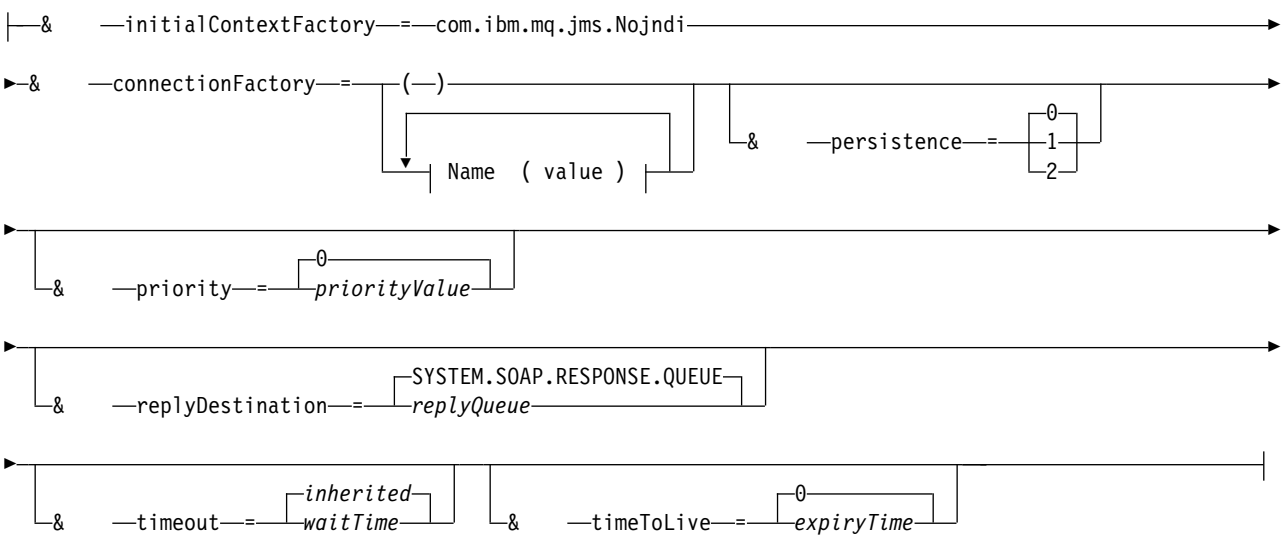




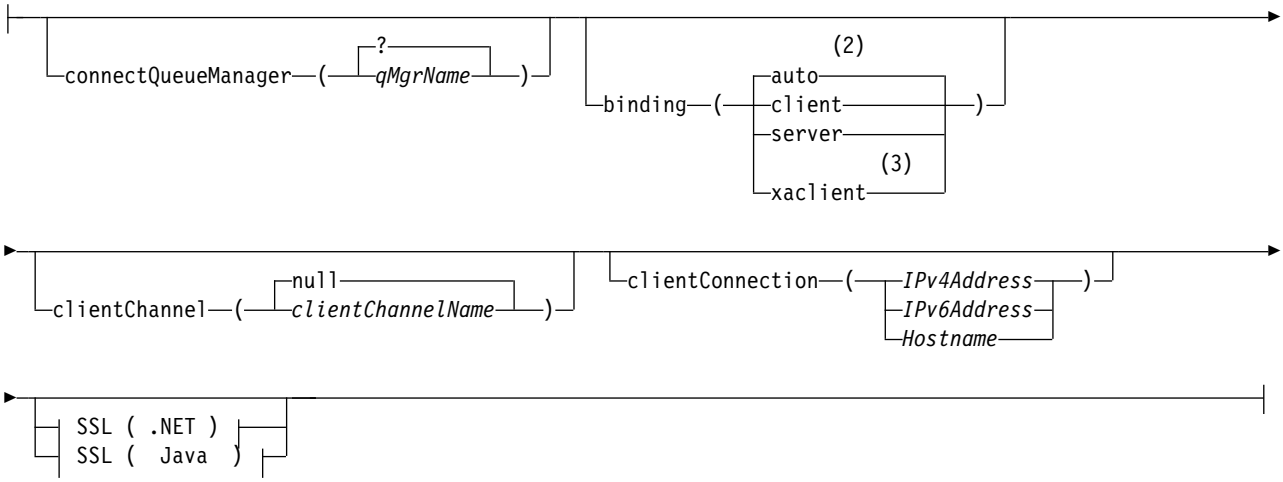
URI (Java service):



URI (Common):



Name (value):



Notes:

1 The queue manager transforms *className* to a queue name following the steps described in "Destination to queue name transformation"

2 `client` is the default if other options appropriate for a client are specified; for example `clientConnection`.

3 `xaclient` applies to .NET only

Destination to queue name transformation

1. *className* is prefixed with `SOAPJ.` for Java services or with `SOAPN.` for .NET services.
2. The file extension is removed from the full path name given in the *className* parameter.
3. The resulting string is truncated to no more than 48 characters
4. Directory separator characters are replaced with period characters.
5. Embedded spaces are replaced with underscore characters.
6. The colon following a drive prefix letter is replaced with a period for a .NET service.

Note: In some environments, a queue name generated by the deployment utility might not be unique. The deployment utility makes checks whether to create the queue. You might choose to override the deployment utility by restructuring the deployment directory hierarchy, or by customizing the supplied deployment process.

Required URI parameters

`destination = queue`

queue is the name of the request destination. It can be a queue or a queue alias. If it is queue alias, the alias can resolve to a topic.

- If the -u parameter is omitted *queue* is generated from *classname* using the steps described in "Destination to queue name transformation."
- If the **`-u`** parameter is specified, *queue* is required and must be the first parameter of the URI after the initial `jms:/queue?` string. Specify either an IBM MQ queue name, or a queue name and queue manager name connected by an @ symbol, for example `SOAPN.trandemos@WMQSOAP.DEMO.QM`.
- The deployment utility checks whether the queue name, generated or provided, matches the name of an existing queue. The action taken is described in Table 380.

Table 380. queue validation

Listener script exists?	**Listener script exists in the `./generated/server` directory**		**Listener script does not exist in the `./generated/server` directory**
Queue in listener script matches *queue* ?	***queue* does not match request queue being used in listener script**	***queue* matches request queue being used in listener script**	
queue exists	• *Deployment exits with an error.* • The service has already been deployed in `./generated/server`, but using a different queue.	• **Deployment continues normally.** • The service has already been deployed in `./generated/server`	• *Deployment exits with an error.* • The listener startup script is not found in `./generated/server`, but *queue* is in use by a different service or application.
queue does not exist		• **Deployment continues with a warning.** • The previous deployment might have failed, because the startup is valid, but *queue* is missing.	• **Deployment continues normally.** • No service has been deployed from this directory.

&connectionFactory = *Name(value)*

Name is one of the following parameters:

- connectQueueManager(qMgrName)
- binding(bindingType)
- clientChannel(channel)
- clientConnection(connection)
- "Required SSL parameters (Java)" on page 3508

See "Connection factory parameters" on page 3522 for a description of the values of these parameters.

&targetService = *serviceName*

[8]On .NET, *serviceName* is the name of a .NET service located in the deployment directory, for example: `targetService=myService.asmx`. In the .NET environment, the targetService parameter makes it possible for a single IBM MQ SOAP listener to be able to process requests for multiple services. These services must be deployed from the same directory.

Optional URI parameters

&initialContextFactory = *contextFactory*

contextFactory is required and must be set to `com.ibm.mq.jms.Nojndi`. Make sure `Nojndi.jar` is in the class path for a WebSphere Application Server web services client. `Nojndi.jar` returns Java objects based on the contents of the `connectionFactory` and `destination` parameters, rather than by reference to a directory.

&targetService = *serviceName*

[9]On Axis, *serviceName* is the fully qualified name of a Java service, for example: `targetService=javaDemos.service.StockQuoteAxis`. If targetService is not specified, a service is loaded using the default Axis mechanism.

&persistence = *messagePersistence*

messagePersistence takes one of the following values:

0 Persistence is inherited from the queue definition.

1 The message is non-persistent.

2 The message is persistent

&priority = *priorityValue*

priorityValue is in the range 0 - 9. 0 is low priority. The default value is environment-specific, which in the case of IBM MQ is 0.

&replyDestination = *replyToQueue*

The queue at the client side to be used for the response message. The default reply queue is `SYSTEM.SOAP.RESPONSE.QUEUE`.

- Run the `setupWMQSOAP` script to create the default IBM MQ SOAP objects.
- Specify a model queue for *replyToQueue* to create either a temporary or permanent dynamic response queue. For both temporary and permanent dynamic response queues, a separate instance of dynamic queue is created for each request. If any of the following events happen the queue is deleted:
  - The response arrives and is processed.
  - The request times out.
  - The requesting program terminates.

8. .NET service only

9. Java service only

For the best performance, use temporary dynamic queues rather than permanent dynamic queues. Do not send a persistent request message to a URI with a temporary dynamic queue. The IBM MQ listener SOAP fails to process the message and outputs an error. The client times out waiting for the reply.

- The `setupWMQSOAP` script creates a default permanent dynamic model queue called `SYSTEM.SOAP.MODEL.RESPONSE.QUEUE`.

&timeout = *waitTime*

The time, in milliseconds, that the client waits for a response message. *waitTime* overrides values set by the infrastructure or client application. If not specified the application value, if specified, or infrastructure default is inherited.

Note: No relationship is enforced between timeout and timeToLive.

&timeToLive = *expiryTime*

expiryTime is the time, specified in milliseconds, before the message expires. The default is zero, which indicates an unlimited lifetime.

Note: No relationship is enforced between timeout and timeToLive.

Connection factory parameters

connectQueueManager (*qMgrName*)

qMgrName specifies the queue manager to which the client connects. The default is blank.

binding (*bindingType*)

bindingType specifies how the client is connected to *qMgrName*. The default is auto. *bindingType* takes the following values:

auto The sender tries the following connection types, in order:

1. If other options appropriate to a client connection are specified, the sender uses a client binding. The other options are clientConnection or clientChannel.
2. Use a server connection.
3. Use a client connection.

Use `binding(auto)` in the URI if there is no local queue manager at the SOAP client. A client connection is built for the SOAP client.

client Use `binding(client)` in the URI to build a client configuration for the SOAP sender.

server Use `binding(server)` in the URI to build a server configuration for the SOAP sender. If the connection has client type parameters the connection fails and an error is displayed by the IBM MQ SOAP sender. Client type parameters are clientConnection, clientChannel, or SSL parameters.

xaclient

xaclient is applicable only on .NET and not for Java clients. Use an XA-client connection.

clientChannel (*channel*)

The SOAP client uses *channel* to make an IBM MQ client connection. *channel* must match the name of a server connection channel, unless channel auto definition is enabled at the server. clientChannel is a required parameter, unless you have provided a Client Connection Definition table (CCDT).

Provide a CCDT in Java by setting `com.ibm.mq.soap.transport.jms.mqchlurl`. In .NET set the `MQCHLLIB` and `MQCHLTAB` environment variables; see "Use a channel definition table with the IBM MQ SOAP transport for SOAP sender" on page 3517.

clientConnection (*connection*)

The SOAP client uses *connection* to make an IBM MQ client connection. The default host name is localhost, and default port is 1414. If *connection* is a TCP/IP address, it takes one of three formats, and can be suffixed with a port number.

JMS clients can either use the format `: hostname:port` or 'escape' the brackets using the format `%X` where `X` is the hexadecimal value that represents the bracket character in the code page of the URI. For example, in ASCII, `%28` and `%29` for `(` and `)` respectively.

.Net clients can use the brackets explicitly `: hostname(port)` or use the 'escaped' format.

IPv4 address

For example, `192.0.2.0`.

IPv6 address

For example, `2001:DB8:0:0:0:0:0:0`.

Host name

For example, `www.example.com%281687%29`, `www.example.com:1687`, or `www.example.com(1687)`.

SSL *platform*

See "Required SSL parameters (Java)" on page 3508

Sample URIs

Note:

1. `&` in the URI is encoded as `&`
2. All the parameter listed previously are applicable to the clients.
3. Only **destination**, **connectionFactory** and **initialContextFactory** are applicable to the WCF service.

```
jms:/queue?destination=myQ&amp;connectionFactory=()&amp;initialContextFactory=com.ibm.mq.jms.Nojndi
```

Figure 75. URI for an Axis service, supplying only required parameters

```
jms:/queue?destination=myQ&amp;connectionFactory=()&amp;targetService=MyService.asmx
&amp;initialContextFactory=com.ibm.mq.jms.Nojndi
```

Figure 76. URI for a .NET service, supplying only required parameters

```
jms:/queue?destination=myQ@myRQM&amp;connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
&amp;initialContextFactory=com.ibm.mq.jms.Nojndi
```

Figure 77. URI for an Axis service, supplying some optional connectionFactory parameters

```
jms:/queue?destination=myQ@myRQM&amp;connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
sslPeerName(CN=MQ Test 1,O=IBM,S=Hampshire,C=GB)
&amp;initialContextFactory=com.ibm.mq.jms.Nojndi
```

Figure 78. URI for an Axis service, supplying the sslPeerName option of the connectionFactory parameter

The `Nojndi` mechanism:

The `Nojndi` mechanism enables JMS programs, which use JNDI interfaces, to use the same URI as IBM MQ programs, which do not use JNDI.

You can use the IBM MQ transport for SOAP to invoke web services on WebSphere Application Server. WebSphere Application Server SOAP over JMS looks up the JMS resources using JNDI. The web service client might be running on .NET, or using Axis 1.4, to invoke the web service and not using JNDI. To use the same URL for the client and server, it must provide the same information whether the environment is using JNDI or not.

The URI passed to the IBM MQ transport for SOAP by a web service client contains a specific IBM MQ queue manager and queue names. These names are parsed and used directly by IBM MQ SOAP support.

The `Nojndi` mechanism directs the initialContextFactory used by a JMS program to `com.ibm.mq.jms.Nojndi`. The `com.ibm.mq.jms.Nojndi` class is an implementation of the JNDI interface that returns the connectionFactory and destination from the URL as ConnectionFactory and Queue Java objects. If the JMS implementation is IBM MQ, MQConnectionFactory and MQQueue inherit from the ConnectionFactory and Queue classes.

By using the `Nojndi` mechanism, you are able to provide the same connection information to WebSphere Application server and .NET using the same URL.

W3C SOAP over JMS URI for the IBM MQ Axis 2 client

Define a W3C SOAP over JMS URI to call a web service from an Axis 2 client using IBM MQ JMS as the SOAP transport. The web service must be provided by a server that supports IBM MQ JMS and the W3C SOAP over JMS candidate recommendation for the SOAP/JMS binding.

Description

The W3C candidate recommendation defines the SOAP over JMS binding; SOAP over Java Message Service 1.0. Also useful for its examples is URI Scheme for Java(tm) Message Service 1.0[10].

Use the syntax diagram to create W3C SOAP over JMS URIs that are syntactically correct, and are accepted by the IBM MQ Axis 2 client. It is limited to defining the URI that is accepted by the IBM MQ Axis 2 client. It is a subset of the W3C recommendation in two respects:

1. The jms-variant topic is not supported, and must not be specified in a URI passed to the IBM MQ Axis 2 client.
2. The following properties are omitted from the syntax diagram because they are JMS properties, and not part of the URI.
   a. bindingVersion
   b. contentType
   c. soapAction
   d. requestURI
   e. isFault

The JMS properties are set by the Axis 2 client or the server.

The diagram extends the W3C recommendation by defining a custom parameter, connectionFactory. connectionFactory is used as an alternative to JNDI to specify how the Axis 2 client connects to a queue manager using a queue.

The IBM MQ Axis 2 client only accepts properties as part of the URI passed to the client by the client application or as environment variables. The IBM MQ Axis 2 client has no capability to process a WSDL

10. Look for *URI Scheme for JMS*, in the W3C specification references, for the latest draft.

document. The client application or a development tool might process the WSDL and create the URI to pass to the Axis 2 client. An IBM MQ Axis 2 client application cannot set the JMS message properties directly.

Syntax

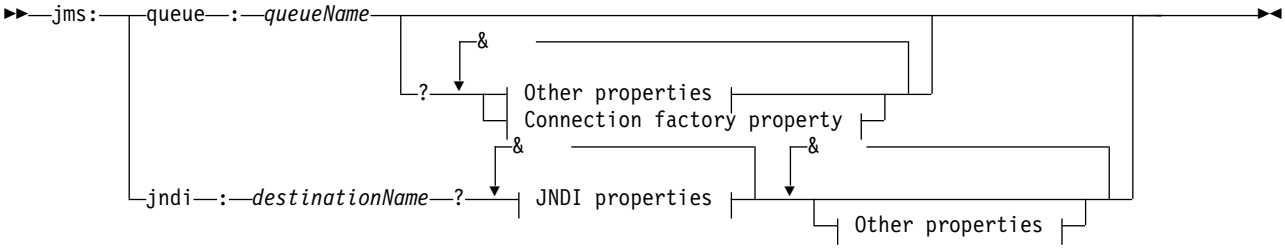
In accordance with the W3C recommendation, all the parameters can be obtained from environment variables. The environment variable names are formed by prefacing the parameter name with `soapjms_`. The syntax is: `soapjms_`*parameterName* ; for example,

```
set soapjms_targetServer=com.example.org.stockquote
```

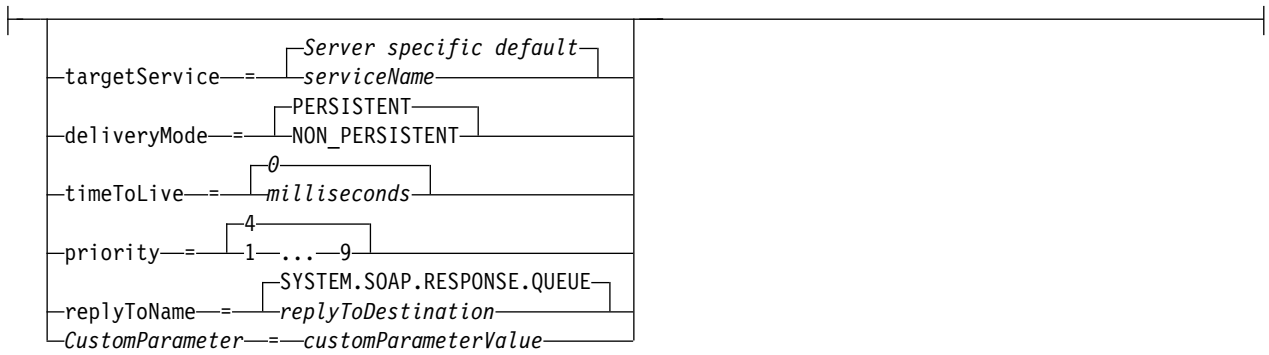
If a parameter is set using an environment variable it overrides the value set in the URI.

In accordance with the W3C recommendation, all the parameters can be repeated. The last instance of a parameter is used, unless overridden by an environment variable.

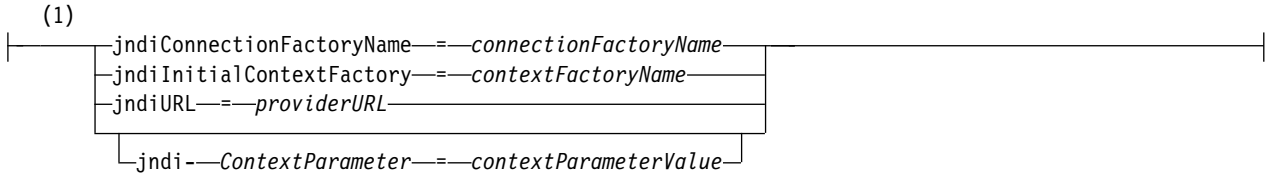
jms-uri



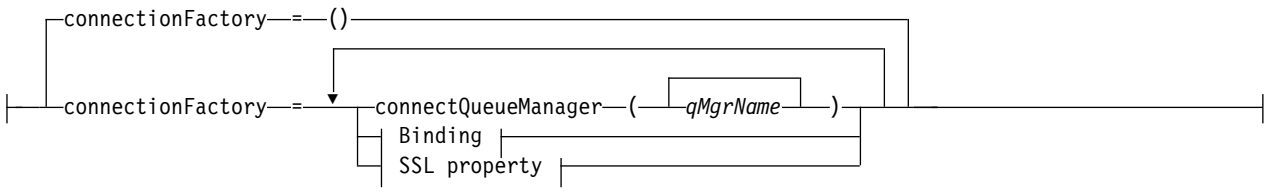
Other properties:



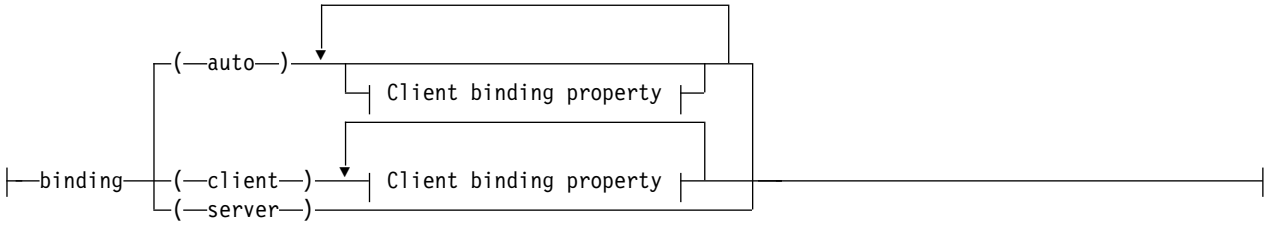
JNDI properties:



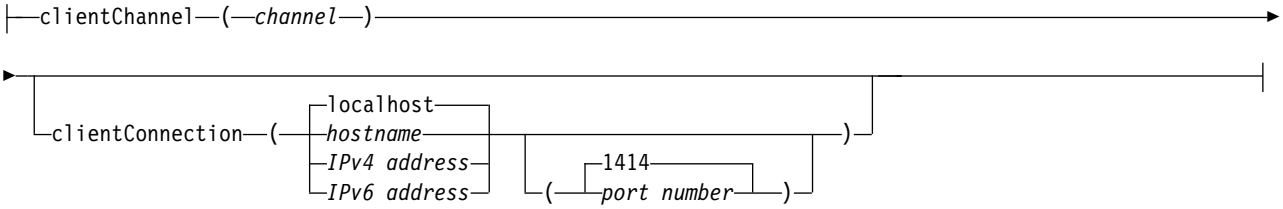
Connection factory property:



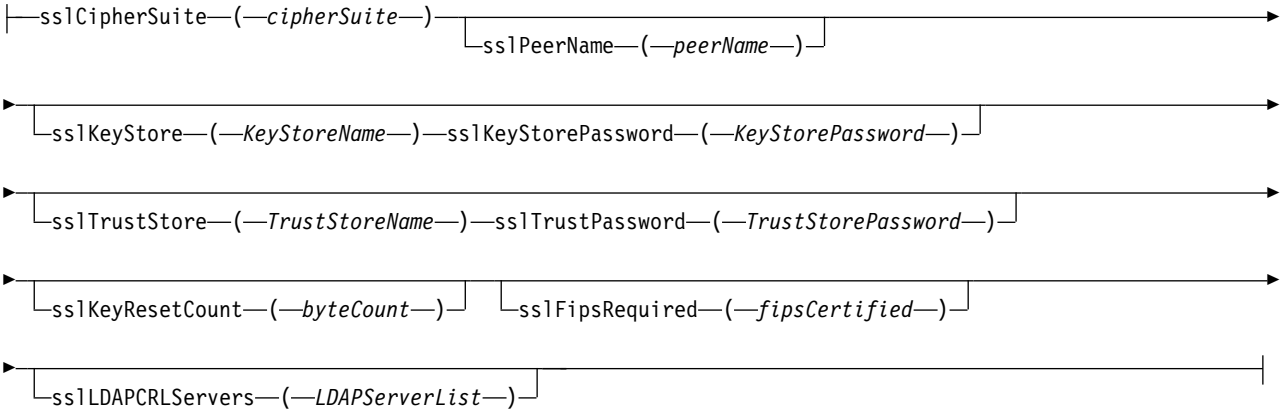
Binding:



Client binding property:



SSL property:



Notes:

1 **jndiConnectionFactoryName**, **jndiConnectionFactoryName** and **jndiURL** are all required parameters. **jndi-** *ContextParameter* is optional.

Parameters

connectionFactory = *connectionFactoryParameterList*

connectionFactoryParameterList are parameters that qualify how the Axis 2 client connects to a queue manager when the destination variant is queue.

connectionFactory must not be specified with the jndi destination variant.

The parameters are not passed to the server in the request URI.

If connectionFactory is omitted, the queue must belong to a default queue manager running on the same server as the Axis 2 client.

The *connectionFactoryParameterList*:

binding (*bindingType*)

bindingType specifies how the client is connected to *qMgrName*. The default is auto. *bindingType* takes the following values:

auto The sender tries the following connection types, in order:

1. If other options appropriate to a client connection are specified, the sender uses a client binding. The other options are clientConnection or clientChannel.
2. Use a server connection.
3. Use a client connection.

Use `binding(auto)` in the URI if there is no local queue manager at the SOAP client. A client connection is built for the SOAP client.

client Use `binding(client)` in the URI to build a client configuration for the SOAP sender.

server Use `binding(server)` in the URI to build a server configuration for the SOAP sender. If the connection has client type parameters the connection fails and an error is displayed by the IBM MQ SOAP sender. Client type parameters are clientConnection, clientChannel, or SSL parameters.

xaclient

xaclient is applicable only on .NET and not for Java clients. Use an XA-client connection.

clientChannel (*channel*)

The SOAP client uses *channel* to make an IBM MQ client connection. *channel* must match the name of a server connection channel, unless channel auto definition is enabled at the server. clientChannel is a required parameter, unless you have provided a Client Connection Definition table (CCDT).

Provide a CCDT in Java by setting `com.ibm.mq.soap.transport.jms.mqchlurl`. In .NET set the `MQCHLLIB` and `MQCHLTAB` environment variables; see "Use a channel definition table with the IBM MQ SOAP transport for SOAP sender" on page 3517.

clientConnection (*connection*)

The SOAP client uses *connection* to make an IBM MQ client connection. The default host name is localhost, and default port is 1414. If *connection* is a TCP/IP address, it takes one of three formats, and can be suffixed with a port number.

JMS clients can either use the format `: hostname:port` or 'escape' the brackets using the format `%X` where `X` is the hexadecimal value that represents the bracket character in the code page of the URI. For example, in ASCII, `%28` and `%29` for `(` and `)` respectively.

.Net clients can use the brackets explicitly `: hostname(port)` or use the 'escaped' format.

IPv4 address

For example, `192.0.2.0`.

IPv6 address

For example, `2001:DB8:0:0:0:0:0:0`.

Host name

For example, `www.example.com%281687%29`, `www.example.com:1687`, or `www.example.com(1687)`.

sslCipherSuite (*CipherSuite*)

CipherSuite specifies the sslCipherSuite used on the channel. The CipherSuite specified by the client must match the CipherSuite specified on the server connection channel.

sslFipsRequired (*fipsCertified*)
fipsCertified specifies whether *CipherSpec* or *CipherSuite* must use FIPS-certified cryptography in IBM MQ on the channel. The effect of setting *fipsCertified* is the same as setting the `FipsRequired` field of the **MQSCO** structure on an MQCONNX call.

sslKeyStore (*KeyStoreName*)
KeyStoreName specifies the sslKeyStoreName used on the channel. The keystore holds the private key of the client used to authenticate the client to the server. The keystore is optional if the TLS connection is configured to accept anonymous client connections.

sslKeyResetCount (*bytecount*)
bytecount specifies the number of bytes passed across a TLS channel before the TLS secret key must be renegotiated. To disable the renegotiation of TLS keys omit the field or set it to zero. Zero is the only value supported in some environments, see Renegotiating the secret key in IBM MQ classes for Java . The effect of setting sslKeyResetCount is the same as setting the `KeyResetCount` field in the **MQSCO** structure on an MQCONNX call.

sslKeyStorePassword (*KeyStorePassword*)
KeyStorePassword specifies the sslKeyStorePassword used on the channel.

sslLDAPCRLServers (*LDAPServerList*)

LDAPServerList specifies a list of LDAP servers to be used for Certificate Revocation List checking.

For TLS enabled client connections, *LDAPServerList* is a list of LDAP servers to be used for Certificate Revocation List (CRL) checking. The certificate provided by the queue manager is checked against one of the listed LDAP CRL servers; if found, the connection fails. Each LDAP server is tried in turn until connectivity is established to one of them. If it is impossible to connect to any of the servers, the certificate is rejected. Once a connection has been successfully established to one of them, the certificate is accepted or rejected depending on the CRLs present on that LDAP server.

If *LDAPServerList* is blank, the certificate belonging to the queue manager is not checked against a Certificate Revocation List. An error message is displayed if the supplied list of LDAP URIs is not valid. The effect of setting this field is the same as that of including `MQAIR` records and accessing them from an **MQSCO** structure on an MQCONNX.

sslPeerName (*peerName*)
peerName specifies the sslPeerName used on the channel.

sslTrustStore (*TrustStoreName*)
TrustStoreName specifies the sslTrustStoreName used on the channel. The truststore holds the public certificate of the server, or its key chain, to authenticate the server to the client. The truststore is optional if the root certificate of a certificate authority is used to authenticate the server. In Java, root certificates are held in the JRE certificate store, `cacerts`.

sslTrustStorePassword (*TrustStorePassword*)
TrustStorePassword specifies the sslTrustStorePassword used on the channel.

CustomParameter = *customParameterValue*

CustomParameter is the user-defined name of a custom parameter, and *customParameterValue* is the value of the parameter.

Custom parameters that are not used by the Axis 2 client are sent by the Axis 2 client to the SOAP server. Consult the server documentation. connectionFactory is a custom parameter that is used by the Axis 2 client and is not passed to the server.

CustomParameter must not match the name of an existing parameter.

If *CustomParameter* starts with the string `jndi-` it is used in looking up a JNDI destination; see jndi-.

deliveryMode = *deliveryMode*
deliveryMode sets the message persistence. The default is `PERSISTENT`.

jndi : *destinationName*
destinationName is a JNDI destination name that maps to a JMS queue. If the jndi destination variant is specified, you must provide a *destinationName*.

jndiConnectionFactoryName = *connectionFactoryName*
connectionFactoryName sets the JNDI name of the connection factory. If the destination variant is jndi, *connectionFactoryName* must be provided.

jndiInitialContextFactory = *contextFactoryName*
contextFactoryName sets the JNDI name of the initial context factory. If the destination variant is jndi, *contextFactoryName* must be provided. See Using JNDI to retrieve administered objects in a JMS application.

jndiURL = *providerURL*
jndiURL sets the URL name of the JNDI provider. If the destination variant is jndi, *jndiURL* must be specified.

jndi- *ContextParameter* = *contextParameterValue*
jndi- *ContextParameter* is the user-defined name of a custom parameter that used to pass information to the JNDI provider. *contextParameterValue* is the information that is passed.

priority = *priorityValue*
priorityValue sets the JMS message priority. `0` is low, `9` is high. The default value is `4`.

queue : *queueName*
queueName is the name of a JMS queue on which the SOAP request is placed. If the queue variant is specified, a queue name must be provided. If the queue does not belong to a default queue manager on the same server as the client, set the connectionFactory parameter.

replyToName = *replyToDestination*
replyToDestination sets the destination queue name. If the destination variant is jndi, the name is a JNDI name that must map to a queue. If the variant is queue the name is a JMS queue. The default value is `SYSTEM.SOAP.RESPONSE.QUEUE`.

targetService = *serviceName*
The name used by the SOAP server to start the target web service.

On Axis, *serviceName* is the fully qualified name of a Java service, for example: `targetService=www.example.org.StockQuote`. If targetService is not specified, a service is loaded using the default Axis mechanism.

timeToLive = *milliseconds*
Set *milliseconds* to the time before the message expires. The default, `0`, is the message never expires.

Examples

```
jms:jndi:REQUESTQ
    ?jndiURL=file:/C:/JMSAdmin
    &jndiInitialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory
    &jndiConnectionFactoryName=ConnectionFactory
    &replyToName=RESPONSEQ
    &deliveryMode(NON_PERSISTENT)
```

Figure 79. Use jms:jndi to send a SOAP/JMS request

```
jms:queue:SOAPJ.demos
    ?connectionFactory=connectQueueManager(QM1)
     Bind(Client)
     ClientChannel(SOAPClient)
     ClientConnection(www.example.org(1418))
     &deliveryMode(NON_PERSISTENT)
```

Figure 80. Use jms:queue to send a SOAP/JMS request

Supported web services

Code that has been written to run as a web service does not need to be modified to use the IBM MQ transport for SOAP. You do need to deploy services differently to run with the IBM MQ transport for SOAP rather than using HTTP.

Description

From IBM MQ Version 8.0.0, Fix Pack 2, the IBM MQ transport for SOAP is deprecated. For more information, see Deprecations.

The IBM MQ transport for SOAP provides a SOAP listener to run services for .NET Framework 1 and .NET 2, and for Axis 1.4. The IBM MQ custom channel for Microsoft Windows Communication Foundation runs services for .NET Framework 3. WebSphere Application Server and CICS provide support for running services over IBM MQ transport for SOAP. Create a custom Export to use WebSphere Enterprise Service Bus or WebSphere Process Server.

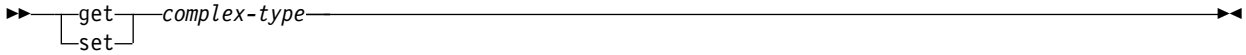
The IBM MQ SOAP listener can process SOAP requests transactionally. Run **amqwdeployWMQService** using the -x option. The two-phase option is only supported for listeners using server bindings. Other environments might provide transactional support for the IBM MQ transport for SOAP. Consult their documentation.

IBM MQ transport for SOAP currently does not support the emerging industry standard SOAP over JMS protocol that has been submitted to W3C. You can distinguish a SOAP/JMS message written to the new standard by looking for the JMS BindingVersion property. IBM MQ transport for SOAP does not set the BindingVersion property.

Axis 1.4

A Java class can typically be used without modification. The types of any arguments to the methods in the web service must be supported by the Axis engine. Refer to Axis documentation for further details. If the service uses a complex object as an argument, or returns one, that object must comply to the Java bean specification. See the examples in Figure 83 on page 3532, Figure 84 on page 3533, and Figure 85 on page 3533:

1. Have a public parameter-less constructor.
2. Any complex types of the bean must have public getters and setters of the form:



Prepare the service for deployment using the **amqwdeployWMQService** utility. The service is invoked by the IBM MQ SOAP listener which uses `axis.jar` to run the service.

The only two-phase transaction manager supported for Axis 1.4 is IBM MQ.

The supplied deployment utility does not support the case where a service returns an object in a different package to the service itself. To use an object returned in a different package, write your own deployment

utility. You can base your deployment utility on the supplied sample, or capture the commands it produces, using the -v option. Amend the commands to produce a tailored script.

If the service uses classes that are external to the Axis infrastructure and the IBM MQ SOAP run time environment, you must set the correct `CLASSPATH`. To change `CLASSPATH`, amend the generated script that starts or defines the listeners to include the services required, in one of the following ways:

- Amend the `CLASSPATH` directly in the script after the call to **amqwsetcp**.
- Create a service-specific script to customize the `CLASSPATH` and invoke this script in the generated script after the call to **amqwsetcp**.
- Create a customized deployment process to customize the `CLASSPATH` in the generated script automatically.

.NET Framework 1 and .NET Framework 2

A service that has already been prepared as an HTTP web service does not need to be modified for use as an IBM MQ web service. It needs to be deployed using the **amqwdeployWMQService** utility.

The only two-phase transaction manager supported for .NET Framework 1 and .NET 2 is Microsoft Transaction Server (MTS).

If the service code has not been prepared as an HTTP web service you must convert it to a web service. Declare the class as a web service and identify how the parameters of each method are formatted. You must check that any arguments to the methods of the service are compatible with the environment. Figure 81 on page 3532 and Figure 82 on page 3532 show a .NET class that has been prepared as a web service. The additions made are shown in bold type.

Figure 81 on page 3532 uses the code-behind programming model for a .NET web service. In the code-behind model, the source for the service is separated from the `.asmx` file. The `.asmx` file declares the name of the associated source file with the Codebehind keyword. IBM MQ has samples of both inline and code-behind .NET web services.

.NET web services source must be compiled before deployment by the **amqwdeployWMQService** deployment utility. The service is compiled into a library (`.dll`). The library must be placed in the `./bin` subdirectory of the deployment directory.

.NET Framework 3

Create an IBM MQ custom channel for Microsoft Windows Communication Foundation (WCF) to invoke services deployed to .NET Framework 3. See Developing WCF applications with IBM MQ for a description of how to configure WCF to use the IBM MQ transport for SOAP.

WebSphere Application Server

You can invoke web services hosted by WebSphere Application Server using the IBM MQ Transport for SOAP; see Using SOAP over JMS to transport web services.

You need to modify the WSDL generated by deployment of a JMS service to WebSphere Application Server in order to generate a web services client. The WSDL created by deployment to WebSphere Application Server includes a URI with a JNDI reference to the JMS `InitialContextFactory`. You need to modify the JNDI reference to Nojndi and provide connection attributes as described in “URI syntax and parameters for web service deployment” on page 3518.

CICS

You can invoke CICS applications using the IBM MQ Transport for SOAP. See Configuring your CICS system for web services.

WebSphere Enterprise Service Bus and WebSphere Process Server for Multiplatforms

WebSphere ESB and WebSphere Process Server for Multiplatforms support SOAP over JMS, with a ready built binding, only when using the default WebSphere Application Server messaging provider. Create a custom binding for JMS to support IBM MQ transport for SOAP. See JMS data bindings. See also Web services with SOAP over JMS in IBM WebSphere Process Server or IBM WebSphere Enterprise Service Bus, Part 2: Using the IBM MQ JMS provider.

Example

```
<%@ WebService Language="C#" CodeBehind="Quote.asmx.cs" Class="Quote.QuoteDotNet" %>
```

Figure 81. Service definition for .NET Framework 2: `Quote.asmx`

```
 <%@ WebService Language="C#" CodeBehind="Quote.asmx.cs" Class="Quote.QuoteDotNet" %>
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

namespace Quote {
    [WebService(Namespace = "http://www.example.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    public class QuoteDotNet : System.Web.Services.WebService {
        [WebMethod]
        public string getQuote(String symbol){
            return symbol.ToUpper();
        }
    }
}
```

Figure 82. Service implementation for .NET Framework 2: `Quote.asmx.cs`

```
package org.example.www;
public interface CustomerInfoInterface extends java.rmi.Remote {
    public org.example.www.CustomerRecord
           getCustomerName(org.example.www.CustomerRecord request)
           throws java.rmi.RemoteException, org.example.www.GetCustomerName_faultMsg;
}
```

Figure 83. Java JAX-RPC service interface using a complex type

```
package org.example.www;
public class CustomerInfoPortImpl implements org.example.www.CustomerInfoInterface{
    public org.example.www.CustomerRecord
           getCustomerName(org.example.www.CustomerRecord request)
           throws java.rmi.RemoteException, org.example.www.GetCustomerName_faultMsg {
     request.setName(request.getID().toString());
     return request;
    }
}
```

Figure 84. Java JAX-RPC service implementation using a complex type

```
package org.example.www;
public class CustomerRecord {
    private java.lang.String name;
    private java.lang.Integer ID;
    public CustomerRecord() {}
    public java.lang.String getName() {
        return name; }
    public void setName(java.lang.String name) {
        this.name = name; }
    public java.lang.Integer getID() {
        return ID; }
    public void setID(java.lang.Integer ID) {
        this.ID = ID; }
}
```

Figure 85. Java JAX-RPC service bean implementation of a complex type

IBM MQ transport for SOAP web service clients

You can reuse an existing SOAP over HTTP client with IBM MQ transport for SOAP. You must make some small modifications to the code and build process to convert the client to work with IBM MQ transport for SOAP.

Coding

JAX-RPC clients must be written in Java. .NET Framework 1 and 2 clients can be written in any language that uses the Common Language Runtime. Code examples are provided in C# and Visual Basic.

The level of transactional support depends on the client environment and the pattern of the SOAP interaction. The SOAP request and SOAP reply can not be part of the same atomic transaction.

You must call `IBM.WMQSOAP.Register.Extension()` in a .NET Framework 1, .NET Framework 2 client. In a JAX-RPC Java web service client call `com.ibm.mq.soap.Register.extension` to register the IBM MQ SOAP sender. The method registers the IBM MQ transport for SOAP sender as the handler for SOAP messages using the `jms:` protocol.

To create a .NET Framework 3 client, generate a Windows Communication Foundation client proxy using the **`svcutil`** tool; see Generating a WCF client proxy and application configuration files using the svcutil tool with metadata from a running service.

Libraries required to build and run .NET Framework 1 and 2 clients

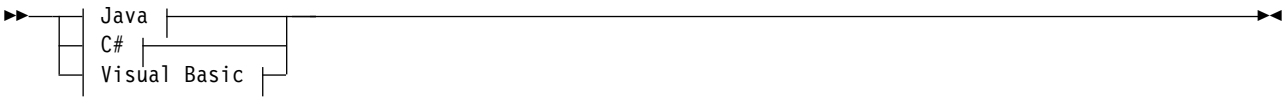
- amqsoap
- System
- System.Web.Services
- System.Xml

Libraries required to build and run Axis 1.4 clients

- *MQ_Install*\java\lib\com.ibm.mq.soap.jar;
- *MQ_Install*\java\lib\com.ibm.mq.commonservices.jar;
- *MQ_Install*\java\lib\soap\axis.jar;
- *MQ_Install*\java\lib\soap\jaxrpc.jar
- *MQ_Install*\java\lib\soap\saaj.jar;
- *MQ_Install*\java\lib\soap\commons-logging-1.0.4.jar;
- *MQ_Install*\java\lib\soap\commons-discovery-0.2.jar;
- *MQ_Install*\java\lib\soap\wsdl4j-1.5.1.jar;
- *MQ_Install*\java\jre\lib\xml.jar;
- *MQ_Install*\java\lib\soap\servlet.jar;
- *MQ_Install*\java\lib\com.ibm.mq.jar;
- *MQ_Install*\java\lib\com.ibm.mq.headers.jar;
- *MQ_Install*\java\lib\com.ibm.mq.pcf.jar;
- *MQ_Install*\java\lib\com.ibm.mq.jmqi.jar;

Note: From IBM MQ Version 8.0, `ldap.jar`, `jndi.jar` and `jta.jar` have been removed from this list as they are now part of the JDK.

Register SOAP extension



Java:

```
com.ibm.mq.soap.Register.extension()
```

C#:

```
IBM.WMQSOAP.Register.Extension();
```

Visual Basic:

```
IBM.WMQSOAP.Register.Extension
```

Client examples

Figure 86 on page 3535 is an example of a .NET Framework 1 or .NET Framework 2 C# client that uses the inline programming model. The **`IBM.WMQSOAP.Register.Extension()`** method registers the IBM MQ SOAP sender with .NET as the `jms:` protocol handler.

```
using System;
namespace QuoteClientProgram {
    class QuoteMain {
        static void Main(string[] args) {
            try {
                IBM.WMQSOAP.Register.Extension();
                Quote q = new Quote();
                Console.WriteLine("Response is: " + q.getQuote("ibm"));
             }  catch (Exception e) {
                Console.WriteLine("Exception is: " + e);
             }
        }
    }
}
```

Figure 86. C# web service client sample

Figure 87 is an example of a Java client that uses the JAX-RPC static proxy client interface. The **com.ibm.mq.soap.Register.extension();** method registers the IBM MQ SOAP sender with the service proxy to handle the `jms:` protocol.

```
package org.example.www;
import com.ibm.mq.soap.Register;
public class QuoteClient {
   public static void main(String[] args) {
      try {
         Register.extension();
         QuoteSOAPImplServiceLocator locator = new QuoteSOAPImplServiceLocator();
         System.out.println("Response = "
                  + locator.getOrgExampleWwwQuoteSOAPImpl_Wmq().getQuote("IBM"));
      } catch (Exception e) {
         System.out.println("Exception = " + e.getMessage());
      }
   }
}
```

Figure 87. Java web service client example

User exits, API exits, and installable services reference

Use the links provided in this section to help you develop your User exits, API exits, and installable services applications:

- "MQIEP structure" on page 3536
- "Data-conversion exit reference" on page 3540
- "MQ_PUBLISH_EXIT - Publish exit" on page 3544
- "Channel-exit calls and data structures" on page 3552
- "API exit reference" on page 3650
- "Installable services interface reference information" on page 3715
- IBM i "Installable services interface reference information on IBM i" on page 3782

Related concepts:

"MQI applications reference" on page 1981
Use the links provided in this section to help you develop your Message Queue Interface (MQI) applications.

Related reference:

"SOAP reference" on page 3485
IBM MQ transport for SOAP reference information arranged alphabetically.

"Reference material for IBM MQ bridge for HTTP" on page 3823
Reference topics for IBM MQ bridge for HTTP, arranged alphabetically

"The IBM MQ .NET classes and interfaces" on page 3859
IBM MQ .NET classes and interfaces are listed alphabetically. The properties, methods and constructors are described.

"IBM MQ C++ classes" on page 3922
The IBM MQ C++ classes encapsulate the IBM MQ Message Queue Interface (MQI). There is a single C++ header file, **imqi.hpp**, which covers all of these classes.

Related information:

Developing applications

The IBM MQ Classes for Java libraries

IBM MQ Classes for JMS

MQIEP structure

The MQIEP structure contains an entry point for each function call that exits are permitted to make.

Fields

StrucId

Type: MQCHAR4 - input

Structure identifier. The value is as follows:

MQIEP_STRUC_ID

Version

Type: MQLONG - input

Structure version number. The value is as follows:

MQIEP_VERSION_1
Version 1 structure version number.

MQIEP_CURRENT_VERSION
Current version of the structure.

StrucLength

Type: MQLONG

Size of the MQIEP structure in bytes. The value is as follows:

MQIEP_LENGTH_1

Flags

Type: MQLONG

Provides information about the function addresses. A flag to indicate if the library is threaded can be used with a flag to indicate if the library is a client or server library.

The following value is used to specify no library information:

MQIEPF_NONE

One of the following values is used to specify if the shared library is threaded or non-threaded:

MQIEPF_NON_THREADED_LIBRARY
A non-threaded shared library

MQIEPF_THREADED_LIBRARY
A threaded shared library

One of the following values is used to specify if the shared library is a client or a server shared library:

MQIEPF_CLIENT_LIBRARY
A client shared library

MQIEPF_LOCAL_LIBRARY
A server shared library

Reserved
Type: MQPTR

MQBACK_Call
Type: PMQ_BACK_CALL

Address of the MQBACK call.

MQBEGIN_Call
Type: PMQ_BEGIN_CALL

Address of the MQBEGIN call.

MQBUFMH_Call
Type: PMQ_BUFMH_CALL

Address of the MQBUFMH call.

MQCB_Call
Type: PMQ_CB_CALL

Address of the MQCB call.

MQCLOSE_Call
Type: PMQ_CLOSE_CALL

Address of the MQCLOSE call.

MQCMIT_Call
Type: PMQ_CMIT_CALL

Address of the MQCMIT call.

MQCONN_Call
Type: PMQ_CONN_CALL

Address of the MQCONN call.

MQCONNX_Call
Type: PMQ_CONNX_CALL

Address of the MQCONNX call.

MQCRTMH_Call
Type: PMQ_CRTMH_CALL

Address of the MQCRTMH call.

MQCTL_Call
Type: PMQ_CTL_CALL

Address of the MQCTL call.

MQDISC_Call
Type: PMQ_DISC_CALL

Address of the MQDISC call.

MQDLTMH_Call
Type: PMQ_DLTMH_CALL

Address of the MQDLTMH call.

MQDLTMP_Call
Type: PMQ_DLTMP_CALL

Address of the MQDLTMP call.

MQGET_Call
Type: PMQ_GET_CALL

Address of the MQGET call.

MQINQ_Call
Type: PMQ_INQ_CALL

Address of the MQINQ call.

MQINQMP_Call
Type: PMQ_INQMP_CALL

Address of the MQINQMP call.

MQMHBUF_Call
Type: PMQ_MHBUF_CALL

Address of the MQMHBUF call.

MQOPEN_Call
Type: PMQ_OPEN_CALL

Address of the MQOPEN call.

MQPUT_Call
Type: PMQ_PUT_CALL

Address of the MQPUT call.

MQPUT1_Call
Type: PMQ_PUT1_CALL

Address of the MQPUT1 call.

MQSET_Call
Type: PMQ_SET_CALL

Address of the MQSET call.

MQSETMP_Call
Type: PMQ_SETMP_CALL

Address of the MQSETMP call.

MQSTAT_Call
Type: PMQ_STAT_CALL

Address of the MQSTAT call.

MQSUB_Call
Type: PMQ_SUB_CALL

Address of the MQSUB call.

MQSUBRQ_Call
Type: PMQ_SUBRQ_CALL

Address of the MQSUBRQ call.

MQXCNVC_Call
Type: PMQ_XCNVC_CALL

Address of the MQXCNVC call.

MQXCLWLN_Call
Type: PMQ_XCLWLN_CALL

Address of the MQXCLWLN call.

MQXDX_Call
Type: PMQ_XDX_CALL

Address of the MQXDX call.

MQXEP_Call
Type: PMQ_XEP_CALL

Address of the MQXEP call.

MQZEP_Call
Type: PMQ_ZEP_CALL

Address of the MQZEP call.

C Declaration

```
struct tagMQIEP {
   MQCHAR4          StrucId;          /* Structure identifier */
   MQLONG           Version;          /* Structure version number */
   MQLONG           StrucLength;      /* Structure length */
   MQLONG           Flags;            /* Flags */
   MQPTR            Reserved;         /* Reserved */
   PMQ_BACK_CALL    MQBACK_Call;      /* Address of MQBACK */
   PMQ_BEGIN_CALL   MQBEGIN_Call;     /* Address of MQBEGIN */
   PMQ_BUFMH_CALL   MQBUFMH_Call;     /* Address of MQBUFMH */
   PMQ_CB_CALL      MQCB_Call;        /* Address of MQCB */
   PMQ_CLOSE_CALL   MQCLOSE_Call;     /* Address of MQCLOSE */
   PMQ_CMIT_CALL    MQCMIT_Call;      /* Address of MQCMIT */
   PMQ_CONN_CALL    MQCONN_Call;      /* Address of MQCONN */
   PMQ_CONNX_CALL   MQCONNX_Call;     /* Address of MQCONNX */
   PMQ_CRTMH_CALL   MQCRTMH_Call;     /* Address of MQCRTMH */
   PMQ_CTL_CALL     MQCTL_Call;       /* Address of MQCTL */
   PMQ_DISC_CALL    MQDISC_Call;      /* Address of MQDISC */
   PMQ_DLTMH_CALL   MQDLTMH_Call;     /* Address of MQDLTMH */
   PMQ_DLTMP_CALL   MQDLTMP_Call;     /* Address of MQDLTMP */
   PMQ_GET_CALL     MQGET_Call;       /* Address of MQGET */
   PMQ_INQ_CALL     MQINQ_Call;       /* Address of MQINQ */
   PMQ_INQMP_CALL   MQINQMP_Call;     /* Address of MQINQMP */
   PMQ_MHBUF_CALL   MQMHBUF_Call;     /* Address of MQMHBUF */
   PMQ_OPEN_CALL    MQOPEN_Call;      /* Address of MQOPEN */
   PMQ_PUT_CALL     MQPUT_Call;       /* Address of MQPUT */
   PMQ_PUT1_CALL    MQPUT1_Call;      /* Address of MQPUT1 */
   PMQ_SET_CALL     MQSET_Call;       /* Address of MQSET */
   PMQ_SETMP_CALL   MQSETMP_Call;     /* Address of MQSETMP */
   PMQ_STAT_CALL    MQSTAT_Call;      /* Address of MQSTAT */
   PMQ_SUB_CALL     MQSUB_Call;       /* Address of MQSUB */
   PMQ_SUBRQ_CALL   MQSUBRQ_Call;     /* Address of MQSUBRQ */
   PMQ_XCLWLN_CALL  MQXCLWLN_Call;    /* Address of MQXCLWLN */
   PMQ_XCNVC_CALL   MQXCNVC_Call;     /* Address of MQXCNVC */
```

```
   PMQ_XDX_CALL     MQXDX_Call;       /* Address of MQXDX */
   PMQ_XEP_CALL     MQXEP_Call;       /* Address of MQXEP */
   PMQ_ZEP_CALL     MQZEP_Call;       /* Address of MQZEP */
};
```

Data-conversion exit reference

For z/OS, you must write data-conversion exits in assembler language. For other platforms, it is recommended that you use the C programming language.

To help you to create a data-conversion exit program, the following resources are supplied:

- A skeleton source file
- A convert characters call
- A utility that creates a fragment of code that performs data conversion on data type structures This utility takes C input only. On z/OS, it produces assembler code.

For the procedure for writing the programs see:

- IBM i Writing a data-conversion exit program for IBM i
- z/OS Writing a data-conversion exit program for IBM MQ for z/OS
- Writing a data-conversion exit for IBM MQ on UNIX and Linux systems
- Writing a data-conversion exit for IBM MQ for Windows

Skeleton source file:

These can be used as your starting point when writing a data-conversion exit program.

The files supplied are listed in Table 381.

Table 381. Skeleton source files

Platform	File
AIX	amqsvfc0.c
IBM i	QMQMSAMP/QCSRC(AMQSVFC4)
HP-UX	amqsvfc0.c
Linux	amqsvfc0.c
z/OS	CSQ4BAX8 (1) CSQ4BAX9 (2) CSQ4CAX9 (3)
Solaris	amqsvfc0.c
Windows systems	amqsvfc0.c

Notes:

1. Illustrates the MQXCVNC call.
2. A wrapper for the code fragments generated by the utility for use in all environments except CICS.
3. A wrapper for the code fragments generated by the utility for use in the CICS environment.

Convert characters call:

Use the MQXCNVC (convert characters) call from within a data-conversion exit program to convert character message data from one character set to another. For certain multibyte character sets (for example, V 9.0.0 UTF-16 character sets), the appropriate options must be used.

No other MQI calls can be made from within the exit; an attempt to make such a call fails with reason code MQRC_CALL_IN_PROGRESS.

See "MQXCNVC - Convert characters" on page 2922 for further information on the MQXCNVC call and appropriate options.

Utility for creating conversion-exit code:

Use this information to learn more about creating conversion-exit code.

The commands for creating conversion-exit code are:

IBM i CVTMQMDTA (Convert IBM MQ Data Type)

Windows, UNIX and Linux systems
`crtmqcvx` (Create IBM MQ conversion-exit)

z/OS **z/OS**
CSQUCVX

The command for your platform produces a fragment of code that performs data conversion on data type structures, for use in your data-conversion exit program. The command takes a file containing one or more C language structure definitions. z/OS On z/OS, it then generates a data set containing assembler code fragments and conversion functions. On other platforms, it generates a file with a C function to convert each structure definition. On z/OS, the utility requires access to the LE/370 runtime library SCEERUN.

z/OS
Invoking the CSQUCVX utility on z/OS

Figure 88 shows an example of the JCL used to invoke the CSQUCVX utility.

```
//CVX    EXEC PGM=CSQUCVX
//STEPLIB  DD DISP=SHR,DSN=thlqual.SCSQANLE
//         DD DISP=SHR,DSN=thlqual.SCSQLOAD
//         DD DISP=SHR,DSN=le370qual.SCEERUN
//SYSPRINT DD SYSOUT=*
//CSQUINP  DD DISP=SHR,DSN=MY.MQSERIES.FORMATS(MSG1)
//CSQUOUT  DD DISP=OLD,DSN=MY.MQSERIES.EXITS(MSG1)
```

Figure 88. Sample JCL used to invoke the CSQUCVX utility

z/OS
z/OS data definition statements

The CSQUCVX utility requires DD statements with the following DDnames:

DD statement	description
SYSPRINT	Specifies a data set or print spool class for reports and error messages.
CSQUINP	Specifies the partitioned data set containing the definitions of the data structures to be converted.
CSQUOUT	Specifies the partitioned data set where the conversion code fragments are to be written. The logical record length (LRECL) must be 80 and the record format (RECFM) must be FB.

Error messages in Windows, UNIX and Linux systems

The crtmqcvx command returns messages in the range AMQ7953 through AMQ7970.

These messages are listed in Reason codes *IBM MQ Messages*.

There are two main types of error:

- Major errors, such as syntax errors, when processing cannot continue.

  A message is displayed on the screen giving the line number of the error in the input file. The output file might have been partially created.
- Other errors when a message is displayed stating that a problem has been found but that parsing of the structure can continue.

  The output file has been created and contains error information about the problems that have occurred. This error information is prefixed by #error so that the code produced is not accepted by any compiler without intervention to rectify the problems.

Valid syntax:

Your input file for the utility must conform to the C language syntax.

If you are unfamiliar with C, refer to the C example in this topic.

In addition, be aware of the following rules:

- typedef is recognized only before the struct keyword.
- A structure tag is required on your structure declarations.
- You can use empty square brackets [ ] to denote a variable length array or string at the end of a message.
- Multidimensional arrays and arrays of strings are not supported.
- The following additional data types are recognized:
  - MQBOOL
  - MQBYTE
  - MQCHAR
  - MQFLOAT32
  - MQFLOAT64
  - MQSHORT
  - MQLONG
  - MQINT8
  - MQUINT8
  - MQINT16
  - MQUINT16
  - MQINT32
  - MQUINT32

  - MQINT64
  - MQUINT64

  MQCHAR fields are code page converted, but MQBYTE, MQINT8 and MQUINT8 are left untouched. If the encoding is different, MQSHORT, MQLONG, MQINT16, MQUINT16, MQINT32, MQUINT32, MQINT64, MQUINT64, MQFLOAT32, MQFLOAT64 and MQBOOL are converted accordingly.
- Do not use the following types of data:
  - double
  - pointers
  - bit-fields

  This is because the utility for creating conversion-exit code does not provide the facility to convert these data types. To overcome this, you can write your own routines and call them from the exit.

Other points to note:

- Do not use sequence numbers in the input data set.
- If there are fields for which you want to provide your own conversion routines, declare them as MQBYTE, and then replace the generated CMQXCFBA macros with your own conversion code.

C example

```
struct TEST { MQLONG    SERIAL_NUMBER;
              MQCHAR    ID[5];
              MQINT16   VERSION;
              MQBYTE    CODE[4];
              MQLONG    DIMENSIONS[3];
              MQCHAR    NAME[24];
            } ;
```

This corresponds to the following declarations in other programming languages:

COBOL

```
      10 TEST.
       15 SERIAL-NUMBER  PIC S9(9) BINARY.
       15 ID             PIC X(5).
       15 VERSION        PIC S9(4) BINARY.
     * CODE IS NOT TO BE CONVERTED
       15 CODE           PIC X(4).
       15 DIMENSIONS     PIC S9(9) BINARY OCCURS 3 TIMES.
       15 NAME           PIC X(24).
```

System/390

```
TEST           EQU *
SERIAL_NUMBER  DS  F
ID             DS  CL5
VERSION        DS  H
CODE           DS  XL4
DIMENSIONS     DS  3F
NAME           DS  CL24
```

PL/I

Supported on z/OS only

```
 DCL 1 TEST,
       2 SERIAL_NUMBER  FIXED BIN(31),
       2 ID             CHAR(5),
       2 VERSION        FIXED BIN(15),
       2 CODE           CHAR(4),        /* not to be converted */
       2 DIMENSIONS(3)  FIXED BIN(31),
       2 NAME           CHAR(24);
```

MQ_PUBLISH_EXIT - Publish exit

The MQ_PUBLISH_EXIT call can inspect and alter messages delivered to subscribers.

Purpose

Use the publish exit to inspect and alter messages delivered to subscribers:

- Examine the contents of a message published to each subscriber
- Modify the contents of a message published to each subscriber
- Alter the queue to which a message is put
- Stop the delivery of a message to a subscriber

This exit is not available on IBM MQ for z/OS.

Syntax

```
MQ_PUBLISH_EXIT (ExitParms, PubContext, SubContext)
```

Parameters

ExitParms **(MQPSXP)- Input/Output**
ExitParms contains information about the invocation of the exit.

PubContext **(MQPBC) - Input**
PubContext contains contextual information about the publisher of the publication.

SubContext **(MQSBC) - Input/Output**
SubContext contains contextual information about the subscriber receiving the publication.

MQPSXP - Publish exit data structure:

The MQPSXP structure describes the information that is passed to and returned from the publish exit.

Table 382 summarizes the fields in the structure:

Table 382. Fields in MQPSXP

Field	Description
StrucID	Structure identifier
Version	Structure version number
ExitId	Type of exit that is being called
ExitReason	Reason for calling the exit
ExitResponse	Response from the exit
ExitResponse2	Secondary response from exit
Feedback	Feedback code
ExitUserArea	Exit user area
ExitData	Exit data
QMgrName	Name of local queue manager
Hconn	Connection handle
MsgDescPtr	Address of message descriptor (MQMD)
MsgHandle	Handle to message properties (MQHMSG)
MsgInPtr	Address of input message
MsgInLength	Length of input message

Table 382. Fields in MQPSXP (continued)

Field	Description
MsgOutPtr	Address of output message
MsgOutLength	Length of output message
pEntryPoints	Address of the MQIEP structure

Fields

StrucID **(MQCHAR4)**

StrucID is the structure identifier. The value is as follows:

MQPSXP_STRUCID

MQPSXP_STRUCID is the identifier for the publish exit parameter structure. For the C programming language, the constant MQPSXP_STRUC_ID_ARRAY is also defined; it has the same value as MQPSXP_STRUC_ID, but is an array of characters instead of a string.

StrucID is an input field to the exit.

Version **(MQLONG)**

Version is the structure version number. The value is as follows:

MQPSXP_VERSION_1

MQPSXP_VERSION_1 is the Version 1 publish exit parameter structure. The constant MQPSXP_CURRENT_VERSION is also defined with the same value.

Version is an input field to the exit.

ExitId **(MQLONG)**

ExitId is the type of exit that is being called. The value is as follows:

MQXT_PUBLISH_EXIT

Publish exit.

ExitId is an input field to the exit.

ExitReason **(MQLONG)**

ExitReason is the reason for calling the exit. The possible values are:

MQXR_INIT

The exit for this connection is called for initialization. The exit might acquire and initialize the resources that it needs; for example, main storage.

MQXR_TERM

The exit for this connection is called because the exit is about to be stopped. The exit must free any resources that it has acquired since it was initialized; for example, main storage.

MQXR_PUBLICATION

The exit is called by the queue manager before it puts a publication onto a message queue of a subscriber. The exit can change the message, not put the message on the queue, or halt publication.

ExitReason is an input field to the exit.

ExitResponse **(MQLONG)**

Set *ExitResponse* in the exit to specify how processing must continue. *ExitResponse* is one of the following values:

MQXCC_OK

Set MQXCC_OK to continue processing normally. Set MQXCC_OK in response to any values of ExitReason.

If ExitReason has the value MQXR_PUBLICATION, the *DestinationQName* and *DestinationQMgrName* fields of the MQSBC structure identify the destination to which the message is sent.

MQXCC_FAILED
Set MQXCC_FAILED to stop the publish operation. The completion code MQCC_FAILED and reason code 2557 (09FD) (RC2557): MQRC_PUBLISH_EXIT_ERROR is set on return from the exit.

MQXCC_SUPPRESS_FUNCTION
Set MQXCC_SUPPRESS_FUNCTION to stop normal processing of the message. Only set MQXCC_SUPPRESS_FUNCTION if *ExitReason* has the value MQXR_PUBLICATION.

The message continues to be processed by the queue manager according to the MQRO_DISCARD_MSG option in the *Report* field in the message descriptor of the message.

- If the MQRO_DISCARD_MSG option is specified, the message is not delivered to the subscriber.
- If the MQRO_DISCARD_MSG option is not specified, the message is placed on the dead-letter queue. If there is no dead-letter queue, or the message cannot be placed successfully on the dead-letter queue, the publication is not delivered to the subscriber. The delivery of the publication to other subscribers depends on the values of the PMSGDLV and NPMSGDLV topic object attributes. For an explanation of these attributes, see the parameter descriptions for the DEFINE TOPIC command.

ExitResponse is an output field from the exit.

ExitResponse2 **(MQLONG)**
ExitResponse2 is reserved for future use.

Feedback **(MQLONG)**
Feedback is the feedback code to be used if the exit returns MQXCC_SUPPRESS_FUNCTION in *ExitResponse*.

On input to the exit, *Feedback* always has the value MQFB_NONE. If the exit returns MQXCC_SUPPRESS_FUNCTION, set *Feedback* to the value to be used for the message when the queue manager places it on the dead-letter queue. On return from the exit, if *Feedback* has the original value MQFB_NONE, the queue manager sets *Feedback* to MQFB_STOPPED_BY_PUBSUB_EXIT.

Feedback is an input/output field to the exit.

ExitUserArea **(MQBYTE16)**
ExitUserArea is a field that is available for the exit to use. Each connection has a separate *ExitUserArea*. The length of *ExitUserArea* is given by MQ_EXIT_USER_AREA_LENGTH.

The *ExitReason* field has the value MQXR_INIT on the first invocation of the exit. *ExitUserArea* is initialized to MQXUA_NONE on the first invocation of the exit for a connection. Subsequent changes to *ExitUserArea* are preserved across invocations of the exit.

ExitUserArea is an input/output field to the exit.

ExitData **(MQCHAR32)**
ExitData is fixed exit data defined by the **PublishExitData** parameter of the stanza in the initialization file of the queue manager. The data is padded with blanks to the full length of the field. If there is no fixed exit data defined in the initialization file, *ExitData* is blank. The length of *ExitData* is given by MQ_EXIT_DATA_LENGTH.

ExitData is an input field to the exit.

QMgrName **(MQCHAR48)**
QMgrName is the name of the local queue manager. The name is padded with blanks to the full length of the field. The length of this field is given by MQ_Q_MGR_NAME_LENGTH.

QMgrName is an input field to the exit.

Hconn **(MQHCONN)**

Hconn is the handle representing a connection to the queue manager. Only use *Hconn* as a parameter to the MQSETMP, MQINQMMP, or MQDLTMP message property function calls to work with message properties.

Hconn is an input field to the exit.

MsgDescPtr **(PMQMD)**

MsgDescPtr is the address of message descriptor (MQMD) of the message being processed, and is a copy of the MQMD returned from the MQPUT call. The exit can change the contents of the message descriptor. Any change to the contents of the message descriptor must be done with care. In particular, in the case where the *SubType* field of the MQSBC structure is of value MQSUBTYPE_PROXY, the *CorrelId* field in the message descriptor must not be changed.

No message descriptor is passed to the exit if *ExitReason* is MQXR_INIT or MQXR_TERM ; in these cases, *MsgDescPtr* is the null pointer.

MsgDescPtr is an input field to the exit.

MsgHandle **(MQHMSG)**

MsgHandle is the handle to message properties. Only use *MsgHandle* with the MQSETMP, MQINQMMP, or MQDLTMP message property function calls to work with message properties.

MsgHandle is an input field to the exit.

MsgInPtr **(PMQVOID)**

MsgInPtr is the address of the input message data. The contents of the buffer addressed by *MsgInPtr* can be modified by the exit; see *MsgOutPtr* .

MsgInPtr is an input field to the exit.

MsgInLength **(MQLONG)**

MsgInLength is the length in bytes of the message data passed to the exit. The address of the data is given by *MsgInPtr*.

MsgInLength is an input field to the exit.

MsgOutPtr **(PMQVOID)**

MsgOutPtr is the address of a buffer containing message data that is returned from the exit. On entry to the exit, *MsgOutPtr* is null. On return from the exit, if the value is still null, the queue manager sends the message specified by *MsgInPtr*, with the length given by *MsgInLength*.

If the exit modifies the message data, use one of the following procedures:

- If the length of the data does not change, the data can be modified in the buffer addressed by *MsgInPtr*. In this case, do not change *MsgOutPtr* and *MsgOutLength*.
- If the modified data is shorter than the original data, the data can be modified in the buffer addressed by *MsgInPtr*. In this case *MsgOutPtr* must be set to the address of the input message buffer, and *MsgOutLength* set to the new length of the message data.
- If the modified data is, or might be, longer than the original data, the exit must obtain a new message buffer. Copy the modified data into it. Set *MsgOutPtr* to the address of the new buffer, and set *MsgOutLength* to the length of the new message data. The exit is responsible for freeing the buffer addressed by *MsgOutPtr* when the exit is next called.

Note: *MsgOutPtr* is always the null pointer on input to the exit, and not the address of a previously obtained message buffer. To free the previously obtained buffer, the exit must save its address and length. Save the information either in *ExitUserArea*, or in a control block that has its address saved in *ExitUserArea*.

MsgOutPtr is an input/output field to the exit.

MsgOutLength **(MQLONG)**

MsgOutLength is the length in bytes of the message data returned by the exit. On input to the exit,

this field is always zero. On return from the exit, this field is ignored if *MsgOutPtr* is null. See *MsgOutPtr* for information about modifying the message data.

MsgOutLength is an input/output field to the exit.

pEntryPoints **(PMQIEP)**
pEntryPoints is the address of an MQIEP structure through which MQI and DCI calls can be made.

C language declaration - MQPSXP

```
typedef struct tagMQPSXP {
 MQCHAR4    StrucId;             /* Structure identifier */
 MQLONG     Version;             /* Structure version number */
 MQLONG     ExitId;              /* Type of exit */
 MQLONG     ExitReason;          /* Reason for invoking exit */
 MQLONG     ExitResponse;        /* Response from exit */
 MQLONG     ExitResponse2;       /* Reserved */
 MQLONG     Feedback;            /* Feedback code */
 MQBYTE16   ExitUserArea;        /* Exit user area */
 MQCHAR32   ExitData;            /* Exit data */
 MQCHAR48   QMgrName;            /* Name of local queue manager */
 MQHCONN    Hconn;               /* Connection handle */
 MQHMSG     MsgHandle;           /* Handle to message properties */
 PMQMD      MsgDescPtr;          /* Address of message descriptor */
 PMQVOID    MsgInPtr;            /* Address of input message data */
 MQLONG     MsgInLength;         /* Length of input message data */
 PMQVOID    MsgOutPtr;           /* Address of output message data */
 MQLONG     MsgOutLength;        /* Length of output message data */
 /* Ver:1 */
 PMQIEP     pEntryPoints;        /* Address of the MQIEP structure */
 /* Ver:2 */
} MQPSXP;
```

MQPBC - Publication context data structure:

The MQPBC structure contains the contextual information, relating to the publisher of the publication, that is passed to the publish exit.

Table 383 summarizes the fields in the structure:

Table 383. Fields in MQPBC

Field	Description
StrucID	Structure identifier
Version	Structure version number
PubTopicString	Publish topic string
MsgDescPtr	Address of message descriptor (MQMD)

Fields

StrucID **(MQCHAR4)**
StrucID is the structure identifier. The value is as follows:

MQPBC_STRUCID
MQPBC_STRUCID is the identifier for the publication context structure. For the C programming language, the constant MQPBC_STRUC_ID_ARRAY is also defined; it has the same value as MQPBC_STRUC_ID, but is an array of characters instead of a string.

StrucID is an input field to the exit.

Version **(MQLONG)**

Version is the structure version number. The value is as follows:

MQPBC_VERSION_1
MQPBC_VERSION_1 is the Version 1 publish exit parameter structure.

MQPBC_VERSION_2
MQPBC_VERSION_2 is the Version 2 publish exit parameter structure. The constant MQPBC_CURRENT_VERSION is also defined with the same value.

Version is an input field to the exit.

PubTopicString **(MQCHARV)**

PubTopicString is the topic string being published to.

PubTopicString is an input field to the exit.

MsgDescPtr **(PMQMD)**

MsgDescPtr is the address of a copy of the message descriptor (MQMD) for the message being processed.

MsgDescPtr is an input field to the exit.

C language declaration - MQPBC

```
typedef struct tagMQPBC {
  MQCHAR4    StrucId;             /* Structure identifier */
  MQLONG     Version;             /* Structure version number */
  MQCHARV    PubTopicString;      /* Publish topic string */
  PMQMD      MsgDescPtr;          /* Address of message descriptor */
} MQPBC;
```

MQSBC - Subscription context data structure:

The MQSBC structure contains the contextual information, relating to the subscriber that is receiving the publication, that is passed to the publish exit.

Table 384 summarizes the fields in the structure:

Table 384. Fields in MQSBC

Field	Description
StrucID	Structure identifier
Version	Structure version number
DestinationQMgrName	Name of destination queue manager
DestinationQName	Name of destination queue
SubType	Type of subscription
SubOptions	Subscription options
ObjectName	Object name
ObjectString	Object string
SubTopicString	Subscription topic string
SubName	Subscription name
SubId	Subscription identifier
SelectionString	Address of selection string
SubLevel	Subscription level
PSProperties	Publish/subscribe properties

Fields

StrucID **(MQCHAR4)**
Structure identifier. The value is as follows:

MQSBC_STRUCID
MQSBC_STRUCID is the identifier for the publish exit parameter structure. For the C programming language, the constant MQSBC_STRUC_ID_ARRAY is also defined; MQSBC_STRUC_ID_ARRAY has the same value as MQSBC_STRUC_ID, but is an array of characters instead of a string.

StrucID is an input field to the exit.

Version **(MQLONG)**
Structure version number. The value is as follows:

MQSBC_VERSION_1
Version 1 publish exit parameter structure. The constant MQSBC_CURRENT_VERSION is also defined with the same value.

Version is an input field to the exit.

DestinationQMgrName **(MQCHAR48)**
DestinationQMgrName is the name of the queue manager to which the message is being sent. The name is padded with blanks to the full length of the field. The name can be altered by the exit. The length of this field is given by MQ_Q_MGR_NAME_LENGTH.

DestinationQMgrName is an input/output field to the exit; see note.

DestinationQName **(MQCHAR48)**
DestinationQName is the name of the queue to which the message is being sent. The name is padded with blanks to the full length of the field. The name can be altered by the exit. The length of this field is given by MQ_Q_NAME_LENGTH.

DestinationQName is an input/output field to the exit; see note.

SubType **(MQLONG)**
SubType indicates how the subscription was created. Valid values are MQSUBTYPE_API, MQSUBTYPE_ADMIN and MQSUBTYPE_PROXY ; see Inquire Subscription Status (Response).

SubType is an input field to the exit.

SubOptions **(MQLONG)**
SubOptions are the subscription options; see "Options (MQLONG)" on page 2553 for a description of values this field can take.

SubOptions is an input field to the exit.

ObjectName **(MQCHAR48)**
ObjectName is the name of the topic object as defined on the local queue manager. The length of this field is given by MQ_TOPIC_NAME_LENGTH. The object name is the name of the administrative topic object that the queue manager has associated with the topic string. Even if the subscriber provided a topic object as part of the subscription, the *ObjectName* might be a different topic object. The association of a topic object with a subscription depends upon the full resolution of *SubTopicString*.

ObjectName is an input field to the exit.

ObjectString **(MQCHARV)**
ObjectString is the full topic string of the publication that was subscribed to. Any wildcards in the original subscription string are resolved. It is different to the MQSD subscription *ObjectString* field described in "ObjectString (MQCHARV)" on page 2553, which might contain wildcards, and is exclusive of any object name provided by the subscriber.

ObjectString is an input field to the exit.

SubTopicString **(MQCHARV)**
SubTopicString is the complete topic string as supplied by the subscriber. *SubTopicString* is the combination of the topic string defined in a topic object, and a topic string. A subscriber must provide either a topic object, a topic string, or both. If the subscriber provides a topic string, it might contain wildcards.

SubTopicString is an input field to the exit.

SubName **(MQCHARV)**
SubName is the subscription name that is provided either by the subscriber, or is a generated name.

SubName is an input field to the exit.

SubId **(MQBYTE 24)**
SubId is the unique internal subscription identifier.

SubId is an input field to the exit.

SelectionString **(MQCHARV)**
SelectionString is the selection criteria used when subscribing for messages from a topic; see Selectors.

SelectionString is an input field to the exit.

SubLevel **(MQLONG)**
SubLevel is the interception level associated with the subscription; see "SubLevel (MQLONG)" on page 2565 for further details.

SubLevel is an input field to the exit.

PSProperties **(MQLONG)**
PSProperties are the publish/subscribe properties. They specify how publish/subscribe related message properties are added to messages sent to this subscription. Possible values are MQPSPROP_NONE, MQPSPROP_COMPAT, MQPSPROP_RFH2, MQPSPROP_MSGPROP. See Optional parameters (Change, Copy, and Create Subscription) for a description of these values.

PSProperties is an input field to the exit.

Note: Authorization checks are only performed on the original values of *DestinationQMgrName* and *DestinationQName* before they are passed to the publish exit. No new authorization checks are performed when the exit changes the destination queue, either by changing *DestinationQMgrName* or *DestinationQName*.

C language declaration - MQSBC

```
typedef struct tagMQSBC {
  MQCHAR4    StrucId;             /* Structure identifier */
  MQLONG     Version;             /* Structure version number */
  MQCHAR48   DestinationQMgrName; /* Destination queue manager */
  MQCHAR48   DestinationQName;    /* Destination queue name */
  MQLONG     SubType;             /* Type of subscription */
  MQLONG     SubOptions;          /* Subscription options */
  MQCHAR48   ObjectName;          /* Object name */
  MQCHARV    ObjectString;        /* Object string */
  MQCHARV    SubTopicString;      /* Subscription topic string */
  MQCHARV    SubName;             /* Subscription name */
  MQBYTE24   SubId;               /* Subscription identifier */
  MQCHARV    SelectionString;     /* Subscription selection string */
  MQLONG     SubLevel;            /* Subscription level */
  MQLONG     PSProperties;        /* Publish/subscribe properties */
} MQSBC;
```

Channel-exit calls and data structures

This collection of topics provide reference information about the special IBM MQ calls and data structures that you can use when you write channel exit programs.

This information is product-sensitive programming interface information. You can write IBM MQ user exits in the following programming languages:

Platform	Programming languages
IBM MQ for z/OS	Assembler and C (which must conform to the C system programming environment for system exits, described in the *z/OS C/C++ Programming Guide*.)
IBM MQ for IBM i	ILE C, ILE COBOL, and ILE RPG
All other IBM MQ platforms	C

You can also write user exits in Java for use only with Java and JMS applications. For more information about creating and using channel exits with the IBM MQ classes for Java, see Using channel exits in IBM MQ classes for Java and for IBM MQ classes for JMS, see Using channel exits with IBM MQ classes for JMS.

You cannot write IBM MQ user exits in TAL or Visual Basic. However, a declaration for the MQCD structure is provided in Visual Basic for use on the MQCONNX call from an IBM MQ MQI client program.

In a number of cases in the descriptions that follow, parameters are arrays or character strings with a size that is not fixed. For these parameters, a lowercase "n" is used to represent a numeric constant. When the declaration for that parameter is coded, the "n" must be replaced by the numeric value required. For further information about the conventions used in these descriptions, see the "Elementary data types" on page 2196.

Data definition files

Data definition files are supplied with IBM MQ for each of the supported programming languages. For details of these files, see Copy, header, include, and module files.

MQ_CHANNEL_EXIT - Channel exit:

The MQ_CHANNEL_EXIT call describes the parameters that are passed to each of the channel exits called by the Message Channel Agent.

No entry point called MQ_CHANNEL_EXIT is provided by the queue manager; the name MQ_CHANNEL_EXIT is of no special significance since the names of the channel exits are provided in the channel definition MQCD.

There are five types of channel exit:

- Channel security exit
- Channel message exit
- Channel send exit
- Channel receive exit
- Channel message-retry exit

The parameters are similar for each type of exit, and the description given here applies to all of them, except where specifically noted.

Syntax

MQ_CHANNEL_EXIT *(ChannelExitParms, ChannelDefinition, DataLength, AgentBufferLength, AgentBuffer, ExitBufferLength, ExitBufferAddr)*

Parameters

The MQ_CHANNEL_EXIT call has the following parameters.

ChannelExitParms (MQCXP) - input/output

Channel exit parameter block.

This structure contains additional information relating to the invocation of the exit. The exit sets information in this structure to indicate how the MCA proceeds.

ChannelDefinition (MQCD) - input/output

Channel definition.

This structure contains parameters set by the administrator to control the behavior of the channel.

DataLength (MQLONG) - input/output

Length of data.

The data depends on the type of exit:

- For a channel security exit, when the exit is invoked this parameter contains the length of any security message in the *AgentBuffer* field, if *ExitReason* is MQXR_SEC_MSG. It is zero if there is no message. The exit must set this field to the length of any security message to be sent to its partner if it sets *ExitResponse* to MQXCC_SEND_SEC_MSG or MQXCC_SEND_AND_REQUEST_SEC_MSG. The message data is in either *AgentBuffer* or *ExitBufferAddr*.

  The content of security messages is the sole responsibility of the security exits.
- For a channel message exit, when the exit is invoked this parameter contains the length of the message (including the transmission queue header). The exit must set this field to the length of the message in either *AgentBuffer* or *ExitBufferAddr* that is to proceed. This must be greater than or equal to the length of the transmission queue header (MQXQH).
- For a channel send or channel receive exit, when the exit is invoked this parameter contains the length of the transmission. The exit must set this field to the length of the transmission in either *AgentBuffer* or *ExitBufferAddr* that is to proceed.

If a security exit sends a message, and there is no security exit at the other end of the channel, or the other end sets an *ExitResponse* of MQXCC_OK, the initiating exit is re-invoked with MQXR_SEC_MSG and a null response (*DataLength* =0).

AgentBufferLength (MQLONG) - input

Length of agent buffer.

This parameter can be greater than *DataLength* on invocation.

For channel message, send, and receive exits, any unused space on invocation can be used by the exit to expand the data in place. If this is done, the **DataLength** parameter must be set appropriately by the exit.

In the C programming language, this parameter is passed by address.

AgentBuffer (MQBYTE x AgentBufferLength) - input/output

Agent buffer.

The contents of this parameter depend upon the exit type:

- For a channel security exit, on invocation of the exit it contains a security message if *ExitReason* is MQXR_SEC_MSG. To send a security message back, the exit can either use this buffer or its own buffer (*ExitBufferAddr*).
- For a channel message exit, on invocation of the exit this parameter contains:
  - The transmission queue header (MQXQH), which includes the message descriptor (which itself contains the context information for the message), immediately followed by
  - The message data

  If the message is to proceed, the exit can do one of the following:
  - Leave the contents of the buffer untouched
  - Modify the contents in place (returning the new length of the data in *DataLength* ; this must not be greater than *AgentBufferLength*)
  - Copy the contents to the *ExitBufferAddr*, making any required changes

  Any changes that the exit makes to the transmission queue header are not checked; however, erroneous modifications might mean that the message cannot be put at the destination.
- For a channel send or receive exit, on invocation of the exit this contains the transmission data. The exit can do one of the following:
  - Leave the contents of the buffer untouched
  - Modify the contents in place (returning the new length of the data in *DataLength* ; this must not be greater than *AgentBufferLength*)
  - Copy the contents to the *ExitBufferAddr*, making any required changes

  The first 8 bytes of the data must not be changed by the exit.

ExitBufferLength (MQLONG) - input/output

Length of exit buffer.

On the first invocation of the exit, this parameter is set to zero. Thereafter whatever value is passed back by the exit, on each invocation, is presented to the exit next time it is invoked. The value is not used by the MCA.

Note: This parameter must not be used by exits written in programming languages which do not support the pointer data type.

ExitBufferAddr (MQPTR) - input/output

Address of exit buffer.

This parameter is a pointer to the address of a buffer of storage managed by the exit, where it can choose to return message or transmission data (depending upon the type of exit) to the agent if the buffer of the agent is or might not be large enough, or if it is more convenient for the exit to do so.

On the first invocation of the exit, the address passed to the exit is null. Thereafter whatever address is passed back by the exit, on each invocation, is presented to the exit the next time it is invoked.

If ExitBufferAddr is null the data used is taken from the AgentBuffer parameter.

If ExitBufferAddr is not null the data used is taken from the buffer pointed to by the ExitBufferAddr parameter.

Note: This parameter must not be used by exits written in programming languages that do not support the pointer data type.

C invocation

```
exitname (&ChannelExitParms, &ChannelDefinition,
&DataLength, &AgentBufferLength, AgentBuffer,
&ExitBufferLength, &ExitBufferAddr);
```

The parameters passed to the exit are declared as follows:

```
MQCXP   ChannelExitParms;   /* Channel exit parameter block */
MQCD    ChannelDefinition;  /* Channel definition */
MQLONG  DataLength;         /* Length of data */
MQLONG  AgentBufferLength;  /* Length of agent buffer */
MQBYTE  AgentBuffer[n];     /* Agent buffer */
MQLONG  ExitBufferLength;   /* Length of exit buffer */
MQPTR   ExitBufferAddr;     /* Address of exit buffer */
```

COBOL invocation

```
     CALL 'exitname' USING CHANNELEXITPARMS, CHANNELDEFINITION,
                           DATALENGTH, AGENTBUFFERLENGTH, AGENTBUFFER,
                           EXITBUFFERLENGTH, EXITBUFFERADDR.
```

The parameters passed to the exit are declared as follows:

```
**   Channel exit parameter block
 01  CHANNELEXITPARMS.
     COPY CMQCXPV.
**   Channel definition
 01  CHANNELDEFINITION.
     COPY CMQCDV.
**   Length of data
 01  DATALENGTH         PIC S9(9) BINARY.
**   Length of agent buffer
 01  AGENTBUFFERLENGTH  PIC S9(9) BINARY.
**   Agent buffer
 01  AGENTBUFFER        PIC X(n).
**   Length of exit buffer
 01  EXITBUFFERLENGTH   PIC S9(9) BINARY.
**   Address of exit buffer
 01  EXITBUFFERADDR     POINTER.
```

RPG invocation (ILE)

```
     C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     C                     CALLP     exitname(MQCXP : MQCD : DATLEN :
     C                                        ABUFL : ABUF : EBUFL :
     C                                        EBUF)
```

The prototype definition for the call is:

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     Dexitname         PR                  EXTPROC('exitname')
     D* Channel exit parameter block
     D MQCXP                         160A
     D* Channel definition
     D MQCD                         1328A
     D* Length of data
     D DATLEN                         10I 0
     D* Length of agent buffer
     D ABUFL                          10I 0
     D* Agent buffer
     D ABUF                             *   VALUE
     D* Length of exit buffer
     D EBUFL                          10I 0
     D* Address of exit buffer
     D EBUF                             *
```

System/390 assembler invocation

```
        CALL EXITNAME,(CHANNELEXITPARMS,CHANNELDEFINITION,DATALENGTH, X
              AGENTBUFFERLENGTH,AGENTBUFFER,EXITBUFFERLENGTH,         X
              EXITBUFFERADDR)
```

The parameters passed to the exit are declared as follows:

```
CHANNELEXITPARMS   CMQCXPA  ,      Channel exit parameter block
CHANNELDEFINITION  CMQCDA   ,      Channel definition
DATALENGTH         DS       F      Length of data
AGENTBUFFERLENGTH  DS       F      Length of agent buffer
AGENTBUFFER        DS       CL(n)  Agent buffer
EXITBUFFERLENGTH   DS       F      Length of exit buffer
EXITBUFFERADDR     DS       F      Address of exit buffer
```

Usage notes

1. The function performed by the channel exit is defined by the provider of the exit. The exit, however, must conform to the rules defined here and in the associated control block, the MQCXP.
2. The **ChannelDefinition** parameter passed to the channel exit might be one of several versions. See the *Version* field in the MQCD structure for more information.
3. If the channel exit receives an MQCD structure with the *Version* field set to a value greater than MQCD_VERSION_1, the exit must use the *ConnectionName* field in MQCD, in preference to the *ShortConnectionName* field.
4. In general, channel exits are allowed to change the length of message data. This can arise as a result of the exit adding data to the message, or removing data from the message, or compressing or encrypting the message. However, special restrictions apply if the message is a segment that contains only part of a logical message. In particular, there must be no net change in the length of the message as a result of the actions of complementary sending and receiving exits.

   For example, it is permissible for a sending exit to shorten the message by compressing it, but the complementary receiving exit must restore the original length of the message by decompressing it, so that there is no net change in the length of the message.

   This restriction arises because changing the length of a segment would cause the offsets of later segments in the message to be incorrect, and this would inhibit the ability of the queue manager to recognize that the segments formed a complete logical message.

MQ_CHANNEL_AUTO_DEF_EXIT - Channel auto-definition exit:

The MQ_CHANNEL_AUTO_DEF_EXIT call describes the parameters that are passed to the channel auto-definition exit called by the Message Channel Agent.

No entry point called MQ_CHANNEL_AUTO_DEF_EXIT is provided by the queue manager; the name MQ_CHANNEL_AUTO_DEF_EXIT is of no special significance because the names of the auto-definition exits are provided in the queue manager.

Syntax

MQ_CHANNEL_AUTO_DEF_EXIT *(ChannelExitParms, ChannelDefinition)*

Parameters

The MQ_CHANNEL_AUTO_DEF_EXIT call has the following parameters.

ChannelExitParms (MQCXP) - input/output

Channel exit parameter block.

This structure contains additional information relating to the invocation of the exit. The exit sets information in this structure to indicate how the MCA proceeds.

ChannelDefinition (MQCD) - input/output

Channel definition.

This structure contains parameters set by the administrator to control the behavior of channels which are created automatically. The exit sets information in this structure to modify the default behavior set by the administrator.

The MQCD fields listed must not be altered by the exit:

- *ChannelName*
- *ChannelType*
- *StrucLength*
- *Version*

If other fields are changed, the value set by the exit must be valid. If the value is not valid, an error message is written to the error log file or displayed on the console (as appropriate to the environment).

Attention: Auto-defined channels created by a channel automatic definition (CHAD) exit cannot set the certificate label, because the TLS handshake has occurred by the time the channel is created. Setting the certificate label in a CHAD exit for inbound channels has no effect.

C invocation

```
exitname (&ChannelExitParms, &ChannelDefinition);
```

The parameters passed to the exit are declared as follows:

```
MQCXP  ChannelExitParms;   /* Channel exit parameter block */
MQCD   ChannelDefinition;  /* Channel definition */
```

COBOL invocation

```
     CALL 'exitname' USING CHANNELEXITPARMS, CHANNELDEFINITION.
```

The parameters passed to the exit are declared as follows:

```
**   Channel exit parameter block
 01  CHANNELEXITPARMS.
     COPY CMQCXPV.
**   Channel definition
 01  CHANNELDEFINITION.
     COPY CMQCDV.
```

RPG invocation (ILE)

```
     C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     C                    CALLP     exitname(MQCXP : MQCD)
```

The prototype definition for the call is:

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     Dexitname         PR                  EXTPROC('exitname')
     D* Channel exit parameter block
     D MQCXP                         160A
     D* Channel definition
     D MQCD                         1328A
```

System/390 assembler invocation

```
         CALL EXITNAME,(CHANNELEXITPARMS,CHANNELDEFINITION)
```

The parameters passed to the exit are declared as follows:

```
CHANNELEXITPARMS   CMQCXPA  ,  Channel exit parameter block
CHANNELDEFINITION  CMQCDA   ,  Channel definition
```

Usage notes

1. The function performed by the channel exit is defined by the provider of the exit. The exit, however, must conform to the rules defined here and in the associated control block, the MQCXP.

2. The **ChannelExitParms** parameter passed to the channel auto-definition exit is an MQCXP structure. The version of MQCXP passed depends on the environment in which the exit is running; see the description of the *Version* field in "MQCXP - Channel exit parameter" on page 3608 for details.
3. The **ChannelDefinition** parameter passed to the channel auto-definition exit is an MQCD structure. The version of MQCD passed depends on the environment in which the exit is running; see the description of the *Version* field in "MQCD - Channel definition" on page 3559 for details.

MQXWAIT - Wait in exit:

The MQXWAIT call waits for an event to occur. It can be used only from a channel exit on z/OS.

The use of MQXWAIT helps to avoid performance problems that might otherwise occur if a channel exit does something that causes a wait. The event MQXWAIT is waiting on is signaled by an MVS ECB (event control block). The ECB is described in the MQXWD control block description.

z/OS For more information about the use of MQXWAIT and writing channel-exit programs, see Writing channel exit programs on z/OS

Syntax

MQXWAIT *(Hconn, WaitDesc, CompCode, Reason)*

Parameters

The MQXWAIT call has the following parameters.

Hconn (MQHCONN) - input

Connection handle.

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN call issued in the same or earlier invocation of the exit.

WaitDesc (MQXWD) - input/output

Wait descriptor.

This parameter describes the event to wait for. See "MQXWD - Exit wait descriptor" on page 3626 for details of the fields in this structure.

CompCode (MQLONG) - output

Completion code.

It is one of the following codes:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason (MQLONG) - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_XWAIT_CANCELED
(2107, X'83B') MQXWAIT call canceled.

MQRC_XWAIT_ERROR
(2108, X'83C') Invocation of MQXWAIT call not valid.

C invocation

```
MQXWAIT (Hconn, &WaitDesc, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;     /* Connection handle */
MQXWD    WaitDesc;  /* Wait descriptor */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
        CALL MQXWAIT,(HCONN,WAITDESC,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN     DS       F  Connection handle
WAITDESC  CMQXWDA  ,  Wait descriptor
COMPCODE  DS       F  Completion code
REASON    DS       F  Reason code qualifying COMPCODE
```

MQCD - Channel definition:

The MQCD structure contains the parameters which control execution of a channel. It is passed to each channel exit that is called from a Message Channel Agent (MCA).

For more information about channel exits, see "MQ_CHANNEL_EXIT - Channel exit" on page 3552. The description in this topic relates both to message channels and to MQI channels.

Exit name fields

When an exit is called, the relevant field from *SecurityExit*, *MsgExit*, *SendExit*, *ReceiveExit*, and *MsgRetryExit* contains the name of the exit currently being invoked. The meaning of the name in these fields depends on the environment in which the MCA is running. Except where noted, the name is left-aligned within the field, with no embedded blanks; the name is padded with blanks to the length of the field. In the descriptions that follow, square brackets ([ ]) denote optional information:

UNIX The exit name is the name of a dynamically loadable module or library, suffixed with the name of a function residing in that library. The function name must be enclosed in parentheses. The library name can optionally be prefixed with a directory path:

[*path*] *library* (*function*)

The name is limited to a maximum of 128 characters.

z/OS The exit name is the name of a load module that is valid for specification on the EP parameter of the LINK or LOAD macro. The name is limited to a maximum of eight characters.

Windows
The exit name is the name of a dynamic-link library, suffixed with the name of a function residing in that library. The function name must be enclosed in parentheses. The library name can optionally be prefixed with a directory path and drive:

[d:][*path*] *library* (*function*)

The name is limited to a maximum of 128 characters.

IBM i The exit name is a 10 byte program name followed by a 10 byte library name. If the names are less than 10 bytes long, each name is padded with blanks to make it 10 bytes. The library name can be `*LIBL` except when calling a channel auto-definition exit, in which case a fully qualified name is required.

Changing MQCD fields in a channel exit

A channel exit can change fields in the MQCD. The changed value remains in the MQCD and is passed to any remaining exits in an exit chain and to any conversation sharing the channel instance. The changed MQCD is also used by the MCA for its normal processing during the continuing lifetime of the channel.

The following MQCD fields must not be altered by the exit:

- ChannelName
- ChannelType
- StrucLength
- Version

Related reference:

"Fields"
This topic lists all the fields in the MQCD structure and describes each field.

"C declaration" on page 3593
This declaration is the C declaration for the MQCD structure.

"COBOL declaration" on page 3595
This declaration is the COBOL declaration for the MQCD structure.

"RPG declaration (ILE)" on page 3598
This declaration is the RPG declaration for the MQCD structure.

"System/390 assembler declaration" on page 3601
This declaration is the System/390 assembler declaration for the MQCD structure.

"Visual Basic declaration" on page 3603
This declaration is the Visual Basic declaration of the MQCD structure.

"Changing MQCD fields in a channel exit" on page 3605
A channel exit can change fields in the MQCD. However, these changes are not typically acted on, except in the circumstances listed.

Fields:

This topic lists all the fields in the MQCD structure and describes each field.

BatchDataLimit (MQLONG):

This field specifies the the limit, in kilobytes, of the amount of data that can be sent through a channel before taking a sync point.

A sync point is taken after the message that caused the limit to be reached has flowed across the channel.

The batch is terminated when one of the following conditions is met:

- **BatchSize** messages have been sent.
- **BatchDataLimit** bytes have been sent.
- The transmission queue is empty and **BatchInterval** is exceeded.

The value must be in the range 0 - 999999. The default value is 5000.

A value of zero in this attribute means that no data limit is applied to batches over this channel.

This parameter only applies to channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSRCVR, or MQCHT_CLUSSDR.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_11.

BatchHeartbeat (MQLONG):

This field specifies the time interval that is used to trigger a batch heartbeat for the channel.

Batch heartbeating allows sender channels to determine whether the remote channel instance is still active before going indoubt. A batch heartbeat occurs if a sender channel has not communicated with the remote channel instance within the specified time interval.

The value is in the range 0 through 999 999; the units are milliseconds. A value of zero indicates that batch heartbeating is not enabled.

This field is relevant only for channels that have a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_7.

BatchInterval (MQLONG):

This field specifies the approximate time in milliseconds that a channel keeps a batch open, if fewer than *BatchSize* messages have been transmitted in the current batch.

If *BatchInterval* is greater than zero, the batch is terminated by whichever of the following events occur first:

- *BatchSize* messages have been sent, or
- *BatchInterval* milliseconds have elapsed since the start of the batch.

If *BatchInterval* is zero, the batch is terminated by whichever of the following events occur first:

- *BatchSize* messages have been sent, or
- the transmission queue becomes empty.

BatchInterval must be in the range zero through 999 999 999.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

This is an input field to the exit. The field is not present when *Version* is less than MQCD_VERSION_4.

BatchSize (MQLONG):

This field specifies the maximum number of messages that can be sent through a channel before synchronizing the channel.

This field is not relevant for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN.

CertificateLabel (MQCHAR64):

This field gives details of the certificate label being used.

IBM MQ initializes the default value for the *CertificateLabel* field as blanks.

This is interpreted at runtime as the default value, and is backwards compatible.

For example, specifying a MQCD version less than 11, or using the default value of blanks for the *CertificateLabel* field, means that this field is ignored.

The length of this field is given by MQ_CERT_LABEL_LENGTH.

ChannelMonitoring (MQLONG):

This field specifies the current level of monitoring data collection for the channel.

This field is not relevant for channels with a ChannelType of MQCHT_CLNTCONN.

It is one of the following values:
- MQMON_OFF
- MQMON_LOW
- MQMON_MEDIUM
- MQMON_HIGH

This is an input field to the exit. It is not present if *Version* is less than MQCD_VERSION_8.

ChannelName (MQCHAR20):

This field specifies the channel definition name.

There must be a channel definition of the same name at the remote machine to be able to communicate.

The name must use only the characters:
- Uppercase A-Z
- Lowercase a-z
- Numerics 0-9
- Period (.)
- Forward slash (/)
- Underscore (_)
- Percent sign (%)

and be padded to the right with blanks. Leading or embedded blanks are not allowed.

The length of this field is given by MQ_CHANNEL_NAME_LENGTH.

ChannelStatistics (MQLONG):

This field specifies the current level of statistics data collection for the channel.

This field is not relevant for channels with a ChannelType of MQCHT_CLNTCONN or MQCHT_SVRCONN.

It is one of the following values:

- MQMON_OFF
- MQMON_LOW
- MQMON_MEDIUM
- MQMON_HIGH

This is an input field to the exit. It is not present if *Version* is less than MQCD_VERSION_8.

ChannelType (MQLONG):

This field specifies the type of channel.

It is one of the following values:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_CLNTCONN
Client connection.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLUSSDR
Cluster sender.

MQCHT_CLUSRCVR
Cluster receiver.

ClientChannelWeight (MQLONG):

This field specifies a weighting to influence which client-connection channel definition is used.

The ClientChannelWeight attribute is used so that client channel definitions can be selected at random based on their weighting when more than one suitable definition is available. When a client issues an MQCONN requesting connection to a queue manager group, by specifying a queue manager name starting with an asterisk, and more than one suitable channel definition is available in the client channel definition table (CCDT), the definition to use is randomly selected based on the weighting, with any applicable ClientChannelWeight(0) definitions selected first in alphabetical order.

Specify a value in the range 0 - 99. The default is 0.

A value of 0 indicates that no load balancing is performed and applicable definitions are selected in alphabetical order. To enable load balancing choose a value in the range 1 - 99 where 1 is the lowest weighting and 99 is the highest. The distribution of messages between two or more channels with non-zero weightings is proportional to the ratio of those weightings. For example, three channels with ClientChannelWeight values of 2, 4, and 14 are selected approximately 10%, 20%, and 70% of the time. This distribution is not guaranteed.

This attribute is valid for the client-connection channel type only.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_9.

ClusterPtr (MQPTR):

This field specifies the address a list of cluster names.

If *ClustersDefined* is greater than zero, this address is the address of a list of cluster names. The channel belongs to each cluster listed.

This field is relevant only for channels with a *ChannelType* of MQCHT_CLUSSDR or MQCHT_CLUSRCVR.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_5.

ClustersDefined (MQLONG):

This field specifies the number of clusters to which the channel belongs.

This field is the number of cluster names pointed to by *ClusterPtr*. It is zero or greater.

This field is relevant only for channels with a *ChannelType* of MQCHT_CLUSSDR or MQCHT_CLUSRCVR.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_5.

CLWLChannelPriority (MQLONG):

This field specifies the cluster workload channel priority.

The workload manager choose algorithm selects a destination with the highest priority from the set of destinations selected based on rank. If there are two possible destination queue managers, this attribute can be used to make one queue manager failover onto the other queue manager. All the messages go to the queue manager with the highest priority until that ends, then the messages go to the queue manager with the next highest priority.

The value is in the range 0 through 9. The default is 0.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_8.

For further information, see Configuring a queue manager cluster.

CLWLChannelRank (MQLONG):

This field specifies the cluster workload channel rank.

The workload manager choose algorithm selects a destination with the highest rank. When the final destination is a queue manager on a different cluster, you can set the rank of intermediate gateway queue managers (at the intersection of neighboring clusters) so the choose algorithm correctly chooses a destination queue manager nearer the final destination.

The value is in the range 0 through 9. The default is 0.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_8.

For further information, see Configuring a queue manager cluster.

CLWLChannelWeight (MQLONG):

This field specifies the cluster workload channel weight.

Cluster workload channel weight.

The workload manager choose algorithm uses the "weight" attribute of the channel to the skew the destination choice so that more messages can be sent to a particular machine. For example, you can give a channel on a large UNIX server a larger "weight" than another channel on small desktop PC, and the choose algorithm chooses the UNIX server more frequently than the PC.

The value is in the range 1 through 99. The default is 50.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_8.

For further information, see Configuring a queue manager cluster.

ConnectionAffinity (MQLONG):

This field specifies whether client applications that connect multiple times using the same queue manager name, use the same client channel.

Use this attribute when multiple applicable channel definitions are available.

The value is one of the following:

MQCAFTY_PREFERRED
The first connection in a process reading a client channel definition table (CCDT) creates a list of applicable definitions based on the weighting with any applicable CLNTWGHT(0) definitions first and in alphabetical order. Each connection in the process attempts to connect using the first definition in the list. If a connection is unsuccessful the next definition is used. Unsuccessful definitions with CLNTWGHT values other than 0 are moved to the end of the list. CLNTWGHT(0) definitions remain at the start of the list and are selected first for each connection.

Each client process with the same host name always creates the same list.

For client applications written in C, C++, or the .NET programming framework (including fully managed .NET) the list is updated if the CCDT has been modified since the list was created.

This value is the default value.

MQCAFTY_NONE
The first connection in a process reading a CCDT creates a list of applicable definitions. All connections in a process select an applicable definition based on the weighting with any applicable CLNTWGHT(0) definitions selected first in alphabetical order.

For client applications written in C, C++, or the .NET programming framework (including fully managed .NET) the list is updated if the CCDT has been modified since the list was created.

This attribute is valid for the client-connection channel type only.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_9.

ConnectionName (MQCHAR264):

This field specifies the connection name for the channel.

For cluster-receiver channels (when specified) CONNAME relates to the local queue manager, and for other channels it relates to the target queue manager. The value you specify depends on the transmission protocol (*TransportType*) to be used:

- For MQXPT_LU62, it is the fully-qualified name of the partner Logical Unit.
- For MQXPT_NETBIOS, it is the NetBIOS name defined on the remote machine.
- For MQXPT_TCP, it is either the host name, the network address of the remote machine specified in IPv4 dotted decimal or IPv6 hexadecimal format, or the local machine for cluster-receiver channels.
- For MQXPT_SPX, it is an SPX-style address comprising a 4 byte network address, a 6 byte node address, and a 2 byte socket number.

When defining a channel, this field is not relevant for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_RECEIVER. However, when the channel definition is passed to an exit, this field contains the address of the partner, whatever the channel type.

The length of this field is given by MQ_CONN_NAME_LENGTH. This field is not present if *Version* is less than MQCD_VERSION_2.

DataConversion (MQLONG):

This field specifies whether the sending message channel agent attempts conversion of the application message data if the receiving message channel agent is unable to perform this conversion.

This field applies only to messages that are not segments of logical messages; the MCA never attempts to convert messages which are segments.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR. It is one of the following:

MQCDC_SENDER_CONVERSION
Conversion by sender.

MQCDC_NO_SENDER_CONVERSION
No conversion by sender.

DefReconnect (MQLONG):

The DefReconnect channel attribute sets the default reconnection attribute value for a client connection channel.

The default automatic client reconnection option. You can configure a IBM MQ MQI client to automatically reconnect a client application. The IBM MQ MQI client tries to reconnect to a queue manager after a connection failure. It tries to reconnect without the application client issuing an MQCONN or MQCONNX MQI call.

Reconnection is an MQCONNX option. By using the DefReconnect channel attribute you can add reconnection behavior to existing applications that use MQCONN. You can also change the reconnection behavior of applications that use MQCONNX.

You can also set the DefRecon value from the `mqclient.ini` file to set or modify reconnection behavior. The DefRecon value from the `mqclient.ini` file takes precedence over the DefReconnect channel attribute.

Syntax

```
DefReconnect ( MQRCN_NO | MQRCN_YES | MQRCN_Q_MGR |
MQRCN_DISABLED )
```

Parameters

MQRCN_NO
MQRCN_NO is the default value.

Unless overridden by **MQCONNX**, the client is not reconnected automatically.

MQRCN_YES
Unless overridden by **MQCONNX**, the client reconnects automatically.

MQRCN_Q_MGR
Unless overridden by **MQCONNX**, the client reconnects automatically, but only to the same queue manager. The `QMGR` option has the same effect as `MQCNO_RECONNECT_Q_MGR`.

MQRCN_DISABLED
Reconnection is disabled, even if requested by the client program using the **MQCONNX** MQI call.

Automatic client reconnection is not supported by IBM MQ classes for Java.

Table 385. Automatic reconnection depends on the values set in the application and in the channel definition

DefReconnect	**Reconnection options set in the application**			
	MQCNO_RECONNECT	MQCNO_RECONNECT_Q_MGR	MQCNO_RECONNECT_AS_DEF	MQCNO_RECONNECT_DISABLED
MQRCN_NO	YES	QMGR	NO	NO
MQRCN_YES	YES	QMGR	YES	NO
MQRCN_Q_MGR	YES	QMGR	QMGR	NO
MQRCN_DISABLED	NO	NO	NO	NO

Related reference:

Connection options
Options that control the action of MQCONNX.

Related information:

Automatic client reconnection

Channel and client reconnection

CHANNELS stanza of the client configuration file

Desc (MQCHAR64):

This field can be used for descriptive commentary.

The content of the field is of no significance to Message Channel Agents. However, it must contain only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, the field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the character set of the queue manager (as defined by the **CodedCharSetId** queue manager attribute), those characters might be translated incorrectly if this field is sent to another queue manager.

The length of this field is given by MQ_CHANNEL_DESC_LENGTH.

DiscInterval (MQLONG):

This field specifies the maximum time in seconds for which the channel waits for a message to arrive on the transmission queue, before terminating the channel.

In other words, it specifies the disconnect interval.

The A value of zero causes the MCA to wait indefinitely.

For server-connection channels using the TCP protocol, the interval represents the client inactivity disconnect value, specified in seconds. If a server-connection has received no communication from its partner client for this duration, it terminates the connection. The server-connection inactivity interval only applies between IBM MQ API calls from a client, so no client is disconnected during a long-running MQGET with wait call.

This attribute is not applicable for server-connection channels using protocols other than TCP.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, MQCHT_CLUSRCVR, or MQCHT_SVRCONN.

ExitDataLength (MQLONG):

This field specifies length in bytes of each of the user data items in the lists of exit user data items addressed by the *MsgUserDataPtr*, *SendUserDataPtr*, and *ReceiveUserDataPtr* fields.

This length is not necessarily the same as MQ_EXIT_DATA_LENGTH.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

ExitNameLength (MQLONG):

This field specifies the length in bytes of each of the names in the lists of exit names addressed by the *MsgExitPtr*, *SendExitPtr*, and *ReceiveExitPtr* fields.

This length is not necessarily the same as MQ_EXIT_NAME_LENGTH.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

HdrCompList [2] (MQLONG):

This field specifies the list of header data compression techniques which are supported by the channel.

The list contains one or more of the following values:

MQCOMPRESS_NONE
No header data compression is performed.

MQCOMPRESS_SYSTEM
Header data compression is performed.

Unused values in the array are set to MQCOMPRESS_NOT_AVAILABLE.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_8.

HeartbeatInterval (MQLONG):

This field specifies the time in seconds between heartbeat flows.

The interpretation of this field depends on the channel type, as follows:

- For a channel type of MQCHT_SENDER, MQCHT_SERVER, MQCHT_RECEIVER MQCHT_REQUESTER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR, this field is the time in seconds between heartbeat flows passed from the sending MCA when there are no messages on the transmission queue. This gives the receiving MCA the opportunity to quiesce the channel. To be useful, *HeartbeatInterval* must be less than *DiscInterval*.
- For a channel type of MQCHT_CLNTCONN or MQCHT_SVRCONN with the MQCD Sharing Conversations field set to zero, this field is the time in seconds between heartbeat flows passed from the server MCA when that MCA has issued an MQGET call with the MQGMO_WAIT option on behalf of a client application. This allows the server MCA to handle situations where the client connection fails during an MQGET with MQGMO_WAIT.
- For a channel type of MQCHT_CLNTCONN or MQCHT_SVRCONN with the MQCD Sharing Conversations field set to a non-zero value, this field is the time in seconds between heartbeat flow when there are no data flows sent or received. This allows the channel to be quiesced efficiently.

The value is in the range 0 through 999 999. The value that is used is the larger of the values specified at the sending side and receiving side unless a value of 0 is specified at either side, in which case no heartbeat exchange occurs.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

KeepAliveInterval (MQLONG):

This field specifies the value passed to the communications stack for keepalive timing for the channel.

The value is applicable for the TCP/IP and SPX communications protocols, though not all implementations support this parameter.

The value is in the range 0 through 99 999; the units are seconds. A value of zero indicates that channel keepalive is not enabled, although keepalive might still occur if TCP/IP keepalive (rather than channel keepalive) is enabled. The following special value is also valid:

MQKAI_AUTO
Automatic.

This value indicates that the keepalive interval is calculated from the negotiated heartbeat interval, as follows:

- If the negotiated heartbeat interval is greater than zero, the keepalive interval that is used is the heartbeat interval plus 60 seconds.
- If the negotiated heartbeat interval is zero, the keepalive interval that is used is zero.

- On z/OS, TCP/IP keepalive occurs when TCPKEEP(YES) is specified on the queue manager object.
- In other environments, TCP/IP keepalive occurs when the **KEEPALIVE=YES** parameter is specified in the TCP stanza in the distributed queuing configuration file.

This field is relevant only for channels that have a *TransportType* of MQXPT_TCP or MQXPT_SPX.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_7.

LocalAddress (MQCHAR48):

This field specifies the local TCP/IP address defined for the channel for outbound communications.

This field is blank if no specific address is defined for outbound communications. The address can optionally include a port number or range of port numbers. The format of this address is:

```
[ip-addr][(low-port[,high-port])]
```

where square brackets ([ ]) denote optional information, `ip-addr` is specified in IPv4 dotted decimal, IPv6 hexadecimal, or alphanumeric form, and `low-port` and `high-port` are port numbers enclosed in parentheses. All are optional.

A specific IP address, port, or port range for outbound communications is useful in recovery scenarios where a channel is restarted on a different TCP/IP stack.

LocalAddress is similar in form to *ConnectionName*, but must not be confused with it. *LocalAddress* specifies the characteristics of the local communications, whereas *ConnectionName* specifies how to reach a remote queue manager.

This field is relevant only for channels with a *TransportType* of MQXPT_TCP, and a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, MQCHT_CLNTCONN, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

The length of this field is given by MQ_LOCAL_ADDRESS_LENGTH. This field is not present if *Version* is less than MQCD_VERSION_7.

LongMCAUserIdLength (MQLONG):

This field specifies the length in bytes of the full MCA user identifier pointed to by *LongMCAUserIdPtr*.

This field is not relevant for channels with a *ChannelType* of MQCHT_CLNTCONN.

This is an input/output field to the exit. The field is not present if *Version* is less than MQCD_VERSION_6.

LongMCAUserIdPtr (MQPTR):

This field specifies the address of the long MCA user identifier.

If *LongMCAUserIdLength* is greater than zero, this field is the address of the full MCA user identifier. The length of the full identifier is given by *LongMCAUserIdLength*. The first 12 bytes of the MCA user identifier are also contained in the field *MCAUserIdentifier*.

See the description of the *MCAUserIdentifier* field for details of the MCA user identifier.

This field is not relevant for channels with a *ChannelType* of MQCHT_SDR, MQCHT_SVR, MQCHT_CLNTCONN, or MQCHT_CLUSSDR.

This is an input/output field to the exit. The field is not present if *Version* is less than MQCD_VERSION_6.

LongRemoteUserIdLength (MQLONG):

This field specifies the length in bytes of the full remote user identifier pointed to by *LongRemoteUserIdPtr*.

This field is relevant only for channels with a *ChannelType* of MQCHT_CLNTCONN or MQCHT_SVRCONN.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_6.

LongRemoteUserIdPtr (MQPTR):

This field specifies the address of the long remote user identifier.

If *LongRemoteUserIdLength* is greater than zero, this flag is the address of the full remote user identifier. The length of the full identifier is given by *LongRemoteUserIdLength*. The first 12 bytes of the remote user identifier are also contained in the field *RemoteUserIdentifier*.

See the description of the *RemoteUserIdentifier* field for details of the remote user identifier.

This field is relevant only for channels with a *ChannelType* of MQCHT_CLNTCONN or MQCHT_SVRCONN.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_6.

LongRetryCount (MQLONG):

This field specifies the count used after the count specified by the *ShortRetryCount* has been exhausted.

It specifies the maximum number of further attempts that are made to connect to the remote machine, at intervals specified by *LongRetryInterval*, before logging an error to the operator.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

LongRetryInterval (MQLONG):

This field specifies the maximum number of seconds to wait before reattempting connection to the remote machine.

The interval between retries can be extended if the channel has to wait to become active.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

MaxInstances (MQLONG):

This field specifies the maximum number of simultaneous instances of an individual server-connection channel that can be started.

This field is used only on server-connection channels.

The field can have a value in the range 0 - 999 999 999. A value of zero prevents all client access.

The default value of this field is 999 999 999.

If the value of this field is reduced to a number that is less than the number of instances of the server-connection channel that are currently running, then those running instances are not affected. However, new instances cannot start until sufficient existing instances have ceased to run so that the number of currently running instances is less than the value of the field.

MaxInstancesPerClient (MQLONG):

This field specifies the maximum number of simultaneous instances of an individual server-connection channel that can be started from a single client.

In this context, connections that originate from the same remote network address are regarded as coming from the same client.

This field is used only on server-connection channels.

The field can have a value in the range 0 - 999 999 999. A value of zero prevents all client access.

The default value of this field is 999 999 999.

If the value of this field is reduced to a number that is less than the number of instances of the server-connection channel that are currently running from individual clients, then those running instances are not affected. However, new instances from any of those clients cannot start until sufficient existing instances have ceased to run such that the number of currently running instances, originating from the client attempting to start a new one, is less than the value of the field.

MaxMsgLength (MQLONG):

This field specifies the maximum message length that can be transmitted on the channel.

This is compared with the value for the remote channel and the actual maximum is the lower of the two values.

MCAName (MQCHAR20):

This field is a reserved field.

The value of this field is blank.

The length of this field is given by MQ_MCA_NAME_LENGTH.

MCASecurityId (MQBYTE40):

This field specifies the security identifier for the MCA.

This field is not relevant for channels with a *ChannelType* of MQCHT_CLNTCONN.

The following special value indicates that there is no security identifier:

MQSID_NONE
No security identifier specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQSID_NONE_ARRAY is also defined; this constant has the same value as MQSID_NONE, but is an array of characters instead of a string.

This is an input/output field to the exit. The length of this field is given by MQ_SECURITY_ID_LENGTH. This field is not present if *Version* is less than MQCD_VERSION_6.

MCAType (MQLONG):

This field specifies the type of message channel agent program.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

The value is one of the following:

MQMCAT_PROCESS
Process.

The message channel agent runs as a separate process.

MQMCAT_THREAD
Thread (IBM i, UNIX, and Windows).

The message channel agent runs as a separate thread.

This field is not present when *Version* is less than MQCD_VERSION_2.

MCAUserIdentifier (MQCHAR12):

This field specifies the user identifier for the message channel agent (MCA).

This field uses the first 12 bytes of the MCA user identifier, and can be set by a security agent.

There are two fields that contain the MCA user identifier:

- *MCAUserIdentifier* contains the first 12 bytes of the MCA user identifier, and is padded with blanks if the identifier is shorter than 12 bytes. *MCAUserIdentifier* can be blank.
- *LongMCAUserIdPtr* points to the full MCA user identifier, which can be longer than 12 bytes. Its length is given by *LongMCAUserIdLength*. The full identifier contains no trailing blanks, and is not null-terminated. If the identifier is blank, *LongMCAUserIdLength* is zero, and the value of *LongMCAUserIdPtr* is undefined.

  Note: *LongMCAUserIdPtr* is not present if *Version* is less than MQCD_VERSION_6.

If the MCA user identifier is nonblank, it specifies the user identifier to be used by the message channel agent for authorization to access IBM MQ resources. For channel types MQCHT_REQUESTER, MQCHT_RECEIVER, and MQCHT_CLUSRCVR, if PutAuthority is MQPA_DEFAULT this is the user identifier used for authorization checks for the put operation to destination queues.

If the MCA user identifier is blank, the message channel agent uses its default user identifier.

The MCA user identifier can be set by a security exit to indicate the user identifier that the message channel agent must use. The exit can change either *MCAUserIdentifier*, or the string pointed at by *LongMCAUserIdPtr*. If both are changed but differ from each other, the MCA uses *LongMCAUserIdPtr* in preference to *MCAUserIdentifier*. If the exit changes the length of the string addressed by *LongMCAUserIdPtr*, *LongMCAUserIdLength* must be set correspondingly. If the exit increases the length of the identifier, the exit must allocate storage of the required length, set that storage to the required identifier, and place the address of that storage in *LongMCAUserIdPtr*. The exit is responsible for freeing that storage when the exit is later invoked with the MQXR_TERM reason.

For channels with a *ChannelType* of MQCHT_SVRCONN, if *MCAUserIdentifier* in the channel definition is blank, any user identifier transferred from the client is copied into it. This user identifier (after any modification by the security exit at the server) is the one which the client application is assumed to be running under.

The MCA user identifier is not relevant for channels with a *ChannelType* of MQCHT_SDR, MQCHT_SVR, MQCHT_CLNTCONN, MQCHT_CLUSSDR.

This is an input/output field to the exit. The length of this field is given by MQ_USER_ID_LENGTH. This field is not present when *Version* is less than MQCD_VERSION_2.

ModeName (MQCHAR8):

This field specifies the LU 6.2 mode name.

This field is relevant only if the transmission protocol (*TransportType*) is MQXPT_LU62, and the *ChannelType* is not MQCHT_SVRCONN or MQCHT_RECEIVER.

This field is always blank. The information is contained in the communications Side Object instead.

The length of this field is given by MQ_MODE_NAME_LENGTH.

MsgCompList [16] (MQLONG):

This field specifies the list of message data compression techniques which are supported by the channel.

The list contains one or more of the following values:

MQCOMPRESS_NONE
No message data compression is performed.

MQCOMPRESS_RLE
Message data compression is performed using run-length encoding.

MQCOMPRESS_ZLIBFAST
Message data compression is performed using the zlib compression technique. A fast compression time is preferred.

MQCOMPRESS_ZLIBHIGH
Message data compression is performed using the zlib compression technique. A high level of compression is preferred.

Unused values in the array are set to MQCOMPRESS_NOT_AVAILABLE.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_8.

MsgExit (MQCHARn):

This field specifies the channel message exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately after a message has been retrieved from the transmission queue (sender or server), or immediately before a message is put to a destination queue (receiver or requester).

  The exit is given the entire application message and transmission queue header for modification.
- At initialization and termination of the channel.

This field is not relevant for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN; a message exit is never invoked for such channels.

See “MQCD - Channel definition” on page 3559 for a description of the content of this field in various environments.

The length of this field is given by MQ_EXIT_NAME_LENGTH.

Note: The value of this constant is environment-specific.

MsgExitPtr (MQPTR):

This field specifies the address of the first *MsgExit* field.

If *MsgExitsDefined* is greater than zero, this address is the address of the list of names of each channel message exit in the chain.

Each name is in a field of length *ExitNameLength*, padded to the right with blanks. There are *MsgExitsDefined* fields adjoining one another - one for each exit.

Any changes made to these names by an exit are preserved, although the message channel exit takes no explicit action - it does not change which exits are invoked.

If *MsgExitsDefined* is zero, this field is the null pointer.

On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

MsgExitsDefined (MQLONG):

This field specifies the number of channel message exits defined in the chain.

It is greater than or equal to zero.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

MsgRetryCount (MQLONG):

This field specifies the number of times MCA tries to put the message, after the first attempt has failed.

This field indicates the number of times that the MCA tries the open or put operation, if the first MQOPEN or MQPUT fails with completion code MQCC_FAILED. The effect of this attribute depends on whether *MsgRetryExit* is blank or nonblank:

- If *MsgRetryExit* is blank, the **MsgRetryCount** attribute controls whether the MCA attempts retries. If the attribute value is zero, no retries are attempted. If the attribute value is greater than zero, the retries are attempted at intervals given by the **MsgRetryInterval** attribute.

  Retries are attempted only for the following reason codes:
  - MQRC_PAGESET_FULL
  - MQRC_PUT_INHIBITED
  - MQRC_Q_FULL

  For other reason codes, the MCA proceeds immediately to its normal failure processing, without retrying the failing message.
- If *MsgRetryExit* is nonblank, the **MsgRetryCount** attribute does not affect the MCA; instead it is the message retry exit that determines how many times the retry is attempted, and at what intervals; the exit is invoked even if the **MsgRetryCount** attribute is zero.

  The **MsgRetryCount** attribute is made available to the exit in the MQCD structure, but the exit it not required to honor it - retries continue indefinitely until the exit returns MQXCC_SUPPRESS_FUNCTION in the *ExitResponse* field of MQCXP.

This field is relevant only for channels with a *ChannelType* of MQCHT_REQUESTER, MQCHT_RECEIVER, or MQCHT_CLUSRCVR.

This field is not present when *Version* is less than MQCD_VERSION_3.

MsgRetryExit (MQCHARn):

This field specifies the channel message retry exit name.

The message retry exit is an exit that is invoked by the MCA when the MCA receives a completion code of MQCC_FAILED from an MQOPEN or MQPUT call. The purpose of the exit is to specify a time interval for which the MCA waits before trying the MQOPEN or MQPUT operation again. Alternatively, the exit can be set to not try the operation again.

The exit is invoked for all reason codes that have a completion code of MQCC_FAILED - the settings of the exit determine which reason codes it wants the MCA to try again, for how many attempts, and at what time intervals.

When the operation is not to be attempted any more, the MCA performs its normal failure processing; this processing includes generating an exception report message (if specified by the sender), and either placing the original message on the dead-letter queue or discarding the message (according to whether the sender specified MQRO_DEAD_LETTER_Q or MQRO_DISCARD_MSG). Failures involving the dead-letter queue (for example, dead-letter queue full) do not cause the message-retry exit to be invoked.

If the exit name is nonblank, the exit is called at the following times:

- Immediately before performing the wait before trying to deliver a message again
- At initialization and termination of the channel

See "MQCD - Channel definition" on page 3559 for a description of the content of this field in various environments.

This field is relevant only for channels with a *ChannelType* of MQCHT_REQUESTER, MQCHT_RECEIVER, or MQCHT_CLUSRCVR.

The length of this field is given by MQ_EXIT_NAME_LENGTH.

Note: The value of this constant is environment-specific.

This field is not present when *Version* is less than MQCD_VERSION_3.

MsgRetryInterval (MQLONG):

This field specifies the minimum interval in milliseconds after which the open or put operation is retried.

The effect of this attribute depends on whether *MsgRetryExit* is blank or nonblank:

- If *MsgRetryExit* is blank, the **MsgRetryInterval** attribute specifies the minimum period that the MCA waits before retrying a message, if the first MQOPEN or MQPUT fails with completion code MQCC_FAILED. A value of zero means that the retry will be performed as soon as possible after the previous attempt. Retries are performed only if *MsgRetryCount* is greater than zero.

  This attribute is also used as the wait time if the message-retry exit returns an invalid value in the *MsgRetryInterval* field in MQCXP.
- If *MsgRetryExit* is not blank, the **MsgRetryInterval** attribute does not affect the MCA; instead it is the message-retry exit which determines how long the MCA waits. The **MsgRetryInterval** attribute is made available to the exit in the MQCD structure, but the exit it not required to honor it.

The value is in the range 0 through 999 999 999.

This field is relevant only for channels with a *ChannelType* of MQCHT_REQUESTER, MQCHT_RECEIVER, or MQCHT_CLUSRCVR.

This field is not present when *Version* is less than MQCD_VERSION_3.

The following fields in this structure are not present if *Version* is less than MQCD_VERSION_4.

MsgRetryUserData (MQCHAR32):

This field specifies the channel message retry exit user data.

This data is passed to the channel message-retry exit in the *ExitData* field of the **ChannelExitParms** parameter (see MQ_CHANNEL_EXIT).

This field initially contains the data that was set in the channel definition. However, during the lifetime of this MCA instance, any changes made to the contents of this field by an exit of any type are preserved by the MCA, and made visible to subsequent invocations of exits (regardless of type) for this MCA instance. Such changes do not affect the channel definition used by other MCA instances. Any characters (including binary data) can be used.

This field is relevant only for channels with a *ChannelType* of MQCHT_REQUESTER, MQCHT_RECEIVER, or MQCHT_CLUSRCVR.

The length of this field is given by MQ_EXIT_DATA_LENGTH. This field is not present when *Version* is less than MQCD_VERSION_3.

This field is not relevant in IBM MQ for IBM i.

MsgUserData (MQCHAR32):

This field specifies channel message exit user data.

This data is passed to the channel message exit in the *ExitData* field of the **ChannelExitParms** parameter (see MQ_CHANNEL_EXIT).

This field initially contains the data that was set in the channel definition. However, during the lifetime of this MCA instance, any changes made to the contents of this field by an exit of any type are preserved by the MCA, and made visible to subsequent invocations of exits (regardless of type) for this MCA instance. Such changes do not affect the channel definition used by other MCA instances. Any characters (including binary data) can be used.

The length of this field is given by MQ_EXIT_DATA_LENGTH.

This field is not relevant in IBM MQ for IBM i.

MsgUserDataPtr (MQPTR):

This field specifies the address of the first *MsgUserData* field.

If *MsgExitsDefined* is greater than zero, this address is the address of the list of user data items for each channel message exit in the chain.

Each user data item is in a field of length *ExitDataLength*, padded to the right with blanks. There are *MsgExitsDefined* fields adjoining one another - one for each exit. If the number of user data items defined is less than the number of exit names, undefined user data items are set to blanks. Conversely, if the number of user data items defined is greater than the number of exit names, the excess user data items are ignored and not presented to the exit.

Any changes made to these values by an exit are preserved. This allows one exit to pass information to another exit. No validation is carried out on any changes so, for example, binary data can be written to these fields if required.

If *MsgExitsDefined* is zero, this field is the null pointer.

On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

NetworkPriority (MQLONG):

This field specifies the priority of the network connection for the channel.

When multiple paths to a particular destination are available, the path with the highest priority is chosen. The value is in the range 0 through 9; 0 is the lowest priority.

This field is relevant only for channels with a *ChannelType* of MQCHT_CLUSSDR or MQCHT_CLUSRCVR.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_5.

The following fields in this structure are not present if *Version* is less than MQCD_VERSION_6.

NonPersistentMsgSpeed (MQLONG):

This field specifies the speed at which nonpersistent messages travel through the channel.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_RECEIVER, MQCHT_REQUESTER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

The value is one of the following:

MQNPMS_NORMAL
Normal speed.

If a channel is defined to be MQNPMS_NORMAL, nonpersistent messages travel through the channel at normal speed. This has the advantage that these messages are not lost if there is a channel failure. Also, persistent and nonpersistent messages on the same transmission queue maintain their order relative to each other.

MQNPMS_FAST
Fast speed.

If a channel is defined to be MQNPMS_FAST, nonpersistent messages travel through the channel at fast speed. This improves the throughput of the channel, but means that nonpersistent messages are lost if there is a channel failure. Also, it is possible for nonpersistent messages to jump ahead of persistent messages waiting on the same transmission queue, that is, the order of nonpersistent messages is not maintained relative to persistent messages. However the order of nonpersistent messages relative to each other is maintained. Similarly, the order of persistent messages relative to each other is maintained.

Password (MQCHAR12):

This field specifies the password used by the message channel agent when attempting to initiate a secure SNA session with a remote message channel agent.

This field can be nonblank only on UNIX, and Windows, and is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, or MQCHT_CLNTCONN. On z/OS, this field is not relevant.

The length of this field is given by MQ_PASSWORD_LENGTH. However, only the first 10 characters are used.

This field is not present if *Version* is less than MQCD_VERSION_2.

PropertyControl (MQLONG):

This field specifies what happens to properties of messages when the message is about to be sent to a V6 or prior queue manager (a queue manager that does not understand the concept of a property descriptor).

The value can be any of the following values:

MQPROP_COMPATIBILITY
If the message contains a property with a prefix of **mcd.**, **jms.**, **usr.**, or **mqext.**, all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except those properties contained in the message descriptor (or extension), are discarded and are no longer accessible to the application.

This value is the default value; it allows applications, which expect JMS-related properties to be in an MQRFH2 header in the message data, to continue to work unmodified.

MQPROP_NONE
All properties of the message, except those properties in the message descriptor (or extension), are removed from the message before the message is sent to the remote queue manager.

MQPROP_ALL
All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except those properties in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

This attribute is applicable to Sender, Server, Cluster Sender, and Cluster Receiver channels.

"MQIA_* (Integer Attribute Selectors)" on page 2105
"MQPROP_* (Queue and Channel Property Control Values and Maximum Properties Length)" on page 2139

PutAuthority (MQLONG):

This field specifies whether the user identifier in the context information associated with a message is used to establish authority to put the message to the destination queue.

This field is relevant only for channels with a *ChannelType* of MQCHT_REQUESTER, MQCHT_RECEIVER, or MQCHT_CLUSRCVR. It is one of the following:

MQPA_DEFAULT
Default user identifier is used.

MQPA_CONTEXT
Context user identifier is used.

MQPA_ALTERNATE_OR_MCA
The user ID from the UserIdentifier field of the message descriptor is used. Any user ID received from the network is not used. This value is supported only on z/OS.

MQPA_ONLY_MCA
The default user ID is used. Any user ID received from the network is not used. This value is supported only on z/OS.

QMgrName (MQCHAR48):

This field specifies the name of the queue manager that an exit can connect to.

For channels with a *ChannelType* other than MQCHT_CLNTCONN, this field is the name of the queue manager that an exit can connect to, which on UNIX, Linux, and Windows, is always nonblank.

The length of this field is given by MQ_Q_MGR_NAME_LENGTH.

ReceiveExit (MQCHARn):

This field specifies the channel receive exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately before the received network data is processed.
  The exit is given the complete transmission buffer as received. The contents of the buffer can be modified as required.
- At initialization and termination of the channel.

See "MQCD - Channel definition" on page 3559 for a description of the content of this field in various environments.

The length of this field is given by MQ_EXIT_NAME_LENGTH.

Note: The value of this constant is environment-specific.

ReceiveExitPtr (MQPTR):

This field specifies the address of the first *ReceiveExit* field.

If *ReceiveExitsDefined* is greater than zero, this address is the address of the list of names of each channel receive exit in the chain.

Each name is in a field of length *ExitNameLength*, padded to the right with blanks. There are *ReceiveExitsDefined* fields adjoining one another - one for each exit.

Any changes made to these names by an exit are preserved, although the message channel exit takes no explicit action - it does not change which exits are invoked.

If *ReceiveExitsDefined* is zero, this field is the null pointer.

On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

ReceiveExitsDefined (MQLONG):

This field specifies the number of channel receive exits defined in the chain.

It is greater than or equal to zero.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

ReceiveUserData (MQCHAR32):

This channel specifies channel receive exit user data.

This data is passed to the channel receive exit in the *ExitData* field of the **ChannelExitParms** parameter (see MQ_CHANNEL_EXIT).

This field initially contains the data that was set in the channel definition. However, during the lifetime of this MCA instance, any changes made to the contents of this field by an exit of any type are preserved by the MCA, and made visible to subsequent invocations of exits (regardless of type) for this MCA instance. This applies to exits on different conversations. Such changes do not affect the channel definition used by other MCA instances. Any characters (including binary data) can be used.

The length of this field is given by MQ_EXIT_DATA_LENGTH.

This field is not relevant in IBM MQ for IBM i.

The following fields in this structure are not present if *Version* is less than MQCD_VERSION_2.

ReceiveUserDataPtr (MQPTR):

This field specifies the address of the first *ReceiveUserData* field.

If *ReceiveExitsDefined* is greater than zero, this address is the address of the list of user data item for each channel receive exit in the chain.

Each user data item is in a field of length *ExitDataLength*, padded to the right with blanks. There are *ReceiveExitsDefined* fields adjoining one another - one for each exit. If the number of user data items defined is less than the number of exit names, undefined user data items are set to blanks. Conversely, if the number of user data items defined is greater than the number of exit names, the excess user data items are ignored and not presented to the exit.

Any changes made to these values by an exit are preserved. This allows one exit to pass information to another exit. No validation is carried out on any changes so, for example, binary data can be written to these fields if required.

If *ReceiveExitsDefined* is zero, this field is the null pointer.

On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

The following fields in this structure are not present if *Version* is less than MQCD_VERSION_5.

RemotePassword (MQCHAR12):

This field specifies the password from a partner.

This field contains valid information only if *ChannelType* is MQCHT_CLNTCONN or MQCHT_SVRCONN.

- For a security exit at an MQCHT_CLNTCONN channel, this password is a password which has been obtained from the environment. The exit can choose to send it to the security exit at the server.
- For a security exit at an MQCHT_SVRCONN channel, this field might contain a password which has been obtained from the environment at the client, if there is no client security exit. The exit can use this password to validate the user identifier in *RemoteUserIdentifier*.

  If there is a security exit at the client, then this information can be obtained in a security flow from the client.

The length of this field is given by MQ_PASSWORD_LENGTH. This field is not present if *Version* is less than MQCD_VERSION_2.

The following fields in this structure are not present if *Version* is less than MQCD_VERSION_3.

RemoteSecurityId (MQBYTE40):

This field specifies the security identifier for the remote user.

This field is relevant only for channels with a *ChannelType* of MQCHT_CLNTCONN or MQCHT_SVRCONN.

The following special value indicates that there is no security identifier:

MQSID_NONE
No security identifier specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQSID_NONE_ARRAY is also defined; this constant has the same value as MQSID_NONE, but is an array of characters instead of a string.

This is an input field to the exit. The length of this field is given by MQ_SECURITY_ID_LENGTH. This field is not present if *Version* is less than MQCD_VERSION_6.

The following fields in this structure are not present if *Version* is less than MQCD_VERSION_7.

RemoteUserIdentifier (MQCHAR12):

This field specifies the first 12 bytes of a user identifier from a partner.

There are two fields that contain the remote user identifier:

- *RemoteUserIdentifier* contains the first 12 bytes of the remote user identifier, and is padded with blanks if the identifier is shorter than 12 bytes. *RemoteUserIdentifier* can be blank.
- *LongRemoteUserIdPtr* points to the full remote user identifier, which can be longer than 12 bytes. Its length is given by *LongRemoteUserIdLength*. The full identifier contains no trailing blanks, and is not null-terminated. If the identifier is blank, *LongRemoteUserIdLength* is zero, and the value of *LongRemoteUserIdPtr* is undefined.

  LongRemoteUserIdPtr is not present if *Version* is less than MQCD_VERSION_6.

The remote user identifier is relevant only for channels with a *ChannelType* of MQCHT_CLNTCONN or MQCHT_SVRCONN.

- For a security exit on an MQCHT_CLNTCONN channel, this value is a user identifier that has been obtained from the environment. The exit can choose to send it to the security exit at the server.
- For a security exit on an MQCHT_SVRCONN channel, this field might contain a user identifier which has been obtained from the environment at the client, if there is no client security exit. The exit might validate this user ID (possibly with the password in *RemotePassword*) and update the value in *MCAUserIdentifier*.

  If there is a security exit at the client, then this information can be obtained in a security flow from the client.

The length of this field is given by MQ_USER_ID_LENGTH. This field is not present if *Version* is less than MQCD_VERSION_2.

SecurityExit (MQCHARn):

This field specifies the channel security exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately after establishing a channel.

  Before any messages are transferred, the exit is given the opportunity to instigate security flows to validate connection authorization.
- Upon receipt of a response to a security message flow.

  Any security message flows received from the remote processor on the remote machine are given to the exit.
- At initialization and termination of the channel.

See "MQCD - Channel definition" on page 3559 for a description of the content of this field in various environments.

The length of this field is given by MQ_EXIT_NAME_LENGTH.

Note: The value of this constant is environment-specific.

SecurityUserData (MQCHAR32):

This channel specifies the channel security exit user data.

This data is passed to the channel security exit in the *ExitData* field of the **ChannelExitParms** parameter (see MQ_CHANNEL_EXIT).

This field initially contains the data that was set in the channel definition. However, during the lifetime of this MCA instance, any changes made to the contents of this field by an exit of any type are preserved by the MCA, and made visible to subsequent invocations of exits (regardless of type) for this MCA instance. This applies to exits on different conversations. Such changes do not effect on the channel definition used by other MCA instances. Any characters (including binary data) can be used.

The length of this field is given by MQ_EXIT_DATA_LENGTH.

This field is not relevant in IBM MQ for IBM i.

SendExit (MQCHARn):

This field specifies the channel send exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately before data is sent out on the network.

  The exit is given the complete transmission buffer before it is transmitted. The contents of the buffer can be modified as required.
- At initialization and termination of the channel.

See "MQCD - Channel definition" on page 3559 for a description of the content of this field in various environments.

The length of this field is given by MQ_EXIT_NAME_LENGTH.

Note: The value of this constant is environment-specific.

SendExitPtr (MQPTR):

This field specifies the address of the first *SendExit* field.

If *SendExitsDefined* is greater than zero, this address is the address of the list of names of each channel send exit in the chain.

Each name is in a field of length *ExitNameLength*, padded to the right with blanks. There are *SendExitsDefined* fields adjoining one another - one for each exit.

Any changes made to these names by an exit are preserved, although the message send exit takes no explicit action - it does not change which exits are invoked.

If *SendExitsDefined* is zero, this field is the null pointer.

On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

SendExitsDefined (MQLONG):

This field specifies the number of channel send exits defined in the chain.

It is greater than or equal to zero.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

SendUserData (MQCHAR32):

This field specifies the channel send exit user data.

This data is passed to the channel send exit in the *ExitData* field of the **ChannelExitParms** parameter (see MQ_CHANNEL_EXIT).

This field initially contains the data that was set in the channel definition. However, during the lifetime of this MCA instance, any changes made to the contents of this field by an exit of any type are preserved by the MCA, and made visible to subsequent invocations of exits (regardless of type) for this MCA instance. This applies to exits on different conversations. Such changes do not affect the channel definition used by other MCA instances. Any characters (including binary data) can be used.

The length of this field is given by MQ_EXIT_DATA_LENGTH.

This field is not relevant in IBM MQ for IBM i.

SendUserDataPtr (MQPTR):

This field specifies the address of the *SendUserData* field.

If *SendExitsDefined* is greater than zero, this address is the address of the list of user data items for each channel message exit in the chain.

Each user data item is in a field of length *ExitDataLength*, padded to the right with blanks. There are *MsgExitsDefined* fields adjoining one another - one for each exit. If the number of user data items defined is less than the number of exit names, undefined user data items are set to blanks. Conversely, if the number of user data items defined is greater than the number of exit names, the excess user data items are ignored and not presented to the exit.

Any changes made to these values by an exit are preserved. This allows one exit to pass information to another exit. No validation is carried out on any changes so, for example, binary data can be written to these fields if required.

If *SendExitsDefined* is zero, this field is the null pointer.

On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

SeqNumberWrap (MQLONG):

This field specifies the highest allowable message sequence number.

When this value is reached, sequence numbers wrap to start again at 1.

This value is non-negotiable and must match in both the local and remote channel definitions.

This field is not relevant for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN.

SharingConversations (MQLONG):

This field specifies the maximum number of conversations that can share a channel instance associated with this channel.

This field is used on client connection and server-connection channels.

A value of 0 means that the channel operates as it did in versions earlier than IBM WebSphere MQ Version 7.0 with respect to the following attributes:

- Conversation sharing
- Read ahead
- `STOP CHANNEL(channelname) MODE(QUIESCE)`
- Heartbeating
- Client asynchronous consumption

A value of 1 is the minimum value for IBM MQ V7.0 behavior. Although only one conversation is allowed on the channel instance, read ahead, asynchronous consumption, and the Version 7 behavior of `CLNTCONN-SVRCONN` heartbeating and quiescent channel stopping are available.

This is an input field to the exit. It is not present if *Version* is less than MQCD_VERSION_9.

The default value of this field is 10.

Note: *MaxInstances* and *MaxInstancesPerClient* limits applied to a channel restrict the number of channel instances, not the number of conversations that might be sharing those instances.

ShortConnectionName (MQCHAR20):

This field specifies the first 20 bytes of a connection name.

If the *Version* field is MQCD_VERSION_1, *ShortConnectionName* contains the full connection name.

If the *Version* field is MQCD_VERSION_2 or greater, *ShortConnectionName* contains the first 20 characters of the connection name. The full connection name is given by the *ConnectionName* field; *ShortConnectionName* and the first 20 characters of *ConnectionName* are identical.

See *ConnectionName* for details of the contents of this field.

Note: The name of this field was changed for MQCD_VERSION_2 and subsequent versions of MQCD; the field was previously called *ConnectionName*.

The length of this field is given by MQ_SHORT_CONN_NAME_LENGTH.

ShortRetryCount (MQLONG):

This field specifies the maximum number of attempts that are made to connect to a remote machine.

This field is the maximum number of attempts that are made to connect to the remote machine, at intervals specified by *ShortRetryInterval*, before the (normally longer) *LongRetryCount* and *LongRetryInterval* are used.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

ShortRetryInterval (MQLONG):

This field specifies the maximum number of seconds to wait before reattempting connection to the remote machine.

The interval between retries might be extended if the channel has to wait to become active.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

SSLCipherSpec (MQCHAR32):

This field specifies the Cipher Spec that is in use when using TLS.

If SSLCipherSpec is blank, the channel is not using TLS. If it is not blank, this field contains a string specifying the CipherSpec in use.

This parameter is valid for all channel types. It is supported on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS. It is valid only for channel types of a transport type (TRPTYPE) of TCP.

This is an input field to the exit. The length of this field is given by MQ_SSL_CIPHER_SPEC_LENGTH. The field is not present if *Version* is less than MQCD_VERSION_7.

SSLClientAuth (MQLONG):

This field specifies whether TLS client authentication is required.

This field is relevant only to SVRCONN channel definitions.

It is one of the following values:

MQSCA_REQUIRED
Client authentication required.

MQSCA_OPTIONAL
Client authentication optional.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_7.

SSLPeerNameLength (MQLONG):

This field specifies the length in bytes of the TLS peer name pointed to by *SSLPeerNamePtr*.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_7.

SSLPeerNamePtr (MQPTR):

This field specifies the address of the TLS peer name.

When a certificate is received during a successful TLS handshake, the Distinguished Name of the subject of the certificate is copied into the MQCD field accessed by SSLPeerNamePtr at the end of the channel which receives the certificate. It overwrites the SSLPeerName value for the channel if this value is present in the channel definition of the local user. If a security exit is specified at this end of the channel it receives the Distinguished Name from the peer certificate in the MQCD.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_7.

Note: Security exit applications constructed prior to the release of IBM WebSphere MQ Version 7.1 may require updating. For more information see Channel security exit programs.

StrucLength (MQLONG):

This field specifies the length in bytes of the MQCD structure.

The length does not include any of the strings addressed by pointer fields contained within the structure. The value is one of the following:

MQCD_LENGTH_4
Length of version-4 channel definition structure.

MQCD_LENGTH_5
Length of version-5 channel definition structure.

MQCD_LENGTH_6
Length of version-6 channel definition structure.

MQCD_LENGTH_7
Length of version-7 channel definition structure.

MQCD_LENGTH_8
Length of version-8 channel definition structure.

MQCD_LENGTH_9
Length of version-9 channel definition structure.

The following constant specifies the length of the current version:

MQCD_CURRENT_LENGTH
Length of current version of channel definition structure.

Note: These constants have values that are environment-specific.

The field is not present if *Version* is less than MQCD_VERSION_4.

TpName (MQCHAR64):

This field specifies the LU 6.2 transaction program name.

This field is relevant only if the transmission protocol (*TransportType*) is MQXPT_LU62, and the *ChannelType* is not MQCHT_SVRCONN or MQCHT_RECEIVER.

This field is always blank on platforms on which the information is contained in the communications Side Object instead.

The length of this field is given by MQ_TP_NAME_LENGTH.

TransportType (MQLONG):

This field specifies the transmission protocol to be used.

The value is not checked if the channel was initiated from the other end.

It is one of the following values:

MQXPT_LU62
LU 6.2 transport protocol.

MQXPT_TCP
TCP/IP transport protocol.

MQXPT_NETBIOS
NetBIOS transport protocol.

This value is supported in the following environments: Windows.

MQXPT_SPX
SPX transport protocol.

This value is supported in the following environments: Windows, plus IBM MQ clients connected to these systems.

UseDLQ (MQLONG):

This field specifies whether the dead-letter queue (or undelivered message queue) is used when messages cannot be delivered by channels.

It can contain one of the following values:

MQUSEDLQ_NO
Messages that cannot be delivered by a channel are treated as a failure. The channel either discards the message, or the channel ends, in accordance with the NPMSPEED setting.

MQUSEDLQ_YES
When the DEADQ queue manager attribute provides the name of a dead-letter queue, then it is used, else the behavior is as for NO. YES is the default value.

UserIdentifier (MQCHAR12):

This field specifies the user identifier used by the message channel agent when attempting to initiate a secure SNA session with a remote message channel agent.

This field can be nonblank only on UNIX and Windows, and is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, or MQCHT_CLNTCONN. On z/OS, this field is not relevant.

The length of this field is given by MQ_USER_ID_LENGTH. However, only the first 10 characters are used.

This field is not present when *Version* is less than MQCD_VERSION_2.

Version (MQLONG):

The Version field specifies the highest version number that you can set for the structure.

The value depends on the environment:

MQCD _VERSION_1
Version 1 channel definition structure.

MQCD _VERSION_2
Version 2 channel definition structure.

MQCD _VERSION_3
Version 3 channel definition structure.

MQCD _VERSION_4
Version 4 channel definition structure.

MQCD _VERSION_5
Version 5 channel definition structure.

MQCD _VERSION_6
Version 6 channel definition structure.

MQCD _VERSION_7
Version 7 channel definition structure.

MQCD _VERSION_8
Version 8 channel definition structure.

MQCD _VERSION_9
Version 9 channel definition structure.

Version 9 is the highest that you can set the field to on IBM WebSphere MQ Version 7.0 and IBM WebSphere MQ Version 7.0.1 on all platforms.

MQCD _VERSION_10
Version 10 channel definition structure.

Version 10 is the highest that you can set the field to on IBM WebSphere MQ Version 7.1 and IBM WebSphere MQ Version 7.5 on all platforms.

MQCD _VERSION_11
Version 11 channel definition structure.

Version 11 is the highest that you can set the field to on IBM MQ Version 8.0 on all platforms.

Fields that exist only in the more recent versions of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQCD_CURRENT_VERSION
The value set in MQCD_CURRENT_VERSION is the current version of the channel definition structure being used.

The value of MQCD_CURRENT_VERSION depends on the environment. It contains the highest value supported by the platform.

MQCD_CURRENT_VERSION is not used to initialize the default structures provided in the header, copy, and include files provided for different programming languages. The default initialization of Version depends on the platform and release.

For IBM WebSphere MQ Version 7.0 and later versions, the MQCD declarations in the header, copy, and include files are initialized to MQCD_VERSION_6. To use additional MQCD fields, applications

must set the version number to MQCD_CURRENT_VERSION. If you are writing an application that is portable between several environments, you must choose a version that is supported in all the environments.

Tip: When a new version of the MQCD structure is introduced, the layout of the existing part is not changed. The exit must check the version number. It must be equal to or greater than the lowest version that contains the fields that the exit needs to use.

XmitQName (MQCHAR48):

This field specifies the name of the transmission queue from which messages are retrieved.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER or MQCHT_SERVER.

The length of this field is given by MQ_Q_NAME_LENGTH.

C declaration:

This declaration is the C declaration for the MQCD structure.

```
typedef struct tagMQCD MQCD;
typedef MQCD MQPOINTER PMQCD;
typedef PMQCD MQPOINTER PPMQCD;

 struct tagMQCD {
   MQCHAR    ChannelName[20];          /* Channel definition name */
   MQLONG    Version;                  /* Structure version number */
   MQLONG    ChannelType;              /* Channel type */
   MQLONG    TransportType;            /* Transport type */
   MQCHAR    Desc[64];                 /* Channel description */
   MQCHAR    QMgrName[48];             /* Queue manager name */
   MQCHAR    XmitQName[48];            /* Transmission queue name */
   MQCHAR    ShortConnectionName[20];  /* First 20 bytes of */
                                       /* connection name */
   MQCHAR    MCAName[20];              /* Reserved */
   MQCHAR    ModeName[8];              /* LU 6.2 Mode name */
   MQCHAR    TpName[64];               /* LU 6.2 transaction program */
                                       /* name */
   MQLONG    BatchSize;                /* Batch size */
   MQLONG    DiscInterval;             /* Disconnect interval */
   MQLONG    ShortRetryCount;          /* Short retry count */
   MQLONG    ShortRetryInterval;       /* Short retry wait interval */
   MQLONG    LongRetryCount;           /* Long retry count */
   MQLONG    LongRetryInterval;        /* Long retry wait interval */
   MQCHAR    SecurityExit[128];        /* Channel security exit name */
   MQCHAR    MsgExit[128];             /* Channel message exit name */
   MQCHAR    SendExit[128];            /* Channel send exit name */
   MQCHAR    ReceiveExit[128];         /* Channel receive exit name */
   MQLONG    SeqNumberWrap;            /* Highest allowable message */
                                       /* sequence number */
   MQLONG    MaxMsgLength;             /* Maximum message length */
   MQLONG    PutAuthority;             /* Put authority */
   MQLONG    DataConversion;           /* Data conversion */
   MQCHAR    SecurityUserData[32];     /* Channel security exit user */
                                       /* data */
   MQCHAR    MsgUserData[32];          /* Channel message exit user */
                                       /* data */
   MQCHAR    SendUserData[32];         /* Channel send exit user */
                                       /* data */
   MQCHAR    ReceiveUserData[32];      /* Channel receive exit user */
                                       /* data */
   /* Ver:1 */
   MQCHAR    UserIdentifier[12];       /* User identifier */
   MQCHAR    Password[12];             /* Password */
```

```
MQCHAR    MCAUserIdentifier[12];     /* First 12 bytes of MCA user */
                                     /* identifier */
MQLONG    MCAType;                   /* Message channel agent type */
MQCHAR    ConnectionName[264];       /* Connection name */
MQCHAR    RemoteUserIdentifier[12];  /* First 12 bytes of user */
                                     /* identifier from partner */
MQCHAR    RemotePassword[12];        /* Password from partner */
/* Ver:2 */
MQCHAR    MsgRetryExit[128];         /* Channel message retry exit */
                                     /* name */
MQCHAR    MsgRetryUserData[32];      /* Channel message retry exit */
                                     /* user data */
MQLONG    MsgRetryCount;             /* Number of times MCA will */
                                     /* try to put the message, */
                                     /* after first attempt has */
                                     /* failed */
MQLONG    MsgRetryInterval;          /* Minimum interval in */
                                     /* milliseconds after which */
                                     /* the open or put operation */
                                     /* will be retried */
/* Ver:3 */
MQLONG    HeartbeatInterval;         /* Time in seconds between */
                                     /* heartbeat flows */
MQLONG    BatchInterval;             /* Batch duration */
MQLONG    NonPersistentMsgSpeed;     /* Speed at which */
                                     /* nonpersistent messages are */
                                     /* sent */
MQLONG    StrucLength;               /* Length of MQCD structure */
MQLONG    ExitNameLength;            /* Length of exit name */
MQLONG    ExitDataLength;            /* Length of exit user data */
MQLONG    MsgExitsDefined;           /* Number of message exits */
                                     /* defined */
MQLONG    SendExitsDefined;          /* Number of send exits */
                                     /* defined */
MQLONG    ReceiveExitsDefined;       /* Number of receive exits */
                                     /* defined */
MQPTR     MsgExitPtr;                /* Address of first MsgExit */
                                     /* field */
MQPTR     MsgUserDataPtr;            /* Address of first */
                                     /* MsgUserData field */
MQPTR     SendExitPtr;               /* Address of first SendExit */
                                     /* field */
MQPTR     SendUserDataPtr;           /* Address of first */
                                     /* SendUserData field */
MQPTR     ReceiveExitPtr;            /* Address of first */
                                     /* ReceiveExit field */
MQPTR     ReceiveUserDataPtr;        /* Address of first */
                                     /* ReceiveUserData field */
/* Ver:4 */
MQPTR     ClusterPtr;                /* Address of a list of */
                                     /* cluster names */
MQLONG    ClustersDefined;           /* Number of clusters to */
                                     /* which the channel belongs */
MQLONG    NetworkPriority;           /* Network priority */
/* Ver:5 */
MQLONG    LongMCAUserIdLength;       /* Length of long MCA user */
                                     /* identifier */
MQLONG    LongRemoteUserIdLength;    /* Length of long remote user */
                                     /* identifier */
MQPTR     LongMCAUserIdPtr;          /* Address of long MCA user */
                                     /* identifier */
MQPTR     LongRemoteUserIdPtr;       /* Address of long remote */
                                     /* user identifier */
MQBYTE40  MCASecurityId;             /* MCA security identifier */
MQBYTE40  RemoteSecurityId;          /* Remote security identifier */
/* Ver:6 */
MQCHAR    SSLCipherSpec[32];         /* TLS CipherSpec */
```

```
   MQPTR     SSLPeerNamePtr;           /* Address of TLS peer name */
   MQLONG    SSLPeerNameLength;        /* Length of TLS peer name */
   MQLONG    SSLClientAuth;            /* Whether TLS client */
                                       /* authentication is required */
   MQLONG    KeepAliveInterval;        /* Keepalive interval */
   MQCHAR    LocalAddress[48];         /* Local communications */
                                       /* address */
   MQLONG    BatchHeartbeat;           /* Batch heartbeat interval */
   /* Ver:7 */
   MQLONG    HdrCompList[2];           /* Header data compression */
                                       /* list */
   MQLONG    MsgCompList[16];          /* Message data compression */
                                       /* list */
   MQLONG    CLWLChannelRank;          /* Channel rank */
   MQLONG    CLWLChannelPriority;      /* Channel priority */
   MQLONG    CLWLChannelWeight;        /* Channel weight */
   MQLONG    ChannelMonitoring;        /* Channel monitoring */
   MQLONG    ChannelStatistics;        /* Channel statistics */
   /* Ver:8 */
   MQLONG    SharingConversations;     /* Limit on sharing */
                                       /* conversations */
   MQLONG    PropertyControl;          /* Message property control */
   MQLONG    MaxInstances;             /* Limit on SVRCONN channel */
                                       /* instances */
   MQLONG    MaxInstancesPerClient;    /* Limit on SVRCONN channel */
                                       /* instances per client */
   MQLONG    ClientChannelWeight;      /* Client channel weight */
   MQLONG    ConnectionAffinity;       /* Connection affinity */
   /* Ver:9 */
   MQLONG    BatchDataLimit;           /* Batch data limit */
   MQLONG    UseDLQ;                   /* Use Dead Letter Queue */
   MQLONG    DefReconnect;             /* Default client reconnect */
                                       /* option */
   /* Ver:10 */
   MQCHAR64  CertificateLabel;         /* Certificate label */
 /* Ver:11 */
};
```

COBOL declaration:

This declaration is the COBOL declaration for the MQCD structure.

```
** MQCD structure
       10  MQCD.
      ** Channel definition name
       15  MQCD-CHANNELNAME PIC X(20).
      ** Structure version number
       15  MQCD-VERSION PIC S9(9) BINARY.
      ** Channel type
       15  MQCD-CHANNELTYPE PIC S9(9) BINARY.
      ** Transport type
       15  MQCD-TRANSPORTTYPE PIC S9(9) BINARY.
      ** Channel description
       15  MQCD-DESC PIC X(64).
      ** Queue manager name
       15  MQCD-QMGRNAME PIC X(48).
      ** Transmission queue name
       15  MQCD-XMITQNAME PIC X(48).
      ** First 20 bytes of connection name
       15  MQCD-SHORTCONNECTIONNAME PIC X(20).
      ** Reserved
       15  MQCD-MCANAME PIC X(20).
      ** LU 6.2 Mode name
       15  MQCD-MODENAME PIC X(8).
      ** LU 6.2 transaction program name
       15  MQCD-TPNAME PIC X(64).
      ** Batch size
```

```
 15  MQCD-BATCHSIZE PIC S9(9) BINARY.
** Disconnect interval
 15  MQCD-DISCINTERVAL PIC S9(9) BINARY.
** Short retry count
 15  MQCD-SHORTRETRYCOUNT PIC S9(9) BINARY.
** Short retry wait interval
 15  MQCD-SHORTRETRYINTERVAL PIC S9(9) BINARY.
** Long retry count
 15  MQCD-LONGRETRYCOUNT PIC S9(9) BINARY.
** Long retry wait interval
 15  MQCD-LONGRETRYINTERVAL PIC S9(9) BINARY.
** Channel security exit name
 15  MQCD-SECURITYEXIT PIC X(20).
** Channel message exit name
 15  MQCD-MSGEXIT PIC X(20).
** Channel send exit name
 15  MQCD-SENDEXIT PIC X(20).
** Channel receive exit name
 15  MQCD-RECEIVEEXIT PIC X(20).
** Highest allowable message sequence number
 15  MQCD-SEQNUMBERWRAP PIC S9(9) BINARY.
** Maximum message length
 15  MQCD-MAXMSGLENGTH PIC S9(9) BINARY.
** Put authority
 15  MQCD-PUTAUTHORITY PIC S9(9) BINARY.
** Data conversion
 15  MQCD-DATACONVERSION PIC S9(9) BINARY.
** Channel security exit user data
 15  MQCD-SECURITYUSERDATA PIC X(32).
** Channel message exit user data
 15  MQCD-MSGUSERDATA PIC X(32).
** Channel send exit user data
 15  MQCD-SENDUSERDATA PIC X(32).
** Channel receive exit user data
 15  MQCD-RECEIVEUSERDATA PIC X(32).
** Ver:1 **
** User identifier
 15  MQCD-USERIDENTIFIER PIC X(12).
** Password
 15  MQCD-PASSWORD PIC X(12).
** First 12 bytes of MCA user identifier
 15  MQCD-MCAUSERIDENTIFIER PIC X(12).
** Message channel agent type
 15  MQCD-MCATYPE PIC S9(9) BINARY.
** Connection name
 15  MQCD-CONNECTIONNAME PIC X(264).
** First 12 bytes of user identifier from partner
 15  MQCD-REMOTEUSERIDENTIFIER PIC X(12).
** Password from partner
 15  MQCD-REMOTEPASSWORD PIC X(12).
** Ver:2 **
** Channel message retry exit name
 15  MQCD-MSGRETRYEXIT PIC X(20).
** Channel message retry exit user data
 15  MQCD-MSGRETRYUSERDATA PIC X(32).
** Number of times MCA will try to put the message, after first
** attempt has failed
 15  MQCD-MSGRETRYCOUNT PIC S9(9) BINARY.
** Minimum interval in milliseconds after which the open or put
** operation will be retried
 15  MQCD-MSGRETRYINTERVAL PIC S9(9) BINARY.
** Ver:3 **
** Time in seconds between heartbeat flows
 15  MQCD-HEARTBEATINTERVAL PIC S9(9) BINARY.
** Batch duration
 15  MQCD-BATCHINTERVAL PIC S9(9) BINARY.
** Speed at which nonpersistent messages are sent
```

```
 15  MQCD-NONPERSISTENTMSGSPEED PIC S9(9) BINARY.
** Length of MQCD structure
 15  MQCD-STRUCLENGTH PIC S9(9) BINARY.
** Length of exit name
 15  MQCD-EXITNAMELENGTH PIC S9(9) BINARY.
** Length of exit user data
 15  MQCD-EXITDATALENGTH PIC S9(9) BINARY.
** Number of message exits defined
 15  MQCD-MSGEXITSDEFINED PIC S9(9) BINARY.
** Number of send exits defined
 15  MQCD-SENDEXITSDEFINED PIC S9(9) BINARY.
** Number of receive exits defined
 15  MQCD-RECEIVEEXITSDEFINED PIC S9(9) BINARY.
** Address of first MsgExit field
 15  MQCD-MSGEXITPTR POINTER.
** Address of first MsgUserData field
 15  MQCD-MSGUSERDATAPTR POINTER.
** Address of first SendExit field
 15  MQCD-SENDEXITPTR POINTER.
** Address of first SendUserData field
 15  MQCD-SENDUSERDATAPTR POINTER.
** Address of first ReceiveExit field
 15  MQCD-RECEIVEEXITPTR POINTER.
** Address of first ReceiveUserData field
 15  MQCD-RECEIVEUSERDATAPTR POINTER.
** Ver:4 **
** Address of a list of cluster names
 15  MQCD-CLUSTERPTR POINTER.
** Number of clusters to which the channel belongs
 15  MQCD-CLUSTERSDEFINED PIC S9(9) BINARY.
** Network priority
 15  MQCD-NETWORKPRIORITY PIC S9(9) BINARY.
** Ver:5 **
** Length of long MCA user identifier
 15  MQCD-LONGMCAUSERIDLENGTH PIC S9(9) BINARY.
** Length of long remote user identifier
 15  MQCD-LONGREMOTEUSERIDLENGTH PIC S9(9) BINARY.
** Address of long MCA user identifier
 15  MQCD-LONGMCAUSERIDPTR POINTER.
** Address of long remote user identifier
 15  MQCD-LONGREMOTEUSERIDPTR POINTER.
** MCA security identifier
 15  MQCD-MCASECURITYID PIC X(40).
** Remote security identifier
 15  MQCD-REMOTESECURITYID PIC X(40).
** Ver:6 **
** TLS CipherSpec
 15  MQCD-SSLCIPHERSPEC PIC X(32).
** Address of TLS peer name
 15  MQCD-SSLPEERNAMEPTR POINTER.
** Length of TLS peer name
 15  MQCD-SSLPEERNAMELENGTH PIC S9(9) BINARY.
** Whether TLS client authentication is required
 15  MQCD-SSLCLIENTAUTH PIC S9(9) BINARY.
** Keepalive interval
 15  MQCD-KEEPALIVEINTERVAL PIC S9(9) BINARY.
** Local communications address
 15  MQCD-LOCALADDRESS PIC X(48).
** Batch heartbeat interval
 15  MQCD-BATCHHEARTBEAT PIC S9(9) BINARY.
** Ver:7 **
** Header data compression list
 15  MQCD-HDRCOMPLIST PIC S9(9) BINARY.
** Message data compression list
 15  MQCD-MSGCOMPLIST PIC S9(9) BINARY.
** Channel rank
 15  MQCD-CLWLCHANNELRANK PIC S9(9) BINARY.
```

```
      ** Channel priority
       15  MQCD-CLWLCHANNELPRIORITY PIC S9(9) BINARY.
      ** Channel weight
       15  MQCD-CLWLCHANNELWEIGHT PIC S9(9) BINARY.
      ** Channel monitoring
       15  MQCD-CHANNELMONITORING PIC S9(9) BINARY.
      ** Channel statistics
       15  MQCD-CHANNELSTATISTICS PIC S9(9) BINARY.
      ** Ver:8 **
      ** Limit on sharing conversations
       15  MQCD-SHARINGCONVERSATIONS PIC S9(9) BINARY.
      ** Message property control
       15  MQCD-PROPERTYCONTROL PIC S9(9) BINARY.
      ** Limit on SVRCONN channel instances
       15  MQCD-MAXINSTANCES PIC S9(9) BINARY.
      ** Limit on SVRCONN channel instances per client
       15  MQCD-MAXINSTANCESPERCLIENT PIC S9(9) BINARY.
      ** Client channel weight
       15  MQCD-CLIENTCHANNELWEIGHT PIC S9(9) BINARY.
      ** Connection affinity
       15  MQCD-CONNECTIONAFFINITY PIC S9(9) BINARY.
      ** Ver:9 **
      ** Batch data limit
       15  MQCD-BATCHDATALIMIT PIC S9(9) BINARY.
      ** Use Dead Letter Queue
       15  MQCD-USEDLQ PIC S9(9) BINARY.
      ** Default client reconnect option
       15  MQCD-DEFRECONNECT PIC S9(9) BINARY.
      ** Ver:10 **
```

RPG declaration (ILE):

This declaration is the RPG declaration for the MQCD structure.

```
D* MQCD Structure
     D*
     D* Channel definition name
     D  CDCHN                  1     20
     D* Structure version number
     D  CDVER                 21     24I 0
     D* Channel type
     D  CDCHT                 25     28I 0
     D* Transport type
     D  CDTRT                 29     32I 0
     D* Channel description
     D  CDDES                 33     96
     D* Queue manager name
     D  CDQM                  97    144
     D* Transmission queue name
     D  CDXQ                 145    192
     D* First 20 bytes of connection name
     D  CDSCN                193    212
     D* Reserved
     D  CDMCA                213    232
     D* LU 6.2 Mode name
     D  CDMOD                233    240
     D* LU 6.2 transaction program name
     D  CDTP                 241    304
     D* Batch size
     D  CDBS                 305    308I 0
     D* Disconnect interval
     D  CDDI                 309    312I 0
     D* Short retry count
     D  CDSRC                313    316I 0
     D* Short retry wait interval
     D  CDSRI                317    320I 0
     D* Long retry count
```

```
D  CDLRC                321    324I 0
D* Long retry wait interval
D  CDLRI                325    328I 0
D* Channel security exit name
D  CDSCX                329    348
D* Channel message exit name
D  CDMSX                349    368
D* Channel send exit name
D  CDSNX                369    388
D* Channel receive exit name
D  CDRCX                389    408
D* Highest allowable message sequence number
D  CDSNW                409    412I 0
D* Maximum message length
D  CDMML                413    416I 0
D* Put authority
D  CDPA                 417    420I 0
D* Data conversion
D  CDDC                 421    424I 0
D* Channel security exit user data
D  CDSCD                425    456
D* Channel message exit user data
D  CDMSD                457    488
D* Channel send exit user data
D  CDSND                489    520
D* Channel receive exit user data
D  CDRCD                521    552
D* Ver:1 **
D* User identifier
D  CDUID                553    564
D* Password
D  CDPW                 565    576
D* First 12 bytes of MCA user identifier
D  CDAUI                577    588
D* Message channel agent type
D  CDCAT                589    592I 0
D* Connection name
D  CDCON                593    848
D  CDCN2                849    856
D* First 12 bytes of user identifier from partner
D  CDRUI                857    868
D* Password from partner
D  CDRPW                869    880
D* Ver:2 **
D* Channel message retry exit name
D  CDMRX                881    900
D* Channel message retry exit user data
D  CDMRD                901    932
D* Number of times MCA will try to put the message, after first
D* attempt has failed
D  CDMRC                933    936I 0
D* Minimum interval in milliseconds after which the open or put
D* operation will be retried
D  CDMRI                937    940I 0
D* Ver:3 **
D* Time in seconds between heartbeat flows
D  CDHBI                941    944I 0
D* Batch duration
D  CDBI                 945    948I 0
D* Speed at which nonpersistent messages are sent
D  CDNPM                949    952I 0
D* Length of MQCD structure
D  CDLEN                953    956I 0
D* Length of exit name
D  CDXNL                957    960I 0
D* Length of exit user data
D  CDXDL                961    964I 0
```

```
D* Number of message exits defined
D  CDMXD                965    968I 0
D* Number of send exits defined
D  CDSXD                969    972I 0
D* Number of receive exits defined
D  CDRXD                973    976I 0
D* Address of first MsgExit field
D  CDMXP                977    992*
D* Address of first MsgUserData field
D  CDMUP                993   1008*
D* Address of first SendExit field
D  CDSXP               1009   1024*
D* Address of first SendUserData field
D  CDSUP               1025   1040*
D* Address of first ReceiveExit field
D  CDRXP               1041   1056*
D* Address of first ReceiveUserData field
D  CDRUP               1057   1072*
D* Ver:4 **
D* Address of a list of cluster names
D  CDCLP               1073   1088*
D* Number of clusters to which the channel belongs
D  CDCLD               1089   1092I 0
D* Network priority
D  CDNP                1093   1096I 0
D* Ver:5 **
D* Length of long MCA user identifier
D  CDLML               1097   1100I 0
D* Length of long remote user identifier
D  CDLRL               1101   1104I 0
D* Address of long MCA user identifier
D  CDLMP               1105   1120*
D* Address of long remote user identifier
D  CDLRP               1121   1136*
D* MCA security identifier
D  CDMSI               1137   1176
D* Remote security identifier
D  CDRSI               1177   1216
D* Ver:6 **
D* TLS CipherSpec
D  CDSCS               1217   1248
D* Address of TLS peer name
D  CDSPN               1249   1264*
D* Length of TLS peer name
D  CDSPL               1265   1268I 0
D* Whether TLS client authentication is required
D  CDSCA               1269   1272I 0
D* Keepalive interval
D  CDKAI               1273   1276I 0
D* Local communications address
D  CDLOA               1277   1324
D* Batch heartbeat interval
D  CDBHB               1325   1328I 0
D* Ver:7 **
D* Header data compression list
D  CDHCL0
D  CDHCL1              1329   1332I 0
D  CDHCL2              1333   1336I 0
D  CDHCL                        10I 0 DIM(2) OVERLAY(CDHCL0)
D* Message data compression list
D  CDMCL0
D  CDMCL1              1337   1340I 0
D  CDMCL2              1341   1344I 0
D  CDMCL3              1345   1348I 0
D  CDMCL4              1349   1352I 0
D  CDMCL5              1353   1356I 0
D  CDMCL6              1357   1360I 0
```

```
D  CDMCL7              1361   1364I 0
D  CDMCL8              1365   1368I 0
D  CDMCL9              1369   1372I 0
D  CDMCL10             1373   1376I 0
D  CDMCL11             1377   1380I 0
D  CDMCL12             1381   1384I 0
D  CDMCL13             1385   1388I 0
D  CDMCL14             1389   1392I 0
D  CDMCL15             1393   1396I 0
D  CDMCL16             1397   1400I 0
D  CDMCL                        10I 0 DIM(16) OVERLAY(CDMCL0)
D* Channel rank
D  CDCWCR              1401   1404I 0
D* Channel priority
D  CDCWCP              1405   1408I 0
D* Channel weight
D  CDCWCW              1409   1412I 0
D* Channel monitoring
D  CDCHLMON            1413   1416I 0
D* Channel statistics
D  CDCHLST             1417   1420I 0
D* Ver:8 **
D* Limit on sharing conversations
D  CDSHC               1421   1424I 0
D* Message property control
D  CDPRC               1425   1428I 0
D* Limit on SVRCONN channel instances
D  CDMXIN              1429   1432I 0
D* Limit on SVRCONN channel instances per client
D  CDMXIC              1433   1436I 0
D* Client channel weight
D  CDCLNCHLW           1437   1440I 0
D* Connection affinity
D  CDCONNAFF           1441   1444I 0
D* Ver:9 **
D* Batch data limit
D  CDBDL               1445   1448I 0
D* Use Dead Letter Queue
D  CDUDLQ              1449   1452I 0
D* Default client reconnect option
D  CDDRCN              1453   1456I 0
D* Ver:10 **
```

System/390 assembler declaration:

This declaration is the System/390 assembler declaration for the MQCD structure.

```
MQCD                     DSECT
MQCD_CHANNELNAME         DS   CL20   Channel definition name
MQCD_VERSION             DS   F      Structure version number
MQCD_CHANNELTYPE         DS   F      Channel type
MQCD_TRANSPORTTYPE       DS   F      Transport type
MQCD_DESC                DS   CL64   Channel description
MQCD_QMGRNAME            DS   CL48   Queue manager name
MQCD_XMITQNAME           DS   CL48   Transmission queue name
MQCD_SHORTCONNECTIONNAME DS   CL20   First 20 bytes of connection
*                                    name
MQCD_MCANAME             DS   CL20   Reserved
MQCD_MODENAME            DS   CL8    LU 6.2 Mode name
MQCD_TPNAME              DS   CL64   LU 6.2 transaction program name
MQCD_BATCHSIZE           DS   F      Batch size
MQCD_DISCINTERVAL        DS   F      Disconnect interval
MQCD_SHORTRETRYCOUNT     DS   F      Short retry count
MQCD_SHORTRETRYINTERVAL  DS   F      Short retry wait interval
MQCD_LONGRETRYCOUNT      DS   F      Long retry count
MQCD_LONGRETRYINTERVAL   DS   F      Long retry wait interval
MQCD_SECURITYEXIT        DS   CLn    Channel security exit name
```

```
MQCD_MSGEXIT                 DS   CLn    Channel message exit name
MQCD_SENDEXIT                DS   CLn    Channel send exit name
MQCD_RECEIVEEXIT             DS   CLn    Channel receive exit name
MQCD_SEQNUMBERWRAP           DS   F      Highest allowable message
*                                        sequence number
MQCD_MAXMSGLENGTH            DS   F      Maximum message length
MQCD_PUTAUTHORITY            DS   F      Put authority
MQCD_DATACONVERSION          DS   F      Data conversion
MQCD_SECURITYUSERDATA        DS   CL32   Channel security exit user data
MQCD_MSGUSERDATA             DS   CL32   Channel message exit user data
MQCD_SENDUSERDATA            DS   CL32   Channel send exit user data
MQCD_RECEIVEUSERDATA         DS   CL32   Channel receive exit user data
MQCD_USERIDENTIFIER          DS   CL12   User identifier
MQCD_PASSWORD                DS   CL12   Password
MQCD_MCAUSERIDENTIFIER       DS   CL12   First 12 bytes of MCA user
*                                        identifier
MQCD_MCATYPE                 DS   F      Message channel agent type
MQCD_CONNECTIONNAME          DS   CL264  Connection name
MQCD_REMOTEUSERIDENTIFIER    DS   CL12   First 12 bytes of user
*                                        identifier from partner
MQCD_REMOTEPASSWORD          DS   CL12   Password from partner
MQCD_MSGRETRYEXIT            DS   CLn    Channel message retry exit name
MQCD_MSGRETRYUSERDATA        DS   CL32   Channel message retry exit user
*                                        data
MQCD_MSGRETRYCOUNT           DS   F      Number of times MCA will try to
*                                        put the message, after the
*                                        first attempt has failed
MQCD_MSGRETRYINTERVAL        DS   F      Minimum interval in
*                                        milliseconds after which the
*                                        open or put operation will be
*                                        retried
MQCD_HEARTBEATINTERVAL       DS   F      Time in seconds between
*                                        heartbeat flows
MQCD_BATCHINTERVAL           DS   F      Batch duration
MQCD_NONPERSISTENTMSGSPEED   DS   F      Speed at which nonpersistent
*                                        messages are sent
MQCD_STRUCLENGTH             DS   F      Length of MQCD structure
MQCD_EXITNAMELENGTH          DS   F      Length of exit name
MQCD_EXITDATALENGTH          DS   F      Length of exit user data
MQCD_MSGEXITSDEFINED         DS   F      Number of message exits defined
MQCD_SENDEXITSDEFINED        DS   F      Number of send exits defined
MQCD_RECEIVEEXITSDEFINED     DS   F      Number of receive exits defined
MQCD_MSGEXITPTR              DS   F      Address of first MSGEXIT field
MQCD_MSGUSERDATAPTR          DS   F      Address of first MSGUSERDATA
*                                        field
MQCD_SENDEXITPTR             DS   F      Address of first SENDEXIT field
MQCD_SENDUSERDATAPTR         DS   F      Address of first SENDUSERDATA
*                                        field
MQCD_RECEIVEEXITPTR          DS   F      Address of first RECEIVEEXIT
*                                        field
MQCD_RECEIVEUSERDATAPTR      DS   F      Address of first
*                                        RECEIVEUSERDATA field
MQCD_CLUSTERPTR              DS   F      Address of a list of cluster
*                                        names
MQCD_CLUSTERSDEFINED         DS   F      Number of clusters to which the
*                                        channel belongs
MQCD_NETWORKPRIORITY         DS   F      Network priority
MQCD_LONGMCAUSERIDLENGTH     DS   F      Length of long MCA user
*                                        identifier
MQCD_LONGREMOTEUSERIDLENGTH  DS   F      Length of long remote user
*                                        identifier
MQCD_LONGMCAUSERIDPTR        DS   F      Address of long MCA user
*                                        identifier
MQCD_LONGREMOTEUSERIDPTR     DS   F      Address of long remote user
*                                        identifier
MQCD_MCASECURITYID           DS   XL40   MCA security identifier
MQCD_REMOTESECURITYID        DS   XL40   Remote security identifier
```

```
MQCD_SSLCIPHERSPEC           DS   CL32   TLS CipherSpec
MQCD_SSLPEERNAMEPTR          DS   F      Address of TLS peer name
MQCD_SSLPEERNAMELENGTH       DS   F      Length of TLS peer name
MQCD_SSLCLIENTAUTH           DS   F      Whether TLS client
*                                        authentication is required
MQCD_KEEPALIVEINTERVAL       DS   F      Keepalive interval
MQCD_LOCALADDRESS            DS   CL48   Local communications address
MQCD_BATCHHEARTBEAT          DS   F      Batch heartbeat interval
MQCD_HDRCOMPLIST             DS   CL2    Header data compression list
MQCD_MSGCOMPLIST             DS   CL16   Message data compression list
MQCD_CLWLCHANNELRANK         DS   F      Channel rank
MQCD_CLWLCHANNELPRIORITY     DS   F      Channel priority
MQCD_CLWLCHANNELWEIGHT       DS   F      Channel weight
MQCD_CHANNELMONITORING       DS   F      Channel monitoring
MQCD_CHANNELSTATISTICS       DS   F      Channel statistics
MQCD_SHARINGCONVERSATIONS    DS   F      Limit on sharing
*                                        conversations
MQCD_PROPERTYCONTROL         DS   F      Message property
*                                        control
MQCD_SHARINGCONVERSATIONS    DS   F      Limit on sharing conversations
MQCD_PROPERTYCONTROL         DS   F      Message property control
MQCD_MAXINSTANCES            DS   F      Limit on SVRCONN chl instances
MQCD_MAXINSTANCESPERCLIENT   DS   F      Limit on SVRCONN chl instances
                                         per client
MQCD_CLIENTCHANNELWEIGHT     DS   F      Channel weight
MQCD_CONNECTIONAFFINITY      DS   F      Connection Affinty
MQCD_BATCHDATALIMIT          DS   F      Batch data limit
MQCD_USEDLQ                  DS   F      Use dead-letter queue
MQCD_DEFRECONNECT            DS   F      Default client reconnect option
MQCD_LENGTH                  EQU  *-MQCD
                             ORG  MQCD
MQCD_AREA                    DS   CL(MQCD_LENGTH)
```

Visual Basic declaration:

This declaration is the Visual Basic declaration of the MQCD structure.

In Visual Basic, the MQCD structure can be used with the MQCNO structure on the MQCONNX call.

```
Type MQCD
  ChannelName            As String*20  'Channel definition name'
  Version                As Long       'Structure version number'
  ChannelType            As Long       'Channel type'
  TransportType          As Long       'Transport type'
  Desc                   As String*64  'Channel description'
  QMgrName               As String*48  'Queue manager name'
  XmitQName              As String*48  'Transmission queue name'
  ShortConnectionName    As String*20  'First 20 bytes of connection'
                                       'name'
  MCAName                As String*20  'Reserved'
  ModeName               As String*8   'LU 6.2 Mode name'
  TpName                 As String*64  'LU 6.2 transaction program name'
  BatchSize              As Long       'Batch size'
  DiscInterval           As Long       'Disconnect interval'
  ShortRetryCount        As Long       'Short retry count'
  ShortRetryInterval     As Long       'Short retry wait interval'
  LongRetryCount         As Long       'Long retry count'
  LongRetryInterval      As Long       'Long retry wait interval'
  SecurityExit           As String*128 'Channel security exit name'
  MsgExit                As String*128 'Channel message exit name'
  SendExit               As String*128 'Channel send exit name'
  ReceiveExit            As String*128 'Channel receive exit name'
  SeqNumberWrap          As Long       'Highest allowable message'
                                       'sequence number'
  MaxMsgLength           As Long       'Maximum message length'
  PutAuthority           As Long       'Put authority'
  DataConversion         As Long       'Data conversion'
```

```
SecurityUserData       As String*32  'Channel security exit user data'
MsgUserData            As String*32  'Channel message exit user data'
SendUserData           As String*32  'Channel send exit user data'
ReceiveUserData        As String*32  'Channel receive exit user data'
UserIdentifier         As String*12  'User identifier'
Password               As String*12  'Password'
MCAUserIdentifier      As String*12  'First 12 bytes of MCA user'
                                     'identifier'
MCAType                As Long       'Message channel agent type'
ConnectionName         As String*264 'Connection name'
RemoteUserIdentifier   As String*12  'First 12 bytes of user'
                                     'identifier from partner'
RemotePassword         As String*12  'Password from partner'
MsgRetryExit           As String*128 'Channel message retry exit name'
MsgRetryUserData       As String*32  'Channel message retry exit user'
                                     'data'
MsgRetryCount          As Long       'Number of times MCA will try to'
                                     'put the message, after the'
                                     'first attempt has failed'
MsgRetryInterval       As Long       'Minimum interval in'
                                     'milliseconds after which the'
                                     'open or put operation will be'
                                     'retried'
HeartbeatInterval      As Long       'Time in seconds between'
                                     'heartbeat flows'
BatchInterval          As Long       'Batch duration'
NonPersistentMsgSpeed  As Long       'Speed at which nonpersistent'
                                     'messages are sent'
StrucLength            As Long       'Length of MQCD structure'
ExitNameLength         As Long       'Length of exit name'
ExitDataLength         As Long       'Length of exit user data'
MsgExitsDefined        As Long       'Number of message exits defined'
SendExitsDefined       As Long       'Number of send exits defined'
ReceiveExitsDefined    As Long       'Number of receive exits defined'
MsgExitPtr             As MQPTR      'Address of first MsgExit field'
MsgUserDataPtr         As MQPTR      'Address of first MsgUserData'
                                     'field'
SendExitPtr            As MQPTR      'Address of first SendExit field'
SendUserDataPtr        As MQPTR      'Address of first SendUserData'
                                     'field'
ReceiveExitPtr         As MQPTR      'Address of first ReceiveExit'
                                     'field'
ReceiveUserDataPtr     As MQPTR      'Address of first'
                                     'ReceiveUserData field'
ClusterPtr             As MQPTR      'Address of a list of cluster'
                                     'names'
ClustersDefined        As Long       'Number of clusters to which the'
                                     'channel belongs'
NetworkPriority        As Long       'Network priority'
LongMCAUserIdLength    As Long       'Length of long MCA user'
                                     'identifier'
LongRemoteUserIdLength As Long       'Length of long remote user'
                                     'identifier'
LongMCAUserIdPtr       As MQPTR      'Address of long MCA user'
                                     'identifier'
LongRemoteUserIdPtr    As MQPTR      'Address of long remote user'
                                     'identifier'
MCASecurityId          As MQBYTE40   'MCA security identifier'
RemoteSecurityId       As MQBYTE40   'Remote security identifier'
SSLCipherSpec          As String*32  'TLS CipherSpec'
SSLPeerNamePtr         As MQPTR      'Address of TLS peer name'
SSLPeerNameLength      As Long       'Length of TLS peer name'
SSLClientAuth          As Long       'Whether TLS client'
                                     'authentication is required'
KeepAliveInterval      As Long       'Keepalive interval'
LocalAddress           As String*48  'Local communications address'
BatchHeartbeat         As Long       'Batch heartbeat interval'
```

```
  HdrCompList(0 to 1)    As Long2      'Header data compression list'
  MsgCompList(0 To 15)   As Long16     'Message data compression list'
  CLWLChannelRank        As Long       'Channel Rank'
  CLWLChannelPriority    As Long       'Channel priority'
  CLWLChannelWeight      As Long       'Channel Weight'
  ChannelMonitoring      As Long       'Channel Monitoring control'
  ChannelStatistics      As Long       'Channel Statistics'
 End Type
```

Changing MQCD fields in a channel exit:

A channel exit can change fields in the MQCD. However, these changes are not typically acted on, except in the circumstances listed.

If a channel exit program changes a field in the MQCD data structure, the new value is typically ignored by the IBM MQ channel process. However, the new value remains in the MQCD and is passed to any remaining exits in an exit chain and to any conversation sharing the channel instance.

If SharingConversations is set to FALSE in the MQCXP structure, changes to certain fields can be acted on, depending on the type of exit program, the type of channel, and the exit reason code. The following table shows the fields that can be changed and affect the behavior of the channel, and in what circumstances. If an exit program changes one of these fields in any other circumstances, or any field not listed, the new value is ignored by the channel process. The new value remains in the MQCD and is passed to any remaining exits in an exit chain and to any conversation sharing the channel instance.

Any type of exit program when called for initialization (MQXR_INIT) can change the ChannelName field of any type of channel, as long as MQCXP SharingConverstions is set to FALSE. Only a security exit can change the MCAUserIdentifier field, regardless of the value of MQCXP SharingConverstions.

Field	Exit reason code	Exit type	Channel type
ChannelName	MQXR_INIT	All	All
TransportType	MQXR_INIT	All	All
XmitQName	MQXR_INIT	All	SDR, RCVR
ModeName	MQXR_INIT	All	All
TpName	MQXR_INIT	All	All
BatchSize	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
DiscInterval	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
ShortRetryCount	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
ShortRetryInterval	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR

Field	Exit reason code	Exit type	Channel type
LongRetryCount	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
LongRetryInterval	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
SeqNumberWrap	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
MaxMsgLength	MQXR_INIT	All	All
PutAuthority	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
DataConversion	MQXR_INIT	All	All
MCAUserIdentifier	MQXR_INIT, MQXR_INIT_SEC, MQXR_SEC_MSG, MQXR_SEC_PARMS	Security	RCVR, RQSTR, SVRCONN, CLUSRCVR
ConnectionName	MQXR_INIT	All	SDR, SVR, RQSTR, CLNTCONN, CLUSSDR, CLUSRCVR
MsgRetryUserData	MQXR_INIT	All	RCVR, RQSTR, CLUSRCVR
MsgRetryCount	MQXR_INIT	All	RCVR, RQSTR, CLUSRCVR
MsgRetryInterval	MQXR_INIT	All	RCVR, RQSTR, CLUSRCVR
HeartbeatInterval	MQXR_INIT	All	All
BatchInterval	MQXR_INIT	All	SDR, SVR, CLUSSDR, CLUSRCVR
NonPersistentMsgSpeed	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR

Field	Exit reason code	Exit type	Channel type
MCASecurityId	MQXR_INIT, MQXR_INIT_SEC, MQXR_SEC_MSG, MQXR_SEC_PARMS	Security	SDR, SVR, RCVR, RQSTR, SVRCONN, CLUSSDR, CLUSRCVR
SSLCipherSpec	MQXR_INIT	All	All
SSLPeerNamePtr	MQXR_INIT	All	All
SSLPeerNameLength	MQXR_INIT	All	All
SSLClientAuth	MQXR_INIT	All	SVR, RCVR, RQSTR, SVRCONN, CLUSRCVR
KeepAliveInterval	MQXR_INIT	All	All
LocalAddress	MQXR_INIT	All	SDR, SVR, RQSTR, CLNTCONN, CLUSSDR, CLUSRCVR
BatchHeartbeat	MQXR_INIT	All	SDR, SVR, CLUSSDR, CLUSRCVR
HdrCompList	MQXR_INIT	All	All
MsgCompList	MQXR_INIT	All	All
ChannelMonitoring	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, SVRCONN, CLUSSDR, CLUSRCVR
ChannelStatistics	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
SharingConversations	MQXR_INIT	All	SVRCONN, CLNTCONN
PropertyControl	MQXR_INIT	All	SDR, SVR, CLUSSDR, CLUSRCVR

MQCXP - Channel exit parameter:

The MQCXP structure is passed to each type of exit called by a Message Channel Agent (MCA), client-connection channel, or server-connection channel.

See MQ_CHANNEL_EXIT.

The fields described as "input to the exit" in the descriptions that follow are ignored by the channel when the exit returns control to the channel. Any input fields that the exit changes in the channel exit parameter block will not be preserved for its next invocation. Changes made to input/output fields (for example, the *ExitUserArea* field), are preserved for invocations of that instance of the exit only. Such changes cannot be used to pass data between different exits defined on the same channel, or between the same exit defined on different channels.

Related reference:

"Fields"
This topic lists all the fields in the MQCXP structure and describes each field.

"C declaration" on page 3622
This declaration is the C declaration for the MQCXP structure.

"COBOL declaration" on page 3623
This declaration is the COBOL declaration for the MQCXP structure.

"RPG declaration (ILE)" on page 3624
This declaration is the RPG declaration for the MQCXP structure.

"System/390 assembler declaration" on page 3625
This declaration is the System/390 assembler declaration for the MQCXP structure.

Fields:

This topic lists all the fields in the MQCXP structure and describes each field.

StrucId (MQCHAR4):

This field specifies the structure identifier.

The value must be:

MQCXP_STRUC_ID
Identifier for channel exit parameter structure.

For the C programming language, the constant MQCXP_STRUC_ID_ARRAY is also defined; this constant has the same value as MQCXP_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

Version (MQLONG):

This field specifies the structure version number.

The value depends on the environment:

MQCXP_VERSION_1
Version-1 channel exit parameter structure.

MQCXP_VERSION_2
Version-2 channel exit parameter structure.

The field has this value in the HP Integrity NonStop Server environment.

MQCXP_VERSION_3
Version-3 channel exit parameter structure.

The field has this value in the following environments: UNIX systems not listed elsewhere.

MQCXP_VERSION_4
Version-4 channel exit parameter structure.

MQCXP_VERSION_5
Version-5 channel exit parameter structure.

MQCXP_VERSION_6
Version-6 channel exit parameter structure.

MQCXP_VERSION_8
Version-8 channel exit parameter structure.

The field has this value in the following environments: z/OS.

MQCXP_VERSION_9
Version-9 channel exit parameter structure.

The field has this value in the following environments: z/OS, AIX, HP-UX, Linux, IBM i, Solaris, Windows.

Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQCXP_CURRENT_VERSION
Current version of channel exit parameter structure.

The value depends on the environment.

Note: When a new version of the MQCXP structure is introduced, the layout of the existing part is not changed. The exit must therefore check that the version number is equal to or greater than the lowest version which contains the fields that the exit needs to use.

This is an input field to the exit.

ExitId (MQLONG):

This field specifies the type of exit being called and is set on entry to the exit routine.

The following values are possible:

MQXT_CHANNEL_SEC_EXIT
Channel security exit.

MQXT_CHANNEL_MSG_EXIT
Channel message exit.

MQXT_CHANNEL_SEND_EXIT
Channel send exit.

MQXT_CHANNEL_RCV_EXIT
Channel receive exit.

MQXT_CHANNEL_MSG_RETRY_EXIT
Channel message-retry exit.

MQXT_CHANNEL_AUTO_DEF_EXIT
Channel auto-definition exit.

On z/OS, this type of exit is supported only for channels of type MQCHT_CLUSSDR and MQCHT_CLUSRCVR.

This is an input field to the exit.

ExitReason (MQLONG):

This field specifies the reason why the exit is being called and is set on entry to the exit routine.

It is not used by the auto-definition exit. The following values are possible:

MQXR_INIT
Exit initialization.

This value indicates that the exit is being invoked for the first time. It allows the exit to acquire and initialize any resources that it needs (for example: memory).

MQXR_TERM
Exit termination.

This value indicates that the exit is about to be terminated. The exit should free any resources that it has acquired since it was initialized (for example: memory).

MQXR_MSG
Process a message.

This value indicates that the exit is being invoked to process a message. This value occurs for channel message exits only.

MQXR_XMIT
Process a transmission.

This value occurs for channel send and receive exits only.

MQXR_SEC_MSG
Security message received.

This value occurs for channel security exits only.

MQXR_INIT_SEC
Initiate security exchange.

This value occurs for channel security exits only.

The security exit of the receiver is always invoked with this reason immediately after being invoked with MQXR_INIT, to give it the opportunity to initiate a security exchange. If it declines the opportunity (by returning MQXCC_OK instead of MQXCC_SEND_SEC_MSG or MQXCC_SEND_AND_REQUEST_SEC_MSG), the security exit of the sender is invoked with MQXR_INIT_SEC.

If the security exit of the receiver does initiate a security exchange (by returning MQXCC_SEND_SEC_MSG or MQXCC_SEND_AND_REQUEST_SEC_MSG), the security exit of the sender is never invoked with MQXR_INIT_SEC; instead it is invoked with MQXR_SEC_MSG to process the message of the receiver. (In either case it is first invoked with MQXR_INIT.)

Unless one of the security exits requests termination of the channel (by setting *ExitResponse* to MQXCC_SUPPRESS_FUNCTION or MQXCC_CLOSE_CHANNEL), the security exchange must complete at the side that initiated the exchange. Therefore, if a security exit is invoked with MQXR_INIT_SEC and it does initiate an exchange, the next time the exit is invoked it will be with MQXR_SEC_MSG. This happens whether there is a security message for the exit to process or not. There is a security message if the partner returns MQXCC_SEND_SEC_MSG or MQXCC_SEND_AND_REQUEST_SEC_MSG, but not if the partner returns MQXCC_OK or there is no security exit at the partner. If there is no security message to process, the security exit at the initiating end is re-invoked with a *DataLength* of zero.

MQXR_RETRY
Retry a message.

This value occurs for message-retry exits only.

MQXR_AUTO_CLUSSDR
Automatic definition of a cluster-sender channel.

This value occurs for channel auto-definition exits only.

MQXR_AUTO_RECEIVER
Automatic definition of a receiver channel.

This value occurs for channel auto-definition exits only.

MQXR_AUTO_SVRCONN
Automatic definition of a server-connection channel.

This value occurs for channel auto-definition exits only.

MQXR_AUTO_CLUSRCVR
Automatic definition of a cluster-receiver channel.

This value occurs for channel auto-definition exits only.

MQXR_SEC_PARMS
Security parameters

This value applies to security exits only and indicates that an MQCSP structure is being passed to the exit. For more information, see "MQCSP - Security parameters" on page 2292

Note:

1. If you have more than one exit defined for a channel, they are each invoked with MQXR_INIT when the MCA is initialized. Also, they are each invoked with MQXR_TERM when the MCA is terminated.
2. For the channel auto-definition exit, *ExitReason* is not set if *Version* is less than MQCXP_VERSION_4. The value MQXR_AUTO_SVRCONN is implied in this case.

This is an input field to the exit.

ExitResponse (MQLONG):

This field specifies the response from the exit.

This field is set by the exit to communicate with the MCA. It must be one of the following values:

MQXCC_OK
Exit completed successfully.

- For the channel security exit, this value indicates that message transfer can now proceed normally.
- For the channel message retry exit, this value indicates that the MCA must wait for the time interval returned by the exit in the *MsgRetryInterval* field in MQCXP, and then try the message again.

The *ExitResponse2* field might contain additional information.

MQXCC_SUPPRESS_FUNCTION
Suppress function.

- For the channel security exit, this value indicates that the channel must be terminated.
- For the channel message exit, this value indicates that the message is not to proceed any further towards its destination. Instead the MCA generates an exception report message (if one was requested by the sender of the original message), and places the message contained in the original buffer on the dead-letter queue (if the sender specified MQRO_DEAD_LETTER_Q), or discards it (if the sender specified MQRO_DISCARD_MSG).

  For persistent messages, if the sender specified MQRO_DEAD_LETTER_Q, but the put to the dead-letter queue fails, or there is no dead-letter queue, the original message is left on the transmission queue and the report message is not generated. The original message is also left on the transmission queue if the report message cannot be generated successfully.

  The *Feedback* field in the MQDLH structure at the start of the message on the dead-letter queue indicates why the message was put on the dead-letter queue; this feedback code is also used in the message descriptor of the exception report message (if one was requested by the sender).
- For the channel message retry exit, this value indicates that the MCA does not wait and try the message again; instead, the MCA continues immediately with its normal failure processing (the message is placed on the dead-letter queue or discarded, as specified by the sender of the message).
- For the channel auto-definition exit, either MQXCC_OK or MQXCC_SUPPRESS_FUNCTION must be specified. If neither of these values is specified, MQXCC_SUPPRESS_FUNCTION is assumed by default and the auto-definition is abandoned.

This response is not supported for the channel send and receive exits.

MQXCC_SEND_SEC_MSG
Send security message.

This value can be set only by a channel security exit. It indicates that the exit has provided a security message which must be transmitted to the partner.

MQXCC_SEND_AND_REQUEST_SEC_MSG
Send security message that requires a reply.

This value can be set only by a channel security exit. It indicates

- that the exit has provided a security message which can be transmitted to the partner, and
- that the exit requires a response from the partner. If no response is received, the channel must be terminated, because the exit has not yet decided whether communications can proceed.

MQXCC_SUPPRESS_EXIT
Suppress exit.

- This value can be set by all types of channel exit other than a security exit or an auto-definition exit. It suppresses any further invocation of that exit (as if its name had been blank in the channel definition), until termination of the channel, when the exit is again invoked with an *ExitReason* of MQXR_TERM.
- If a message retry exit returns this value, message retries for subsequent messages are controlled by the *MsgRetryCount* and *MsgRetryInterval* channel attributes as normal. For the current message, the MCA performs the number of outstanding retries, at intervals given by the *MsgRetryInterval* channel attribute, but only if the reason code is one that the MCA would normally retry (see the *MsgRetryCount* field described in "MQCD - Channel definition" on page 3559). The number of outstanding retries is the value of the **MsgRetryCount** attribute, less the number of times the exit returned MQXCC_OK for the current message; if this number is negative, no further retries are performed by the MCA for the current message.

MQXCC_CLOSE_CHANNEL
Close channel.

This value can be set by any type of channel exit except an auto-definition exit.

If sharing conversations is not enabled, this value closes the channel.

If sharing conversations is enabled, this value ends the conversation. If this conversation is the only conversation on the channel, the channel also closes.

This field is an input/output field from the exit.

ExitResponse2 (MQLONG):

This field specifies the secondary response from the exit.

This field is set to zero on entry to the exit routine. It can be set by the exit to provide further information to the IBM MQ channel functions. It is not used by the auto-definition exit.

The exit can set one or more of the following values. If more than one is required, the values are added. Combinations that are not valid are noted; other combinations are allowed.

MQXR2_PUT_WITH_DEF_ACTION
Put with default action.

This value is set by the channel message exit of the receiver. It indicates that the message is to be put with the default action of the MCA, that is either the default user ID of the MCA, or the context *UserIdentifier* in the MQMD (message descriptor) of the message.

The value is zero, which corresponds to the initial value set when the exit is invoked. The constant is provided for documentation purposes.

MQXR2_PUT_WITH_DEF_USERID
Put with default user identifier.

This value can only be set by the channel message exit of the receiver. It indicates that the message is to be put with the default user identifier of the MCA.

MQXR2_PUT_WITH_MSG_USERID
Put with user identifier of the message.

This value can only be set by the channel message exit of the receiver. It indicates that the message is to be put with the context *UserIdentifier* in the MQMD (message descriptor) of the message (this might have been modified by the exit).

Only one of MQXR2_PUT_WITH_DEF_ACTION, MQXR2_PUT_WITH_DEF_USERID, and MQXR2_PUT_WITH_MSG_USERID should be set.

MQXR2_USE_AGENT_BUFFER
Use agent buffer.

This value indicates that any data to be passed on is in *AgentBuffer*, not *ExitBufferAddr*.

The value is zero, which corresponds to the initial value set when the exit is invoked. The constant is provided for documentation purposes.

MQXR2_USE_EXIT_BUFFER
Use exit buffer.

This value indicates that any data to be passed on is in *ExitBufferAddr*, not *AgentBuffer*.

Only one of MQXR2_USE_AGENT_BUFFER and MQXR2_USE_EXIT_BUFFER should be set.

MQXR2_DEFAULT_CONTINUATION
Default continuation.

Continuation with the next exit in the chain depends on the response from the last exit invoked:

- If MQXCC_SUPPRESS_FUNCTION or MQXCC_CLOSE_CHANNEL are returned, no further exits in the chain are called.
- Otherwise, the next exit in the chain is invoked.

MQXR2_CONTINUE_CHAIN
Continue with the next exit.

MQXR2_SUPPRESS_CHAIN
Skip remaining exits in chain.

This is an input/output field to the exit.

Feedback (MQLONG):

This field specifies the feedback code.

This field is set to MQFB_NONE on entry to the exit routine.

If a channel message exit sets the *ExitResponse* field to MQXCC_SUPPRESS_FUNCTION, the *Feedback* field specifies the feedback code that identifies why the message was put on the dead-letter (undelivered-message) queue, and is also used to send an exception report if one has been requested. In this case, if the *Feedback* field is MQFB_NONE, the following feedback code is used:

MQFB_STOPPED_BY_MSG_EXIT
Message stopped by channel message exit.

The value returned in this field by channel security, send, receive, and message-retry exits is not used by the MCA.

The value returned in this field by auto-definition exits is not used if *ExitResponse* is MQXCC_OK, but otherwise is used for the *AuxErrorDataInt1* parameter in the event message.

This is an input/output field from the exit.

MaxSegmentLength (MQLONG):

This field specifies the maximum length in bytes that can be sent in a single transmission.

It is not used by the auto-definition exit. It is of interest to a channel send exit, because this exit must ensure that it does not increase the size of a transmission segment to a value greater than *MaxSegmentLength*. The length includes the initial 8 bytes that the exit must not change. The value is negotiated between the IBM MQ channel functions when the channel is initiated. See Writing channel-exit programs for more information about segment lengths.

The value in this field is not meaningful if *ExitReason* is MQXR_INIT.

This is an input field to the exit.

ExitUserArea (MQBYTE16):

This field specifies the exit user area - a field available for the exit to use.

It is initialized to binary zero before the first invocation of the exit (which has an *ExitReason* set to MQXR_INIT), and thereafter any changes made to this field by the exit are preserved across invocations of the exit.

The following value is defined:

MQXUA_NONE
No user information.

The value is binary zero for the length of the field.

For the C programming language, the constant MQXUA_NONE_ARRAY is also defined; this constant has the same value as MQXUA_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_EXIT_USER_AREA_LENGTH. This is an input/output field to the exit.

ExitData (MQCHAR32):

This field specifies the exit data.

This field is set on entry to the exit routine to information that IBM MQ channel functions took from the channel definition. If no such information is available, this field is all blanks.

The length of this field is given by MQ_EXIT_DATA_LENGTH.

This is an input field to the exit.

The following fields in this structure are not present if *Version* is less than MQCXP_VERSION_2.

MsgRetryCount (MQLONG):

This field specifies the number of times the message has been retried.

The first time the exit is invoked for a particular message, this field has the value zero (no retries yet attempted). On each subsequent invocation of the exit for that message, the value is incremented by one by the MCA.

This is an input field to the exit. The value in this field is not meaningful if *ExitReason* is MQXR_INIT. The field is not present if *Version* is less than MQCXP_VERSION_2.

MsgRetryInterval (MQLONG):

This field specifies the minimum interval in milliseconds after which the put operation is retried.

The first time the exit is invoked for a particular message, this field contains the value of the *MsgRetryInterval* channel attribute. The exit can leave the value unchanged, or modify it to specify a different time interval in milliseconds. If the exit returns MQXCC_OK in *ExitResponse*, the MCA waits for at least this time interval before retrying the MQOPEN or MQPUT operation. The time interval specified must be zero or greater.

The second and subsequent times the exit is invoked for that message, this field contains the value returned by the previous invocation of the exit.

If the value returned in the *MsgRetryInterval* field is less than zero or greater than 999 999 999, and *ExitResponse* is MQXCC_OK, the MCA ignores the *MsgRetryInterval* field in MQCXP and waits instead for the interval specified by the *MsgRetryInterval* channel attribute.

This is an input/output field to the exit. The value in this field is not meaningful if *ExitReason* is MQXR_INIT. The field is not present if *Version* is less than MQCXP_VERSION_2.

MsgRetryReason (MQLONG):

This field specifies the reason code from the previous attempt to put the message.

This field is the reason code from the previous attempt to put the message; it is one of the MQRC_* values.

This is an input field to the exit. The value in this field is not meaningful if *ExitReason* is MQXR_INIT. The field is not present if *Version* is less than MQCXP_VERSION_2.

The following fields in this structure are not present if *Version* is less than MQCXP_VERSION_3.

HeaderLength (MQLONG):

This field specifies the length of header information.

This field is relevant only for a message exit and a message-retry exit. The value is the length of the routing header structures at the start of the message data; these are the MQXQH structure, the MQMDE (message description extension header), and (for a distribution-list message) the MQDH structure and arrays of MQOR and MQPMR records that follow the MQXQH structure.

The message exit can examine this header information, and modify it if necessary, but the data that the exit returns must still be in the correct format. The exit must not, for example, encrypt or compress the header data at the sending end, even if the message exit at the receiving end makes compensating changes.

If the message exit modifies the header information in such a way as to change its length (for example, by adding another destination to a distribution-list message), it must change the value of *HeaderLength* correspondingly before returning.

This is an input/output field to the exit. The value in this field is not meaningful if *ExitReason* is MQXR_INIT. The field is not present if *Version* is less than MQCXP_VERSION_3.

PartnerName (MQCHAR48):

This field specifies the name of the partner.

The name of the partner, as follows:

- For SVRCONN channels, it is the logged-on user ID at the client.
- For all other types of channel, it is the queue manager name of the partner.

When the exit is initialized this field is blank because the queue manager does not know the name of the partner until after initial negotiation has taken place.

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_3.

FAPLevel (MQLONG):

Negotiated Formats and Protocols level.

This is an input field to the exit. Changes to this field should only be made under the direction of IBM service. The field is not present if *Version* is less than MQCXP_VERSION_3.

CapabilityFlags (MQLONG):

You can set the capability flag to either MQCF_NONE or MQCF_DIST_LISTS.

You can set either of the following capability flags:

MQCF_NONE
No flags.

MQCF_DIST_LISTS
Distribution lists supported.

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_3.

ExitNumber (MQLONG):

This field specifies the ordinal number of the exit.

The ordinal number of the exit, within the type defined in *ExitId*. For example, if the exit being invoked is the third message exit defined, this field contains the value 3. If the exit type is one for which a list of exits cannot be defined (for example, a security exit), this field has the value 1.

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_3.

The following fields in this structure are not present if *Version* is less than MQCXP_VERSION_5.

ExitSpace (MQLONG):

This field specifies the number of bytes in the transmission buffer reserved for the exit to use.

This field is relevant only for a send exit. It specifies the amount of space in bytes that the IBM MQ channel functions reserve in the transmission buffer for the exit to use. This field allows the exit to add to the transmission buffer a small amount of data (typically not exceeding a few hundred bytes) for use by a complementary receive exit at the other end. The data added by the send exit must be removed by the receive exit.

The value is always zero on z/OS.

Note: This facility must not be used to send large amounts of data, as it might degrade performance, or even inhibit operation of the channel.

By setting *ExitSpace* the exit is guaranteed that there is always at least that number of bytes available in the transmission buffer for the exit to use. However, the exit can use less than the amount reserved, or more than the amount reserved if there is space available in the transmission buffer. The exit space in the buffer is provided following the existing data.

ExitSpace can be set by the exit only when *ExitReason* has the value MQXR_INIT; in all other cases the value returned by the exit is ignored. On input to the exit, *ExitSpace* is zero for the MQXR_INIT call, and is the value returned by the MQXR_INIT call in other cases.

If the value returned by the MQXR_INIT call is negative, or there are fewer than 1024 bytes available in the transmission buffer for message data after reserving the requested exit space for all the send exits in the chain, the MCA outputs an error message and closes the channel. Similarly, if during data transfer the exits in the send exit chain allocate more user space than they reserved such that fewer than 1024 bytes remain in the transmission buffer for message data, the MCA outputs an error message and closes the channel. The limit of 1024 allows the control and administrative flows of the channel to be processed by the chain of send exits, without the need for the flows to be segmented.

This is an input/output field to the exit if *ExitReason* is MQXR_INIT, and an input field in all other cases. The field is not present if *Version* is less than MQCXP_VERSION_5.

SSLCertUserId (MQCHAR12):

This field specifies the UserId associated with the remote certificate.

It is blank on all platforms except z/OS

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_6.

SSLRemCertIssNameLength (MQLONG):

This field specifies the length in bytes of the full Distinguished Name of the issuer of the remote certificate pointed to by SSLCertRemoteIssuerNamePtr.

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_6. The value is zero if it is not a TLS channel.

SSLRemCertIssNamePtr (PMQVOID):

This field specifies the address of the full Distinguished Name of the issuer of the remote certificate.

Its value is the null pointer if it is not a TLS channel.

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_6.

Note: The behavior of channel security exits in determining the Subject Distinguished Name and the Issuer Distinguished Name is changed from IBM WebSphere MQ Version 7.1. For more information see Channel security exit programs.

SecurityParms (PMQCSP):

This field specifies the address of the MQSCP structure used to specify a user ID and password.

The initial value of this field is the null pointer.

This is an input/output field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_6.

The value in this field that is returned by the exit must be usable by IBM MQ until MQXR_TERM.

CurHdrCompression (MQLONG):

This field specifies which technique is currently being used to compress the header data.

It is set to one of the following:

MQCOMPRESS_NONE
No header data compression is performed.

MQCOMPRESS_SYSTEM
Header data compression is performed.

The value can be altered by a sending channel's message exit to one of the negotiated supported values accessed from the HdrCompList field of the MQCD. This enables the technique used to compress the header data to be chosen for each message based on the content of the message. The altered value is used for the current message only. The channel ends if the attribute is altered to an unsupported value. The value is ignored if altered outside a sending channel's message exit.

This is an input/output field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_6.

CurMsgCompression (MQLONG):

This field specifies which technique is currently being used to compress the message data.

It is set to one of the following:

MQCOMPRESS_NONE
No header data compression is performed.

MQCOMPRESS_RLE
Message data compression is performed using run-length encoding.

MQCOMPRESS_ZLIBFAST
Message data compression is performed using the zlib compression technique. A fast compression time is preferred.

MQCOMPRESS_ZLIBHIGH
Message data compression is performed using the zlib compression technique. A high level of compression is preferred.

The value can be altered by a sending channel's message exit to one of the negotiated supported values accessed from the MsgCompList field of the MQCD. This enables the technique used to compress the message data to be decided for each message based on the content of the message. The altered value is used for the current message only. The channel ends if the attribute is altered to an unsupported value. The value is ignored if altered outside a sending channel's message exit.

This is an input/output field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_6.

Hconn (MQHCONN):

This field specifies the connection handle that the exit uses if it needs to make any MQI calls within the exit.

This field is not relevant to exits running on client-connection channels, where it contains the value MQHC_UNUSABLE_HCONN (-1).

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_7.

SharingConversations (MQBOOL):

This field specifies whether the conversation is the only one that can currently be running on this channel instance, or whether more than one conversation can currently be running on this channel instance.

It also indicates whether the exit program is subject to the risk of the MQCD being altered by another exit program running at the same time.

This field is only relevant for exit programs running on client-connection or server-connection channels.

It is set to one of the following:

FALSE
The exit instance is the only exit instance that can currently be running on this channel instance. This allows the exit to safely update the MQCD fields without contention from other exits running on other channel instances. Whether changes to the MQCD fields are acted upon by the channel is defined by the table of MQCD fields in "Changing MQCD fields in a channel exit" on page 3605.

TRUE The exit instance is not the only exit instance that can currently be running on this channel instance. Any changes made to the MQCD are not acted upon by the channel, except for changes listed in the table of MQCD fields in "Changing MQCD fields in a channel exit" on page 3605 for Exit Reasons other than MQXR_INIT. If this exit updates the MQCD fields, ensure there is no contention from other exits running on other conversations at the same time by providing serialization among the exits that run on this channel instance.

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_7.

MCAUserSource (MQLONG):

This field specifies the source of the provided MCA user ID.

It can contain one of the following values:

MQUSRC_MAP
The user ID is specified in the MCAUSER attribute.

MQUSRC_CHANNEL
The user ID is flowed from the inbound partner or specified in the MCAUSER field defined in the channel object.

This is an input field to the exit. The field is not present if Version is less than MQCXP_VERSION_8.

pEntryPoints (PMQIEP):

This field specifies the address of the interface entry point for the MQI or DCI call.

The field is not present if *Version* is less than MQCXP_VERSION_8.

RemoteProduct (MQCHAR4):

This field specifies the remote product name.

This field identifies the remote product of the client, for example, C or Java, as displayed in the **RPRODUCT** field of DISPLAY CHSATUS.

The field is not present if *Version* is less than MQCXP_VERSION_9.

RemoteVersion (MQCHAR8):

This field specifies the name of the remote version.

This field identifies the version of the client libraries, as displayed in the **RVERSION** field of DISPLAY CHSTATUS.

The field is not present if *Version* is less than MQCXP_VERSION_9.

C declaration:

This declaration is the C declaration for the MQCXP structure.

```
typedef struct tagMQCXP MQCXP;
struct tagMQCXP {
  MQCHAR4   StrucId;           /* Structure identifier */
  MQLONG    Version;           /* Structure version number */
  MQLONG    ExitId;            /* Type of exit */
  MQLONG    ExitReason;        /* Reason for invoking exit */
  MQLONG    ExitResponse;      /* Response from exit */
  MQLONG    ExitResponse2;     /* Secondary response from exit */
  MQLONG    Feedback;          /* Feedback code */
  MQLONG    MaxSegmentLength;  /* Maximum segment length */
  MQBYTE16  ExitUserArea;      /* Exit user area */
  MQCHAR32  ExitData;          /* Exit data */
  MQLONG    MsgRetryCount;     /* Number of times the message has been
                                  retried */
  MQLONG    MsgRetryInterval;  /* Minimum interval in milliseconds after
                                  which the put operation should be
                                  retried */
  MQLONG    MsgRetryReason;    /* Reason code from previous attempt to
                                  put the message */
  MQLONG    HeaderLength;      /* Length of header information */
  MQCHAR48  PartnerName;       /* Partner Name */
  MQLONG    FAPLevel;          /* Negotiated Formats and Protocols
                                  level */
  MQLONG    CapabilityFlags;   /* Capability flags */
  MQLONG    ExitNumber;        /* Exit number */
  /* Ver:3 */
  /* Ver:4 */
  MQLONG    ExitSpace;         /* Number of bytes in transmission buffer
                                  reserved for exit to use */
  /* Ver:5 */
  MQCHAR12  SSLCertUserid;     /* User identifier associated
                                  with remote TLS certificate */
  MQLONG    SSLRemCertIssNameLength;   /* Length of
                                  distinguished name of issuer
                                  of remote TLS certificate */
```

```
  MQPTR     SSLRemCertIssNamePtr;      /* Address of
                                       distinguished name of issuer
                                       of remote TLS certificate */
  PMQVOID   SecurityParms;     /* Security parameters */
  MQLONG    CurHdrCompression; /* Header data compression
                                       used for current message */
  MQLONG    CurMsgCompression; /* Message data compression
                                       used for current message */
  /* Ver:6 */
  MQHCONN   Hconn;             /* Connection handle */
  MQBOOL    SharingConversations;      /* Multiple conversations
                                       possible on channel inst?  */
  /* Ver:7 */
  MQLONG    MCAUserSource;     /* Source of the provided MCA user ID */
  PMQIEP    pEntryPoints;      /* Address of the MQIEP structure */
  /* Ver:8 */
  MQCHAR4   RemoteProduct;     /* The identifier for the remote product */
  MQCHAR8   RemoteVersion;     /* The version of the remote product */
  /* Ver:9 */
};
```

COBOL declaration:

This declaration is the COBOL declaration for the MQCXP structure.

```
**   MQCXP structure
  10 MQCXP.
**    Structure identifier
   15 MQCXP-STRUCID          PIC X(4).
**    Structure version number
   15 MQCXP-VERSION          PIC S9(9) BINARY.
**    Type of exit
   15 MQCXP-EXITID           PIC S9(9) BINARY.
**    Reason for invoking exit
   15 MQCXP-EXITREASON       PIC S9(9) BINARY.
**    Response from exit
   15 MQCXP-EXITRESPONSE     PIC S9(9) BINARY.
**    Secondary response from exit
   15 MQCXP-EXITRESPONSE2    PIC S9(9) BINARY.
**    Feedback code
   15 MQCXP-FEEDBACK         PIC S9(9) BINARY.
**    Maximum segment length
   15 MQCXP-MAXSEGMENTLENGTH PIC S9(9) BINARY.
**    Exit user area
   15 MQCXP-EXITUSERAREA     PIC X(16).
**    Exit data
   15 MQCXP-EXITDATA         PIC X(32).
**    Number of times the message has been retried
   15 MQCXP-MSGRETRYCOUNT    PIC S9(9) BINARY.
**    Minimum interval in milliseconds after which the put operation
**    should be retried
   15 MQCXP-MSGRETRYINTERVAL PIC S9(9) BINARY.
**    Reason code from previous attempt to put the message
   15 MQCXP-MSGRETRYREASON   PIC S9(9) BINARY.
**    Length of header information
   15 MQCXP-HEADERLENGTH     PIC S9(9) BINARY.
**    Partner Name
   15 MQCXP-PARTNERNAME      PIC X(48).
**    Negotiated Formats and Protocols level
   15 MQCXP-FAPLEVEL         PIC S9(9) BINARY.
**    Capability flags
   15 MQCXP-CAPABILITYFLAGS  PIC S9(9) BINARY.
**    Exit number
   15 MQCXP-EXITNUMBER       PIC S9(9) BINARY.
**    Number of bytes in transmission buffer reserved for exit to use
   15 MQCXP-EXITSPACE        PIC S9(9) BINARY.
**    User Id associated with remote certificate
```

```
   15  MQCXP-SSLCERTUSERID   PIC X(12).
** Length of distinguished name of issuer of remote TLS
** certificate
   15  MQCXP-SSLREMCERTISSNAMELENGTH PIC S9(9) BINARY.
** Address of distinguished name of issuer of remote TLS
** certificate
   15  MQCXP-SSLREMCERTISSNAMEPTR    POINTER.
** Security parameters
   15  MQCXP-SECURITYPARMS           PIC S9(18) BINARY.
** Header data compression used for current message
   15  MQCXP-CURHDRCOMPRESSION       PIC S9(9) BINARY.
** Message data compression used for current message
   15  MQCXP-CURMSGCOMPRESSION       PIC S9(9) BINARY.
** Connection handle
   15 MQCXP-HCONN                    PIC S9(9) BINARY.
** Multiple conversations possible on channel instance?
   15 MQCXP-SHARINGCONVERSATIONS     PIC S9(9) BINARY.
** Source of the provided MCA user ID
   15 MQCXP-MCAUSERSOURCE            PIC S9(9) BINARY.
```

RPG declaration (ILE):

This declaration is the RPG declaration for the MQCXP structure.

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     D* MQCXP Structure
     D*
     D* Structure identifier
     D  CXSID                  1      4
     D* Structure version number
     D  CXVER                  5      8I 0
     D* Type of exit
     D  CXXID                  9     12I 0
     D* Reason for invoking exit
     D  CXREA                 13     16I 0
     D* Response from exit
     D  CXRES                 17     20I 0
     D* Secondary response from exit
     D  CXRE2                 21     24I 0
     D* Feedback code
     D  CXFB                  25     28I 0
     D* Maximum segment length
     D  CXMSL                 29     32I 0
     D* Exit user area
     D  CXUA                  33     48
     D* Exit data
     D  CXDAT                 49     80
     D* Number of times the message has been retried
     D  CXMRC                 81     84I 0
     D* Minimum interval in milliseconds after which the put operation
     D* should be retried
     D  CXMRI                 85     88I 0
     D* Reason code from previous attempt to put the message
     D  CXMRR                 89     92I 0
     D* Length of header information
     D  CXHDL                 93     96I 0
     D* Partner Name
     D  CXPNM                 97    144
     D* Negotiated Formats and Protocols level
     D  CXFAP                145    148I 0
     D* Capability flags
     D  CXCAP                149    152I 0
     D* Exit number
     D  CXEXN                153    156I 0
     D* Number of bytes in transmission buffer reserved for exit to use
     D  CXHDL                157    160I 0
     D* User identifier associated with remote TLS certificate
```

```
D  CXSSLCU              161    172
D* Length of distinguished name of issuer of remote TLS certificate
D  CXSRCINL             173    176I 0
D* Address of distinguished name of issuer of remote TLS certificate
D  CXSRCINP             177    192*
D* Security parameters
D  CXSECP               193    208*
D* Header data compression used for current message
D  CXCHC                209    212I 0
D* Message data compression used for current message
D  CXCMC                213    216I 0
D* Connection handle
D  CXHCONN               217    220I   0
D* Multiple conversations possible on channel instance?
D  CXSHARECONV           221    224I   0
D* Source of the provided MCA user ID
D  MCAUSERSOURCE         225    228I   0
```

System/390 assembler declaration:

This declaration is the System/390 assembler declaration for the MQCXP structure.

```
MQCXP                   DSECT
MQCXP_STRUCID           DS   CL4   Structure identifier
MQCXP_VERSION           DS   F     Structure version number
MQCXP_EXITID            DS   F     Type of exit
MQCXP_EXITREASON        DS   F     Reason for invoking exit
MQCXP_EXITRESPONSE      DS   F     Response from exit
MQCXP_EXITRESPONSE2     DS   F     Secondary response from exit
MQCXP_FEEDBACK          DS   F     Feedback code
MQCXP_MAXSEGMENTLENGTH  DS   F     Maximum segment length
MQCXP_EXITUSERAREA      DS   XL16  Exit user area
MQCXP_EXITDATA          DS   CL32  Exit data
MQCXP_MSGRETRYCOUNT     DS   F     Number of times the message has been
*                                  retried
MQCXP_MSGRETRYINTERVAL  DS   F     Minimum interval in milliseconds
*                                  after which the put operation should
*                                  be retried
MQCXP_MSGRETRYREASON    DS   F     Reason code from previous attempt to
*                                  put the message
MQCXP_HEADERLENGTH      DS   F     Length of header information
MQCXP_PARTNERNAME       DS   CL48  Partner Name
MQCXP_FAPLEVEL          DS   F     Negotiated Formats and Protocols
*                                  level
MQCXP_CAPABILITYFLAGS   DS   F     Capability flags
MQCXP_EXITNUMBER        DS   F     Exit number
MQCXP_EXITSPACE         DS   F     Number of bytes in transmission
*                                  buffer reserved for exit to use
MQCXP_SSLCERTUSERID     DS   CL12  User identifier associated with
*                                  remote TLS certificate
MQCXP_SSLREMCERTISSNAMELENGTH   DS   F     Length of distinguished name
*                                  of issuer of remote TLS certificate
MQCXP_SSLREMCERTISSNAMEPTR      DS   F     Address of distinguished name
*                                  of issuer of remote TLS certificate
MQCXP_SECURITYPARMS     DS   F    Address of security parameters
MQCXP_CURHDRCOMPRESSION         DS   F     Header data compression used for
*                                  current message
MQCXP_CURMSGCOMPRESSION         DS   F      Message data compression used for
*                                  current message
MQCXP_HCONN             DS   F Connection handle
MQCXP_SHARINGCONVERSATIONS      DS   F Multiple conversations possible on
*                                  channel inst?
MQCXP_MCAUSERSOURCE     DS   F    Source of the provided MCA user ID

MQCXP_LENGTH            EQU  *-MQCXP
                        ORG  MQCXP
MQCXP_AREA              DS   CL(MQCXP_LENGTH)
```

MQXWD - Exit wait descriptor:

The MQXWD structure is an input/output parameter on the MQXWAIT call.

This structure is supported only on z/OS.

Related reference:

"Fields"
This topic lists all the fields in the MQXWD structure and describes each field.

"C declaration" on page 3627
This declaration is the C declaration for the MQXWD structure.

"System/390 assembler declaration" on page 3627
This declaration is the System/390 assembler declaration for the MQXWD structure.

Fields:

This topic lists all the fields in the MQXWD structure and describes each field.

StrucId (MQCHAR4):

This field specifies the structure identifier.

The value must be:

MQXWD_STRUC_ID
Identifier for exit wait descriptor structure.

For the C programming language, the constant MQXWD_STRUC_ID_ARRAY is also defined; this constant has the same value as MQXWD_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQXWD_STRUC_ID.

Version (MQLONG):

This field specifies the structure version number.

The value must be:

MQXWD_VERSION_1
Version number for exit wait descriptor structure.

The initial value of this field is MQXWD_VERSION_1.

Reserved1 (MQLONG):

This field is reserved. Its value must be zero.

This is an input field.

Reserved2 (MQLONG):

This field is reserved. Its value must be zero.

This is an input field.

Reserved3 (MQLONG):

This field is reserved. Its value must be zero.

This is an input field.

ECB (MQLONG):

This field specifies the event control block to wait on.

This field is the event control block (ECB) to wait on. It must be set to zero before the MQXWAIT call is issued; on successful completion it contains the post code.

This field is an input/output field.

C declaration:

This declaration is the C declaration for the MQXWD structure.

```
typedef struct tagMQXWD MQXWD;
struct tagMQXWD {
  MQCHAR4  StrucId;    /* Structure identifier */
  MQLONG   Version;    /* Structure version number */
  MQLONG   Reserved1;  /* Reserved */
  MQLONG   Reserved2;  /* Reserved */
  MQLONG   Reserved3;  /* Reserved */
  MQLONG   ECB;        /* Event control block to wait on */
};
```

System/390 assembler declaration:

This declaration is the System/390 assembler declaration for the MQXWD structure.

```
MQXWD            DSECT
MQXWD_STRUCID    DS   CL4  Structure identifier
MQXWD_VERSION    DS   F    Structure version number
MQXWD_RESERVED1  DS   F    Reserved
MQXWD_RESERVED2  DS   F    Reserved
MQXWD_RESERVED3  DS   F    Reserved
MQXWD_ECB        DS   F    Event control block to wait on
*
MQXWD_LENGTH     EQU  *-MQXWD
                 ORG  MQXWD
MQXWD_AREA       DS   CL(MQXWD_LENGTH)
```

Cluster workload exit call and data structures

This section provides reference information for the cluster workload exit and the data structures. This is general-use programming interface information.

You can write cluster workload exits in the following programming languages:

- C
- System/390 assembler (IBM MQ for z/OS)

The call is described in:

- "MQ_CLUSTER_WORKLOAD_EXIT - Call description" on page 3629

The structure data types used by the exit are described in:

- "MQXCLWLN - Navigate Cluster workload records" on page 3630
- "MQWXP - Cluster workload exit parameter structure" on page 3633
- "MQWDR - Cluster workload destination record structure" on page 3640
- "MQWQR - Cluster workload queue record structure" on page 3644
- "MQWCR - Cluster workload cluster record structure" on page 3649
- z/OS Asynchronous behavior of CLUSTER commands on z/OS

Throughout this section, queue manager attributes and queue attributes are shown in full. The equivalent names that are used in the MQSC commands are shown below. For details of MQSC commands, see MQSC commands.

Table 386. Queue manager attributes

Full name	Name used in MQSC
ClusterWorkloadData	CLWLDATA
ClusterWorkloadExit	CLWLEXIT
ClusterWorkloadLength	CLWLLEN

Table 387. Queue attributes

Full name	Name used in MQSC
DefBind	DEFBIND
DefPersistence	DEFPSIST
DefPriority	DEFPRTY
InhibitPut	PUT
QDesc	DESCR

Related information:
Writing and compiling cluster workload exits

MQ_CLUSTER_WORKLOAD_EXIT - Call description:

The cluster workload exit is called by the queue manager to route a message to an available queue manager.

Note: No entry point called MQ_CLUSTER_WORKLOAD_EXIT is provided by the queue manager. Instead, the name of the cluster workload exit is defined by the ClusterWorkloadExit queue manager attribute.

The MQ_CLUSTER_WORKLOAD_EXIT exit is supported on all platforms.

Syntax

```
MQ_CLUSTER_WORKLOAD_EXIT (ExitParms)
```

Parameters for MQ_CLUSTER_WORKLOAD_EXIT:

Description of the parameters in the MQ_CLUSTER_WORKLOAD_EXIT call.

***ExitParms* (MQWXP) - input/output**
Exit parameter block.

- The exit sets information in MQWXP to indicate how to manage the workload.

Usage notes:

The function performed by the cluster workload exit is defined by the provider of the exit. The exit, however, must conform to the rules defined in the associated control block MQWXP.

No entry point called MQ_CLUSTER_WORKLOAD_EXIT is provided by the queue manager. However, a typedef is provided for the name MQ_CLUSTER_WORKLOAD_EXIT in the C programming language. Use the typedef to declare the user-written exit, to ensure that the parameters are correct.

Language invocations for MQ_CLUSTER_WORKLOAD_EXIT:

The MQ_CLUSTER_WORKLOAD_EXIT supports two languages, C and High Level Assembler.

C invocation

```
MQ_CLUSTER_WORKLOAD_EXIT (&ExitParms);
```

Replace *MQ_CLUSTER_WORKLOAD_EXIT* with the name of your cluster workload exit function.

Declare the **MQ_CLUSTER_WORKLOAD_EXIT** parameters as follows:

```
MQWXP ExitParms; /* Exit parameter block */
```

High Level Assembler invocation

```
CALL EXITNAME,(EXITPARMS)
```

Declare the parameters as follows:

```
EXITPARMS      CMQWXPA       Exit parameter block
```

MQXCLWLN - Navigate Cluster workload records:

The MQXCLWLN call is used to navigate through the chains of MQWDR, MQWQR, and MQWCR records stored in the cluster cache.

The cluster cache is an area of main storage used to store information relating to the cluster.

If the cluster cache is static, it has a fixed size. If you set it to dynamic, the cluster cache can expand as required.

Set the type of cluster cache to STATIC or DYNAMIC using either a system parameter or macro.

- Multi Use the system parameter ClusterCacheType on Multiplatforms.
- z/OS Use the CLCACHE parameter in the CSQ6SYSP macro on z/OS.

Syntax

```
MQXCLWLN (ExitParms, CurrentRecord, NextOffset, NextRecord, Compcode, Reason)
```

Parameters for MQXCLWLN - Navigate Cluster workload records:

Description of the parameters in the MQXCLWLN call.

ExitParms (MQWXP) - input/output
Exit parameter block.

This structure contains information relating to the invocation of the exit. The exit sets information in this structure to indicate how to manage the workload.

CurrentRecord (MQPTR) - input
Address of current record.

This structure contains information relating to the address of the record currently being examined by the exit. The record must be one of the following types:

- Cluster workload destination record (MQWDR)
- Cluster workload queue record (MQWQR)
- Cluster workload cluster record (MQWCR)

NextOffset (MQLONG) - input
Offset of next record.

This structure contains information relating to the offset of the next record or structure. *NextOffset* is the value of the appropriate offset field in the current record, and must be one of the following fields:

- ChannelDefOffset field in MQWDR
- ClusterRecOffset field in MQWDR
- ClusterRecOffset field in MQWQR
- ClusterRecOffset field in MQWCR

NextRecord (MQPTR) - output
Address of next record or structure.

This structure contains information relating to the address of the next record or structure. If *CurrentRecord* is the address of an MQWDR, and *NextOffset* is the value of the ChannelDefOffset field, *NextRecord* is the address of the channel definition structure (MQCD).

If there is no next record or structure, the queue manager sets *NextRecord* to the null pointer, and the call returns completion code MQCC_WARNING and reason code MQRC_NO_RECORD_AVAILABLE.

CompCode (MQLONG) - output
Completion code.

The completion code has one of the following values:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying CompCode

If CompCode is MQCC_OK:

MQRC_NONE
(0, X'0000')
No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_NO_RECORD_AVAILABLE
(2359, X'0937')
No record available. An MQXCLWLN call was issued from a cluster workload exit to obtain the address of the next record in the chain. The current record is the last record in the chain. Corrective action: None.

If *CompCode* is MQCC_FAILED:

MQRC_CURRENT_RECORD_ERROR
(2357, X'0935')
CurrentRecord parameter not valid. An MQXCLWLN call was issued from a cluster workload exit to obtain the address of the next record in the chain. The address specified by the **CurrentRecord** parameter is not the address of a valid record.

CurrentRecord must be the address of a destination record, MQWDR, queue record (MQWQR), or cluster record (MQWCR) residing within the cluster cache. Corrective action: Ensure that the cluster workload exit passes the address of a valid record residing in the cluster cache.

MQRC_ENVIRONMENT_ERROR
(2012, X'07DC')
Call not valid in environment. An MQXCLWLN call was issued, but not from a cluster workload exit.

MQRC_NEXT_OFFSET_ERROR
(2358, X'0936')
NextOffset parameter not valid. An MQXCLWLN call was issued from a cluster workload exit to obtain the address of the next record in the chain. The offset specified by the **NextOffset** parameter is not valid. **NextOffset** must be the value of one of the following fields:

- ChannelDefOffset field in MQWDR
- ClusterRecOffset field in MQWDR
- ClusterRecOffset field in MQWQR
- ClusterRecOffset field in MQWCR

Corrective action: Ensure that the value specified for the **NextOffset** parameter is the value of one of the fields listed previously.

MQRC_NEXT_RECORD_ERROR
(2361, X'0939')
NextRecord parameter not valid.

MQRC_WXP_ERROR
(2356, X'0934')

Workload exit parameter structure not valid. An MQXCLWLN call was issued from a cluster workload exit to obtain the address of the next record in the chain. The workload exit parameter structure **ExitParms** is not valid, for one of the following reasons:

- The parameter pointer is not valid. It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.
- The StrucId field is not MQWXP_STRUC_ID.
- The Version field is not MQWXP_VERSION_2.
- The Context field does not contain the value passed to the exit by the queue manager.

Corrective action: Ensure that the parameter specified for **ExitParms** is the MQWXP structure that was passed to the exit when the exit was invoked.

Usage notes for MQXCLWLN - Navigate Cluster workload records:

Use MQXCLWLN to navigate through cluster records, even if the cache is static.

If the cluster cache is dynamic, the MQXCLWLN call must be used to navigate through the records. The exit ends abnormally if simple pointer-and-offset arithmetic is used to navigate through the records.

If the cluster cache is static, MQXCLWLN need not be used to navigate through the records. Typically use MQXCLWLN even when the cache is static. You can then change the cluster cache to being dynamic without needing to change the workload exit.

Language invocations of MQXCLWLN:

MQXCLWLN supports two languages, C and High Level Assembler.

C invocation

```
MQXCLWLN (&ExitParms, CurrentRecord, NextOffset, &NextRecord, &CompCode, &Reason) ;
```

Declare the parameters as follows:

```
Typedef struct tagMQXCLWLN {
MQWXP   ExitParms;      /* Exit parameter block */
MQPTR   CurrentRecord;  /* Address of current record*/
MQLONG  NextOffset;     /* Offset of next record */
MQPTR   NextRecord;     /* Address of next record or structure */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQXCLWLN,(CLWLEXITPARMS,CURRENTRECORD,NEXTOFFSET,NEXTRECORD,COMPCODE,REASON)
```

Declare the parameters as follows:

```
CLWLEXITPARMS CMQWXPA, Cluster workload exit parameter block
CURRENTRECORD CMQWDRA, Current record
NEXTOFFSET    DS F     Next offset
NEXTRECORD    DS F     Next record
COMPCODE      DS F     Completion code
REASON        DS F     Reason code qualifying COMPCODE
```

MQWXP - Cluster workload exit parameter structure:

The following table summarizes the fields in the MQWXP - Cluster workload exit parameter structure.

Table 388. Fields in MQWXP

Field	Description	Page
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
ExitId	Type of exit	ExitId
ExitReason	Reason for invoking exit	ExitReason
ExitResponse	Response from exit	ExitResponse
ExitResponse2	Secondary response from exit	ExitResponse2
Feedback	Feedback code	Feedback
Flags	Flags values. These bit flags are used to indicate information about the message being put	Flags
ExitUserArea	Exit user area	ExitUserArea
ExitData	Exit data	ExitData
MsgDescPtr	Address of message descriptor (MQMD)	MsgDescPtr
MsgBufferPtr	Address of buffer containing some or all the message data	MsgBufferPtr
MsgBufferLength	Length of buffer containing message data	MsgBufferLength
MsgLength	Length of complete message	MsgLength
QName	Name of queue	QName
QMgrName	Name of local queue manager	QMgrName
DestinationCount	Number of possible destinations	DestinationCount
DestinationChosen	Destination chosen	DestinationChosen
DestinationArrayPtr	Address of an array of pointers to destination records (MQWDR)	DestinationArrayPtr
QArrayPtr	Address of an array of pointers to queue records (MQWQR)	QArrayPtr
Note: The remaining fields are ignored if Version is less than MQWXP_VERSION_2.		
CacheContext	Context information	CacheContext
CacheType	Type of cluster cache	CacheType
Note: The remaining fields are ignored if Version is less than MQWXP_VERSION_3.		
CLWLMRUChannels	Maximum number of allowed active outbound cluster channels	CLWLMRUChannels
Note: The remaining fields are ignored if Version is less than MQWXP_VERSION_4.		
pEntryPoints	Address of the MQIEP structure to allow MQI and DCI calls to be made	pEntryPoints

The cluster workload exit parameter structure describes the information that is passed to the cluster workload exit.

The cluster workload exit parameter structure is supported on all platforms

Additionally, the MQWXP1, MQWXP2 and MQWXP3 structures are available for backwards compatibility.

Fields in MQWXP - Cluster workload exit parameter structure:

Description of the fields in the MQWXP - Cluster workload exit parameter structure

StrucId (MQCHAR4) - input
The structure identifier for the cluster workload exit parameter structure.

- The StrucId value is MQWXP_STRUC_ID.
- For the C programming language, the constant MQWXP_STRUC_ID_ARRAY is also defined. It has the same value as MQWXP_STRUC_ID. It is an array of characters instead of a string.

Version (MQLONG) - input
Indicates the structure version number. Version takes one of the following values:

MQWXP_VERSION_1
Version-1 cluster workload exit parameter structure.

MQWXP_VERSION_1 is supported in all environments.

MQWXP_VERSION_2
Version-2 cluster workload exit parameter structure.

MQWXP_VERSION_2 is supported in the following environments: AIX, HP-UX, Linux, IBM i, Solaris and Windows.

MQWXP_VERSION_3
Version-3 cluster workload exit parameter structure.

MQWXP_VERSION_3 is supported in the following environments: AIX, HP-UX, Linux, IBM i, Solaris and Windows.

MQWXP_VERSION_4
Version-4 cluster workload exit parameter structure.

MQWXP_VERSION_4 is supported in the following environments: AIX, HP-UX, Linux, IBM i, Solaris and Windows.

MQWXP_CURRENT_VERSION
Current version of cluster workload exit parameter structure.

ExitId (MQLONG) - input
Indicates the type of exit being called. The cluster workload exit is the only supported exit.

- The ExitId value must be MQXT_CLUSTER_WORKLOAD_EXIT

ExitReason (MQLONG) - input
Indicates the reason for invoking the cluster workload exit. ExitReason takes one of the following values:

MQXR_INIT
Indicates that the exit is being invoked for the first time.

Acquire and initialize any resources that the exit might need, such as main storage.

MQXR_TERM
Indicates that the exit is about to be terminated.

Free any resources that the exit might have acquired since it was initialized, such as main storage.

MQXR_CLWL_OPEN
Called by MQOPEN.

MQXR_CLWL_PUT
Called by MQPUT or MQPUT1.

MQXR_CLWL_MOVE
Called by MCA when the channel state has changed.

MQXR_CLWL_REPOS
Called by MQPUT or MQPUT1 for a repository-manager PCF message.

MQXR_CLWL_REPOS_MOVE
Called by MCA for a repository-manager PCF message if the channel state has changed.

ExitResponse (MQLONG) - output
Set ExitResponse to indicate whether processing of the message continues. It must be one of the following values:

MQXCC_OK
Continue processing the message normally.

- DestinationChosen identifies the destination to which the message is to be sent.

MQXCC_SUPPRESS_FUNCTION
Discontinue processing the message.

- The actions taken by the queue manager depend on the reason the exit was invoked:

Table 389. Actions taken by the queue manager.

There are two columns in this table. The first column lists the exit reasons and the second column describes the action taken for each exit reason.

ExitReason	Action taken
• MQXR_CLWL_OPEN • MQXR_CLWL_REPOS • MQXR_CLWL_PUT	MQOPEN, MQPUT, or MQPUT1 call fail with completion code MQCC_FAILED and reason code MQRC_STOPPED_BY_CLUSTER_EXIT.
• MQXR_CLWL_MOVE • MQXR_CLWL_REPOS_MOVE	The message is placed on the dead-letter queue.

MQXCC_SUPPRESS_EXIT
Continue processing the current message normally. Do not invoke the exit again until the queue manager shuts down.

The queue manager processes subsequent messages as if the ClusterWorkloadExit queue manager attribute is blank. DestinationChosen identifies the destination to which the current message is sent.

Any other value
Process the message as if MQXCC_SUPPRESS_FUNCTION is specified.

ExitResponse2 (MQLONG) - input/output
Set ExitResponse2 to provide the queue manager with more information.

- MQXR2_STATIC_CACHE is the default value, and is set on entry to the exit.
- When ExitReason has the value MQXR_INIT, the exit can set one of the following values in ExitResponse2:

MQXR2_STATIC_CACHE
The exit requires a static cluster cache.

- If the cluster cache is static, the exit need not use the MQXCLWLN call to navigate the chains of records in the cluster cache.
- If the cluster cache is dynamic, the exit cannot navigate correctly through the records in the cache.

Note: The queue manager processes the return from the MQXR_INIT call as though the exit had returned MQXCC_SUPPRESS_EXIT in the ExitResponse field.

MQXR2_DYNAMIC_CACHE
The exit can operate with either a static or dynamic cache.
- If the exit returns this value, the exit must use the MQXCLWLN call to navigate the chains of records in the cluster cache.

Feedback (MQLONG) - input
A reserved field. The value is zero.

Flags (MQLONG) - input
Indicates information about the message being put.
- The value of Flags is MQWXP_PUT_BY_CLUSTER_CHL. The message originates from a cluster channel, rather than locally or from a non-cluster channel. In other words, the message has come from another cluster queue manager.

Reserved (MQLONG) - input
A reserved field. The value is zero.

ExitUserArea (MQBYTE16) - input/output
Set ExitUserArea to communicate between calls to the exit.
- ExitUserArea is initialized to binary zero before the first invocation of the exit. Any changes made to this field by the exit are preserved across the invocations of the exit that occur between the MQCONN call and the matching MQDISC call. The field is reset to binary zero when the MQDISC call occurs.
- The first invocation of the exit is indicated by the ExitReason field having the value MQXR_INIT.
- The following constants are defined:

  MQXUA_NONE - string
  MQXUA_NONE_ARRAY - character array
  No user information. Both constants are binary zero for the length of the field.

  MQ_EXIT_USER_AREA_LENGTH
  The length of ExitUserArea.

ExitData (MQCHAR32) - input
The value of the ClusterWorkloadData queue manager attribute. If no value has been defined for that attribute, this field is all blanks.
- The length of ExitData is given by MQ_EXIT_DATA_LENGTH.

MsgDescPtr (PMQMD) - input
The address of a copy of the message descriptor (MQMD) for the message being processed.
- Any changes made to the message descriptor by the exit are ignored by the queue manager.
- If ExitReason has one of the following values MsgDescPtr is set to the null pointer, and no message descriptor is passed to the exit:
  - MQXR_INIT
  - MQXR_TERM
  - MQXR_CLWL_OPEN

MsgBufferPtr (PMQVOID) - input
The address of a buffer containing a copy of the first MsgBufferLength bytes of the message data.
- Any changes made to the message data by the exit are ignored by the queue manager.
- No message data is passed to the exit when:
  - MsgDescPtr is the null pointer.
  - The message has no data.
  - The ClusterWorkloadLength queue manager attribute is zero.

  In these cases, MsgBufferPtr is the null pointer.

MsgBufferLength (MQLONG) - input
The length of the buffer containing the message data passed to the exit.

- The length is controlled by the ClusterWorkloadLength queue manager attribute.
- The length might be less than the length of the complete message, see MsgLength.

MsgLength (MQLONG) - input
The length of the complete message passed to the exit.

- MsgBufferLength might be less than the length of the complete message.
- MsgLength is zero if ExitReason is MQXR_INIT, MQXR_TERM, or MQXR_CLWL_OPEN.

QName (MQCHAR48) - input
The name of the destination queue. The queue is a cluster queue.

- The length of QName is MQ_Q_NAME_LENGTH.

QMgrName (MQCHAR48) - input
The name of the local queue manager that has invoked the cluster workload exit.

- The length of QMgrName is MQ_Q_MGR_NAME_LENGTH.

DestinationCount (MQLONG) - input
The number of possible destinations. Destinations are instances of the destination queue and are described by destination records.

- A destination record is a MQWDR structure. There is one structure for each possible route to each instance of the queue.
- MQWDR structures are addressed by an array of pointers, see DestinationArrayPtr.

DestinationChosen (MQLONG) - input/output
The chosen destination.

- The number of the MQWDR structure that identifies the route and queue instance where the message is to be sent.
- The value is in the range 1 - DestinationCount.
- On input to the exit, DestinationChosen indicates the route and queue instance that the queue manager has selected. The exit can accept this choice, or choose a different route and queue instance.
- The value set by the exit must be in the range 1 - DestinationCount. If any other value is returned, the queue manager uses the value of DestinationChosen on input to the exit.

DestinationArrayPtr (PPMQWDR) - input
The address of an array of pointers to destination records (MQWDR).

- There are DestinationCount destination records.

QArrayPtr (PPMQWQR) - input
The address of an array of pointers to queue records (MQWQR).

- If queue records are available, there are DestinationCount of them.
- If no queue records are available, QArrayPtr is the null pointer.

Note: QArrayPtr can be the null pointer even when DestinationCount is greater than zero.

CacheContext (MQPTR) : Version 2 - input
The CacheContext field is reserved for use by the queue manager. The exit must not alter the value of this field.

CacheType (MQLONG) : Version 2 - input
The cluster cache has one of the following types:

MQCLCT_STATIC
The cache is static.

- The size of the cache is fixed, and cannot grow as the queue manager operates.

- You do not need to use the MQXCLWLN call to navigate the records in this type of cache.

MQCLCT_DYNAMIC
The cache is dynamic.
- The size of the cache can increase in order to accommodate the varying cluster information.
- You must use the MQXCLWLN call to navigate the records in this type of cache.

CLWLMRUChannels (MQLONG) : Version 3 - input
Indicates the maximum number of active outbound cluster channels, to be considered for use by the cluster workload choice algorithm.
- `CLWLMRUChannels` is a value 1 - 999 999 999.

pEntryPoints (PMQIEP) : Version 4
The address of an MQIEP structure through which MQI and DCI calls can be made.

Initial values and language declarations for MQWXP:

Initial values and C and High Level Assembler Language declarations for `MQWXP` - Cluster workload exit parameter structure.

Table 390. Initial values of fields in MQWXP

Field name	Name of constant	Value of constant
StrucId	MQWXP_STRUC_ID	'WXP? '
Version	MQWXP_VERSION_2	2
ExitId	None	0
ExitReason	MQXCC_OK	0
ExitResponse	None	0
ExitResponse2	None	0
Flags	None	0
ExitUserArea	{MQXUA_NONE_ARRAY}	0
ExitData	None	""
MsgDescPtr	None	NULL
MsgBufferPtr	None	NULL
MsgBufferLength	None	0
MsgBufferPtr	None	0
QName	None	""
QMgrName	None	""
DestinationCount	None	0
DestinationChosen	None	0
DestinationArrayPtr	None	NULL
QArrayPtr	None	NULL
CacheContext	None	NULL
CacheType	MQCLCT_DYNAMIC	1
CLWLMRUChannels	None	0
pEntryPoints	None	NULL

Table 390. Initial values of fields in MQWXP (continued)

Field name	Name of constant	Value of constant
Notes:		
1. The symbol ? represents a single blank character.		
2. In the C programming language, the macro variable MQWXP_DEFAULT contains the default values. Use it in the following way to provide initial values for the fields in the structure: `MQWDR MyWXP = {MQWXP_DEFAULT};`		

C declaration

```
typedef struct tagMQWXP {
  MQCHAR4   StrucId;              /* Structure identifier */
  MQLONG    Version;              /* Structure version number */
  MQLONG    ExitId;               /* Type of exit */
  MQLONG    ExitReason;           /* Reason for invoking exit */
  MQLONG    ExitResponse;         /* Response from exit */
  MQLONG    ExitResponse2;        /* Reserved */
  MQLONG    Feedback;             /* Reserved */
  MQLONG    Flags;                /* Flags */
  MQBYTE16  ExitUserArea;         /* Exit user area */
  MQCHAR32  ExitData;             /* Exit data */
  PMQMD     MsgDescPtr;           /* Address of message descriptor */
  PMQVOID   MsgBufferPtr;         /* Address of buffer containing some
                                     or all of the message data */
  MQLONG    MsgBufferLength;      /* Length of buffer containing message
                                     data */
  MQLONG    MsgLength;            /* Length of complete message */
  MQCHAR48  QName;                /* Queue name */
  MQCHAR48  QMgrName;             /* Name of local queue manager */
  MQLONG    DestinationCount;     /* Number of possible destinations */
  MQLONG    DestinationChosen;    /* Destination chosen */
  PPMQWDR   DestinationArrayPtr;  /* Address of an array of pointers to
                                     destination records */
  PPMQWQR   QArrayPtr;            /* Address of an array of pointers to
                                     queue records */
  /* version 1 */
  MQPTR     CacheContext;         /* Context information */
  MQLONG    CacheType;            /* Type of cluster cache */
  /* version 2 */
  MQLONG    CLWLMRUChannnels;     /* Maximum number of most recently
                                     used cluster channels */
  /* version 3 */
  PMQIEP    pEntryPoints;         /* Address of the MQIEP structure */
  /* version 4 */
};
```

High Level Assembler

```
MQWXP                   DSECT
MQWXP_STRUCID           DS   CL4      Structure identifier
MQWXP_VERSION           DS   F        Structure version number
MQWXP_EXITID            DS   F        Type of exit
MQWXP_EXITREASON        DS   F        Reason for invoking exit
MQWXP_EXITRESPONSE      DS   F        Response from exit
MQWXP_EXITRESPONSE2     DS   F        Reserved
MQWXP_FEEDBACK          DS   F        Reserved
MQWXP_RESERVED          DS   F        Reserved
MQWXP_EXITUSERAREA      DS   XL16     Exit user area
MQWXP_EXITDATA          DS   CL32     Exit data
MQWXP_MSGDESCPTR        DS   F        Address of message
*                                     descriptor
```

```
MQWXP_MSGBUFFERPTR            DS   F        Address of buffer containing
*                                           some or all of the message
*                                           data
MQWXP_MSGBUFFERLENGTH         DS   F        Length of buffer containing
*                                           message data
MQWXP_MSGLENGTH               DS   F        Length of complete message
MQWXP_QNAME                   DS   CL48     Queue name
MQWXP_QMGRNAME                DS   CL48     Name of local queue manager
MQWXP_DESTINATIONCOUNT        DS   F        Number of possible
*                                           destinations
MQWXP_DESTINATIONCHOSEN       DS   F        Destination chosen
MQWXP_DESTINATIONARRAYPTR     DS   F        Address of an array of
*                                           pointers to destination
*                                           records
MQWXP_QARRAYPTR               DS   F        Address of an array of
*                                           pointers to queue records
MQWXP_CACHECONTEXT            DS   F        Context information
MQWXP_CACHETYPE               DS   F        Type of cluster cache
MQWXP_CLWLMRUCHANNELS         DS   F        Number of most recently used
*                                           channels for workload balancing

MQWXP_LENGTH                  EQU  *-MQWXP  Length of structure
                              ORG  MQWXP
MQWXP_AREA                    DS   CL(MQWXP_LENGTH)
```

MQWDR - Cluster workload destination record structure:

The following table summarizes the fields in the MQWDR - Cluster workload destination record structure.

Table 391. Fields in MQWDR

Field	Description	Page
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
StrucLength	Length of MQWDR structure	StrucLength
QMgrFlags	Queue manager flags	QMgrFlags
QMgrIdentifier	Queue manager identifier	QMgrIdentifier
QMgrName	Queue manager name	QMgrName
ClusterRecOffset	Logical offset of first cluster record (MQWCR)	ClusterRecOffset
ChannelState	Channel state	ChannelState
ChannelDefOffset	Logical offset of channel-definition structure (MQCD)	ChannelDefOffset
Note: The remaining fields are ignored if Version is less than MQWDR_VERSION_2.		
DestSeqNumber	Channel destination sequence number	DestSeqNumber
DestSeqFactor	Channel destination sequence factor for weighting	DestSeqFactor

The cluster workload destination record structure contains information relating to one of the possible destinations for the message. There is one cluster workload destination record structure for each instance of the destination queue.

The cluster workload destination record structure is supported in all environments.

Additionally, the MQWDR1 and MQWDR2 structures are available for backwards compatibility.

Fields in MQWDR - Cluster workload destination record structure:

Description of the parameters in the MQWDR - Cluster workload destination record structure.

StrucId (MQCHAR4) - input
The structure identifier for the cluster workload destination record structure.

- The StrucId value is MQWDR_STRUC_ID.
- For the C programming language, the constant MQWDR_STRUC_ID_ARRAY is also defined. It has the same value as MQWDR_STRUC_ID. It is an array of characters instead of a string.

Version (MQLONG) - input
The structure version number. Version takes one of the following values:

MQWDR_VERSION_1
Version-1 cluster workload destination record.

MQWDR_VERSION_2
Version-2 cluster workload destination record.

MQWDR_CURRENT_VERSION
Current version of cluster workload destination record.

StrucLength (MQLONG) - input
The length of MQWDR structure. StrucLength takes one of the following values:

MQWDR_LENGTH_1
Length of version-1 cluster workload destination record.

MQWDR_LENGTH_2
Length of version-2 cluster workload destination record.

MQWDR_CURRENT_LENGTH
Length of current version of cluster workload destination record.

QMgrFlags (MQLONG) - input
Queue manager flags indicating properties of the queue manager that hosts the instance of the destination queue described by the MQWDR structure. The following flags are defined:

MQQMF_REPOSITORY_Q_MGR
Destination is a full repository queue manager.

MQQMF_CLUSSDR_USER_DEFINED
Cluster-sender channel was defined manually.

MQQMF_CLUSSDR_AUTO_DEFINED
Cluster-sender channel was defined automatically.

MQQMF_AVAILABLE
Destination queue manager is available to receive messages.

Other values
Other flags in the field might be set by the queue manager for internal purposes.

QMgrIdentifier (MQCHAR48) - input
The queue manager identifier is a unique identifier for the queue manager that hosts the instance of the destination queue described by the MQWDR structure.

- The identifier is generated by the queue manager.
- The length of QMgrIdentifier is MQ_Q_MGR_IDENTIFIER_LENGTH.

QMgrName (MQCHAR48) - input
The name of the queue manager that hosts the instance of the destination queue described by the MQWDR structure.

- QMgrName can be the name of the local queue manager, as well another queue manager in the cluster.
- The length of QMgrName is MQ_Q_MGR_NAME_LENGTH.

ClusterRecOffset (MQLONG) - input
The logical offset of the first MQWCR structure that belongs to the MQWDR structure.

- For static caches, ClusterRecOffset is the offset of the first MQWCR structure that belongs to the MQWDR structure.
- The offset is measured in bytes from the start of the MQWDR structure.
- Do not use the logical offset for pointer arithmetic with dynamic caches. To obtain the address of the next record, the MQXCLWLN call must be used.

ChannelState (MQLONG) - input
The state of the channel that links the local queue manager to the queue manager identified by the MQWDR structure. The following values are possible:

MQCHS_BINDING
Channel is negotiating with the partner.

MQCHS_INACTIVE
Channel is not active.

MQCHS_INITIALIZING
Channel is initializing.

MQCHS_PAUSED
Channel has paused.

MQCHS_REQUESTING
Requester channel is requesting connection.

MQCHS_RETRYING
Channel is reattempting to establish connection.

MQCHS_RUNNING
Channel is transferring or waiting for messages.

MQCHS_STARTING
Channel is waiting to become active.

MQCHS_STOPPING
Channel is stopping.

MQCHS_STOPPED
Channel has stopped.

ChannelDefOffset (MQLONG) - input
The logical offset of the channel definition (MQCD) for the channel that links the local queue manager to the queue manager identified by the MQWDR structure.

- ChannelDefOffset is like ClusterRecOffset
- The logical offset cannot be used in pointer arithmetic. To obtain the address of the next record, the MQXCLWLN call must be used.

DestSeqFactor (MQLONG) - input
The destination sequence factor that allows a choice of the channel based on weight.

- DestSeqFactor is used before the queue manager changes it.
- The workload manager increases DestSeqFactor in a way that ensures messages are distributed down channels according to their weight.

DestSeqNumber (MQLONG) - input
The cluster channel destination value before the queue manager changes it.

- The workload manager increases DestSeqNumber every time a message is put down that channel.
- Workload exits can use DestSeqNumber to decide which channel to put a message down.

Initial values and language declarations for MQWDR:

Initial values and C and High Level Assembler Language declarations for MQWDR - Cluster workload destination record.

Table 392. Initial values of fields in MQWDR

Field name	Name of constant	Value of constant
StrucId	MQWDR_STRUC_ID	'WDR? '
Version	MQWDR_VERSION_1	1
StrucLength	MQWDR_CURRENT_LENGTH [3]	136
QMgrFlags	MQWDR_NONE	0
QMgrIdentifier	None	""
QMgrName	None	""
ClusterRecOffset	None	0
ChannelState	None	0
ChannelDefOffset	None	0
DestSeqNumber	None	0
DestSeqFactor	None	0

Notes:

1. The symbol ? represents a single blank character.
2. In the C programming language, the macro variable MQWDR_DEFAULT contains the default values. Use it in the following way to provide initial values for the fields in the structure:

   ```
   MQWDR MyWDR = {MQWDR_DEFAULT};
   ```

3. The initial values intentionally set the length of the structure to the length of the current version, and not version 1 of the structure.

High Level Assembler

```
MQWDR                   DSECT
MQWDR_STRUCID           DS   CL4      Structure identifier
MQWDR_VERSION           DS   F        Structure version number
MQWDR_STRUCLENGTH       DS   F        Length of MQWDR structure
MQWDR_QMGRFLAGS         DS   F        Queue manager flags
MQWDR_QMGRIDENTIFIER    DS   CL48     Queue manager identifier
MQWDR_QMGRNAME          DS   CL48     Queue manager name
MQWDR_CLUSTERRECOFFSET  DS   F        Offset of first cluster
*                                     record
MQWDR_CHANNELSTATE      DS   F        Channel state
MQWDR_CHANNELDEFOFFSET  DS   F        Offset of channel definition
*                                     structure
MQWDR_LENGTH            EQU  *-MQWDR  Length of structure
                        ORG  MQWDR
MQWDR_AREA              DS   CL(MQWDR_LENGTH)
```

C declaration

```
typedef struct tagMQWDR {
  MQCHAR4   StrucId;          /* Structure identifier */
  MQLONG    Version;          /* Structure version number */
  MQLONG    StrucLength;      /* Length of MQWDR structure */
  MQLONG    QMgrFlags;        /* Queue managerflags */
  MQCHAR48  QMgrIdentifier;   /* Queue manageridentifier */
```

```
  MQCHAR48  QMgrName;          /* Queue manager name */
  MQLONG    ClusterRecOffset;  /* Offset of first cluster record */
  MQLONG    ChannelState;      /* Channel state */
  MQLONG    ChannelDefOffset;  /* Offset of channel definition structure */
  /* Ver:1 */
  MQLONG    DestSeqNumber;     /* Cluster channel destination sequence number */
  MQINT64   DestSeqFactor;     /* Cluster channel factor sequence number */
  /* Ver:2 */
};
```

MQWQR - Cluster workload queue record structure:

The following table summarizes the fields in the MQWQR - Cluster workload queue record structure.

Table 393. Fields in MQWQR

Field	Description	Page
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
StrucLength	Length of MQWQR structure	StrucLength
QFlags	Queue flags	QFlags
QName	Queue name	QName
QMgrIdentifier	Queue manager identifier	QMgrIdentifier
ClusterRecOffset	Offset of first cluster record (MQWCR)	ClusterRecOffset
QType	Queue type	QType
QDesc	Queue description	QDesc
DefBind	Default binding	DefBind
DefPersistence	Default message persistence	DefPersistence
DefPriority	Default message priority	DefPriority
InhibitPut	Whether put operations on the queue are allowed	InhibitPut
Note: The remaining fields are ignored if Version is less than MQWQR_VERSION_2.		
CWLQueuePriority	A value 0 - 9 representing the priority of the queue	CLWLQueuePriority
CLWLQueueRank	A value 0 - 9 representing the rank of the queue	CLWLQueueRank
Note: The remaining fields are ignored if Version is less than MQWQR_VERSION_3.		
DefPutResponse	Default put response	DefPutResponse

The cluster workload queue record structure contains information relating to one of the possible destinations for the message. There is one cluster workload queue record structure for each instance of the destination queue.

The cluster workload queue record structure is supported in all environments.

Additionally, the MQWQR1 and MQWQR2 structures are available for backwards compatibility.

Fields in MQWQR - Cluster workload queue record structure:

Description of the fields in the MQWQR - Cluster workload queue record structure.

StrucId (MQCHAR4) - input
The structure identifier for the cluster workload queue record structure.

- The StrucId value is MQWQR_STRUC_ID.
- For the C programming language, the constant MQWQR_STRUC_ID_ARRAY is also defined. It has the same value as MQWQR_STRUC_ID. It is an array of characters instead of a string.

Version (MQLONG) - input
The structure version number. Version takes one of the following values:

MQWQR_VERSION_1
Version-1 cluster workload queue record.

MQWQR_VERSION_2
Version-2 cluster workload queue record.

MQWQR_VERSION_3
Version-3 cluster workload queue record.

MQWQR_CURRENT_VERSION
Current version of cluster workload queue record.

StrucLength (MQLONG) - input
The length of MQWQR structure. StrucLength takes one of the following values:

MQWQR_LENGTH_1
Length of version-1 cluster workload queue record.

MQWQR_LENGTH_2
Length of version-2 cluster workload queue record.

MQWQR_LENGTH_3
Length of version-3 cluster workload queue record.

MQWQR_CURRENT_LENGTH
Length of current version of cluster workload queue record.

QFlags (MQLONG) - input
The queue flags indicate properties of the queue. The following flags are defined:

MQQF_LOCAL_Q
Destination is a local queue.

MQQF_CLWL_USEQ_ANY
Allow use of local and remote queues in puts.

MQQF_CLWL_USEQ_LOCAL
Allow only local queue puts.

Other values
Other flags in the field might be set by the queue manager for internal purposes.

QName (MQCHAR48) - input
The name of the queue that is one of the possible destinations of the message.

- The length of QName is MQ_Q_NAME_LENGTH.

QMgrIdentifier (MQCHAR48) - input
The queue manager identifier is a unique identifier for the queue manager that hosts the instance of the queue described by the MQWQR structure.

- The identifier is generated by the queue manager.
- The length of QMgrIdentifier is MQ_Q_MGR_IDENTIFIER_LENGTH.

ClusterRecOffset (MQLONG) - input
The logical offset of the first MQWCR structure that belongs to the MQWQR structure.

- For static caches, ClusterRecOffset is the offset of the first MQWCR structure that belongs to the MQWQR structure.
- The offset is measured in bytes from the start of the MQWQR structure.
- Do not use the logical offset for pointer arithmetic with dynamic caches. To obtain the address of the next record, the MQXCLWLN call must be used.

QType (MQLONG) - input
The queue type of the destination queue. The following values are possible:

MQCQT_LOCAL_Q
Local queue.

MQCQT_ALIAS_Q
Alias queue.

MQCQT_REMOTE_ Q
Remote queue.

MQCQT_Q_MGR_ALIAS
Queue manager alias.

QDesc (MQCHAR64) - input
The queue description queue attribute defined on the queue manager that hosts the instance of the destination queue described by the MQWQR structure.

- The length of QDesc is MQ_Q_DESC_LENGTH.

DefBind (MQLONG) - input
The default binding queue attribute defined on the queue manager that hosts the instance of the destination queue described by the MQWQR structure. Either MQBND_BIND_ON_OPEN or MQBND_BIND_ON_GROUP must be specified when using groups with clusters. The following values are possible:

MQBND_BIND_ON_OPEN
Binding fixed by MQOPEN call.

MQBND_BIND_NOT_FIXED
Binding not fixed.

MQBND_BIND_ON_GROUP
Allows an application to request that a group of messages are all allocated to the same destination instance.

DefPersistence (MQLONG) - input
The default message persistence queue attribute defined on the queue manager that hosts the instance of the destination queue described by the MQWQR structure. The following values are possible:

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

DefPriority (MQLONG) - input

The default message priority queue attribute defined on the queue manager that hosts the instance of the destination queue described by the MQWQR structure. The priority range is 0 - MaxPriority.

- 0 is the lowest priority.
- MaxPriority is the queue manager attribute of the queue manager that hosts this instance of the destination queue.

InhibitPut (MQLONG) - input
The put inhibited queue attribute defined on the queue manager that hosts the instance of the destination queue described by the MQWQR structure. The following values are possible:

MQQA_PUT_INHIBITED
Put operations are inhibited.

MQQA_PUT_ALLOWED
Put operations are allowed.

CLWLQueuePriority (MQLONG) - input
The cluster workload queue priority attribute defined on the queue manager that hosts the instance of the destination queue described by the MQWQR structure.

CLWLQueueRank (MQLONG) - input
The cluster workload queue rank defined on the queue manager that hosts the instance of the destination queue described by the MQWQR structure.

DefPutResponse (MQLONG) - input
The default put response queue attribute defined on the queue manager that hosts the instance of the destination queue described by the MQWQR structure. The following values are possible:

MQPRT_SYNC_RESPONSE
Synchronous response to MQPUT or MQPUT1 calls.

MQPRT_ASYNC_RESPONSE
Asynchronous response to MQPUT or MQPUT1 calls.

Initial values and language declarations for MQWQR:

Initial values and C and High Level Assembler Language declarations for MQWQR - Cluster workload queue record.

Table 394. Initial values of fields in MQWQR

Field name	Name of constant	Value of constant
StrucId	MQWQR_STRUC_ID_ARRAY	'WQR? '
Version	MQWQR_VERSION_1	1
StrucLength	MQWQR_CURRENT_LENGTH [3]	212
QFlags	None	0
QName	None	""
QMgrIdentifier	None	""
ClusterRecOffset	None	0
QType	None	0
QDesc	None	""
DefBind	None	0
DefPersistence	None	0
DefPriority	None	0
InhibitPut	None	0
CLWLQueuePriority	None	0
CLWLQueueRank	None	0
DefPutResponse	None	1

Table 394. Initial values of fields in MQWQR (continued)

Field name	Name of constant	Value of constant
Notes: 1. The symbol ? represents a single blank character. 2. In the C programming language, the macro variable MQWQR_DEFAULT contains the default values. Use it in the following way to provide initial values for the fields in the structure: `MQWQR MyWQR = {MQWQR_DEFAULT};` 3. The initial values intentionally set the length of the structure to the length of the current version, and not version 1 of the structure.		

C declaration

```
typedef struct tagMQWQR {
  MQCHAR4   StrucId;           /* Structure identifier */
  MQLONG    Version;           /* Structure version number */
  MQLONG    StrucLength;       /* Length of MQWQR structure */
  MQLONG    QFlags;            /* Queue flags */
  MQCHAR48  QName;             /* Queue name */
  MQCHAR48  QMgrIdentifier;    /* Queue manageridentifier */
  MQLONG    ClusterRecOffset;  /* Offset of first cluster record */
  MQLONG    QType;             /* Queue type */
  MQCHAR64  QDesc;             /* Queue description */
  MQLONG    DefBind;           /* Default binding */
  MQLONG    DefPersistence;    /* Default message persistence */
  MQLONG    DefPriority;       /* Default message priority */
  MQLONG    InhibitPut;        /* Whether put operations on the queue
                                  are allowed */
  /* version 2 */
  MQLONG    CLWLQueuePriority; /* Queue priority */
  MQLONG    CLWLQueueRank;     /* Queue rank */
  /* version 3 */
  MQLONG    DefPutResponse;    /* Default put response */
};
```

High Level Assembler

```
MQWQR                   DSECT
MQWQR_STRUCID           DS   CL4      Structure identifier
MQWQR_VERSION           DS   F        Structure version number
MQWQR_STRUCLENGTH       DS   F        Length of MQWQR structure
MQWQR_QFLAGS            DS   F        Queue flags
MQWQR_QNAME             DS   CL48     Queue name
MQWQR_QMGRIDENTIFIER    DS   CL48     Queue manager identifier
MQWQR_CLUSTERRECOFFSET  DS   F        Offset of first cluster
*                                     record
MQWQR_QTYPE             DS   F        Queue type
MQWQR_QDESC             DS   CL64     Queue description
MQWQR_DEFBIND           DS   F        Default binding
MQWQR_DEFPERSISTENCE    DS   F        Default message persistence
MQWQR_DEFPRIORITY       DS   F        Default message priority
MQWQR_INHIBITPUT        DS   F        Whether put operations on
*                                     the queue are allowed
MQWQR_DEFPUTRESPONSE    DS   F        Default put response
MQWQR_LENGTH            EQU  *-MQWQR  Length of structure
                        ORG  MQWQR
MQWQR_AREA              DS   CL(MQWQR_LENGTH)
```

MQWCR - Cluster workload cluster record structure:

The following table summarizes the fields in the MQWCR cluster workload record structure.

Table 395. Fields in MQWCR.

There are three columns in this table. The first column lists the field names, the second column provides a description of each field and the third column provides a link to further information about each field.

Field	Description	Page
ClusterName	Name of cluster	ClusterName
ClusterRecOffset	Offset of next cluster record (MQWCR)	ClusterRecOffset
ClusterFlags	Cluster flags	ClusterFlags

The cluster workload cluster record structure contains information about a cluster. For each cluster the destination queue belongs to, there is one cluster workload cluster record structure.

The cluster workload cluster record structure is supported in all environments.

Fields in the MQWCR - Cluster workload cluster record structure.:

Description of the fields in the MQWCR - Cluster workload cluster record structure.

ClusterName (MQCHAR48) - input
The name of a cluster to which the instance of the destination queue that owns the MQWCR structure belongs. The destination queue instance is described by an MQWDR structure.

- The length of ClusterName is MQ_CLUSTER_NAME_LENGTH.

ClusterRecOffset (MQLONG) - input
The logical offset of the next MQWCR structure.

- If there are no more MQWCR structures, ClusterRecOffset is zero.
- The offset is measured in bytes from the start of the MQWCR structure.

ClusterFlags (MQLONG) - input
The cluster flags indicate properties of the queue manager identified by the MQWCR structure. The following flags are defined:

MQQMF_REPOSITORY_Q_MGR
Destination is a full repository queue manager.

MQQMF_CLUSSDR_USER_DEFINED
Cluster-sender channel was defined manually.

MQQMF_CLUSSDR_AUTO_DEFINED
Cluster-sender channel was defined automatically.

MQQMF_AVAILABLE
Destination queue manager is available to receive messages.

Other values
Other flags in the field might be set by the queue manager for internal purposes.

Initial values and language declarations for MQWCR:

Initial values and C and High Level Assembler Language declarations for MQWCR - Cluster workload cluster record structure.

Table 396. Initial values of fields in MQWCR

Field name	Name of constant	Value of constant
ClusterName	None	""
ClusterRecOffset	None	0
ClusterFlags	None	0

C declaration

```
typedef struct tagMQWCR {
  MQCHAR48  ClusterName;       /* Cluster name */
  MQLONG    ClusterRecOffset;  /* Offset of next cluster record */
  MQLONG    ClusterFlags;      /* Cluster flags */
};
```

High Level Assembler

```
MQWCR                           DSECT
MQWCR_CLUSTERNAME               DS   CL48     Cluster name
MQWCR_CLUSTERRECOFFSET          DS   F        Offset of next cluster
*                                            record
MQWCR_CLUSTERFLAGS              DS   F        Cluster flags
MQWCR_LENGTH                    EQU  *-MQWCR  Length of structure
                                ORG  MQWCR
MQWCR_AREA                      DS   CL(MQWCR_LENGTH)
```

API exit reference

This section provides reference information mainly of interest to a programmer writing API exits.

General usage notes

notes:

1. All exit functions can issue the MQXEP call; this call is designed specifically for use from API exit functions.
2. The MQ_INIT_EXIT function cannot issue any MQ calls other than MQXEP.
3. You cannot issue the MQDISC call for the current connection.
4. If an exit function issues the MQCONN call, or the MQCONNX call with the MQCNO_HANDLE_SHARE_NONE option, the call completes with reason code MQRC_ALREADY_CONNECTED, and the handle returned is the same as the one passed to the exit as a parameter.
5. In general when an API exit function issues an MQI call, API exits are not be called recursively. However, if an exit function issues the MQCONNX call with the MQCNO_HANDLE_SHARE_BLOCK or MQCNO_HANDLE_SHARE_NO_BLOCK options, the call returns a new shared handle. This provides the exit suite with a connection handle of its own, and hence a unit of work that is independent of the application's unit of work. The exit suite can use this handle to put and get messages within its own unit of work, and commit or back out that unit of work; all of this can be done without affecting the application's unit of work in any way.

   Because the exit function is using a connection handle that is different from the handle being used by the application, MQ calls issued by the exit function result in the relevant API exit functions being invoked. Exit functions can therefore be invoked recursively. Note that both the *ExitUserArea*

field in MQAXP and the exit chain area have connection-handle scope. Consequently, an exit function cannot use those areas to signal to another instance of itself invoked recursively that it is already active.

6. Exit functions can also put and get messages within the application's unit of work. When the application commits or backs out the unit of work, all messages within the unit of work are committed or backed out together, regardless of who placed them in the unit of work (application or exit function). However, the exit can cause the application to exceed system limits sooner than would otherwise be the case (for example, by exceeding the maximum number of uncommitted messages in a unit of work).

   When an exit function uses the application's unit of work in this way, the exit function should usually avoid issuing the MQCMIT call, as this commits the application's unit of work and might impair the correct functioning of the application. However, the exit function might sometimes need to issue the MQBACK call, if the exit function encounters a serious error that prevents the unit of work being committed (for example, an error putting a message as part of the application's unit of work). When MQBACK is called, take care to ensure that the application unit of work boundaries are not changed. In this situation the exit function must set the appropriate values to ensure that completion code MQCC_WARNING and reason code MQRC_BACKED_OUT are returned to the application, so that the application can detect the fact that the unit of work has been backed out.

   If an exit function uses the application's connection handle to issue MQ calls, those calls do not themselves result in further invocations of API exit functions.

7. If an MQXR_BEFORE exit function terminates abnormally, the queue manager might be able to recover from the failure. If it can, the queue manager continues processing as though the exit function had returned MQXCC_FAILED. If the queue manager cannot recover, the application is terminated.
8. If an MQXR_AFTER exit function terminates abnormally, the queue manager might be able to recover from the failure. If it can, the queue manager continues processing as though the exit function had returned MQXCC_FAILED. If the queue manager cannot recover, the application is terminated. Be aware that in the latter case, messages retrieved outside a unit of work are lost (this is the same situation as the application failing immediately after removing a message from the queue).
9. The MCA process performs a two phase commit.

   If an API exit intercepts an MQCMIT from a prepared MCA process and attempts to perform an action within the unit of work, then the action will fail with reason code MQRC_UOW_NOT_AVAILABLE.

10. For a multi-installation environment, the only way to have an exit that works with both WebSphere MQ Version 7.0 and Version 7.1 is to write the exit in a way which links at Version 7.0 with mqm.Lib and, for non-primary or relocated exits, to ensure that the application finds the correct mqm.Lib for the installation with which the queue manager is currently associated, prior to the application launch. (For example, run the **setmqenv -m QM** command before launching the application, even if the queue manager is owned by a Version 7.0 installation.)
11. Where multiple installations of IBM MQ are available, use the exits written for an earlier version of IBM MQ, as new functionality added in the later version might not work with earlier versions. For more information about changes between releases, see What's changed in IBM MQ 8.0.

IBM MQ API exit parameter structure (MQAXP):

The MQAXP structure, an external control block, is used as an input or output parameter to the API exit. This topic also gives information about how queue managers process exit functions.

MQAXP has the following C declaration:

```
typedef struct tagMQAXP {
  MQCHAR4   StrucId;           /* Structure identifier */
  MQLONG    Version;           /* Structure version number */
  MQLONG    ExitId;            /* Exit Identifier */
  MQLONG    ExitReason;        /* Exit invocation reason */
  MQLONG    ExitResponse;      /* Response code from exit */
  MQLONG    ExitResponse2;     /* Secondary response code from exit */
  MQLONG    Feedback;          /* Feedback code from exit */
  MQLONG    APICallerType;     /* MQSeries API caller type */
  MQBYTE16  ExitUserArea;      /* User area for use by exit */
  MQCHAR32  ExitData;          /* Exit data area */
  MQCHAR48  ExitInfoName;      /* Exit information name */
  MQBYTE48  ExitPDArea;        /* Problem determination area */
  MQCHAR48  QMgrName;          /* Name of local queue manager */
  PMQACH    ExitChainAreaPtr;  /* Inter exit communication area */
  MQHCONFIG Hconfig;           /* Configuration handle */
  MQLONG    Function;          /* Function Identifier */
  /* Ver:1 */
  MQHMSG    ExitMsgHandle      /* Exit message handle
  /* Ver:2 */
};
```

The following parameter list is passed when functions in an API exit are invoked:

StrucId (MQCHAR4) - input
The exit parameter structure identifier, with a value of:

`MQAXP_STRUC_ID.`

The exit handler sets this field on entry to each exit function.

Version (MQLONG) - input
The structure version number, with a value of:

MQAXP_VERSION_1
Version 1 API exit parameter structure.

MQAXP_VERSION_2
Version 2 API exit parameter structure.

MQAXP_CURRENT_VERSION
Current version number for the API exit parameter structure.

The exit handler sets this field on entry to each exit function.

ExitId (MQLONG) - input
The exit identifier, set on entry to the exit routine, indicating the type of exit:

MQXT_API_EXIT
API exit.

ExitReason (MQLONG) - input
The reason for invoking the exit, set on entry to each exit function:

MQXR_CONNECTION
The exit is being invoked to initialize itself before an MQCONN or MQCONNX call, or to end itself after an MQDISC call.

MQXR_BEFORE
The exit is being invoked before executing an API call, or before converting data on an MQGET.

MQXR_AFTER
The exit is being invoked after executing an API call.

ExitResponse (MQLONG) - output
The response from the exit, initialized on entry to each exit function to:

MQXCC_OK
Continue normally.

This field must be set by the exit function, to communicate to the queue manager the result of executing the exit function. The value must be one of the following:

MQXCC_OK
The exit function completed successfully. Continue normally.

This value can be set by all MQXR_* exit functions. ExitResponse2 is used to decide whether to invoke exit functions later in the chain.

MQXCC_FAILED
The exit function failed because of an error.

This value can be set by all MQXR_* exit functions. The queue manager sets CompCode to MQCC_FAILED, and Reason to:

- MQRC_API_EXIT_INIT_ERROR if the function is MQ_INIT_EXIT
- MQRC_API_EXIT_TERM_ERROR if the function is MQ_TERM_EXIT
- MQRC_API_EXIT_ERROR for all other exit functions

The values set can be altered by an exit function later in the chain.

ExitResponse2 is ignored; the queue manager continues processing as though MQXR2_SUPPRESS_CHAIN had been returned.

MQXCC_SUPPRESS_FUNCTION
Suppress IBM MQ API function.

This value can be set only by an MQXR_BEFORE exit function. It bypasses the API call. If it is returned by the MQ_DATA_CONV_ON_GET_EXIT, data conversion is bypassed. The queue manager sets CompCode to MQCC_FAILED, and Reason to MQRC_SUPPRESSED_BY_EXIT, but the values set can be altered by an exit function later in the chain. Other parameters for the call remain as the exit left them. ExitResponse2 is used to decide whether to invoke exit functions later in the chain.

If this value is set by an MQXR_AFTER or MQXR_CONNECTION exit function, the queue manager continues processing as though MQXCC_FAILED had been returned.

MQXCC_SKIP_FUNCTION
Skip IBM MQ API function.

This value can be set only by an MQXR_BEFORE exit function. It bypasses the API call. If it is returned by the MQ_DATA_CONV_ON_GET_EXIT, data conversion is bypassed. The exit function must set CompCode and Reason to the values to be returned to the application, but the values set can be altered by an exit function later in the chain. Other parameters for the call remain as the exit left them. ExitResponse2 is used to decide whether to invoke exit functions later in the chain.

If this value is set by an MQXR_AFTER or MQXR_CONNECTION exit function, the queue manager continues processing as though MQXCC_FAILED had been returned.

MQXCC_SUPPRESS_EXIT
Suppress all exit functions belonging to the set of exits.

This value can be set only by the MQXR_BEFORE and MQXR_AFTER exit functions. It bypasses *all* subsequent invocations of exit functions belonging to this set of exits for this logical connection. This bypassing continues until the logical disconnect request occurs, when MQ_TERM_EXIT function is invoked with an ExitReason of MQXR_CONNECTION.

The exit function must set CompCode and Reason to the values to be returned to the application, but the values set can be altered by an exit function later in the chain. Other parameters for the call remain as the exit left them. ExitResponse2 is ignored.

If this value is set by an MQXR_CONNECTION exit function, the queue manager continues processing as though MQXCC_FAILED had been returned.

For information about the interaction between ExitResponse and ExitResponse2, and its effect on exit processing, see "How queue managers process exit functions" on page 3656.

ExitResponse2 (MQLONG) - output
This is a secondary exit response code that qualifies the primary exit response code for MQXR_BEFORE exit functions. It is initialized to:

```
MQXR2_DEFAULT_CONTINUATION
```

on entry to an IBM MQ API call exit function. It can then be set to one of the values:

MQXR2_DEFAULT_CONTINUATION
Whether to continue with the next exit in the chain, depending on the value of ExitResponse.

If ExitResponse is MQXCC_SUPPRESS_FUNCTION or MQXCC_SKIP_FUNCTION, bypass exit functions later in the MQXR_BEFORE chain and the matching exit functions in the MQXR_AFTER chain. Invoke exit functions in the MQXR_AFTER chain that match exit functions earlier in the MQXR_BEFORE chain.

Otherwise, invoke the next exit in the chain.

MQXR2_SUPPRESS_CHAIN
Suppress the chain.

Bypass exit functions later in the MQXR_BEFORE chain and the matching exit functions in the MQXR_AFTER chain for this API call invocation. Invoke exit functions in the MQXR_AFTER chain that match exit functions earlier in the MQXR_BEFORE chain.

MQXR2_CONTINUE_CHAIN
Continue with the next exit in the chain.

For information about the interaction between ExitResponse and ExitResponse2, and its effect on exit processing, see "How queue managers process exit functions" on page 3656.

Feedback (MQLONG) - input/output
Communicate feedback codes between exit function invocations. This is initialized to:

```
MQFB_NONE (0)
```

before invoking the first function of the first exit in a chain.

Exits can set this field to any value, including any valid MQFB_* or MQRC_* value. Exits can also set this field to a user-defined feedback value in the range MQFB_APPL_FIRST to MQFB_APPL_LAST.

APICallerType (MQLONG) - input
The API caller type, indicating whether the IBM MQ API caller is external or internal to the queue manager: MQXACT_EXTERNAL or MQXACT_INTERNAL.

ExitUserArea (MQBYTE16) - input/output
A user area, available to all the exits associated with a particular ExitInfoObject. It is initialized to MQXUA_NONE (binary zeros for the length of the ExitUserArea) before invoking the first exit function (MQ_INIT_EXIT) for the hconn. From then on, any changes made to this field by an exit function are preserved across invocations of functions of the same exit.

This field is aligned to a multiple of 4 MQLONGs.

Exits can also anchor any storage that they allocate from this area.

For each hconn, each exit in a chain of exits has a different ExitUserArea. The ExitUserArea cannot be shared by exits in a chain, and the contents of the ExitUserArea for one exit are not available to another exit in a chain.

For C programs, the constant MQXUA_NONE_ARRAY is also defined with the same value as MQXUA_NONE, but as an array of characters instead of a string.

The length of this field is given by MQ_EXIT_USER_AREA_LENGTH.

ExitData (MQCHAR32) - input
Exit data, set on input to each exit function to the 32 characters of exit-specific data that is provided in the exit. If you define no value in the exit this field is all blanks.

The length of this field is given by MQ_EXIT_DATA_LENGTH.

ExitInfoName (MQCHAR48) - input
The exit information name, set on input to each exit function to the ApiExit_name specified in the exit definitions in the stanzas.

ExitPDArea (MQBYTE48) - input/output
A problem determination area, initialized to `MQXPDA_NONE` (binary zeros for the length of the field) for each invocation of an exit function.

For C programs, the constant MQXPDA_NONE_ARRAY is also defined with the same value as MQXPDA_NONE, but as an array of characters instead of a string.

The exit handler always writes this area to the IBM MQ trace at the end of an exit, even when the function is successful.

The length of this field is given by MQ_EXIT_PD_AREA_LENGTH.

QMgrName (MQCHAR48) - input
The name of the queue manager the application is connected to, that has invoked an exit as a result of processing an IBM MQ API call.

If the name of a queue manager supplied on an MQCONN or MQCONNX calls is blank, this field is still set to the name of the queue manager to which the application is connected, whether the application is server or client.

The exit handler sets this field on entry to each exit function.

The length of this field is given by MQ_Q_MGR_NAME_LENGTH.

ExitChainAreaPtr (PMQACH) - input/output
This is used to communicate data across invocations of different exits in a chain. It is set to a NULL pointer before invoking the first function (MQ_INIT_EXIT with ExitReason MQXR_CONNECTION) of the first exit in a chain of exits. The value returned by the exit on one invocation is passed on to the next invocation.

Refer to “The exit chain area and exit chain area header (MQACH)” on page 3660 for more details about how to use the exit chain area.

Hconfig (MQHCONFIG) - input
The configuration handle, representing the set of functions being initialized. This value is

generated by the queue manager on the MQ_INIT_EXIT function, and is later passed to the API exit function. It is set on entry to each exit function.

You can use Hconfig as a pointer to the MQIEP structure to make MQI and DCI calls. You must check that the first 4 bytes of HConfig match the StrucId of the MQIEP structure before using the HConfig parameter as a pointer to the MQIEP structure.

Function (MQLONG) - input
The function identifier, valid values for which are the MQXF_* constants described in "External constants" on page 3662.

The exit handler sets this field to the correct value, on entry to each exit function, depending on the IBM MQ API call that resulted in the exit being invoked.

ExitMsgHandle (MQHMSG) - input/output
When Function is MQXF_GET and ExitReason is MQXR_AFTER, a valid message handle is returned in this field allowing the API exit access to the message descriptor fields and any other properties matching the ExitProperties string specified in the MQXEPO structure when registering the API exit.

Any non-message descriptor properties that are returned in the ExitMsgHandle will not be available from the MsgHandle in the MQGMO structure if one was specified, or in the message data.

When Function is MQXF_GET and ExitReason is MQXR_BEFORE, if the exit program sets this field to MQHM_NONE then it will suppress the populating of the ExitMsgHandle properties.

This field is not set if Version is less then MQAXP_VERSION_2.

How queue managers process exit functions

The processing performed by the queue manager on return from an exit function depends on both ExitResponse and ExitResponse2.

Table 397 on page 3657 summarizes the possible combinations and their effects for an MQXR_BEFORE exit function, showing:

- Who sets the CompCode and Reason parameters of the API call
- Whether the remaining exit functions in the MQXR_BEFORE chain and the matching exit functions in the MQXR_AFTER chain are invoked
- Whether the API call is invoked

For an MQXR_AFTER exit function:

- CompCode and Reason are set in the same way as MQXR_BEFORE
- ExitResponse2 is ignored (the remaining exit functions in the MQXR_AFTER chain are always invoked)
- MQXCC_SUPPRESS_FUNCTION and MQXCC_SKIP_FUNCTION are not valid

For an MQXR_CONNECTION exit function:

- CompCode and Reason are set in the same way as MQXR_BEFORE
- ExitResponse2 is ignored
- MQXCC_SUPPRESS_FUNCTION, MQXCC_SKIP_FUNCTION, MQXCC_SUPPRESS_EXIT are not valid

In all cases, where an exit or the queue manager sets CompCode and Reason, the values set can be changed by an exit invoked later, or by the API call (if the API call is invoked later).

Table 397. MQXR_BEFORE exit processing

Value of ExitResponse	CompCode and Reason set by	Value of ExitResponse2 (default continuation) Chain	Value of ExitResponse2 (default continuation) API
MQXCC_OK	exit	Y	Y
MQXCC_SUPPRESS_EXIT	exit	Y	Y
MQXCC_SUPPRESS_FUNCTION	queue manager	N	N
MQXCC_SKIP FUNCTION	exit	N	N
MQXCC_FAILED	queue manager	N	N

How clients process exit functions

In general, clients process exit functions in the same way that server applications do, and the *QMgrName* attribute in this structure applies whether the function is on a server or a client.

However, the client has no concept of the *mqs.ini* file, so the *ApiExitCommon* and *APIExitTemplate* stanzas do not apply. Only the *ApiExitLocal* stanza applies, and this stanza is configured in the *mqclient.ini* file.

IBM MQ API exit context structure (MQAXC):

The MQAXC structure, an external control block, is used as an input parameter to an API exit.

MQAXC has the following C declaration:

```
typedef struct tagMQAXC {
   MQCHAR4   StrucId;                /* Structure identifier */
   MQLONG    Version;                /* Structure version number */
   MQLONG    Environment;            /* Environment */
   MQCHAR12  UserId;                 /* UserId associated with appl */
   MQBYTE40  SecurityId              /* Extension to UserId running appl */
   MQCHAR264 ConnectionName;         /* Connection name */
   MQLONG    LongMCAUserIdLength;    /* long MCA user identifier length */
   MQLONG    LongRemoteUserIdLength; /* long remote user identifier length */
   MQPTR     LongMCAUserIdPtr;       /* long MCA user identifier address */
   MQPTR     LongRemoteUserIdPtr;    /* long remote user identifier address */
   MQCHAR28  ApplName;               /* Application name */
   MQLONG    ApplType;               /* Application type */
   MQPID     ProcessId;              /* Process identifier */
   MQTID     ThreadId;               /* Thread identifier */

   /* Ver:1 */
   MQCHAR    ChannelName[20]         /* Channel Name */
   MQBYTE4   Reserved1;              /* Reserved */
   PMQCD     pChannelDefinition;     /* Channel Definition pointer */
 };
```

The parameters to MQAXC are:

StrucId (MQCHAR4) - input
The exit context structure identifier, with a value of `MQAXC_STRUC_ID`. For C programs, the constant MQAXC_STRUC_ID_ARRAY is also defined, with the same value as MQAXC_STRUC_ID, but as an array of characters instead of a string.

The exit handler sets this field on entry to each exit function.

Version (MQLONG) - input
The structure version number, with a value of:

MQAXC_VERSION_2
Version number for the exit context structure.

MQAXC_CURRENT_VERSION
Current version number for the exit context structure.

The exit handler sets this field on entry to each exit function.

Environment (MQLONG) - input
The environment from which an IBM MQ API call was issued that resulted in an exit function being driven. Valid values for this field are:

MQXE_OTHER
This value is consistent with invocations an API exit sees if the exit is called from a server application. This means that an API exit runs unchanged on a client and does not see anything different.

If the exit really needs to determine whether it is running on the client, the exit can do so by looking at the *ChannelName* and *ChannelDefinition* fields.

MQXE_MCA
Message channel agent

MQXE_MCA_SVRCONN
A message channel agent acting on behalf of a client

MQXE_COMMAND_SERVER
The command server

MQXE_MQSC
The runmqsc command interpreter

The exit handler sets this field on entry to each exit function.

UserId (MQCHAR12) - input
The user ID associated with the application. In particular, in the case of client connections, this field contains the user ID of the adopted user as opposed to the user ID under which the channel code is running. If a blank user ID flows from the client, then no change is made to the user ID already being used. That is, no new user ID is adopted.

The exit handler sets this field on entry to each exit function. The length of this field is given by MQ_USER_ID_LENGTH.

In the case of a client, this is the user ID sent from the client to the server. Note, that this might not be the effective user ID the client is running against in the queue manager, as there could be an MCAUser or CHLAUTH configuration which changes the user ID.

SecurityId (MQBYTE40) - input
An extension to the user ID running the application. Its length is given by MQ_SECURITY_ID_LENGTH.

In the case of a client, this is the user ID sent from the client to the server. Note, that this might not be the effective user ID the client is running against in the queue manager, as there could be an MCAUser or CHLAUTH configuration which changes the user ID.

ConnectionName (MQCHAR264) - input
The connection name field, set to the address of the client. For example, for TCP/IP, it would be the client IP address.

The length of this field is given by MQ_CONN_NAME_LENGTH.

In the case of a client, this is the partner address of the queue manager.

LongMCAUserIdLength (MQLONG) - input
The length of the long MCA user identifier.

When MCA connects to the queue manager this field is set to the length of the long MCA user identifier (or zero if there is no such identifier).

In the case of a client, this is the client long user identifier.

LongRemoteUserIdLength (MQLONG) - input
The length of the long remote user identifier.

When MCA connects to the queue manager this field is set to the length of the long remote user identifier. Otherwise this field will be set to zero

In the case of a client, set this field to zero.

LongMCAUserIdPtr (MQPTR) - input
Address of long MCA user identifier.

When MCA connects to the queue manager this field is set to the address of the long MCA user identifier (or to a null pointer if there is no such identifier).

In the case of a client, this is the client long user identifier.

LongRemoteUserIdPtr (MQPTR) - input
The address of the long remote user identifier.

When MCA connects to the queue manager this field is set to the address of the long remote user identifier (or to a null pointer if there is no such identifier).

In the case of a client, set this field to zero.

ApplName (MQCHAR28) - input
The name of the application or component that issued the IBM MQ API call.

The rules for generating the ApplName are the same as for generating the default name for an MQPUT.

The value of this field is found by querying the operating system for the program name. Its length is given by MQ_APPL_NAME_LENGTH.

ApplType (MQLONG) - input
The type of application or component that issued the IBM MQ API call.

The value is MQAT_DEFAULT for the platform on which the application is compiled, or it equates to one of the defined MQAT_* values.

The exit handler sets this field on entry to each exit function.

ProcessId (MQPID) - input
The operating system process identifier.

Where applicable, the exit handler sets this field on entry to each exit function.

ThreadId (MQTID) - input
The MQ thread identifier. This is the same identifier used in MQ trace and FFST dumps, but might be different from the operating system thread identifier.

Where applicable, the exit handler sets this field on entry to each exit function.

ChannelName (MQCHAR) - input
The name of the channel, padded with blanks, if applicable and known.

If not applicable, this field is set to NULL characters.

Reserved1 (MQBYTE4) - input
This field is reserved.

ChanneDefinition (PMQCD) - input
A pointer to the channel definition being used, if applicable and known.

If not applicable, this field is set to NULL characters.

Note that the pointer is only completed if the connection is processing on behalf of an IBM MQ channel and that channel definition has been read.

In particular, the channel definition is not given on the server when the first MQCONN call is made for the channel. Furthermore, if the pointer is filled, the structure (and any sub structures) pointed to by the pointer must be treated as read only; any updating of the structure would lead to unpredictable results and is not supported.

In the case of a client, fields other than those with a value specified for a client, contain values that are appropriate for a client application.

The exit chain area and exit chain area header (MQACH):

If required, an exit function can acquire storage for an exit chain area and set the ExitChainAreaPtr in MQAXP to point to this storage.

Exits (either the same or different exit functions) can acquire multiple exit chain areas and link them together. Exit chain areas must only be added or removed from this list while called from the exit handler. This ensures that there are no serialization issues caused by different threads adding or removing areas from the list at the same time.

An exit chain area must start with an MQACH header structure, the C declaration for which is:

```
typedef struct tagMQACH {
   MQCHAR4   StrucId;             /* Structure identifier */
   MQLONG    Version;             /* Structure version number */
   MQLONG    StrucLength;         /* Length of the MQACH structure */
   MQLONG    ChainAreaLength;     /* Exit chain area length */
   MQCHAR48  ExitInfoName         /* Exit information name */
   PMQACH    NextChainAreaPtr;    /* Pointer to next exit chain area */
 };
```

The fields in the exit chain area header are:

StrucId (MQCHAR4) - input
The exit chain area structure identifier, with an initial value, defined by MQACH_DEFAULT, of MQACH_STRUC_ID.

For C programs, the constant MQACH_STRUC_ID_ARRAY is also defined; this has the same value as MQACH_STRUC_ID, but as an array of characters instead of a string.

Version (MQLONG) - input
The structure version number, as follows:

MQACH_VERSION_1
The version number for the exit parameter structure.

MQACH_CURRENT_VERSION
The current version number for the exit context structure.

The initial value of this field, defined by MQACH_DEFAULT, is `MQACH_CURRENT_VERSION`.

Note: If you introduce a new version of this structure, the layout of the existing part does not change. Exit functions must check that the version number is equal to or greater than the lowest version containing the fields that the exit function needs to use.

StrucLength (MQLONG) - input
The length of the MQACH structure. Exits can use this field to determine the start of the exit data, setting it to the length of the structure created by the exit.

The initial value of this field, defined by MQACH_DEFAULT, is MQACH_CURRENT_LENGTH.

ChainAreaLength (MQLONG) - input
The exit chain area length, set to the overall length of the current exit chain area, including the MQACH header.

The initial value of this field, defined by MQACH_DEFAULT, is zero.

ExitInfoName (MQCHAR48) - input
The exit information name.

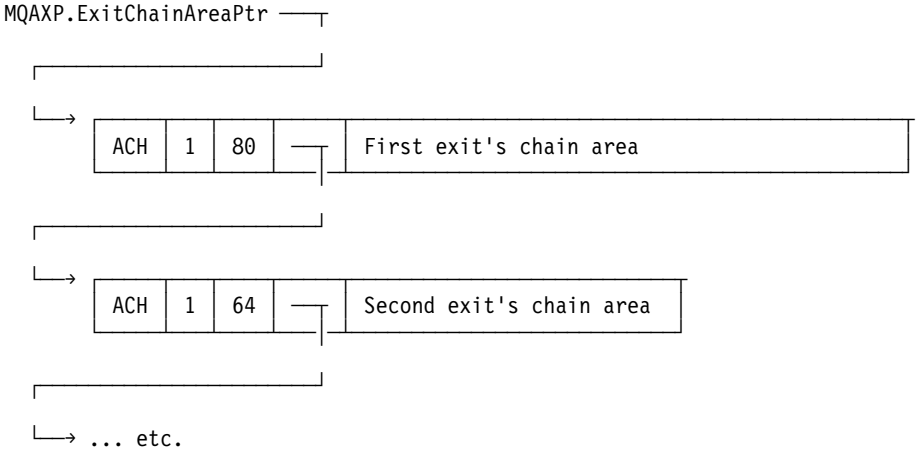
When an exit creates an MQACH structure, it must initialize this field with its own ExitInfoName, so that later this MQACH structure can be found by either another instance of this exit, or by a cooperating exit.

The initial value of this field, defined by MQACH_DEFAULT, is a zero length string ({""}).

NextChainAreaPtr (PMQACH) - input
A pointer to the next exit chain area with an initial value, defined by MQACH_DEFAULT, of null pointer (NULL).

Exit functions must release the storage for any exit chain areas that they acquire, and manipulate the chain pointers to remove their exit chain areas from the list.

An exit chain area can be constructed as follows:



External constants:

Use this topic as reference information for external constants available for API exists.

The following external constants are available for API exits:

MQXF_* (exit function identifiers)

```
MQXF_INIT                   1       X'00000001'
MQXF_TERM                   2       X'00000002'
MQXF_CONN                   3       X'00000003'
MQXF_CONNX                  4       X'00000004'
MQXF_DISC                   5       X'00000005'
MQXF_OPEN                   6       X'00000006'
MQXF_CLOSE                  7       X'00000007'
MQXF_PUT1                   8       X'00000008'
MQXF_PUT                    9       X'00000009'
MQXF_GET                    10      X'0000000A'
MQXF_DATA_CONV_ON_GET       11      X'0000000B'
MQXF_INQ                    12      X'0000000C'
MQXF_SET                    13      X'0000000D'
MQXF_BEGIN                  14      X'0000000E'
MQXF_CMIT                   15      X'0000000F'
MQXF_BACK                   16      X'00000010'
MQXF_STAT                   18      X'00000012'
MQXF_CB                     19      X'00000013'
MQXF_CTL                    20      X'00000014'
MQXF_CALLBACK               21      X'00000015'
MQXF_SUB                    22      X'00000016'
MQXF_SUBRQ                  23      X'00000017'
MQXF_XACLOSE                24      X'00000018'
MQXF_XACOMMIT               25      X'00000019'
MQXF_XACOMPLETE             26      X'0000001A'
MQXF_XAEND                  27      X'0000001B'
MQXF_XAFORGET               28      X'0000001C'
MQXF_XAOPEN                 29      X'0000001D'
MQXF_XAPREPARE              30      X'0000001E'
MQXF_XARECOVER              31      X'0000001F'
MQXF_XAROLLBACK             32      X'00000020'
MQXF_XASTART                33      X'00000021'
MQXF_AXREG                  34      X'00000022'
MQXF_AXUNREG                35      X'00000023'
```

MQXR_* (exit reasons)

```
MQXR_BEFORE                 1       X'00000001'
MQXR_AFTER                  2       X'00000002'
MQXR_CONNECTION             3       X'00000003'
```

MQXE_* (environments)

```
MQXE_OTHER                  0       X'00000000'
MQXE_MCA                    1       X'00000001'
MQXE_MCA_SVRCONN            2       X'00000002'
MQXE_COMMAND_SERVER         3       X'00000003'
MQXE_MQSC                   4       X'00000004'
```

MQ*_* (additional constants)

```
MQAXP_VERSION_1             1
MQAXP_VERSION_2             2
MQAXC_VERSION_1             1
MQACH_VERSION_1             1

MQAXP_CURRENT_VERSION       1
MQAXC_CURRENT_VERSION       1
MQACH_CURRENT_VERSION       1

MQXACT_EXTERNAL             1
MQXACT_INTERNAL             2
```

```
MQXT_API_EXIT                   2

MQACH_LENGTH_1                  68 (32-bit platforms)
                                72 (64-bit platforms)
                                80 (128-bit platforms)

MQACH_CURRENT_LENGTH            68 (32-bit platforms)
                                72 (64-bit platforms)
                                80 (128-bit platforms)
```

MQ*_* (null constants)

```
MQXPDA_NONE                     X'00...00' (48 nulls)
MQXPDA_NONE_ARRAY               '\0','\0',...,'\0','\0'
```

MQXCC_* (completion codes)

```
MQXCC_FAILED                    -8
```

MQRC_* (reason codes)

MQRC_API_EXIT_ERROR 2374 X'00000946'
An exit function invocation has returned an invalid response code, or has failed in some way, and the queue manager cannot determine the next action to take.

Examine both the ExitResponse and ExitResponse2 fields of the MQAXP to determine the bad response code, and change the exit to return a valid response code.

MQRC_API_EXIT_INIT_ERROR 2375 X'00000947'
The queue manager encountered an error while initializing the execution environment for an API exit function.

MQRC_API_EXIT_TERM_ERROR 2376 X'00000948'
The queue manager encountered an error while closing the execution environment for an API exit function.

MQRC_EXIT_REASON_ERROR 2377 X'00000949'
The value of the ExitReason field supplied on an exit entry point registration call (MQXEP) call is in error.

Examine the value of the ExitReason field to determine and correct the bad exit reason value.

MQRC_RESERVED_VALUE_ERROR 2378 X'0000094A'
The value of the Reserved field is in error.

Examine the value of the Reserved field to determine and correct the Reserved value.

C language typedefs:

This topic provides information about typedefs associated with API exits available in C language.

Here are the C language typedefs associated with the API exits:

```
typedef PMQLONG  MQPOINTER PPMQLONG;
typedef PMQBYTE  MQPOINTER PPMQBYTE;
typedef PMQHOBJ  MQPOINTER PPMQHOBJ;
typedef PMQOD    MQPOINTER PPMQOD;
typedef PMQMD    MQPOINTER PPMQMD;
typedef PMQPMO   MQPOINTER PPMQPMO;
typedef PMQGMO   MQPOINTER PPMQGMO;
typedef PMQCNO   MQPOINTER PPMQCNO;
typedef PMQBO    MQPOINTER PPMQBO;

typedef MQAXP    MQPOINTER PMQAXP;
typedef MQACH    MQPOINTER PMQACH;
typedef MQAXC    MQPOINTER PMQAXC;
```

```
typedef MQCHAR   MQCHAR16[16];
typedef MQCHAR16 MQPOINTER PMQCHAR16;

typedef MQLONG   MQPID;
typedef MQLONG   MQTID;
```

The exit entry point registration call (MQXEP):

Use this information to learn about MQXEP, MQXEP C language invocation, and MQXEP C function prototype.

Use the MQXEP call to:

1. Register the before and after IBM MQ API exit invocation points at which to invoke exit functions
2. Specify the exit function entry points
3. Deregister the exit function entry points

You would typically code the MQXEP calls in the MQ_INIT_EXIT exit function, but you can specify them in any subsequent exit function.

If you use an MQXEP call to register an already registered exit function, the second MQXEP call completes successfully, replacing the registered exit function.

If you use an MQXEP call to register a NULL exit function, the MQXEP call completes successfully and the exit function is deregistered.

If MQXEP calls are used to register, deregister, and reregister a particular exit function during the life of a connection request, the previously registered exit function is reactivated. Any storage still allocated and associated with this exit function instance is available for use by the exit's functions. (This storage is typically released during the invocation of the termination exit function.)

The interface to MQXEP is:

```
MQXEP (Hconfig, ExitReason, Function, EntryPoint, &ExitOpts, &CompCode, &Reason)
```

where:

Hconfig (MQHCONFIG) - input
The configuration handle, representing the API exit that includes the set of functions being initialized. This value is generated by the queue manager immediately before invoking the MQ_INIT_EXIT function, and is passed in the MQAXP to each API exit function.

ExitReason (MQLONG) - input
The reason for which the entry point is being registered, from the following reasons:

- Connection level initialization or termination (MQXR_CONNECTION)
- Before an IBM MQ API call (MQXR_BEFORE)
- After an IBM MQ API call (MQXR_AFTER)

Function (MQLONG) - input
The function identifier, valid values for which are the MQXF_* constants (see “External constants” on page 3662).

EntryPoint (PMQFUNC) - input
The address of the entry point for the exit function to be registered. The value NULL indicates either that the exit function has not been provided, or that a previous registration of the exit function is being deregistered.

ExitOpts(MQXEPO)
API exits can specify options that control how API exits are registered. If a null pointer is specified for this field, the default values of the MQXEPO structure are assumed.

CompCode (MQLONG) - output
The completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason (MQLONG) - output
The reason code that qualifies the completion code.

If the completion code is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If the completion code is MQCC_FAILED:

MQRC_HCONFIG_ERROR
(2280, X'8E8') The supplied configuration handle is not valid. Use the configuration handle from the MQAXP.

MQRC_EXIT_REASON_ERROR
(2377, X'949') The supplied exit function invocation reason is either not valid or is not valid for the supplied exit function identifier.

Either use one of the valid exit function invocation reasons (MQXR_* value), or use a valid function identifier and exit reason combination. (See Table 398.)

MQRC_FUNCTION_ERROR
(2281, X'8E9') The supplied function identifier is not valid for API exit reason. The following table shows valid combinations of function identifiers and ExitReasons.

Table 398. Valid combinations of function identifiers and ExitReasons

Function	ExitReason
MQXF_INIT MQXF_TERM	MQXR_CONNECTION
MQXF_CONN MQXF_CONNX MQXF_DISC MQXF_OPEN MQXF_CLOSE MQXF_PUT1 MQXF_PUT MQXF_GET MQXF_INQ MQXF_SET MQXF_BEGIN MQXF_CMIT MQXF_BACK MQXF_STAT MQXF_CB MQXF_CTL MQXF_CALLBACK MQXF_SUB MQXF_SUBRQ	MQXR_BEFORE MQXR_AFTER
MQXF_DATA_CONV_ON_GET	MQXR_BEFORE

MQRC_RESOURCE_PROBLEM
(2102, X'836') An attempt to register or deregister an exit function has failed because of a resource problem.

MQRC_UNEXPECTED_ERROR
(2195, X'893') An attempt to register or deregister an exit function has failed unexpectedly.

MQRC_PROPERTY_NAME_ERROR
(2442, X'098A') Invalid ExitProperties name.

MQRC_XEPO_ERROR
(2507, X'09CB') Exit options structure not valid.

MQXEP C language invocation

```
MQXEP (Hconfig, ExitReason, Function, EntryPoint, &ExitOpts, &CompCode, &Reason);
```

Declaration for parameter list:

```
MQHCONFIG       Hconfig;        /* Configuration handle */
MQLONG          ExitReason;     /* Exit reason */
MQLONG          Function;       /* Function identifier */
PMQFUNC         EntryPoint;     /* Function entry point */
MQXEPO          ExitOpts;       /* Options that control the action of MQXEP */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code qualifying completion
                                   code */
```

MQXEP C function prototype

```
void MQXEP (
MQHCONFIG    Hconfig,        /* Configuration handle */
MQLONG       ExitReason,     /* Exit reason */
MQLONG       Function,       /* Function identifier */
PMQFUNC      EntryPoint,     /* Function entry point */
PMQXEPO      pExitOpts;      /* Options that control the action of MQXEP */
PMQLONG      pCompCode,      /* Address of completion code */
PMQLONG      pReason);       /* Address of reason code qualifying completion
                                code */
```

Exit functions:

This section describes how to invoke the exit functions available.

The descriptions of the individual functions start at “General rules for API exit routines” on page 3667. This topic begins with some general information to help you when using these function calls.

General rules for API exit routines:

Use this information to understand the general rules for API exit routines, and setting up and cleaning up the exit execution environment.

The following general rules apply when invoking API exit routines:

- In all cases, API exit functions are driven before validating API call parameters, and before any security checks (in the case of MQCONN, MQCONNX, or MQOPEN).
- The values of fields entered into and output from an exit routine are:
  - On input to a *before* IBM MQ API exit function, the value of a field can be set by the application program, or by a previous exit function invocation.
  - On output from a *before* IBM MQ API exit function, the value of a field can be left unchanged, or set to some other value by the exit function.
  - On input to an *after* IBM MQ API exit function, the value of a field can be the value set by the queue manager after processing the IBM MQ API call, or can be set to a value by a previous exit function invocation in the chain of exit functions.
  - On output from an *after* IBM MQ API call exit function, the value of a field can be left unchanged, or set to some other value by the exit function.
- Exit functions must communicate with the queue manager by using the ExitResponse and ExitResponse2 fields.
- The CompCode and Reason code fields communicate back to the application. The queue manager and exit functions can set the CompCode and Reason code fields.
- The MQXEP call returns new reason codes to the exit functions that call MQXEP. However, exit functions can translate these new reason codes to any existing reasons codes that existing and new applications can understand.
- Each exit function prototype has similar parameters to the API function with an extra level of indirection except for the CompCode and Reason.
- API exits can issue MQI calls (except MQDISC), but these MQI calls do not themselves invoke API exits.

Note, that whether the application is on a server or a client, you cannot predict the sequencing of the API exit calls. An API exit `BEFORE` call might not be followed immediately by an `AFTER` call.

The `BEFORE` call can be followed by another `BEFORE` call. For example:

```
BEFORE MQCTL
BEFORE Callback
BEFORE MQPUT
AFTER MQPUT
AFTER Callback
AFTER MQCTL
```

or

```
BEFORE XAOPEN
BEFORE MQCONNX
AFTER MQCONNX
AFTER XAOPEN
```

On the client, there is an exit that can modify the behavior of the MQCONN or MQCONNX call, called the `PreConnect` exit. The `PreConnect` exit can modify any of the parameters on the MQCONN or

MQCONNX call including the queue manager name. The client calls this exit first and then invokes the MQCONN or MQCONNX call. Note that only the initial MQCONN or MQCONNX call invokes the API exit; any subsequent reconnect calls have no effect.

The execution environment

In general, all errors from exit functions are communicated back to the exit handler using the ExitResponse and ExitResponse2 fields in MQAXP.

These errors in turn are converted into MQCC_* and MQRC_* values and communicated back to the application in the CompCode and Reason fields. However, any errors encountered in the exit handler logic are communicated back to the application as MQCC_* and MQRC_* values in the CompCode and Reason fields.

If an MQ_TERM_EXIT function returns an error:

- The MQDISC call has already taken place
- There is no other opportunity to drive the *after* MQ_TERM_EXIT exit function (and thus perform exit execution environment cleanup)
- Exit execution environment cleanup is not performed

The exit cannot be unloaded as it might still be in use. Also, other registered exits further down in the exit chain for which the *before* exit was successful, will be driven in the reverse order.

Setting up the exit execution environment

While processing an explicit MQCONN or MQCONNX call, exit handling logic sets up the exit execution environment before invoking the exit initialization function (MQ_INIT_EXIT). Exit execution environment setup involves loading the exit, acquiring storage for, and initializing exit parameter structures. The exit configuration handle is also allocated.

If errors occur during this phase, the MQCONN or MQCONNX call fails with CompCode MQCC_FAILED and one of the following reason codes:

MQRC_API_EXIT_LOAD_ERROR
An attempt to load an API exit module has failed.

MQRC_API_EXIT_NOT_FOUND
An API exit function could not be found in the API exit module.

MQRC_STORAGE_NOT_AVAILABLE
An attempt to initialize the execution environment for an API exit function failed because insufficient storage was available.

MQRC_API_EXIT_INIT_ERROR
An error was encountered while initializing the execution environment for an API exit function.

Cleaning up the exit execution environment

While processing an explicit MQDISC call, or an implicit disconnect request as a result of an application ending, exit handling logic might need to clean up the exit execution environment after invoking the exit termination function (MQ_TERM_EXIT), if registered.

Cleaning up the exit execution environment involves releasing storage for exit parameter structures, possibly deleting any modules previously loaded into memory.

If errors occur during this phase, an explicit MQDISC call fails with CompCode MQCC_FAILED and the following reason code (errors are not highlighted on implicit disconnect requests):

MQRC_API_EXIT_TERM_ERROR
An error was encountered while closing the execution environment for an API exit function. The exit should not return any failure from the MQDISC before or after the MQ_TERM* API exit function calls.

API exits on clients:

A client uses the PreConnect exit to modify the behavior of the MQCONN and MQCONNX calls and does not support API exit properties.

PreConnect exit

On a client, the PreConnect exit can be used to look up the channel definition from a central repository, such as an LDAP server.

The PreConnect exit can also modify any parameter, or all the parameters, on an MQCONN or MQCONNX call itself, for example, the queue manager name.

In the case of client applications, the PreConnect exit must be called before the API exit because the MQCONN or MQCONNX API exit is called only once the name of the queue manager is known and this name can be changed by the PreConnect exit.

Note that only the initial MQCONN or MQCONNX call invokes the exit.

API exit properties

On a server, API exits can register an MQXEPO structure at initialization time. The MQXEPO structure contains the ExitProperties field which details the group of properties the exit is interested in. This has the effect of generating a separate message property handle which the exit can manipulate separately from any application message property handle.

On a client, API exit properties are not supported. If an attempt is made to register a property group name on a client, the function fails with a reason code of MQRC_EXIT_PROPS_NOT_SUPPORTED.

Backout - MQ_BACK_EXIT:

MQ_BACK_EXIT provides a backout exit function to perform *before* and *after* backout processing. Use function identifier MQXF_BACK with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* backout call exit functions.

The interface to this function is:

```
MQ_BACK_EXIT (&ExitParms, &ExitContext, &Hconn, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;      /* Exit parameter structure */
MQAXC    ExitContext;    /* Exit context structure */
MQHCONN  Hconn;          /* Connection handle */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying completion code */
```

The queue manager then logically calls the exit as follows:

```
MQ_BACK_EXIT (&ExitParms, &ExitContext, &Hconn, &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_BACK_EXIT (
PMQAXP    pExitParms,       /* Address of exit parameter structure */
PMQAXC    pExitContext,     /* Address of exit context structure */
PMQHCONN  pHconn,           /* Address of connection handle */
PMQLONG   pCompCode,        /* Address of completion code */
PMQLONG   pReason);         /* Address of reason code qualifying completion
                               code */
```

Begin - MQ_BEGIN_EXIT:

MQ_BEGIN_EXIT provides a begin exit function to perform *before* and *after* MQBEGIN call processing. Use function identifier MQXF_BEGIN with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQBEGIN call exit functions.

The interface to this function is:

```
MQ_BEGIN_EXIT (&ExitParms, &ExitContext, &Hconn, &pBeginOptions, &CompCode,
               &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pBeginOptions (PMQBO)- input/output
Pointer to begin options.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;      /* Exit parameter structure */
MQAXC    ExitContext;    /* Exit context structure */
MQHCONN  Hconn;          /* Connection handle */
PMQBO    pBeginOptions;  /* Ptr to begin options */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying completion code */
```

The queue manager then logically calls the exit as follows:

```
MQ_BEGIN_EXIT (&ExitParms, &ExitContext, &Hconn, &pBeginOptions, &CompCode,
               &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_BEGIN_EXIT (
PMQAXP    pExitParms,       /* Address of exit parameter structure */
PMQAXC    pExitContext,     /* Address of exit context structure */
PMQHCONN  pHconn,           /* Address of connection handle */
PPMQBO    ppBeginOptions,   /* Address of ptr to begin options */
PMQLONG   pCompCode,        /* Address of completion code */
PMQLONG   pReason);         /* Address of reason code qualifying completion
                               code */
```

Callback - MQ_CALLBACK_EXIT:

MQ_CALLBACK_EXIT provides an exit function to perform *before* and *after* callback processing. Use function identifier MQXF_CALLBACK with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* callback call exit functions.

The interface to this function is:

```
MQ_CALLBACK_EXIT (&ExitParms, &ExitContext, &Hconn, &pMsgDesc, &pGetMsgOpts,
                  &pBuffer, &pMQCBContext)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure

ExitContext (MQAXC) - input/output
Exit context structure

Hconn (MQHCONN) - input/output
Connection handle

pMsgDesc
Message descriptor

pGetMsgOpts
Options that control the action of MQGET

pBuffer
Area to contain the message data

pMQCBContext
Context data for the callback

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;      /* Exit parameter structure */
MQAXC    ExitContext;    /* Exit context structure */
MQHCONN  Hconn;          /* Connection handle */
PMQMD    pMsgDesc;       /* Message descriptor */
PMQGMO   pGetMsgOpts;    /* Options that define the operation of the consumer */
PMQVOID  pBuffer;        /* Area to contain the message data */
PMQCBC   pContext;       /* Context data for the callback */
```

The queue manager then logically calls the exit as follows:

```
MQ_SUBRQ_EXIT (&ExitParms, &ExitContext, &Hconn, &pMsgDesc, &pGetMsgOpts, &pBuffer,
               &pContext);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_CALLBACK_EXIT (
PMQAXP    pExitParms;    /* Exit parameter structure */
PMQAXC    pExitContext;  /* Exit context structure */
PMQHCONN  pHconn;        /* Connection handle */
PPMQMD    ppMsgDesc;     /* Message descriptor */
PPMQGMO   ppGetMsgOpts;  /* Options that define the operation of the consumer */
PPMQVOID  ppBuffer;      /* Area to contain the message data */
PPMQCBC   ppContext;)    /* Context data for the callback */
```

Usage notes

1. The Callback exit is invoked before the consumer is invoked and after the consumer's consumer function has completed. Although the MQMD and MQGMO structures are alterable, changing the

values in the before exit does not redrive the retrieval of a message from the queue as the message has already been removed from the queue to be delivered to the consumer function

Manage callback functions - MQ_CB_EXIT:

MQ_CB_EXIT provides an exit function to perform *before* and *after* the MQCB call. Use function identifier MQXF_CB with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQCB call exit functions.

The interface to this function is:

```
MQ_CB_EXIT (&ExitParms, &ExitContext, &Hconn, &Operation, &pCallbackDesc,
            &Hobj, &pMsgDesc, &pGetMsgOpts, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure

ExitContext (MQAXC) - input/output
Exit context structure

Hconn (MQHCONN) - input/output
Connection handle

Operation (MQLONG) - input/output
Operation value

pCallbackDesc (PMQCBD) - input/output
Callback descriptor

Hobj (MQHOBJ) - input/output
Object handle

pMsgDesc (PMQMD) - input/output
Message descriptor

pGetMsgOpts (PMQGMO) - input/output
Options that control the action of MQCB

CompCode (MQLONG) - input/output
Completion code

Reason (MQLONG) - input/output
Reason code qualifying CompCode

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;      /* Exit parameter structure */
MQAXC    ExitContext;    /* Exit context structure */
MQHCONN  Hconn;          /* Connection handle */
MQLONG   Operation;      /* Operation value. */
MQCBD    pMsgDesc;       /* Callback descriptor. */
MQHOBJ   Hobj;           /* Object handle. */
PMQMD    pMsgDesc;       /* Message descriptor */
PMQGMO   pGetMsgOpts;    /* Options that define the operation of the consumer */
PMQLONG  CompCode;       /* Completion code. */
PMQLONG  Reason;         /* Reason code qualifying CompCode. */
```

The queue manager then logically calls the exit as follows:

```
MQ_CB_EXIT (&ExitParms, &ExitContext, &Hconn, &Operation, &Hobj, &pMsgDesc,
            &pGetMsgOpts, &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_CB_EXIT (
PMQAXP     pExitParms;     /* Exit parameter structure */
PMQAXC     pExitContext;   /* Exit context structure */
PMQHCONN   pHconn;         /* Connection handle */
PMQLONG    pOperation;     /* Callback operation */
PMQHOBJ    pHobj;          /* Object handle */
PPMQMD     ppMsgDesc;      /* Message descriptor */
PPMQGMO    ppGetMsgOpts;   /* Options that control the action of MQCB */
PMQLONG    pCompCode;      /* Completion code */
PMQLONG    pReason;        /* Reason code qualifying CompCode */
```

Close - MQ_CLOSE_EXIT:

MQ_CLOSE_EXIT provides a close exit function to perform *before* and *after* MQCLOSE call processing. Use function identifier MQXF_CLOSE with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQCLOSE call exit functions.

The interface to this function is:

```
MQ_CLOSE_EXIT (&ExitParms, &ExitContext, &Hconn, &pHobj,
               &Options, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pHobj (PMQHOBJ) - input
Pointer to object handle.

Options (MQLONG)- input/output
Close options.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP           ExitParms;      /* Exit parameter structure */
MQAXC           ExitContext;    /* Exit context structure */
MQHCONN         Hconn;          /* Connection handle */
PMQHOBJ         pHobj;          /* Ptr to object handle */
MQLONG          Options;        /* Close options */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code */
```

The queue manager then logically calls the exit as follows:

```
MQ_CLOSE_EXIT (&ExitParms, &ExitContext,&Hconn, &pHobj, &Options,
               &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_CLOSE_EXIT (
PMQAXP          pExitParms,     /* Address of exit parameter structure */
PMQAXC          pExitContext,   /* Address of exit context structure */
PMQHCONN        pHconn,         /* Address of connection handle */
PPMQHOBJ        ppHobj,         /* Address of ptr to object handle */
PMQLONG         pOptions,       /* Address of close options */
PMQLONG         pCompCode,      /* Address of completion code */
PMQLONG         pReason);       /* Address of reason code qualifying
                                   completion code */
```

Commit - MQ_CMIT_EXIT:

MQ_CMIT_EXIT provides a commit exit function to perform *before* and *after* commit processing. Use function identifier MQXF_CMIT with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* commit call exit functions.

If a commit operation fails, and the transaction is backed out, the MQCMIT call fails with MQCC_WARNING and MQRC_BACKED_OUT. These return and reason codes are passed into any *after* MQCMIT exit functions to give the exit functions an indication that the unit of work has been backed out.

The interface to this function is:

```
MQ_CMIT_EXIT (&ExitParms, &ExitContext, &Hconn, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;      /* Exit parameter structure */
MQAXC    ExitContext;    /* Exit context structure */
MQHCONN  Hconn;          /* Connection handle */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying completion code */
```

The queue manager then logically calls the exit as follows:

```
MQ_CMIT_EXIT (&ExitParms, &ExitContext,&Hconn, &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_CMIT_EXIT (
PMQAXP    pExitParms,       /* Address of exit parameter structure */
PMQAXC    pExitContext,     /* Address of exit context structure */
PMQHCONN  pHconn,           /* Address of connection handle */
PMQLONG   pCompCode,        /* Address of completion code */
PMQLONG   pReason);         /* Address of reason code qualifying completion
                               code */
```

Usage notes

1. The MQ_GET_EXIT function interface described here is used for both the MQXF_GET exit function and the "MQXF_DATA_CONV_ON_GET" on page 3683 exit function.

   Separate entry points are defined for these two exit functions, so to intercept *both* the MQXEP call must be used twice; for this call use function identifier MQXF_GET.

   Because the MQ_GET_EXIT interface is the same for MQXF_GET and MQXF_DATA_CONV_ON_GET, a single exit function can be used for both; the *Function* field in the MQAXP structure indicates which exit function has been invoked. Alternatively, the MQXEP call can be used to register different exit functions for the two cases.

Connect and connect extension - MQ_CONNX_EXIT:

MQ_CONNX_EXIT provides:

- Connection exit function to perform *before* and *after* MQCONN processing
- Connection extension exit function to perform *before* and *after* MQCONNX processing

The same interface, described here, is invoked for both MQCONN and MQCONNX call exit functions.

When the message channel agent (MCA) responds to an inbound client connection, the MCA can connect and make a number of IBM MQ API calls before the client state is fully known. These API calls call the API exit functions with the MQAXC based on the MCA program itself (for example in the UserId and ConnectionName fields of the MQAXC).

When the MCA responds to subsequent inbound client API calls, the MQAXC structure is based on the inbound client, setting the UserId and ConnectionName fields appropriately.

The queue manager name set by the application on an MQCONN or MQCONNX call is passed to the underlying connect call. Any attempt by a *before* MQ_CONNX_EXIT to change the name of the queue manager has no effect.

Use function identifiers MQXF_CONN and MQXF_CONNX with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQCONN and MQCONNX call exit functions.

An MQ_CONNX_EXIT exit called for reason MQXR_BEFORE *must not* issue any IBM MQ API calls, as the correct environment has not been set up at this time.

An MQ_CONNX_EXIT cannot call MQDISC from an API exit call for the connection for which it is being called. This restriction is applicable to both client and server API exits.

The interface to MQCONN and MQCONNX is identical:

```
MQ_CONNX_EXIT (&ExitParms, &ExitContext, &pQMgrName, &pConnectOpts,
&pHconn, &CompCode, &Reason);
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

pQMgrName (PMQCHAR) - input
Pointer to the queue manager name supplied on the MQCONNX call. The exit must not change this name on the MQCONN or MQCONNX call.

pConnectOpts (PMQCNO) - input/output
Pointer to the options that control the action of the MQCONNX call.

See "MQCNO - Connect options" on page 2276 for details.

For exit function MQXF_CONN, pConnectOpts points to the default connect options structure (MQCNO_DEFAULT).

pHconn (PMQHCONN) - input
Pointer to the connection handle.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion)

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP           ExitParms;      /* Exit parameter structure */
MQAXC           ExitContext;    /* Exit context structure */
PMQCHAR         pQMgrName;      /* Ptr to Queue manager name */
PMQCNO          pConnectOpts;   /* Ptr to Connection options */
PMQHCONN        pHconn;         /* Ptr to Connection handle */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code */
```

The queue manager then logically calls the exit as follows:

```
MQ_CONNX_EXIT (&ExitParms, &ExitContext, &pQMgrName, &pConnectOps,
               &pHconn, &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_CONNX_EXIT (
PMQAXP          pExitParms,     /* Address of exit parameter structure */
PMQAXC          pExitContext,   /* Address of exit context structure */
PPMQCHAR        ppQMgrName,     /* Address of ptr to queue manager name */
PPMQCNO         ppConnectOpts,  /* Address of ptr to connection options */
PPMQHCONN       ppHconn,        /* Address of ptr to connection handle */
PMQLONG         pCompCode,      /* Address of completion code */
PMQLONG         pReason);       /* Address of reason code qualifying
                                   completion code */
```

Usage notes

1. The MQ_CONNX_EXIT function interface described here is used for both the MQCONN call and the MQCONNX call. However, separate entry points are defined for these two calls. To intercept *both* calls, the MQXEP call must be used at least twice - once with function identifier MQXF_CONN, and again with MQXF_CONNX.

   Because the MQ_CONNX_EXIT interface is the same for MQCONN and MQCONNX, a single exit function can be used for both calls; the *Function* field in the MQAXP structure indicates which call is in progress. Alternatively, the MQXEP call can be used to register different exit functions for the two calls.
2. When a message channel agent (MCA) responds to an inbound client connection, the MCA can issue a number of MQ calls before the client state is fully known. These MQ calls result in the API exit functions being invoked with the MQAXC structure containing data relating to the MCA, and not to the client (for example, user identifier and connection name). However, once the client state is fully known, subsequent MQ calls result in the API exit functions being invoked with the appropriate client data in the MQAXC structure.
3. All MQXR_BEFORE exit functions are invoked before any parameter validation is performed by the queue manager. The parameters might therefore be invalid (including invalid pointers for the addresses of parameters).

   The MQ_CONNX_EXIT function is invoked before any authorization checks are performed by the queue manager.
4. The exit function must not change the name of the queue manager specified on the MQCONN or MQCONNX call. If the name is changed by the exit function, the results are undefined.
5. An MQXR_BEFORE exit function for the MQ_CONNX_EXIT cannot issue MQ calls other than MQXEP.

Control callback - MQ_CTL_EXIT:

MQ_CTL_EXIT provides a subscription request exit function to perform *before* and *after* control callback processing. Use function identifier MQXF_CTL with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* control callback call exit functions.

The interface to this function is:

```
MQ_CTL_EXIT (&Hconn, &Operation, &ControlOpts, &CompCode, &Reason)
```

where the parameters are:

Hconn (MQHCONN) - input/output
Connection handle.

Operation (MQLONG) input/output
The operation being processed on the callback defined for the specified object handle

ControlOpts (MQCTLO) input/output
Options that control the action of MQCTL

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQHCONN  Hconn;          /* Connection handle */
MQLONG   Operation;      /* Operation being processed */
MQCTLO   ControlOpts;    /* Options that control the action of MQCTL  */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying completion code */
```

The queue manager then logically calls the exit as follows:

```
MQ_CTL_EXIT (&Hconn, &Operation, &ControlOpts, &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_CTL_EXIT (
PMQHCONN  pHconn;          /* Address of connection handle */
PMQLONG   pOperation;      /* Address of operation being processed */
PMQCTLO   pControlOpts;    /* Address of options that control the action of MQCTL  */
PMQLONG   pCompCode;       /* Address of completion code */
PMQLONG   pReason;)        /* Address of reason code qualifying completion code */
```

Disconnect - MQ_DISC_EXIT:

MQ_DISC_EXIT provides a disconnect exit function to perform *before* and *after* MQDISC exit processing. Use function identifier MQXF_DISC with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQDISC call exit functions.

The interface to this function is

```
MQ_DISC_EXIT (&ExitParms, &ExitContext, &pHconn,
&CompCode, &Reason);
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

pHconn (PMQHCONN) - input
Pointer to the connection handle.

For the before MQDISC call, the value of this field is one of:

- The connection handle returned on the MQCONN or MQCONNX call
- Zero, for environments where an environment-specific adapter has connected to the queue manager
- A value set by a previous exit function invocation

For the after MQDISC call, the value of this field is zero or a value set by a previous exit function invocation.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP           ExitParms;      /* Exit parameter structure */
MQAXC           ExitContext;    /* Exit context structure */
PMQHCONN        pHconn;         /* Ptr to Connection handle */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code */
```

The queue manager then logically calls the exit as follows:

```
MQ_DISC_EXIT (&ExitParms, &ExitContext, &pHconn,
              &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_DISC_EXIT (
PMQAXP          pExitParms,     /* Address of exit parameter structure */
PMQAXC          pExitContext,   /* Address of exit context structure */
PPMQHCONN       ppHconn,        /* Address of ptr to connection handle */
PMQLONG         pCompCode,      /* Address of completion code */
PMQLONG         pReason);       /* Address of reason code qualifying
                                   completion code */
```

Get - MQ_GET_EXIT:

MQ_GET_EXIT provides a get exit function to perform *before* and *after* MQGET call processing.

There are two function identifiers:

1. Use MQXF_GET with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQGET call exit functions.
2. See "MQXF_DATA_CONV_ON_GET" on page 3683 for information on using the MQXF_DATA_CONV_ON_GET function identifier.

The interface to this function is:

```
MQ_GET_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &pMsgDesc,
             &pGetMsgOpts, &BufferLength, &pBuffer, &pDataLength,
             &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

Hobj (MQHOBJ) - input/output
Object handle.

pMsgDesc (PMQMD) - input/output
Pointer to message descriptor.

pGetMsgOpts (PMQGMO) - input/output
Pointer to get message options.

BufferLength (MQLONG) - input/output
Message buffer length.

pBuffer (PMQBYTE) - input/output
Pointer to message buffer.

pDataLength (PMQLONG) - input/output
Pointer to data length field.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP           ExitParms;      /* Exit parameter structure */
MQAXC           ExitContext;    /* Exit context structure */
MQHCONN         Hconn;          /* Connection handle */
MQHOBJ          Hobj;           /* Object handle */
PMQMD           pMsgDesc;       /* Ptr to message descriptor */
PMQPMO          pGetMsgOpts;    /* Ptr to get message options */
MQLONG          BufferLength;   /* Message buffer length */
PMQBYTE         pBuffer;        /* Ptr to message buffer */
PMQLONG         pDataLength;    /* Ptr to data length field */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code */
```

The queue manager then logically calls the exit as follows:

```
MQ_GET_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &pMsgDesc,
             &pGetMsgOpts, &BufferLength, &pBuffer, &pDataLength,
             &CompCode, &Reason)
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_GET_EXIT (
PMQAXP          pExitParms,     /* Address of exit parameter structure */
PMQAXC          pExitContext,   /* Address of exit context structure */
PMQHCONN        pHconn,         /* Address of connection handle */
PMQHOBJ         pHobj,          /* Address of object handle */
PPMQMD          ppMsgDesc,      /* Address of ptr to message descriptor */
PPMQGMO         ppGetMsgOpts,   /* Address of ptr to get message options */
PMQLONG         pBufferLength,  /* Address of message buffer length */
PPMQBYTE        ppBuffer,       /* Address of ptr to message buffer */
PPMQLONG        ppDataLength,   /* Address of ptr to data length field */
PMQLONG         pCompCode,      /* Address of completion code */
PMQLONG         pReason);       /* Address of reason code qualifying
                                   completion code */
```

Usage notes

1. The MQ_GET_EXIT function interface described here is used for both the MQXF_GET exit function and the "MQXF_DATA_CONV_ON_GET" on page 3683 exit function.

   Separate entry points are defined for these two exit functions, so to intercept *both* the MQXEP call must be used twice; for this call use function identifier MQXF_GET.

   Because the MQ_GET_EXIT interface is the same for MQXF_GET and MQXF_DATA_CONV_ON_GET, a single exit function can be used for both; the *Function* field in the MQAXP structure indicates which exit function has been invoked. Alternatively, the MQXEP call can be used to register different exit functions for the two cases.

MQXF_DATA_CONV_ON_GET:
See MQ_GET_EXIT for information about the interface to this call, and a sample C language declaration.

Usage notes

If registered, this entry point is called when messages arrive at the application but before any data conversion has occurred. This can be useful if the API exit needs to perform processing, such as decryption or decompression, before the message is passed to data conversion. The exit can, if necessary, cause data conversion to be bypassed by returning MQXCC_SUPPRESS_FUNCTION; for more information, see MQAXP structure.

Registering for this entry point on a client has the effect of causing the data conversion to be performed locally on the client machine. For correct operation it might, therefore, be necessary to install the application conversion exits on the client. Note that MQXF_DATA_CONV_ON_GET is also used for asynchronous consume.

When using the MQ_GET_EXIT call, use MQXF_DATA_CONV_ON_GET, with exit reason MQXR_BEFORE, to register a *before* MQGET data conversion exit function.

There is no MQXR_AFTER exit function for MQXF_DATA_CONV_ON_GET; the MQXR_AFTER exit function for MQXF_GET provides the required capability for exit processing after data conversion.

Separate entry points are defined for the MQ_GET_EXIT call, so to intercept *both* exit functions, the MQXEP call must be used twice; for this call use function identifier MQXF_DATA_CONV_ON_GET.

Because the MQ_GET_EXIT interface is the same for MQXF_GET and MQXF_DATA_CONV_ON_GET, a single exit function can be used for both; the *Function* field in the MQAXP structure indicates which exit function has been invoked. Alternatively, the MQXEP call can be used to register different exit functions for the two cases.

Initialization - MQ_INIT_EXIT:

MQ_INIT_EXIT provides connection level initialization, indicated by setting ExitReason in MQAXP to MQXR_CONNECTION.

During the initialization, note the following:

- The MQ_INIT_EXIT function calls MQXEP to register the IBM MQ API verbs and the ENTRY and EXIT points in which it is interested.
- Exits do not need to intercept all the IBM MQ API verbs. Exit functions are invoked only if an interest has been registered.
- Storage that is to be used by the exit can be acquired while initializing it.
- If a call to this function fails, the MQCONN or MQCONNX call that invoked it also fails with a CompCode and Reason that depend on the value of the ExitResponse field in MQAXP.
- An MQ_INIT_EXIT exit must not issue IBM MQ API calls, because the correct environment has not been set up at this time.
- If an MQ_INIT_EXIT fails with MQXCC_FAILED, the queue manager returns from the MQCONN or MQCONNX call that called it with MQCC_FAILED and MQRC_API_EXIT_ERROR.
- If the queue manager encounters an error while initializing the API exit function execution environment before invoking the first MQ_INIT_EXIT, the queue manager returns from the MQCONN or MQCONNX call that invoked MQ_INIT_EXIT with MQCC_FAILED and MQRC_API_EXIT_INIT_ERROR.

The interface to MQ_INIT_EXIT is:

```
MQ_INIT_EXIT (&ExitParms, &ExitContext, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

CompCode (MQLONG) - input/output
Pointer to completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Pointer to reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

The CompCode and Reason returned to the application depend on the value of the ExitResponse field in MQAXP.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP           ExitParms;      /* Exit parameter structure */
MQAXC           ExitContext;    /* Exit context structure */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code */
```

The queue manager then logically calls the exit as follows:

```
MQ_INIT_EXIT (&ExitParms, &ExitContext, &CompCode, &Reason)
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_INIT_EXIT (
PMQAXP          pExitParms,     /* Address of exit parameter structure */
PMQAXC          pExitContext,   /* Address of exit context structure */
PMQLONG         pCompCode,      /* Address of completion code */
PMQLONG         pReason);       /* Address of reason code qualifying
                                   completion code */
```

Usage notes

1. The MQ_INIT_EXIT function can issue the MQXEP call to register the addresses of the exit functions for the particular MQ calls to be intercepted. It is not necessary to intercept all MQ calls, or to intercept both MQXR_BEFORE and MQXR_AFTER calls. For example, an exit suite could choose to intercept only the MQXR_BEFORE call of MQPUT.
2. Storage that is to be used by exit functions in the exit suite can be acquired by the MQ_INIT_EXIT function. Alternatively, exit functions can acquire storage when they are invoked, as and when

needed. However, all storage should be freed before the exit suite is terminated; the MQ_TERM_EXIT function can free the storage, or an exit function invoked earlier.

3. If MQ_INIT_EXIT returns MQXCC_FAILED in the `ExitResponse` field of MQAXP, or fails in some other way, the MQCONN or MQCONNX call that caused MQ_INIT_EXIT to be invoked also fails, with the **CompCode** and **Reason** parameters set to appropriate values.
4. An MQ_INIT_EXIT function cannot issue MQ calls other than MQXEP.

Inquire - MQ_INQ_EXIT:

MQ_INQ_EXIT provides an inquire exit function to perform *before* and *after* MQINQ call processing. Use function identifier MQXF_INQ with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQINQ call exit functions.

The interface to this function is:

```
MQ_INQ_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &SelectorCount,
             &pSelectors, &IntAttrCount, &pIntAttrs, &CharAttrLength,
             &pCharAttrs, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

Hobj (MQHOBJ) - input
Object handle.

SelectorCount (MQLONG) - input
Count of selectors

pSelectors (PMQLONG) - input/output
Pointer to array of selector values.

IntAttrCount (MQLONG) - input
Count of integer attributes.

pIntAttrs (PMQLONG) - input/output
Pointer to array of integer attribute values.

CharAttrLength (MQLONG) - input/output
Character attributes array length.

pCharAttrs (PMQCHAR) - input/output
Pointer to character attributes array.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output

Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE

(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;       /* Exit parameter structure */
MQAXC    ExitContext;     /* Exit context structure */
MQHCONN  Hconn;           /* Connection handle */
MQHOBJ   Hobj;            /* Object handle */
MQLONG   SelectorCount;   /* Count of selectors */
PMQLONG  pSelectors;      /* Ptr to array of attribute selectors */
MQLONG   IntAttrCount;    /* Count of integer attributes */
PMQLONG  pIntAttrs;       /* Ptr to array of integer attributes */
MQLONG   CharAttrLength;  /* Length of char attributes array */
PMQCHAR  pCharAttrs;      /* Ptr to character attributes */
MQLONG   CompCode;        /* Completion code */
MQLONG   Reason;          /* Reason code qualifying completion code */
```

The queue manager then logically calls the exit as follows:

```
MQ_INQ_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &SelectorCount,
             &pSelectors, &IntAttrCount, &pIntAttrs, &CharAttrLength,
             &pCharAttrs, &CompCode, &Reason)
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_INQ_EXIT (
PMQAXP    pExitParms,       /* Address of exit parameter structure */
PMQAXC    pExitContext,     /* Address of exit context structure */
PMQHCONN  pHconn,           /* Address of connection handle */
PMQHOBJ   pHobj,            /* Address of object handle */
PMQLONG   pSelectorCount,   /* Address of selector count */
PPMQLONG  ppSelectors,      /* Address of ptr to array of selectors */
PMQLONG   pIntAttrCount;    /* Address of count of integer attributes */
PPMQLONG  ppIntAttrs,       /* Address of ptr to array of integer attributes */
PMQLONG   pCharAttrLength,  /* Address of character attribute length */
PPMQCHAR  ppCharAttrs,      /* Address of ptr to character attributes array */
PMQLONG   pCompCode,        /* Address of completion code */
PMQLONG   pReason);         /* Address of reason code qualifying completion
                               code */
```

Open - MQ_OPEN_EXIT:

MQ_OPEN_EXIT provides an open exit function to perform *before* and *after* MQOPEN call processing. Use function identifier MQXF_OPEN with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQOPEN call exit functions.

The interface to this function is

```
MQ_OPEN_EXIT (&ExitParms, &ExitContext, &Hconn, &pObjDesc, &Options,
&pHobj, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pObjDesc (PMQOD) - input/output
Pointer to object descriptor.

Options (MQLONG) - input/output
Open options.

pHobj (PMQHOBJ) - input
Pointer to object handle.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP           ExitParms;      /* Exit parameter structure */
MQAXC           ExitContext;    /* Exit context structure */
MQHCONN         Hconn;          /* Connection handle */
PMQOD           pObjDesc;       /* Ptr to object descriptor */
MQLONG          Options;        /* Open options */
```

```
PMQHOBJ         pHobj;          /* Ptr to object handle */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code */
```

The queue manager then logically calls the exit as follows:

```
MQ_OPEN_EXIT (&ExitParms, &ExitContext, &Hconn, &pObjDesc, &Options,
              &pHobj, &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_OPEN_EXIT (
PMQAXP          pExitParms,     /* Address of exit parameter structure */
PMQAXC          pExitContext,   /* Address of exit context structure */
PMQHCONN        pHconn,         /* Address of connection handle */
PPMQOD          ppObjDesc,      /* Address of ptr to object descriptor */
PMQLONG         pOptions,       /* Address of open options */
PPMQHOBJ        ppHobj,         /* Address of ptr to object handle */
PMQLONG         pCompCode,      /* Address of completion code */
PMQLONG         pReason);       /* Address of reason code qualifying
                                   completion code */
```

Put - MQ_PUT_EXIT:

MQ_PUT_EXIT provides a put exit function to perform *before* and *after* MQPUT call processing. Use function identifier MQXF_PUT with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQPUT call exit functions.

The interface to this function is:

```
MQ_PUT_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &pMsgDesc,
             &pPutMsgOpts, &BufferLength, &pBuffer, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

Hobj (MQHOBJ) - input/output
Object handle.

pMsgDesc (PMQMD) - input/output
Pointer to message descriptor.

pPutMsgOpts (PMQPMO) - input/output
Pointer to put message options.

BufferLength (MQLONG) - input/output
Message buffer length.

pBuffer (PMQBYTE) - input/output
Pointer to message buffer.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP           ExitParms;      /* Exit parameter structure */
MQAXC           ExitContext;    /* Exit context structure */
MQHCONN         Hconn;          /* Connection handle */
MQHOBJ          Hobj;           /* Object handle */
PMQMD           pMsgDesc;       /* Ptr to message descriptor */
PMQPMO          pPutMsgOpts;    /* Ptr to put message options */
MQLONG          BufferLength;   /* Message buffer length */
PMQBYTE         pBuffer;        /* Ptr to message data */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code */
```

The queue manager then logically calls the exit as follows:

```
MQ_PUT_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &pMsgDesc,
             &pPutMsgOpts, &BufferLength, &pBuffer, &CompCode, &Reason)
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_PUT_EXIT (
PMQAXP          pExitParms,     /* Address of exit parameter structure */
PMQAXC          pExitContext,   /* Address of exit context structure */
PMQHCONN        pHconn,         /* Address of connection handle */
PMQHOBJ         pHobj,          /* Address of object handle */
PPMQMD          ppMsgDesc,      /* Address of ptr to message descriptor */
PPMQPMO         ppPutMsgOpts,   /* Address of ptr to put message options */
PMQLONG         pBufferLength,  /* Address of message buffer length */
PPMQBYTE        ppBuffer,       /* Address of ptr to message buffer */
PMQLONG         pCompCode,      /* Address of completion code */
PMQLONG         pReason);       /* Address of reason code qualifying
                                   completion code */
```

Usage notes

- Report messages generated by the queue manager skip the normal call processing. As a result, such messages cannot be intercepted by the MQ_PUT_EXIT function or the MQPUT1 function. However, report messages generated by the message channel agent are processed normally, and hence can be intercepted by the MQ_PUT_EXIT function or the MQ_PUT1_EXIT function. To be sure to intercepting all of the report messages generated by the MCA, both MQ_PUT_EXIT and MQ_PUT1_EXIT should be used.

Put1 - MQ_PUT1_EXIT:

MQ_PUT1_EXIT provides a *put one message only* exit function to perform *before* and *after* MQPUT1 call processing. Use function identifier MQXF_PUT1 with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQPUT1 call exit functions.

The interface to this function is:

```
MQ_PUT1_EXIT (&ExitParms, &ExitContext, &Hconn, &pObjDesc, &pMsgDesc,
&pPutMsgOpts, &BufferLength, &pBuffer, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pObjDesc (PMQOD) - input/output
Pointer to object descriptor.

pMsgDesc (PMQMD) - input/output
Pointer to message descriptor.

pPutMsgOpts (PMQPMO) - input/output
Pointer to put message options.

BufferLength (MQLONG) - input/output
Message buffer length.

pBuffer (PMQBYTE) - input/output
Pointer to message buffer.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP           ExitParms;      /* Exit parameter structure */
MQAXC           ExitContext;    /* Exit context structure */
MQHCONN         Hconn;          /* Connection handle */
PMQOD           pObjDesc;       /* Ptr to object descriptor */
PMQMD           pMsgDesc;       /* Ptr to message descriptor */
PMQPMO          pPutMsgOpts;    /* Ptr to put message options */
MQLONG          BufferLength;   /* Message buffer length */
PMQBYTE         pBuffer;        /* Ptr to message data */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code */
```

The queue manager then logically calls the exit as follows:

```
MQ_PUT1_EXIT (&ExitParms, &ExitContext, &Hconn, &pObjDesc, &pMsgDesc,
              &pPutMsgOpts, &BufferLength, &pBuffer, &CompCode, &Reason)
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_PUT1_EXIT (
PMQAXP          pExitParms,     /* Address of exit parameter structure */
PMQAXC          pExitContext,   /* Address of exit context structure */
PMQHCONN        pHconn,         /* Address of connection handle */
PPMQOD          ppObjDesc,      /* Address of ptr to object descriptor */
PPMQMD          ppMsgDesc,      /* Address of ptr to message descriptor */
PPMQPMO         ppPutMsgOpts,   /* Address of ptr to put message options */
PMQLONG         pBufferLength,  /* Address of message buffer length */
PPMQBYTE        ppBuffer,       /* Address of ptr to message buffer */
PMQLONG         pCompCode,      /* Address of completion code */
PMQLONG         pReason);       /* Address of reason code qualifying
                                   completion code */
```

Set - MQ_SET_EXIT:

MQ_SET_EXIT provides a set exit function to perform *before* and *after* MQSET call processing. Use function identifier MQXF_SET with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQSET call exit functions.

The interface to this function is:

```
MQ_SET_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &SelectorCount,
             &pSelectors, &IntAttrCount, &pIntAttrs, &CharAttrLength,
             &pCharAttr, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

Hobj (MQHOBJ) - input
Object handle.

SelectorCount (MQLONG) - input
Count of selectors

pSelectors (PMQLONG) - input/output
Pointer to array of selector values.

IntAttrCount (MQLONG) - input
Count of integer attributes.

pIntAttrs (PMQLONG) - input/output
Pointer to array of integer attribute values.

CharAttrLength (MQLONG) - input/output
Character attributes array length.

pCharAttrs (PMQCHAR) - input/output
Pointer to character attribute values.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;       /* Exit parameter structure */
MQAXC    ExitContext;     /* Exit context structure */
MQHCONN  Hconn;           /* Connection handle */
MQHOBJ   Hobj;            /* Object handle */
MQLONG   SelectorCount;   /* Count of selectors */
PMQLONG  pSelectors;      /* Ptr to array of attribute selectors */
MQLONG   IntAttrCount;    /* Count of integer attributes */
PMQLONG  pIntAttrs;       /* Ptr to array of integer attributes */
MQLONG   CharAttrLength;  /* Length of char attributes array */
PMQCHAR  pCharAttrs;      /* Ptr to character attributes */
MQLONG   CompCode;        /* Completion code */
MQLONG   Reason;          /* Reason code qualifying completion code */
```

The queue manager then logically calls the exit as follows:

```
MQ_SET_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &SelectorCount,
             &pSelectors, &IntAttrCount, &pIntAttrs, &CharAttrLength,
             &pCharAttrs, &CompCode, &Reason)
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_SET_EXIT (
PMQAXP    pExitParms,       /* Address of exit parameter structure */
PMQAXC    pExitContext,     /* Address of exit context structure */
PMQHCONN  pHconn,           /* Address of connection handle */
PMQHOBJ   pHobj,            /* Address of object handle */
PMQLONG   pSelectorCount,   /* Address of selector count */
PPMQLONG  ppSelectors,      /* Address of ptr to array of selectors */
PMQLONG   pIntAttrCount;    /* Address of count of integer attributes */
PPMQLONG  ppIntAttrs,       /* Address of ptr to array of integer attributes */
PMQLONG   pCharAttrLength,  /* Address of character attribute length */
```

```
PPMQCHAR  ppCharAttrs,      /* Address of ptr to character attributes array */
PMQLONG   pCompCode,        /* Address of completion code */
PMQLONG   pReason);         /* Address of reason code qualifying completion
                               code */
```

Status - MQ_STAT_EXIT:

MQ_STAT_EXIT provides a status exit function to perform *before* and *after* MQSTAT call processing. Use function identifier MQXF_STAT with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQSTAT call exit functions.

The interface to this function is:

```
MQ_STAT_EXIT (&ExitParms, &ExitContext, &Hconn, &Type, &pStatus
              &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

Type (MQLONG) - input
Type of status information to retrieve.

pStatus (PMQSTS) - output
Pointer to status buffer.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

Your exit must match the following C function prototype:

```
void MQENTRY MQ_STAT_EXIT (
PMQAXP    pExitParms,       /* Address of exit parameter structure */
PMQAXC    pExitContext,     /* Address of exit context structure */
PMQHCONN  pHconn,           /* Address of connection handle */
PMQLONG   pType             /* Address of status type */
```

```
PPMQSTS   ppStatus          /* Address of status buffer */
PMQLONG   pCompCode,        /* Address of completion code */
PMQLONG   pReason);         /* Address of reason code qualifying completion
                               code */
```

Termination - MQ_TERM_EXIT:

MQ_TERM_EXIT provides connection level termination, registered with a function identifier of MQXF_TERM and ExitReason MQXR_CONNECTION. If registered, MQ_TERM_EXIT is called once for every disconnect request.

As part of the termination, storage no longer required by the exit can be released, and any clean up required can be performed.

If an MQ_TERM_EXIT fails with MQXCC_FAILED, the queue manager returns from the MQDISC that called it with MQCC_FAILED and MQRC_API_EXIT_ERROR.

If the queue manager encounters an error while terminating the API exit function execution environment after invoking the last MQ_TERM_EXIT, the queue manager returns from the MQDISC call that invoked MQ_TERM_EXIT with MQCC_FAILED and MQRC_API_EXIT_TERM_ERROR

The interface to this function is:

```
MQ_TERM_EXIT (&ExitParms, &ExitContext, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED, the exit function can set the reason code field to any valid MQRC_* value.

The CompCode and Reason returned to the application depend on the value of the ExitResponse field in MQAXP.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP           ExitParms;      /* Exit parameter structure */
MQAXC           ExitContext;    /* Exit context structure */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code */
```

The queue manager then logically calls the exit as follows:

```
MQ_TERM_EXIT (&ExitParms, &ExitContext, &CompCode, &Reason)
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_TERM_EXIT (
PMQAXP          pExitParms,     /* Address of exit parameter structure */
PMQAXC          pExitContext,   /* Address of exit context structure */
PMQLONG         pCompCode,      /* Address of completion code */
PMQLONG         pReason);       /* Address of reason code qualifying
                                   completion code */
```

Usage notes

1. The MQ_TERM_EXIT function is optional. It is not necessary for an exit suite to register a termination exit if there is no termination processing to be done.

   If functions belonging to the exit suite acquire resources during the connection, an MQ_TERM_EXIT function is a convenient point at which to free those resources, for example, freeing storage obtained dynamically.
2. If an MQ_TERM_EXIT function is registered when the MQDISC call is issued, the exit function is invoked after all of the MQDISC exit functions have been invoked.
3. If MQ_TERM_EXIT returns MQXCC_FAILED in the `ExitResponse` field of MQAXP, or fails in some other way, the MQDISC call that caused MQ_TERM_EXIT to be invoked also fails, with the **CompCode** and **Reason** parameters set to appropriate values.

Register subscription - MQ_SUB_EXIT:

MQ_SUB_EXIT provides an exit function to perform *before* and *after* subscription reregistration processing. Use function identifier MQXF_SUB with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* subscription registrationcall exit functions.

The interface to this function is:

```
MQ_SUB_EXIT (&ExitParms, &ExitContext, &Hconn, &pSubDesc, &pHobj, &pHsub, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input/output
Connection handle.

pSubDesc - input/output
Array of attribute selectors.

pHobj - input/output
Object handle

pHsub (MQHOBJ) input/output
Subscription handle

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;      /* Exit parameter structure */
MQAXC    ExitContext;    /* Exit context structure */
MQHCONN  Hconn;          /* Connection handle */
PMQSD    pSubDesc;       /* Subscription descriptor */
PMQHOBJ  pHobj;          /* Object Handle */
PMQHOBJ  pHsub;          /* Subscription handle */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying completion code */
```

The queue manager then logically calls the exit as follows:

```
MQ_SUB_EXIT (&ExitParms, &ExitContext, &Hconn, &pSubDesc, &pHobj, &pHsub,
             &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
PMQAXP    pExitParms;      /* Exit parameter structure */
PMQAXC    pExitContext;    /* Exit context structure */
PMQHCONN  pHconn;          /* Connection handle */
PPMQSD    ppSubDesc;       /* Subscription descriptor */
PPMQHOBJ  ppHobj;          /* Object Handle */
PPMQHOBJ  ppHsub;          /* Subscription handle */
PMQLONG   pCompCode;       /* Completion code */
PMQLONG   pReason;         /* Reason code qualifying completion code */
```

Subscription request - MQ_SUBRQ_EXIT:

MQ_SUBRQ_EXIT provides a subscription request exit function to perform *before* and *after* subscription request processing. Use function identifier MQXF_SUBRQ with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* subscription request call exit functions.

The interface to this function is:

```
MQ_SUBRQ_EXIT (&ExitParms, &ExitContext, &Hconn, &pHsub, &Action, &pSubRqOpts,
               &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input/output
Connection handle.

pHsub (MQHOBJ) input/output
Subscription handle

Action (MQLONG) input/output
Action

pSubRqOpts (MQSRO) input/output

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;      /* Exit parameter structure */
MQAXC    ExitContext;    /* Exit context structure */
MQHCONN  Hconn;          /* Connection handle */
PMQLONG  pHsub;          /* Subscription handle */
MQLONG   Action;         /* Action */
PMQSRO   pSubRqOpts;     /* Subscription Request Options */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying completion code */
```

The queue manager then logically calls the exit as follows:

```
MQ_SUBRQ_EXIT (&ExitParms, &ExitContext, &Hconn, &pHsub, &Action, &pSubRqOpts,
               &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_SUBRQ_EXIT (
PMQAXP    pExitParms,       /* Address of exit parameter structure */
PMQAXC    pExitContext,     /* Address of exit context structure */
PMQHCONN  pHconn,           /* Address of connection handle */
PPMQHOBJ  ppHsub;           /* Address of Subscription handle */
PMQLONG   pAction;          /* Address of Action */
PPMQSRO   ppSubRqOpts;      /* Address of Subscription Request Options */
PMQLONG   pCompCode,        /* Address of completion code */
PMQLONG   pReason);         /* Address of reason code qualifying completion
                               code */
```

xa_close - XA_CLOSE_EXIT:

XA_CLOSE_EXIT provides an xa_close exit function to perform before and after xa_close processing. Use function identifier MQXF_XACLOSE with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_close call exit functions.

The interface to this function is:

```
XA_CLOSE_EXIT (&ExitParms, &ExitContext, &Hconn, &pXa_info, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXa_info (PMQCHAR) - input/output
Instance-specific resource manager information.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;    /* Exit parameter structure */
MQAXC    ExitContext;  /* Exit context structure */
MQHCONN  Hconn;        /* Connection handle */
PMQCHAR  pXa_info;     /* Instance-specific RM info */
MQLONG   Rmid;         /* Resource manager identifier */
MQLONG   Flags;        /* Resource manager options*/
MQLONG   XARetCode;    /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_CLOSE_EXIT (&ExitParms, &ExitContext, &Hconn, &pXa_info, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_CLOSE_EXIT (
     PMQAXP    pExitParms,    /* Address of exit parameter structure */
     PMQAXC    pExitContext,  /* Address of exit context structure */
     PMQHCONN  pHconn,        /* Address of connection handle */
     PPMQCHAR  ppXa_info,     /* Address of instance-specific RM info */
     PMQLONG   pRmid,         /* Address of resource manager identifier */
     PMQLONG   pFlags,        /* Address of resource manager options*/
     PMQLONG   pXARetCode);   /* Address of response from XA call */
```

xa_commit - XA_COMMIT_EXIT:

XA_COMMIT_EXIT provides an xa_commit exit function to perform before and after xa_commit processing. Use function identifier MQXF_XACOMMIT with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_commit call exit functions.

The interface to this function is:

```
XA_COMMIT_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXID (MQPTR) - input/output
Transaction branch ID.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
     MQAXP    ExitParms;   /* Exit parameter structure */
     MQAXC    ExitContext; /* Exit context structure */
     MQHCONN  Hconn;       /* Connection handle */
     MQPTR    pXID;        /* Transaction branch ID */
     MQLONG   Rmid;        /* Resource manager identifier */
     MQLONG   Flags;       /* Resource manager options*/
     MQLONG   XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_COMMIT_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_COMMIT_EXIT (
     PMQAXP    pExitParms,   /* Address of exit parameter structure */
     PMQAXC    pExitContext, /* Address of exit context structure */
     PMQHCONN  pHconn,       /* Address of connection handle */
     PMQPTR    ppXID,        /* Address of transaction branch ID */
```

```
     PMQLONG   pRmid,        /* Address of resource manager identifier */
     PMQLONG   pFlags,       /* Address of resource manager options*/
     PMQLONG   pXARetCode);  /* Address of response from XA call */
```

xa_complete - XA_COMPLETE_EXIT:

XA_COMPLETE_EXIT provides an xa_complete exit function to perform before and after xa_complete processing. Use function identifier MQXF_XACOMPLETE with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_complete call exit functions.

The interface to this function is:

```
XA_COMPLETE_EXIT (&ExitParms, &ExitContext, &Hconn, &pHandle, &pRetval, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pHandle (PMQLONG) - input/output
Pointer to asynchronous operation.

pRetVal (PMQLONG) - input/output
Return value of asynchronous operation.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
     MQAXP   ExitParms;   /* Exit parameter structure */
     MQAXC   ExitContext; /* Exit context structure */
     MQHCONN Hconn;       /* Connection handle */
     PMQLONG pHandle;     /* Ptr to asynchronous op */
     PMQLONG pRetval;     /* Return value of async op */
     MQLONG  Rmid;        /* Resource manager identifier */
     MQLONG  Flags;       /* Resource manager options*/
     MQLONG  XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_COMPLETE_EXIT (&ExitParms, &ExitContext, &Hconn, &pHandle, &pRetval, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_COMPLETE_EXIT (
     PMQAXP   pExitParms,   /* Address of exit parameter structure */
     PMQAXC   pExitContext, /* Address of exit context structure */
     PMQHCONN pHconn,       /* Address of connection handle */
     PPMQLONG ppHandle,     /* Address of ptr to asynchronous op */
     PPMQLONG ppRetval,     /* Address of return value of async op */
```

```
     PMQLONG  pRmid,        /* Address of resource manager identifier */
     PMQLONG  pFlags,       /* Address of resource manager options*/
     PMQLONG  pXARetCode);  /* Address of response from XA call */
```

xa_end - XA_END_EXIT:

XA_END_EXIT provides an xa_end exit function to perform before and after xa_end processing. Use function identifier MQXF_XAEND with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_end call exit functions.

The interface to this function is:

```
XA_END_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXID (MQPTR) - input/output
Transaction branch ID.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
     MQAXP   ExitParms;   /* Exit parameter structure */
     MQAXC   ExitContext; /* Exit context structure */
     MQHCONN Hconn;       /* Connection handle */
     MQPTR   pXID;        /* Transaction branch ID */
     MQLONG  Rmid;        /* Resource manager identifier */
     MQLONG  Flags;       /* Resource manager options*/
     MQLONG  XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_END_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_END_EXIT (
     PMQAXP   pExitParms,   /* Address of exit parameter structure */
     PMQAXC   pExitContext, /* Address of exit context structure */
     PMQHCONN pHconn,       /* Address of connection handle */
     PMQPTR   ppXID,        /* Address of transaction branch ID */
     PMQLONG  pRmid,        /* Address of resource manager identifier */
     PMQLONG  pFlags,       /* Address of resource manager options*/
     PMQLONG  pXARetCode);  /* Address of response from XA call */
```

xa_forget - XA_FORGET_EXIT:

XA_FORGET_EXIT provides an xa_forget exit function to perform before and after xa_forget processing. Use function identifier MQXF_XAFORGET with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_forget call exit functions.

The interface to this function is:

```
XA_FORGET_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXID (MQPTR) - input/output
Transaction branch ID.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP   ExitParms;   /* Exit parameter structure */
MQAXC   ExitContext; /* Exit context structure */
MQHCONN Hconn;       /* Connection handle */
MQPTR   pXID;        /* Transaction branch ID */
MQLONG  Rmid;        /* Resource manager identifier */
MQLONG  Flags;       /* Resource manager options*/
MQLONG  XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_FORGET_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_FORGET_EXIT (
     PMQAXP   pExitParms,   /* Address of exit parameter structure */
     PMQAXC   pExitContext, /* Address of exit context structure */
     PMQHCONN pHconn,       /* Address of connection handle */
     PMQPTR   ppXID,        /* Address of transaction branch ID */
     PMQLONG  pRmid,        /* Address of resource manager identifier */
     PMQLONG  pFlags,       /* Address of resource manager options*/
     PMQLONG  pXARetCode);  /* Address of response from XA call */
```

xa_open - XA_OPEN_EXIT:

XA_OPEN_EXIT provides an xa_open exit function to perform before and after xa_open processing. Use function identifier MQXF_XAOPEN with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_open call exit functions.

The interface to this function is:

```
XA_OPEN_EXIT (&ExitParms, &ExitContext, &Hconn, &pXa_info, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXa_info (PMQCHAR) - input/output
Instance-specific resource manager information.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP   ExitParms;   /* Exit parameter structure */
MQAXC   ExitContext; /* Exit context structure */
MQHCONN Hconn;       /* Connection handle */
PMQCHAR pXa_info;    /* Instance-specific RM info */
MQLONG  Rmid;        /* Resource manager identifier */
MQLONG  Flags;       /* Resource manager options*/
MQLONG  XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_OPEN_EXIT (&ExitParms, &ExitContext, &Hconn, &pXa_info, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_OPEN_EXIT (
     PMQAXP   pExitParms,   /* Address of exit parameter structure */
     PMQAXC   pExitContext, /* Address of exit context structure */
     PMQHCONN pHconn,       /* Address of connection handle */
     PPMQCHAR ppXa_info,    /* Address of instance-specific RM info */
     PMQLONG  pRmid,        /* Address of resource manager identifier */
     PMQLONG  pFlags,       /* Address of resource manager options*/
     PMQLONG  pXARetCode);  /* Address of response from XA call */
```

xa_prepare - XA_PREPARE_EXIT:

XA_PREPARE_EXIT provides an xa_prepare exit function to perform before and after xa_prepare processing. Use function identifier MQXF_XAPREPARE with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_prepare call exit functions.

The interface to this function is:

```
XA_PREPARE_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXID (MQPTR) - input/output
Transaction branch ID.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP  ExitParms;  /* Exit parameter structure */
     MQAXC   ExitContext; /* Exit context structure */
     MQHCONN Hconn;       /* Connection handle */
     MQPTR   pXID;        /* Transaction branch ID */
     MQLONG  Rmid;        /* Resource manager identifier */
     MQLONG  Flags;       /* Resource manager options*/
     MQLONG  XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_PREPARE_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_PREPARE_EXIT (
     PMQAXP   pExitParms,   /* Address of exit parameter structure */
     PMQAXC   pExitContext, /* Address of exit context structure */
     PMQHCONN pHconn,       /* Address of connection handle */
     PMQPTR   ppXID,        /* Address of transaction branch ID */
     PMQLONG  pRmid,        /* Address of resource manager identifier */
     PMQLONG  pFlags,       /* Address of resource manager options*/
     PMQLONG  pXARetCode);  /* Address of response from XA call */
```

xa_recover - XA_RECOVER_EXIT:

XA_RECOVER_EXIT provides an xa_recover exit function to perform before and after xa_recover processing. Use function identifier MQXF_XARECOVER with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_recover call exit functions.

The interface to this function is:

```
XA_RECOVER_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Count, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXID (MQPTR) - input/output
Transaction branch ID.

Count (MQLONG) - input/output
Maximum XIDs in XID array

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP   ExitParms;   /* Exit parameter structure */
MQAXC   ExitContext; /* Exit context structure */
MQHCONN Hconn;       /* Connection handle */
MQPTR   pXID;        /* Transaction branch ID */
MQLONG  Count;       /* Max XIDs in XID array */
MQLONG  Rmid;        /* Resource manager identifier */
MQLONG  Flags;       /* Resource manager options*/
MQLONG  XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_RECOVER_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Count, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_RECOVER_EXIT (
     PMQAXP   pExitParms,   /* Address of exit parameter structure */
     PMQAXC   pExitContext, /* Address of exit context structure */
     PMQHCONN pHconn,       /* Address of connection handle */
     PMQPTR   ppXID,        /* Address of transaction branch ID */
     PMQLONG  pCount,       /* Address of max XIDs in XID array */
     PMQLONG  pRmid,        /* Address of resource manager identifier */
     PMQLONG  pFlags,       /* Address of resource manager options*/
     PMQLONG  pXARetCode);  /* Address of response from XA call */
```

xa_rollback - XA_ROLLBACK_EXIT:

XA_ROLLBACK_EXIT provides an xa_rollback exit function to perform before and after xa_rollback processing. Use function identifier MQXF_XAROLLBACK with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_rollback call exit functions.

The interface to this function is:

```
XA_ROLLBACK_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXID (MQPTR) - input/output
Transaction branch ID.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP   ExitParms;   /* Exit parameter structure */
MQAXC   ExitContext; /* Exit context structure */
MQHCONN Hconn;       /* Connection handle */
MQPTR   pXID;        /* Transaction branch ID */
MQLONG  Rmid;        /* Resource manager identifier */
MQLONG  Flags;       /* Resource manager options*/
MQLONG  XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_ROLLBACK_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_ROLLBACK_EXIT (
     PMQAXP   pExitParms,   /* Address of exit parameter structure */
     PMQAXC   pExitContext, /* Address of exit context structure */
     PMQHCONN pHconn,       /* Address of connection handle */
     PMQPTR   ppXID,        /* Address of transaction branch ID */
     PMQLONG  pRmid,        /* Address of resource manager identifier */
     PMQLONG  pFlags,       /* Address of resource manager options*/
     PMQLONG  pXARetCode);  /* Address of response from XA call */
```

xa_start - XA_START_EXIT:

XA_START_EXIT provides an xa_start exit function to perform before and after xa_start processing. Use function identifier MQXF_XASTART with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_start call exit functions.

The interface to this function is:

```
XA_START_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXID (MQPTR) - input/output
Transaction branch ID.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP   ExitParms;   /* Exit parameter structure */
MQAXC   ExitContext; /* Exit context structure */
MQHCONN Hconn;       /* Connection handle */
MQPTR   pXID;        /* Transaction branch ID */
MQLONG  Rmid;        /* Resource manager identifier */
MQLONG  Flags;       /* Resource manager options*/
MQLONG  XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_START_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_START_EXIT (
     PMQAXP   pExitParms,   /* Address of exit parameter structure */
     PMQAXC   pExitContext, /* Address of exit context structure */
     PMQHCONN pHconn,       /* Address of connection handle */
     PMQPTR   ppXID,        /* Address of transaction branch ID */
     PMQLONG  pRmid,        /* Address of resource manager identifier */
     PMQLONG  pFlags,       /* Address of resource manager options*/
     PMQLONG  pXARetCode);  /* Address of response from XA call */
```

ax_reg - AX_REG_EXIT:

AX_REG_EXIT provides an ax_reg exit function to perform before and after ax_reg processing. Use function identifier MQXF_AXREG with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after ax_reg call exit functions.

The interface to this function is:

```
AX_REG_EXIT (&ExitParms, &ExitContext, &pXID, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXID (MQPTR) - input/output
Transaction branch ID.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP  ExitParms;   /* Exit parameter structure */
MQAXC  ExitContext; /* Exit context structure */
MQPTR  pXID;        /* Transaction branch ID */
MQLONG Rmid;        /* Resource manager identifier */
MQLONG Flags;       /* Resource manager options*/
MQLONG XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
AX_REG_EXIT (&ExitParms, &ExitContext, &pXID, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY AX_REG_EXIT (
     PMQAXP  pExitParms,   /* Address of exit parameter structure */
     PMQAXC  pExitContext, /* Address of exit context structure */
     PMQPTR  ppXID,        /* Address of transaction branch ID */
     PMQLONG pRmid,        /* Address of resource manager identifier */
     PMQLONG pFlags,       /* Address of resource manager options*/
     PMQLONG pXARetCode);  /* Address of response from XA call */
```

ax_unreg - AX_UNREG_EXIT:

AX_UNREG_EXIT provides an ax_unreg exit function to perform before and after ax_unreg processing. Use function identifier MQXF_AXUNREG with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after ax_unreg call exit functions.

The interface to this function is:

```
AX_UNREG_EXIT (&ExitParms, &ExitContext, &Rmid, &Flags, &XARetCode);
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP  ExitParms;   /* Exit parameter structure */
MQAXC  ExitContext; /* Exit context structure */
MQLONG Rmid;        /* Resource manager identifier */
MQLONG Flags;       /* Resource manager options*/
MQLONG XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
AX_UNREG_EXIT (&ExitParms, &ExitContext, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY AX_UNREG_EXIT (
     PMQAXP  pExitParms,   /* Address of exit parameter structure */
     PMQAXC  pExitContext, /* Address of exit context structure */
     PMQLONG pRmid,        /* Address of resource manager identifier */
     PMQLONG pFlags,       /* Address of resource manager options*/
     PMQLONG pXARetCode);  /* Address of response from XA call */
```

General information on invoking exit functions:

This topic provides some general guidance to help you to plan your exits, particularly related to handling errors and unexpected events.

Exit failure:

If an exit function abnormally terminates after a destructive, out of syncpoint, MQGET call but before the message has been passed to the application, the exit handler can recover from the failure, and pass control to the application.

In this case, the message might be lost. This is like what happens when an application fails immediately after receiving a message from a queue.

The MQGET call might complete with MQCC_FAILED and MQRC_API_EXIT_ERROR.

If a *before* API call exit function terminates abnormally, the exit handler can recover from the failure and pass control to the application without processing the API call. In this event, the exit function must recover any resources that it owns.

If chained exits are in use, the *after* API call exits for any *before* API call exits that had successfully been driven can themselves be driven. The API call might fail with MQCC_FAILED and MQRC_API_EXIT_ERROR.

Example error handling for exit functions:

The following diagram shows the points (e *N*) at which errors can occur. It is only an example to show how exits behave and should be read together with the following table. In this example, two exit functions are invoked both before and after each API call to show the behavior with chained exits.

```
Application ErrPt     Exit function            API call
----------- -----     -------------            --------

Start

MQCONN  -->
            e1
                          MQ_INIT_EXIT
            e2
                  before MQ_CONNX_EXIT  1
            e3
                  before MQ_CONNX_EXIT  2
            e4
                                          -->  MQCONN
            e5
                  after  MQ_CONNX_EXIT  2
            e6
                  after  MQ_CONNX_EXIT  1
            e7
        <--
MQOPEN  -->
                  before MQ_OPEN_EXIT  1
            e8
                  before MQ_OPEN_EXIT  2
            e9
                                          -->  MQOPEN
            e10
                  after  MQ_OPEN_EXIT  2
            e11
                  after  MQ_OPEN_EXIT  1
            e12
        <--
```

```
MQPUT   -->
                  before MQ_PUT_EXIT   1
            e13
                  before MQ_PUT_EXIT   2
            e14
                                       -->  MQPUT
            e15
                  after  MQ_PUT_EXIT   2
            e16
                  after  MQ_PUT_EXIT   1
            e17
        <--
MQCLOSE -->
                  before MQ_CLOSE_EXIT 1
            e18
                  before MQ_CLOSE_EXIT 2
            e19
                                       -->  MQCLOSE
            e20
                  after  MQ_CLOSE_EXIT 2
            e21
                  after  MQ_CLOSE_EXIT 1
            e22
        <--
MQDISC  -->
                  before MQ_DISC_EXIT  1
            e23
                  before MQ_DISC_EXIT  2
            e24
                                       -->  MQDISC
            e25
                  after  MQ_DISC_EXIT  2
            e26
                  after  MQ_DISC_EXIT  1
            e27

        <--

end
```

The following table lists the actions to be taken at each error point. Only a subset of the error points have been covered, as the rules shown here can apply to all others. It is the actions that specify the intended behavior in each case.

Table 399. API exit errors and appropriate actions to take

ErrPt	Description	Actions
e1	Error while setting up environment setup.	1. Undo environment setup as required 2. Drive no exit functions 3. Fail MQCONN with MQCC_FAILED, MQRC_API_EXIT_LOAD_ERROR
e2	MQ_INIT_EXIT function completes with: • MQXCC_FAILED • MQXCC_*	• For MQXCC_FAILED: 1. Clean up environment 2. Fail MQCONN with MQCC_FAILED, MQRC_API_EXIT_INIT_ERROR • For MQXCC_* 1. Act as for the values of MQXCC_* and MQXR2_* [1] 2. Clean up environment

Table 399. API exit errors and appropriate actions to take (continued)

ErrPt	Description	Actions
e3	*Before* MQ_CONNX_EXIT 1 function completes with: • MQXCC_FAILED • MQXCC_*	• For MQXCC_FAILED: 1. Drive MQ_TERM_EXIT function 2. Clean up environment 3. Fail MQCONN call with MQCC_FAILED, MQRC_API_EXIT_ERROR • For MQXCC_* 1. Act as for the values of MQXCC_* and MQXR2_* [1] 2. Drive MQ_TERM_EXIT function if required 3. Clean up environment if required
e4	*Before* MQ_CONNX_EXIT 2 function completes with: • MQXCC_FAILED • MQXCC_*	• For MQXCC_FAILED: 1. Drive *after* MQ_CONNX_EXIT 1 function 2. Drive MQ_TERM_EXIT function 3. Clean up environment 4. Fail MQCONN call with MQCC_FAILED, MQRC_API_EXIT_ERROR • For MQXCC_* 1. Act as for the values of MQXCC_* and MQXR2_* [1] 2. Drive *after* MQ_CONNX_EXIT 1 function if exit not suppressed 3. Drive MQ_TERM_EXIT function if required 4. Clean up environment if required
e5	MQCONN call fails.	1. Pass MQCONN CompCode and Reason 2. Drive *after* MQ_CONNX_EXIT 2 function if the *before* MQ_CONNX_EXIT 2 succeeded and the exit is not suppressed 3. Drive *after* MQ_CONNX_EXIT 1 function if the *before* MQ_CONNX_EXIT 1 succeeded and the exit is not suppressed 4. Drive MQ_TERM_EXIT function 5. Clean up environment
e6	*After* MQ_CONNX_EXIT 2 function completes with: • MQXCC_FAILED • MQXCC_*	• For MQXCC_FAILED: 1. Drive *after* MQ_CONNX_EXIT 1 function 2. Complete MQCONN call with MQCC_FAILED, MQRC_API_EXIT_ERROR • For MQXCC_* 1. Act as for the values of MQXCC_* and MQXR2_* [1] 2. Drive *after* MQ_CONNX_EXIT 1 function if required
e7	*After* MQ_CONNX_EXIT 1 function completes with: • MQXCC_FAILED • MQXCC_*	• For MQXCC_FAILED, complete MQCONN call with MQCC_FAILED, MQRC_API_EXIT_ERROR • For MQXCC_*, act as for the values of MQXCC_* and MQXR2_* [1]
e8	*Before* MQ_OPEN_EXIT 1 function completes with: • MQXCC_FAILED • MQXCC_*	• For MQXCC_FAILED, complete MQOPEN call with MQCC_FAILED, MQRC_API_EXIT_ERROR • For MQXCC_*, act as for the values of MQXCC_* and MQXR2_* [1]

Table 399. API exit errors and appropriate actions to take (continued)

ErrPt	Description	Actions
e9	*Before* MQ_OPEN_EXIT 2 function completes with: • MQXCC_FAILED • MQXCC_*	• For MQXCC_FAILED: 1. Drive *after* MQ_OPEN_EXIT 1 function 2. Complete MQOPEN call with MQCC_FAILED, MQRC_API_EXIT_ERROR • For MQXCC_*, act as for the values of MQXCC_* and MQXR2_* [1]
e10	MQOPEN call fails	1. Pass MQOPEN CompCode and Reason 2. Drive *after* MQ_OPEN_EXIT 2 function if exit not suppressed 3. Drive *after* MQ_OPEN_EXIT 1 function if exit not suppressed and if chained exits not suppressed
e11	*After* MQ_OPEN_EXIT 2 function completes with: • MQXCC_FAILED • MQXCC_*	• For MQXCC_FAILED: 1. Drive *after* MQ_OPEN_EXIT 1 function 2. Complete MQOPEN call with MQCC_FAILED, MQRC_API_EXIT_ERROR • For MQXCC_* 1. Act as for the values of MQXCC_* and MQXR2_* [1] 2. Drive *after* MQ_OPEN_EXIT 1 function if exit not suppressed
e25	*After* MQ_DISC_EXIT 2 function completes with: • MQXCC_FAILED • MQXCC_*	• For MQXCC_FAILED: 1. Drive *after* MQ_DISC_EXIT 1 function 2. Drive MQ_TERM_EXIT function 3. Clean up exit execution environment 4. Complete MQDISC call with MQCC_FAILED, MQRC_API_EXIT_ERROR • For MQXCC_* 1. Act as for the values of MQXCC_* and MQXR2_* [1] 2. Drive MQ_TERM_EXIT function 3. Clean up exit execution environment

Note:

1. The values of MQXCC_* and MQXR2_* and their corresponding actions are defined in How queue managers process exit functions.

ExitResponse fields set incorrectly:

This topic gives information about what would happen when the ExitResponse field is set to anything but the supported values.

If the ExitResponse field is set to a value other than one of the supported values, the following actions apply:

- For a *before* MQCONN or MQDISC API exit function:
  - The ExitResponse2 value is ignored.
  - No further *before* exit functions in the exit chain (if any) are invoked; the API call itself is not issued.
  - For any *before* exits that were successfully called, the *after* exits are called in reverse order.
  - If registered, the termination exit functions for those *before* MQCONN or MQDISC exit functions in the chain that were successfully invoked are driven to clean up after these exit functions.
  - The MQCONN or MQDISC call fails with MQRC_API_EXIT_ERROR.
- For a *before* IBM MQ API exit function other than MQCONN or MQDISC:
  - The ExitResponse2 value is ignored.
  - No further *before* or *after* data conversion functions in the exit chain (if any) are invoked.
  - For any *before* exits that were successfully called, the *after* exits are called in reverse order.
  - The IBM MQ API call itself is not issued.
  - The IBM MQ API call fails with MQRC_API_EXIT_ERROR.
- For an *after* MQCONN or MQDISC API exit function:
  - The ExitResponse2 value is ignored.
  - The remaining exit functions that were successfully called before the API call are called in reverse order.
  - If registered, the termination exit functions for those *before* or *after* MQCONN or MQDISC exit functions in the chain that were successfully invoked are driven to clean up after the exit.
  - A CompCode of the more severe of MQCC_WARNING and the CompCode returned by the exit is returned to the application.
  - A Reason of MQRC_API_EXIT_ERROR is returned to the application.
  - The IBM MQ API call is successfully issued.
- For an *after* IBM MQ API call exit function other than MQCONN or MQDISC:
  - The ExitResponse2 value is ignored.
  - The remaining exit functions that were successfully called before the API call are called in reverse order.
  - A CompCode of the more severe of MQCC_WARNING and the CompCode returned by the exit is returned to the application.
  - A Reason of MQRC_API_EXIT_ERROR is returned to the application.
  - The IBM MQ API call is successfully issued.
- For the *before* data conversion on get exit function:
  - The ExitResponse2 value is ignored.
  - The remaining exit functions that were successfully called before the API call are called in reverse order.
  - The message is not converted, and the unconverted message is returned to the application.
  - A CompCode of the more severe of MQCC_WARNING and the CompCode returned by the exit is returned to the application.
  - A Reason of MQRC_API_EXIT_ERROR is returned to the application.
  - The IBM MQ API call is successfully issued.

Note: As the error is with the exit, it is better to return MQRC_API_EXIT_ERROR than to return MQRC_NOT_CONVERTED.

If an exit function sets the ExitResponse2 field to a value other than one of the supported values, a value of MQXR2_DEFAULT_CONTINUATION is assumed instead.

Installable services interface reference information

This collection of topics provides reference information for the installable services.

The functions and data types are listed in alphabetical order within the group for each service type.

Related information:

ULW Installable services and components for UNIX, Linux and Windows

ULW Configuring installable services

IBM i Installable services and components for IBM i

IBM i Installable services interface reference information for IBMi

Use this information to understand the reference information for the installable services for IBM i.

How the functions are shown:

How the installable services functions are documented.

For each function there is a description, including the function identifier (for MQZEP).

The *parameters* are shown listed in the order they must occur. They must all be present.

Each parameter name is followed by its data type. These are the elementary data types described in the "Elementary data types" on page 2196.

The C language invocation is also given, after the description of the parameters.

MQZ_AUTHENTICATE_USER - Authenticate user:

This function is provided by an MQZAS_VERSION_5 authorization service component, and is invoked by the queue manager to authenticate a user, or to set identity context fields. It is invoked when the IBM MQ user application context is established.

The application context is established during connect calls at the point where the application's user context is initialized, and at each point where the application's user context is changed. Each time a connect call is made, the application's user context information is reacquired in the *IdentityContext* field.

The function identifier for this function (for MQZEP) is MQZID_AUTHENTICATE_USER.

Syntax

MQZ_AUTHENTICATE_USER (*QMgrName* , *SecurityParms* , *ApplicationContext* , *IdentityContext* , *CorrelationPtr* , *ComponentData* , *Continuation* , *CompCode* , *Reason*)

Parameters

QMgrName

Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

SecurityParms
Type: MQCSP - input

Security parameters. Data relating to the user ID, password, and authentication type. If the AuthenticationType attribute of the MQCSP structure is specified as MQCSP_AUTH_USER_ID_AND_PWD, both the user ID and password are compared against the equivalent fields in the IdentityContext (MQZIC) parameter to determine whether they match . For more information, see "MQCSP - Security parameters" on page 2292.

During an MQCONN MQI call this parameter contains null, or default values.

ApplicationContext
Type: MQZAC - input

Application context. Data relating to the calling application. See MQZAC - Application context for details.

During every MQCONN or MQCONNX MQI call, the user context information in the MQZAC structure is reacquired.

IdentityContext
Type: MQZIC - input/output

Identity context. On input to the authenticate user function, this identifies the current identity context. The authenticate user function can change this, at which point the queue manager adopts the new identity context. See MQZIC - Identity context for more details on the MQZIC structure.

CorrelationPtr
Type: MQPTR - output

Correlation pointer. Specifies the address of any correlation data. This pointer is subsequently passed on to other OAM calls.

ComponentData
Type: MQBYTE x ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the ComponentDataLength parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation flag. You can specify the following values:

MQZCI_DEFAULT
Continuation dependent on other components.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

For more information about these reason codes, see Reason codes.

C invocation

```
MQZ_AUTHENTICATE_USER (QMgrName, SecurityParms, ApplicationContext,
                  IdentityContext, &CorrelationPtr, ComponentData,
                  &Continuation, &CompCode, &Reason);
```

Declare the parameters passed to the service as follows:

```
MQCHAR48  QMgrName;           /* Queue manager name */
MQCSP     SecurityParms;      /* Security parameters */
MQZAC     ApplicationContext; /* Application context */
MQZIC     IdentityContext;    /* Identity context */
MQPTR     CorrelationPtr;     /* Correlation pointer */
MQBYTE    ComponentData[n];   /* Component data */
MQLONG    Continuation;       /* Continuation indicator set by
                                 component */
MQLONG    CompCode;           /* Completion code */
MQLONG    Reason;             /* Reason code qualifying CompCode */
```

MQZ_CHECK_AUTHORITY - Check authority:

This function is provided by a MQZAS_VERSION_1 authorization service component, and is started by the queue manager to check whether an entity has authority to perform a particular action, or actions, on a specified object.

The function identifier for this function (for MQZEP) is MQZID_CHECK_AUTHORITY.

Syntax

```
MQZ_CHECK_AUTHORITY( QMgrName , EntityName , EntityType , ObjectName , ObjectType ,
 Authority , ComponentData , Continuation , CompCode , Reason )
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityName
Type: MQCHAR12 - input

Entity name. The name of the entity whose authorization to the object is to be checked. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

It is not essential for this entity to be known to the underlying security service. If it is not known, the authorizations of the special **nobody** group (to which all entities are assumed to belong) are used for the check. An all-blank name is valid and can be used in this way.

EntityType
Type: MQLONG - input

Entity type. The type of entity specified by EntityName. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object to which access is required. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType
Type: MQLONG - input

Object type. The type of entity specified by *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

Authority
Type: MQLONG - input

Authority to be checked. If one authorization is being checked, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authorization is being checked, it is the bitwise OR of the corresponding MQZAO_* constants.

The following authorizations apply to use of the MQI calls:

MQZAO_CONNECT
Ability to use the MQCONN call.

MQZAO_BROWSE
Ability to use the MQGET call with a browse option.

This allows the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_MSG_UNDER_CURSOR, or MQGMO_BROWSE_NEXT option to be specified on the MQGET call.

MQZAO_INPUT
Principal. Ability to use the MQGET call with an input option.

This allows the MQOO_INPUT_SHARED, MQOO_INPUT_EXCLUSIVE, or MQOO_INPUT_AS_Q_DEF option to be specified on the MQOPEN call.

MQZAO_OUTPUT
Ability to use the MQPUT call.

This allows the MQOO_OUTPUT option to be specified on the MQOPEN call.

MQZAO_INQUIRE
Ability to use the MQINQ call.

This allows the MQOO_INQUIRE option to be specified on the MQOPEN call.

MQZAO_SET
Ability to use the MQSET call.

This allows the MQOO_SET option to be specified on the MQOPEN call.

MQZAO_PASS_IDENTITY_CONTEXT
Ability to pass identity context.

This allows the MQOO_PASS_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_PASS_ALL_CONTEXT
Ability to pass all context.

This allows the MQOO_PASS_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_IDENTITY_CONTEXT
Ability to set identity context.

This allows the MQOO_SET_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_ALL_CONTEXT
Ability to set all context.

This allows the MQOO_SET_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_ALTERNATE_USER_AUTHORITY
Ability to use alternate user authority.

This allows the MQOO_ALTERNATE_USER_AUTHORITY option to be specified on the MQOPEN call, and the MQPMO_ALTERNATE_USER_AUTHORITY option to be specified on the MQPUT1 call.

MQZAO_ALL_MQI
All of the MQI authorizations.

This enables all of the authorizations.

The following authorizations apply to administration of a queue manager:

MQZAO_CREATE
Ability to create objects of a specified type.

MQZAO_DELETE
Ability to delete a specified object.

MQZAO_DISPLAY
Ability to display the attributes of a specified object.

MQZAO_CHANGE
Ability to change the attributes of a specified object.

MQZAO_CLEAR
Ability to delete all messages from a specified queue.

MQZAO_AUTHORIZE
Ability to authorize other users for a specified object.

MQZAO_CONTROL
Ability to start or stop a listener, service, or non-client channel object, and the ability to ping a non-client channel object.

MQZAO_CONTROL_EXTENDED
Ability to reset a sequence number, or resolve an indoubt message on a non-client channel object.

MQZAO_ALL_ADMIN
Ability to set identity context.

All of the administration authorizations, other than MQZAO_CREATE.

The following authorizations apply to both use of the MQI and to administration of a queue manager:

MQZAO_ALL
All authorizations, other than MQZAO_CREATE.

MQZAO_NONE
No authorizations.

ComponentData
Type: MQBYTE x ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

If the call to a component fails (that is, *CompCode* returns MQCC_FAILED), and the *Continuation* parameter is MQZCI_DEFAULT or MQZCI_CONTINUE, the queue manager continues to call other components if there are any.

If the call succeeds (that is, *CompCode* returns MQCC_OK) no other components are called no matter what the setting of *Continuation* is.

If the call fails and the *Continuation* parameter is MQZCI_STOP then no other components are called and the error is returned to the queue manager. Components have no knowledge of previous calls, so the *Continuation* parameter is always set to MQZCI_DEFAULT before the call.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_CHECK_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                     ObjectType, Authority, ComponentData,
                     &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority to be checked */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
```

```
                              component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_CHECK_AUTHORITY_2 - Check authority (extended):

This function is provided by a MQZAS_VERSION_2 authorization service component, and is started by the queue manager to check whether an entity has authority to perform a particular action, or actions, on a specified object.

The function identifier for this function (for MQZEP) is MQZID_CHECK_AUTHORITY.

MQZ_CHECK_AUTHORITY_2 is like MQZ_CHECK_AUTHORITY, but with the **EntityName** parameter replaced by the **EntityData** parameter.

Syntax

```
MQZ_CHECK_AUTHORITY_2( QMgrName , EntityData , EntityType , ObjectName , ObjectType ,
 Authority , ComponentData , Continuation , CompCode , Reason )
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityData
Type: MQZED - input

Entity data. Data relating to the entity with authorization to the object that is to be checked. See "MQZED - Entity descriptor" on page 3777 for details.

It is not essential for this entity to be known to the underlying security service. If it is not known, the authorizations of the special **nobody** group (to which all entities are assumed to belong) are used for the check. An all-blank name is valid and can be used in this way.

EntityType
Type: MQLONG - input

Entity type. The type of entity specified by *EntityData*. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object to which access is required. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType

Type: MQLONG - input

Object type. The type of entity specified by *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

MQOT_TOPIC
Topic.

Authority

Type: MQLONG - input

Authority to be checked. If one authorization is being checked, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authorization is being checked, it is the bitwise OR of the corresponding MQZAO_* constants.

The following authorizations apply to use of the MQI calls:

MQZAO_CONNECT
Ability to use the MQCONN call.

MQZAO_BROWSE
Ability to use the MQGET call with a browse option.

This allows the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_MSG_UNDER_CURSOR, or MQGMO_BROWSE_NEXT option to be specified on the MQGET call.

MQZAO_INPUT
Principal. Ability to use the MQGET call with an input option.

This allows the MQOO_INPUT_SHARED, MQOO_INPUT_EXCLUSIVE, or MQOO_INPUT_AS_Q_DEF option to be specified on the MQOPEN call.

MQZAO_OUTPUT
Ability to use the MQPUT call.

This allows the MQOO_OUTPUT option to be specified on the MQOPEN call.

MQZAO_INQUIRE
Ability to use the MQINQ call.

This allows the MQOO_INQUIRE option to be specified on the MQOPEN call.

MQZAO_SET
Ability to use the MQSET call.

This allows the MQOO_SET option to be specified on the MQOPEN call.

MQZAO_PASS_IDENTITY_CONTEXT
Ability to pass identity context.

This allows the MQOO_PASS_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_PASS_ALL_CONTEXT
Ability to pass all context.

This allows the MQOO_PASS_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_IDENTITY_CONTEXT
Ability to set identity context.

This allows the MQOO_SET_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_ALL_CONTEXT
Ability to set all context.

This allows the MQOO_SET_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_ALTERNATE_USER_AUTHORITY
Ability to use alternate user authority.

This allows the MQOO_ALTERNATE_USER_AUTHORITY option to be specified on the MQOPEN call, and the MQPMO_ALTERNATE_USER_AUTHORITY option to be specified on the MQPUT1 call.

MQZAO_ALL_MQI
All of the MQI authorizations.

This enables all of the authorizations.

The following authorizations apply to administration of a queue manager:

MQZAO_CREATE
Ability to create objects of a specified type.

MQZAO_DELETE
Ability to delete a specified object.

MQZAO_DISPLAY
Ability to display the attributes of a specified object.

MQZAO_CHANGE
Ability to change the attributes of a specified object.

MQZAO_CLEAR
Ability to delete all messages from a specified queue.

MQZAO_AUTHORIZE
Ability to authorize other users for a specified object.

MQZAO_CONTROL
Ability to start or stop a listener, service, or non-client channel object, and the ability to ping a non-client channel object.

MQZAO_CONTROL_EXTENDED
Ability to reset a sequence number, or resolve an indoubt message on a non-client channel object.

MQZAO_ALL_ADMIN
Ability to set identity context.

All of the administration authorizations, other than MQZAO_CREATE.

The following authorizations apply to both use of the MQI and to administration of a queue manager:

MQZAO_ALL
All authorizations, other than MQZAO_CREATE.

MQZAO_NONE
No authorizations.

ComponentData
Type: MQBYTE x ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_CHECK_AUTHORITY_2 (QMgrName, &EntityData, EntityType,
                       ObjectName, ObjectType, Authority, ComponentData,
                       &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQZED     EntityData;        /* Entity data */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority to be checked */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_CHECK_PRIVILEGED - Check if user is privileged:

This function is provided by an MQZAS_VERSION_6 authorization service component, and is invoked by the queue manager to determine whether a specified user is a privileged user.

The function identifier for this function (for MQZEP) is MQZID_CHECK_PRIVILEGED.

Syntax

```
MQZ_CHECK_PRIVILEGED( QMgrName , EntityData , EntityType , ComponentData , Continuation ,
 CompCode , Reason )
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityData
Type: MQZED - input

Entity data. Data relating to the entity that is to be checked. For more information, see "MQZED - Entity descriptor" on page 3777.

EntityType
Type: MQLONG - input

Entity type. The type of entity specified by EntityData. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ComponentData
Type: MQBYTExComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

If the call to a component fails (that is, *CompCode* returns MQCC_FAILED), and the *Continuation* parameter is MQZCI_DEFAULT or MQZCI_CONTINUE, the queue manager continues to call other components if there are any.

If the call succeeds (that is, *CompCode* returns MQCC_OK) no other components are called no matter what the setting of *Continuation* is.

If the call fails and the *Continuation* parameter is MQZCI_STOP then no other components are called and the error is returned to the queue manager. Components have no knowledge of previous calls, so the *Continuation* parameter is always set to MQZCI_DEFAULT before the call.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_PRIVILEGED
(2584, X'A18') This user is not a privileged user ID.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_CHECK_PRIVILEGED (QMgrName, &EntityData, EntityType,
                      ComponentData, &Continuation,
                      &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQZED     EntityData;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_COPY_ALL_AUTHORITY - Copy all authority:

This function is provided by an authorization service component. It is started by the queue manager to copy all of the authorizations that are currently in force for a reference object to another object.

The function identifier for this function (for MQZEP) is MQZID_COPY_ALL_AUTHORITY.

Syntax

```
MQZ_COPY_ALL_AUTHORITY( QMgrName , RefObjectName , ObjectName , ObjectType , ComponentData
, Continuation , CompCode , Reason )
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

RefObjectName
Type: MQCHAR48 - input

Reference object name. The name of the reference object, the authorizations for which are to be copied. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

ObjectName

Type: MQCHAR48 - input

Object name. The name of the object for which accesses are to be set. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

ObjectType

Type: MQLONG - input

Object type. The type of entity specified by *RefObjectName* and *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

MQOT_TOPIC
Topic.

ComponentData

Type: MQBYTExComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the ComponentDataLength parameter of the MQZ_INIT_AUTHORITY call.

Continuation

Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_REF_OBJECT
(2294, X'8F6') Reference object unknown.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_COPY_ALL_AUTHORITY (QMgrName, RefObjectName, ObjectName, ObjectType,
                        ComponentData, &Continuation, &CompCode,
                        &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  RefObjectName;     /* Reference object name */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_DELETE_AUTHORITY - Delete authority:

This function is provided by an authorization service component, and is started by the queue manager to delete all of the authorizations associated with the specified object.

The function identifier for this function (for MQZEP) is MQZID_DELETE_AUTHORITY.

Syntax

```
MQZ_DELETE_AUTHORITY( QMgrName , ObjectName , ObjectType , ComponentData , Continuation ,
 CompCode , Reason )
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object for which accesses are to be deleted. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType
Type: MQLONG - input

Object type. The type of entity specified by *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

MQOT_TOPIC
Topic.

ComponentData
Type: MQBYTE x ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the ComponentDataLength parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_DELETE_AUTHORITY (QMgrName, ObjectName, ObjectType, ComponentData,
                      &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_ENUMERATE_AUTHORITY_DATA - Enumerate authority data:

This function is provided by an MQZAS_VERSION_4 authorization service component, and is started repeatedly by the queue manager to retrieve all of the authority data that matches the selection criteria specified on the first invocation.

The function identifier for this function (for MQZEP) is MQZID_ENUMERATE_AUTHORITY_DATA.

Syntax

```
MQZ_ENUMERATE_AUTHORITY_DATA( QMgrName , StartEnumeration , Filter , AuthorityBufferLength ,
 AuthorityBuffer , AuthorityDataLength , ComponentData , Continuation , CompCode , Reason )
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

StartEnumeration
Type: MQLONG - input

Flag indicating whether call can start enumeration. This indicates whether the call can start the enumeration of authority data, or continue the enumeration of authority data started by a previous call to MQZ_ENUMERATE_AUTHORITY_DATA. The value is one of the following values:

MQZSE_START
Start enumeration. The call is started with this value to start the enumeration of authority data. The **Filter** parameter specifies the selection criteria to be used to select the authority data returned by this and successive calls.

MQZSE_CONTINUE
Continue enumeration. The call is started with this value to continue the enumeration of authority data. The **Filter** parameter is ignored in this case, and can be specified as the null pointer (the selection criteria are determined by the **Filter** parameter specified by the call that had *StartEnumeration* set to MQZSE_START).

Filter
Type: MQZAD - input

Filter. If *StartEnumeration* is MQZSE_START, *Filter* specifies the selection criteria to be used to select the authority data to return. If *Filter* is the null pointer, no selection criteria are used, that is, all authority data is returned. See "MQZAD - Authority data" on page 3774 for details of the selection criteria that can be used.

If *StartEnumeration* is MQZSE_CONTINUE, *Filter* is ignored, and can be specified as the null pointer.

AuthorityBufferLength
Type: MQLONG - input

Length of *AuthorityBuffer*. This is the length in bytes of the **AuthorityBuffer** parameter. The authority buffer must be large enough to accommodate the data to be returned.

AuthorityBuffer
Type: MQZAD - output

Authority data. This is the buffer in which the authority data is returned. The buffer must be large enough to accommodate an MQZAD structure, an MQZED structure, plus the longest entity name and longest domain name defined.

Note: Note: This parameter is defined as an MQZAD, as the MQZAD always occurs at the start of the buffer. However, if the buffer is declared as an MQZAD, the buffer will be too small - it must be bigger than an MQZAD so that it can accommodate the MQZAD, MQZED, plus entity and domain names.

AuthorityDataLength
Type: MQLONG - output

Length of data returned in *AuthorityBuffer*. If the authority buffer is too small, *AuthorityDataLength* is set to the length of the buffer required, and the call returns completion code MQCC_FAILED and reason code MQRC_BUFFER_LENGTH_ERROR.

ComponentData
Type: MQBYTE x ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the ComponentDataLength parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_ENUMERATE_AUTHORITY_DATA, this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_NO_DATA_AVAILABLE
(2379, X'94B') No data available.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_ENUMERATE_AUTHORITY_DATA (QMgrName, StartEnumeration, &Filter,
                              AuthorityBufferLength,
                              &AuthorityBuffer,
                              &AuthorityDataLength, ComponentData,
                              &Continuation, &CompCode,
                              &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;               /* Queue manager name */
MQLONG    StartEnumeration;       /* Flag indicating whether call should
                                     start enumeration */
MQZAD     Filter;                 /* Filter */
MQLONG    AuthorityBufferLength;  /* Length of AuthorityBuffer */
MQZAD     AuthorityBuffer;        /* Authority data */
MQLONG    AuthorityDataLength;    /* Length of data returned in
                                     AuthorityBuffer */
MQBYTE    ComponentData[n];       /* Component data */
MQLONG    Continuation;           /* Continuation indicator set by
                                     component */
MQLONG    CompCode;               /* Completion code */
MQLONG    Reason;                 /* Reason code qualifying CompCode */
```

MQZ_FREE_USER - Free user:

This function is provided by a MQZAS_VERSION_5 authorization service component, and is started by the queue manager to free associated allocated resource.

It is started when an application has finished running under all user contexts, for example during an MQDISC MQI call.

The function identifier for this function (for MQZEP) is MQZID_FREE_USER.

Syntax

```
MQZ_FREE_USER( QMgrName , FreeParms , ComponentData , Continuation , CompCode , Reason )
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

FreeParms
Type: MQZFP - input

Free parameters. A structure containing data relating to the resource to be freed. See “MQZFP - Free parameters” on page 3779 for details.

ComponentData
Type: MQBYTE x ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the ComponentDataLength parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation flag. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on other components.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_AUTHENTICATE_USER (QMgrName, SecurityParms, ApplicationContext,
                       IdentityContext, CorrelationPtr, ComponentData,
                       &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;           /* Queue manager name */
MQZFP     FreeParms;          /* Resource to be freed */
MQBYTE    ComponentData[n];   /* Component data */
MQLONG    Continuation;       /* Continuation indicator set by
                                 component */
MQLONG    CompCode;           /* Completion code */
MQLONG    Reason;             /* Reason code qualifying CompCode */
```

MQZ_GET_AUTHORITY - Get authority:

This function is provided by a MQZAS_VERSION_1 authorization service component, and is started by the queue manager to retrieve the authority that an entity has to access the specified object, including (if the entity is a principal) authorities possessed by the groups in which the principal is a member. Authorities from generic profiles are included in the returned authority set.

The function identifier for this function (for MQZEP) is MQZID_GET_AUTHORITY.

Syntax

```
MQZ_GET_AUTHORITY( QMgrName , EntityName , EntityType , ObjectName , ObjectType ,
 Authority , ComponentData , Continuation , CompCode , Reason )
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityName
Type: MQCHAR12 - input

Entity name. The name of the entity whose access to the object is to be retrieved. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

EntityType
Type: MQLONG - input

Entity type. The type of entity specified by *EntityName*. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object to which access is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType
Type: MQLONG - input

Object type. The type of entity specified by *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

MQOT_TOPIC
Topic.

Authority
Type: MQLONG - input

Authority of entity. If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_GET_AUTHORITY, this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_GET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                   ObjectType, &Authority, ComponentData,
                   &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority of entity */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_GET_AUTHORITY_2 - Get authority (extended):

This function is provided by a MQZAS_VERSION_2 authorization service component, and is started by the queue manager to retrieve the authority that an entity has to access the specified object.

The function identifier for this function (for MQZEP) is MQZID_GET_AUTHORITY.

MQZ_GET_AUTHORITY_2 is like MQZ_GET_AUTHORITY, but with the **EntityName** parameter replaced by the **EntityData** parameter.

Syntax

```
MQZ_GET_AUTHORITY_2( QMgrName , EntityData , EntityType , ObjectName , ObjectType ,
 Authority , ComponentData , Continuation , CompCode , Reason )
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityData
Type: MQZED - input

Entity data. Data relating to the entity for which authorization to the object is to be retrieved. See "MQZED - Entity descriptor" on page 3777 for details.

EntityType
Type: MQLONG - input

Entity type. The type of entity specified by *EntityData*. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object for which the entity authority is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType
Type: MQLONG - input

Object type. The type of entity specified by *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

MQOT_TOPIC
Topic.

Authority
Type: MQLONG - input

Authority of entity. If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information about these reason codes, see API reason codes.

Syntax

```
MQZ_GET_AUTHORITY_2 (QMgrName, EntityData, EntityType, ObjectName, ObjectType, Authority,
ComponentData, Continuation, CompCode, Reason)
```

C invocation

```
MQZ_GET_AUTHORITY_2 (QMgrName, &EntityData, EntityType, ObjectName,
                     ObjectType, &Authority, ComponentData,
                     &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQZED     EntityData;        /* Entity data */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority of entity */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_GET_EXPLICIT_AUTHORITY - Get explicit authority:

This function is provided by a MQZAS_VERSION_1 authorization service component, and is started by the queue manager to retrieve the authority that an entity has to access the specified object, including (if the entity is a principal) authorities possessed by the groups in which the principal is a member. Authorities from generic profiles are included in the returned authority set.

On UNIX, for the built-in IBM MQ object authority manager (OAM), the returned authority is that possessed only by the principal's primary group.

The function identifier for this function (for MQZEP) is MQZID_GET_EXPLICIT_AUTHORITY.

Syntax

```
MQZ_GET_EXPLICIT_AUTHORITY( QMgrName , EntityName , EntityType , ObjectName , ObjectType ,
 Authority , ComponentData , Continuation , CompCode , Reason )
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityName
Type: MQCHAR12 - input

Entity name. The name of the entity for which access to the object is to be retrieved. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

EntityType
Type: MQLONG - input

Entity type. The type of entity specified by *EntityName*. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object for which the entity authority is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType
Type: MQLONG - input

Object type. The type of entity specified by *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

MQOT_TOPIC
Topic.

Authority
Type: MQLONG - input

Authority of entity. If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData
Type: MQBYTE x ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_GET_AUTHORITY, this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_GET_EXPLICIT_AUTHORITY (QMgrName, EntityName, EntityType,
                            ObjectName, ObjectType, &Authority,
                            ComponentData, &Continuation,
                            &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority of entity */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_GET_EXPLICIT_AUTHORITY_2 - Get explicit authority (extended):

This function is provided by a MQZAS_VERSION_2 authorization service component, and is started by the queue manager to retrieve the authority that a named group has to access a specified object (but without the additional authority of the **nobody** group), or the authority that the primary group of the named principal has to access a specified object.

The function identifier for this function (for MQZEP) is MQZID_GET_EXPLICIT_AUTHORITY.

MQZ_GET_EXPLICIT_AUTHORITY_2 is like MQZ_GET_EXPLICIT_AUTHORITY, but with the **EntityName** parameter replaced by the **EntityData** parameter.

Syntax

```
MQZ_GET_EXPLICIT_AUTHORITY_2( QMgrName , EntityData , EntityType , ObjectName , ObjectType
, Authority , ComponentData , Continuation , CompCode , Reason )
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityData
Type: MQZED - input

Entity data. Data relating to the entity whose authorization to the object is to be retrieved. See "MQZED - Entity descriptor" on page 3777 for details.

EntityType
Type: MQLONG - input

Entity type. The type of entity specified by *EntityData*. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object for which the entity authority is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType
Type: MQLONG - input

Object type. The type of entity specified by *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

MQOT_TOPIC
Topic.

Authority
Type: MQLONG - input

Authority of entity. If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_GET_EXPLICIT_AUTHORITY_2 (QMgrName, &EntityData, EntityType,
                              ObjectName, ObjectType, &Authority,
                              ComponentData, &Continuation,
                              &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQZED     EntityData;        /* Entity data */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority of entity */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_INIT_AUTHORITY - Initialize authorization service:

This function is provided by an authorization service component, and is started by the queue manager during configuration of the component. It is expected to call MQZEP in order to provide information to the queue manager.

The function identifier for this function (for MQZEP) is MQZID_INIT_AUTHORITY.

Syntax

```
MQZ_INIT_AUTHORITY( Hconfig , Options , QMgrName , ComponentDataLength , ComponentData ,
 Version , CompCode , Reason )
```

Parameters

Hconfig
Type: MQHCONFIG - input

Configuration handle. This handle represents the particular component being initialized. It is to be used by the component when calling the queue manager with the MQZEP function.

Options
Type: MQLONG - input

Initialization options. It must be one of the following values:

MQZIO_PRIMARY
Primary initialization.

MQZIO_SECONDARY
Secondary initialization.

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

ComponentDataLength
Type: MQLONG - input

Length of component data. Length in bytes of the *ComponentData* area. This length is defined in the component configuration data.

ComponentData
Type: MQBYTE x ComponentDataLength - input/output

Component data. This is initialized to all zeros before calling the component primary initialization function. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions (including the initialization function) provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Version
Type: MQLONG - input/output

Version number. On input to the initialization function, this identifies the highest version number that the queue manager supports. The initialization function must change this, if necessary, to the version

of the interface which it supports. If on return the queue manager does not support the version returned by the component, it calls the component MQZ_TERM_AUTHORITY function and makes no further use of this component.

The following values are supported:

MQZAS_VERSION_1
Version 1.

MQZAS_VERSION_2
Version 2.

MQZAS_VERSION_3
Version 3.

MQZAS_VERSION_4
Version 4.

MQZAS_VERSION_5
Version 5.

MQZAS_VERSION_6
Version 6.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_INITIALIZATION_FAILED
(2286, X'8EE') Initialization failed for an undefined reason.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_INIT_AUTHORITY (Hconfig, Options, QMgrName, ComponentDataLength,
                    ComponentData, &Version, &CompCode,
                    &Reason);
```

The parameters passed to the service are declared as follows:

```
MQHCONFIG  Hconfig;              /* Configuration handle */
MQLONG     Options;              /* Initialization options */
MQCHAR48   QMgrName;             /* Queue manager name */
MQLONG     ComponentDataLength;  /* Length of component data */
MQBYTE     ComponentData[n];     /* Component data */
```

```
MQLONG     Version;              /* Version number */
MQLONG     CompCode;             /* Completion code */
MQLONG     Reason;               /* Reason code qualifying CompCode */
```

MQZ_INQUIRE - Inquire authorization service:

This function is provided by a MQZAS_VERSION_5 authorization service component, and is started by the queue manager to query the supported functionality.

Where multiple service components are used, service components are called in reverse order to the order they were installed in.

The function identifier for this function (for MQZEP) is MQZID_INQUIRE.

Syntax

```
MQZ_INQUIRE( QMgrName , SelectorCount , Selectors , IntAttrCount , IntAttrs ,
 CharAttrLength , CharAttrs , SelectorReturned , ComponentData , Continuation , CompCode ,
 Reason )
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

SelectorCount
Type: MQLONG - input

Number of selectors. The number of selectors supplied in the **Selectors** parameter.

The value must be in the range 0 through 256.

Selectors
Type: MQLONGxSelectorCount - input

Array of selectors. Each selector identifies a required attribute and must be one of the following:

- MQIACF_INTERFACE_VERSION (integer)
- MQIACF_USER_ID_SUPPORT (integer)
- MQCACF_SERVICE_COMPONENT (character)

Selectors can be specified in any order. The number of selectors in the array is indicated by the **SelectorCount** parameter.

Integer attributes identified by selectors are returned in the **IntAttrs** parameter in the same order as they appear in *Selectors*.

Character attributes identified by selectors are returned in the **CharAttrs** parameter in the same order as they in appear *Selectors*.

IntAttrCount
Type: MQLONG - input

Number of integer attributes supplied in the IntAttrs parameter.

The value must be in the range 0 through 256.

IntAttrs
Type: MQLONG x IntAttrCount - output

Integer attributes. Array of integer attributes. The integer attributes are returned in the same order as the corresponding integer selectors in the *Selectors* array.

CharAttrCount
Type: MQLONG - input

Length of the character attributes buffer. The length in bytes of the **CharAttrs** parameter.

The value must be at least the sum of the lengths of the requested character attributes. If no character attributes are requested, zero is a valid value.

CharAttrs
Type: MQLONG x CharAttrCount - output

Character attributes buffer. Buffer containing character attributes, concatenated together. The character attributes are returned in the same order as the corresponding character selectors in the *Selectors* array.

The length of the buffer is given by the CharAttrCount parameter.

SelectorReturned
Type: MQLONG x SelectorCount - input

Selector returned. Array of values identifying which attributes have been returned from the set requested for by the selectors in the Selectors parameter. The number of values in this array is indicated by the **SelectorCount** parameter. Each value in the array relates to the selector from the corresponding position in the Selectors array. Each value is one of the following:

MQZSL_RETURNED
The attribute requested by the corresponding selector in the **Selectors** parameter has been returned.

MQZSL_NOT_RETURNED
The attribute requested by the corresponding selector in the **Selectors** parameter has not been returned.

The array is initialized with all values as *MQZSL_NOT_RETURNED*. When an authorization service component returns an attribute, it sets the appropriate value in the array to *MQZSL_NOT_RETURNED*. This allows any other authorization service components, to which the inquire call is made, to identify which attributes have already been returned.

ComponentData
Type: MQBYTE x ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_CHAR_ATTRS_TOO_SHORT
Not enough space for character attributes.

MQRC_INT_COUNT_TOO_SMALL
Not enough space for integer attributes.

If *CompCode* is MQCC_FAILED:

MQRC_SELECTOR_COUNT_ERROR
Number of selectors is not valid.

MQRC_SELECTOR_ERROR
Attribute selector not valid.

MQRC_SELECTOR_LIMIT_EXCEEDED
Too many selectors specified.

MQRC_INT_ATTR_COUNT_ERROR
Number of integer attributes is not valid.

MQRC_INT_ATTRS_ARRAY_ERROR
Integer attributes array not valid.

MQRC_CHAR_ATTR_LENGTH_ERROR
Number of character attributes is not valid.

MQRC_CHAR_ATTRS_ERROR
Character attributes string is not valid.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_INQUIRE (QMgrName, SelectorCount, Selectors, IntAttrCount,
                  &IntAttrs, CharAttrLength, &CharAttrs,
                  SelectorReturned, ComponentData, &Continuation,
                  &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48   QMgrName;              /* Queue manager name */
MQLONG     SelectorCount;         /* Selector count */
MQLONG     Selectors[n];          /* Selectors */
MQLONG     IntAttrCount;          /* IntAttrs count */
MQLONG     IntAttrs[n];           /* Integer attributes */
MQLONG     CharAttrCount;         /* CharAttrs count */
MQLONG     CharAttrs[n];          /* Chatacter attributes */
MQLONG     SelectorReturned[n];   /* Selector returned */
MQBYTE     ComponentData[n];      /* Component data */
MQLONG     Continuation;          /* Continuation indicator set by
                                     component */
MQLONG     CompCode;              /* Completion code */
MQLONG     Reason;                /* Reason code qualifying CompCode */
```

MQZ_REFRESH_CACHE - Refresh all authorizations:

This function is provided by an MQZAS_VERSION_3 authorization service component, and is invoked by the queue manager to refresh the list of authorizations held internally by the component.

The function identifier for this function (for MQZEP) is MQZID_REFRESH_CACHE (8L).

Syntax

```
MQZ_REFRESH_CACHE( QMgrName , ComponentData , Continuation , CompCode , Reason )
```

Parameters

QMgrName

Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

ComponentData

Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation

Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT

Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

C invocation

```
MQZ_REFRESH_CACHE (QMgrName, ComponentData,
                   &Continuation, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_SET_AUTHORITY - Set authority:

This function is provided by a MQZAS_VERSION_1 authorization service component, and is started by the queue manager to set the authority that an entity has to access the specified object.

The function identifier for this function (for MQZEP) is MQZID_SET_AUTHORITY.

Note: This function overrides any existing authorities. To preserve any existing authorities you must set them again with this function.

Syntax

```
MQZ_SET_AUTHORITY( QMgrName , EntityName , EntityType , ObjectName , ObjectType ,
 Authority , ComponentData , Continuation , CompCode , Reason )
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityName
Type: MQCHAR12 - input

Entity name. The name of the entity for which access to the object is to be retrieved. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

EntityType
Type: MQLONG - input

Entity type. The type of entity specified by *EntityName*. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object to which access is required. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType
Type: MQLONG - input

Object type. The type of entity specified by *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

MQOT_TOPIC
Topic.

Authority
Type: MQLONG - input

Authority of entity. If one authority is being set, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authority is being set, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentDatarname>
Type: MQBYTExComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_GET_AUTHORITY, this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_SET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                   ObjectType, Authority, ComponentData,
                   &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority to be checked */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_SET_AUTHORITY_2 - Set authority (extended):

This function is provided by a MQZAS_VERSION_2 authorization service component, and is started by the queue manager to set the authority that an entity has to access the specified object.

The function identifier for this function (for MQZEP) is MQZID_SET_AUTHORITY.

Note: This function overrides any existing authorities. To preserve any existing authorities you must set them again with this function.

MQZ_SET_AUTHORITY_2 is like MQZ_SET_AUTHORITY, but with the **EntityName** parameter replaced by the **EntityData** parameter.

Syntax

```
MQZ_SET_AUTHORITY_2( QMgrName , EntityData , EntityType , ObjectName , ObjectType ,
 Authority , ComponentData , Continuation , CompCode , Reason )
```

Parameters

QMgrName

Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityData

Type: MQZED - input

Entity data. Data relating to the entity whose authorization to the object is to be set. See "MQZED - Entity descriptor" on page 3777 for details.

EntityType

Type: MQLONG - input

Entity type. The type of entity specified by *EntityData*. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object to which the entity authority is to be set. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType
Type: MQLONG - input

Object type. The type of entity specified by *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

MQOT_TOPIC
Topic.

Authority
Type: MQLONG - input

Authority of entity. If one authority is being set, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authority is being set, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_SET_AUTHORITY_2 (QMgrName, &EntityData, EntityType, ObjectName,
                     ObjectType, Authority, ComponentData,
                     &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQZED     EntityData;        /* Entity data */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority to be checked */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
```

```
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_TERM_AUTHORITY - Terminate authorization service:

This function is provided by an authorization service component, and is started by the queue manager when it no longer requires the services of this component. The function must perform any cleanup required by the component.

The function identifier for this function (for MQZEP) is MQZID_TERM_AUTHORITY.

Syntax

```
MQZ_TERM_AUTHORITY( Hconfig , Options , QMgrName , ComponentData , CompCode , Reason )
```

Parameters

Hconfig
Type: MQHCONFIG - input

Configuration handle. This handle represents the particular component being terminated. It is to be used by the component when calling the queue manager with the MQZEP function.

Options
Type: MQLONG - input

Termination options. It must be one of the following values:

MQZTO_PRIMARY
Primary termination.

MQZTO_SECONDARY
Secondary termination.

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

ComponentData
Type: MQBYTE x ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the ComponentDataLength parameter on the MQZ_INIT_AUTHORITY call.

When the MQZ_TERM_AUTHORITY call has completed, the queue manager discards this data.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_TERMINATION_FAILED
(2287, X'8FF') Termination failed for an undefined reason.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_TERM_AUTHORITY (Hconfig, Options, QMgrName, ComponentData,
                    &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQHCONFIG  Hconfig;           /* Configuration handle */
MQLONG     Options;           /* Termination options */
MQCHAR48   QMgrName;          /* Queue manager name */
MQBYTE     ComponentData[n];  /* Component data */
MQLONG     CompCode;          /* Completion code */
MQLONG     Reason;            /* Reason code qualifying CompCode */
```

MQZ_DELETE_NAME - Delete name:

This function is provided by a name service component, and is started by the queue manager to delete an entry for the specified queue.

The function identifier for this function (for MQZEP) is MQZID_DELETE_NAME.

Syntax

```
MQZ_DELETE_NAME( QMgrName , QName , ComponentData , Continuation , CompCode , Reason )
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

QName
Type: MQCHAR48 - input

Queue name. The name of the queue for which an entry is to be deleted. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ComponentData
Type: MQBYTE x ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the ComponentDataLength parameter on the MQZ_INIT_NAME call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. It must be one of the following values:

MQZCI_DEFAULT
Continuation dependent on queue manager.

MQZCI_STOP
Do not continue with next component.

For the **MQZ_DELETE_NAME** command, the queue manager does not attempt to start another component, no matter what is returned in the **Continuation** parameter.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_UNKNOWN_NAME
(2288, X'8F0') Queue name not found.

Note: It might not be possible to return this code if the underlying service responds with success for this case.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_DELETE_NAME (QMgrName, QName, ComponentData, &Continuation,
                 &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  QName;             /* Queue name */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_INIT_NAME - Initialize name service:

This function is provided by a name service component, and is started by the queue manager during configuration of the component. It is expected to call MQZEP in order to provide information to the queue manager.

The function identifier for this function (for MQZEP) is MQZID_INIT_NAME.

Syntax

```
MQZ_INIT_NAME( Hconfig , Options , QMgrName , ComponentDataLength , ComponentData ,
 Version , CompCode , Reason )
```

Parameters

Hconfig
Type: MQHCONFIG - input

Configuration handle. This handle represents the particular component being initialized. It is to be used by the component when calling the queue manager with the MQZEP function.

Options
Type: MQLONG - input

Initialization options. It must be one of the following values:

MQZIO_PRIMARY
Primary initialization.

MQZIO_SECONDARY
Secondary initialization.

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

ComponentDataLength
Type: MQLONG - input

Length of component data. Length in bytes of the *ComponentData* area. This length is defined in the component configuration data.

ComponentData
Type: MQBYTE x ComponentDataLength - input/output

Component data. This is initialized to all zeros before calling the component primary initialization function. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions (including the initialization function) provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Version

Type: MQLONG - input/output

Version number. On input to the initialization function, this identifies the highest version number that the queue manager supports. The initialization function must change this, if necessary, to the version of the interface which it supports. If on return the queue manager does not support the version returned by the component, it calls the component MQZ_TERM_NAME function and makes no further use of this component.

The following values are supported:

MQZAS_VERSION_1
Version 1.

CompCode

Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_INITIALIZATION_FAILED
(2286, X'8EE') Initialization failed for an undefined reason.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_INIT_NAME (Hconfig, Options, QMgrName, ComponentDataLength,
               ComponentData, &Version, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQHCONFIG  Hconfig;              /* Configuration handle */
MQLONG     Options;              /* Initialization options */
MQCHAR48   QMgrName;             /* Queue manager name */
MQLONG     ComponentDataLength;  /* Length of component data */
MQBYTE     ComponentData[n];     /* Component data */
```

```
MQLONG     Version;              /* Version number */
MQLONG     CompCode;             /* Completion code */
MQLONG     Reason;               /* Reason code qualifying CompCode */
```

MQZ_INSERT_NAME - Insert name:

This function is provided by a name service component, and is started by the queue manager to insert an entry for the specified queue, containing the name of the queue manager that owns the queue. If the queue is already defined in the service, the call fails.

The function identifier for this function (for MQZEP) is MQZID_INSERT_NAME.

Syntax

```
MQZ_INSERT_NAME( QMgrName , QName , ResolvedQMgrName , ComponentData , Continuation ,
 CompCode , Reason )
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

QName
Type: MQCHAR48 - input

Queue name. The name of the queue for which an entry is to be inserted. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ResolvedQMgrName
Type: MQCHAR48 - input

Resolved queue manager name. The name of the queue manager to which the queue resolves. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions (including the initialization function) provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_NAME call.

Continuation
Type: MQLONG - input/output

Continuation indicator set by component. For MQZ_INSERT_NAME, the queue manager does not attempt to start another component, whatever is returned in the **Continuation** parameter.

The following values are supported:

MQZCI_DEFAULT
Continuation dependent on queue manager.

MQZCI_STOP
Do not continue with next component.

CompCode

Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_Q_ALREADY_EXISTS
(2290, X'8F2') Queue object already exists.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_INSERT_NAME (QMgrName, QName, ResolvedQMgrName, ComponentData,
                 &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  QName;             /* Queue name */
MQCHAR48  ResolvedQMgrName;  /* Resolved queue manager name */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_LOOKUP_NAME - Lookup name:

This function is provided by a name service component, and is started by the queue manager to retrieve the name of the owning queue manager, for a specified queue.

The function identifier for this function (for MQZEP) is MQZID_LOOKUP_NAME.

Syntax

```
MQZ_LOOKUP_NAME( QMgrName , QName , ResolvedQMgrName , ComponentData , Continuation ,
 CompCode , Reason )
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

QName
Type: MQCHAR48 - input

Queue name. The name of the queue for which an entry is to be resolved. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ResolvedQMgrName
Type: MQCHAR48 - output

Resolved queue manager name. If the function completes successfully, this is the name of the queue manager that owns the queue.

The name returned by the service component must be padded on the right with blanks to the full length of the parameter; the name must not be terminated by a null character, or contain leading or embedded blanks.

ComponentData
Type: MQBYTExComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions (including the initialization function) provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_NAME call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. For MQZ_LOOKUP_NAME, the queue manager specifies whether to start another name service component, as follows:

- If *CompCode* is MQCC_OK, no further components are started, whatever value is returned in *Continuation*.
- If *CompCode* is not MQCC_OK, a further component is started, unless *Continuation* is MQZCI_STOP.

The following values are supported:

MQZCI_DEFAULT
Continuation dependent on queue manager.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_Q_NAME
(2288, X'8F0') Queue name not found.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_LOOKUP_NAME (QMgrName, QName, ResolvedQMgrName, ComponentData,
                 &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  QName;             /* Queue name */
MQCHAR48  ResolvedQMgrName;  /* Resolved queue manager name */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_TERM_NAME - Terminate name service:

This function is provided by a name service component, and is started by the queue manager when it no longer requires the services of this component. The function must perform any cleanup required by the component.

The function identifier for this function (for MQZEP) is MQZID_TERM_NAME.

Syntax

```
MQZ_TERM_NAME( Hconfig , Options , QMgrName , ComponentData , CompCode , Reason )
```

Parameters

Hconfig
Type: MQHCONFIG - input

Configuration handle. This handle represents the particular component being terminated. It is used by the component when calling the queue manager with the MQZEP function.

Options
Type: MQLONG - input

Termination options. It must be one of the following values:

MQZTO_PRIMARY
Primary termination.

MQZTO_SECONDARY
Secondary termination.

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

ComponentData
Type: MQBYTE x ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions (including the initialization function) provided by this component are preserved, and presented the next time one of these component functions is called.

Component data is in shared memory accessible to all processes.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_NAME call.

When the MQZ_TERM_NAME call has completed, the queue manager discards this data.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_TERMINATION_FAILED
(2287, X'8FF') Termination failed for an undefined reason.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_TERM_NAME (Hconfig, Options, QMgrName, ComponentData, &CompCode,
               &Reason);
```

The parameters passed to the service are declared as follows:

```
MQHCONFIG  Hconfig;           /* Configuration handle */
MQLONG     Options;           /* Termination options */
MQCHAR48   QMgrName;          /* Queue manager name */
MQBYTE     ComponentData[n];  /* Component data */
MQLONG     CompCode;          /* Completion code */
MQLONG     Reason;            /* Reason code qualifying CompCode */
```

MQZAC - Application context:

The MQZAC structure is used on the MQZ_AUTHENTICATE_USER call for the *ApplicationContext* parameter. This parameter specifies data related to the calling application.

Table 1 summarizes the fields in the structure.

Table 400. Fields in MQZAC

Field	Description
StrucId	Structure identifier
Version	Structure version number
ProcessId	Process identifier
ThreadId	Thread identifier
ApplName	Application name
UserID	User identifier
EffectiveUserID	Effective user identifier
Environment	Environment
CallerType	Caller type
AuthenticationType	Authentication type
BindType	Bind type

Fields

StrucId
Type: MQCHAR4 - input

Structure identifier. The value is as follows:

MQZAC_STRUC_ID
Identifier for application context structure.

For the C programming language, the constant MQZAC_STRUC_ID_ARRAY is also defined; this has the same value as MQZAC_STRUC_ID, but is an array of characters instead of a string.

Version
Type: MQLONG - input

Structure version number. The value is as follows:

MQZAC_VERSION_1
Version-1 application context structure. The constant MQZAC_CURRENT_VERSION specifies the version number of the current version.

ProcessId
Type: MQPID - input

Process identifier of the application.

ThreadId
Type: MQTID - input

Thread identifier of the application.

ApplName
Type: MQCHAR28 - input

Application name.

UserID
Type: MQCHAR12 - input

User identifier. On UNIX this field specifies the application's real user ID. On Windows this field specifies the application's user ID.

EffectiveUserID
Type: MQCHAR12 - input

Effective user identifier. On UNIX this field specifies the application's effective user ID. On Windows this field is blank.

Environment
Type: MQLONG - input

Environment. This field specifies the environment from which the call was made. The field is one of the following values:

MQXE_COMMAND_SERVER
Command server

MQXE_MQSC
runmqsc command interpreter

MQXE_MCA
Message channel agent MQXE_OTHER

MQXE_OTHER
Undefined environment

CallerType
Type: MQLONG - input

Caller Type. This field specifies the type of program that made the call. The field is one of the following values:

MQXACT_EXTERNAL
The call is external to the queue manager.

MQXACT_INTERNAL
The call is internal to the queue manager.

AuthenticationType
Type: MQLONG - input

Authentication Type. This field specifies the type of authentication being performed. The field is one of the following values:

MQZAT_INITIAL_CONTEXT
The authentication call is due to user context being initialized. This value is used during an MQCONN or MQCONNX call.

MQZAT_CHANGE_CONTEXT
The authentication call is due to the user context being changed. This value is used when the MCA changes the user context. Parent topic: MQZAC -

BindType
Type: MQLONG - input

Bind Type. This field specifies the type of binding in use. The field is one of the following values:

MQCNO_FASTPATH_BINDING
Fastpath binding.

MQCNO_SHARED_BINDING
Shared binding.

MQCNO_ISOLATED_BINDING
Isolated binding.

C declaration

Declare the fields of the structure as follows:

```
typedef struct tagMQZAC MQZAC;
struct tagMQZAC {
  MQCHAR4   StrucId;            /* Structure identifier */
  MQLONG    Version;            /* Structure version number */
  MQPID     ProcessId;          /* Process identifier */
  MQTID     ThreadId;           /* Thread identifier */
  MQCHAR28  ApplName;           /* Application name */
  MQCHAR12  UserID;             /* User identifier */
  MQCHAR12  EffectiveUserID;    /* Effective user identifier */
  MQLONG    Environment;        /* Environment */
  MQLONG    CallerType;         /* Caller type */
  MQLONG    AuthenticationType; /* Authentication type */
  MQLONG    BindType;           /* Bind type */
};
```

MQZAD - Authority data:

The MQZAD structure is used on the MQZ_ENUMERATE_AUTHORITY_DATA call for two parameters, one input and one output.

- MQZAD is used for the **Filter** parameter which is input to the call. This parameter specifies the selection criteria that are to be used to select the authority data returned by the call.
- MQZAD is also used for the **AuthorityBuffer** parameter which is output from the call. This parameter specifies the authorizations for one combination of profile name, object type, and entity.

Table 1. summarizes the fields in the structure.

Table 401. Fields in MQZAD

Field	Description
StrucId	Structure identifier
Version	Structure version number
ProfileName	Process identifier
ObjectType	Thread identifier
Authority	Application name
EntityDataPtr	User identifier
EntityType	Environment
Options	Caller type

Fields

StrucId
Type: MQCHAR4 - input

Structure identifier. The value is as follows:

MQZAC_STRUC_ID
Identifier for application context structure.

For the C programming language, the constant MQZAC_STRUC_ID_ARRAY is also defined; this has the same value as MQZAC_STRUC_ID, but is an array of characters instead of a string.

Version
Type: MQLONG - input

Structure version number. The value is as follows:

MQZAC_VERSION_1
Version-1 application context structure. The constant MQZAC_CURRENT_VERSION specifies the version number of the current version.

The following constant specifies the version number of the current version:

MQZAD_CURRENT_VERSION
Current version of authority data structure.

ProfileName
Type: MQCHAR48 - input

Profile name.

For the *Filter* parameter, this field is the profile name for which authority data is required. If the name is entirely blank up to the end of the field or the first null character, authority data for all profile names is returned.

For the *AuthorityBuffer* parameter, this field is the name of a profile that matches the specified selection criteria.

ObjectType
Type: MQLONG - input

Object type.

For the *Filter* parameter, this field is the object type for which authority data is required. If the value is MQOT_ALL, authority data for all object types is returned.

For the *AuthorityBuffer* parameter, this field is the object type to which the profile identified by the *ProfileName* parameter applies.

The value is one of the following; for the *Filter* parameter, the value MQOT_ALL is also valid:

MQOT_AUTH_INFO
Authentication information

MQOT_CHANNEL
Channel

MQOT_CLNTCONN_CHANNEL
Client connection channel

MQOT_LISTENER
Listener

MQOT_NAMELIST
Namelist

MQOT_PROCESS
Process definition

MQOT_Q
Queue

MQOT_Q_MGR
Queue manager

MQOT_SERVICE
Service

Authority
Type: MQLONG - input

Authority.

For the *Filter* parameter, this field is ignored.

For the *AuthorityBuffer* parameter, this field represents the authorizations that the entity has to the objects identified by *ProfileName* and *ObjectType*. If the entity has only one authority, the field is equal to the appropriate authorization value (MQZAO_* constant). If the entity has more than one authority, the field is the bitwise OR of the corresponding MQZAO_* constants.

EntityDataPtr
Type: PMQZED - input

Address of MQZED structure identifying an entity.

For the *Filter* parameter, this field points to an MQZED structure that identifies the entity for which authority data is required. If *EntityDataPtr* is the null pointer, authority data for all entities is returned.

For the *AuthorityBuffer* parameter, this field points to an MQZED structure that identifies the entity for which authority data has been returned.

EntityType

Type: MQLONG - input

Entity type.

For the *Filter* parameter, this field specifies the entity type for which authority data is required. If the value is MQZAET_NONE, authority data for all entity types is returned.

For the *AuthorityBuffer* parameter, this field specifies the type of the entity identified by the MQZED structure pointed to by the *EntityDataPtr* parameter.

The value is one of the following; for the *Filter* parameter, the value MQZAET_NONE is also valid:

MQZAET_PRINCIPAL
Principal

MQZAET_GROUP
Group

Options

Type: MQAUTHOPT - input

Options. This field specifies options that give control over the profiles that are displayed. One of the following values must be specified:

MQAUTHOPT_NAME_ALL_MATCHING
Displays all profiles

MQAUTHOPT_NAME_EXPLICIT
Displays profiles that have exactly the same name as specified in the *ProfileName* field.

In addition, one of the following must also be specified:

MQAUTHOPT_ENTITY_SET
Display all profiles that are used to calculate the cumulative authority that the entity has to the object specified by the *ProfileName* parameter. The *ProfileName* parameter must not contain any wildcard characters.

If the specified entity is a principal, for each member of the set {entity, groups} the most applicable profile that applies to the object is displayed.

If the specified entity is a group, the most applicable profile from the group that applies to the object is displayed.

If this value is specified, the values of *ProfileName*, *ObjectType*, *EntityType*, and the entity name that is specified in the *EntityDataPtr* MQZED structure, must all be non-blank.

If you have specified MQAUTHOPT_NAME_ALL_MATCHING, you can also specify the following value:

MQAUTHOPT_ENTITY_EXPLICIT
Displays profiles that have exactly the same entity name as the entity name specified in the *EntityDataPtr* MQZED structure.

C declaration

```
typedef struct tagMQZAD MQZAD;
struct tagMQZAD {
  MQCHAR4   StrucId;        /* Structure identifier */
  MQLONG    Version;        /* Structure version number */
  MQCHAR48  ProfileName;    /* Profile name */
  MQLONG    ObjectType;     /* Object type */
  MQLONG    Authority;      /* Authority */
  PMQZED    EntityDataPtr;  /* Address of MQZED structure identifying an
```

```
                              entity */
  MQLONG    EntityType;     /* Entity type */
  MQAUTHOPT Options;        /* Options */
};
```

Fields

MQZED - Entity descriptor:

The MQZED structure is used in a number of authorization service calls to specify the entity for which authorization is to be checked.

Table 1. summarizes the fields in the structure.

Table 402. Fields in MQZED

Field	Description
StrucId	Structure identifier
Version	Version
EntityName Ptr	Entity name
EntityDomainPtr	Entity domain pointer
SecurityId	Security identifier
CorrelationPtr	Correlation pointer

Fields

StrucId
Type: MQCHAR4 - input

Structure identifier. The value is as follows:

MQZED_STRUC_ID
Identifier for entity descriptor structure.

For the C programming language, the constant MQZED_STRUC_ID_ARRAY is also defined; this has the same value as MQZED_STRUC_ID, but is an array of characters instead of a string.

Version
Type: MQLONG - input

Structure version number. The value is as follows:

MQZED_VERSION_1
Version-1 entity descriptor structure.

The following constant specifies the version number of the current version:

MQZED_CURRENT_VERSION
Current version of entity descriptor structure.

EntityNamePtr
Type: PMQCHAR - input

Profile name.

Address of entity name. This is a pointer to the name of the entity whose authorization is to be checked.

EntityDomainPtr
Type: PMQCHAR - input

Address of entity domain name. This is a pointer to the name of the domain containing the definition of the entity whose authorization is to be checked.

SecurityId
Type: MQBYTE40 - input

Authority.

Security identifier. This is the security identifier whose authorization is to be checked.

CorrelationPtr
Type: MQPTR - input

Correlation pointer. This facilitates the passing of correlational data between the authenticate user function and other appropriate OAM functions.

C declaration

```
typedef struct tagMQZED MQZED;
struct tagMQZED {
  MQCHAR4   StrucId;          /* Structure identifier */
  MQLONG    Version;          /* Structure version number */
  PMQCHAR   EntityNamePtr;    /* Address of entity name */
  PMQCHAR   EntityDomainPtr;  /* Address of entity domain name */
  MQBYTE40  SecurityId;       /* Security identifier */
  MQPTR     CorrelationPtr;   /* Address of correlation data */
```

Fields

MQZEP - Add component entry point:

A service component starts this function, during initialization, to add an entry point to the entry point vector for that service component.

Syntax

MQZEP (*Hconfig* , *Function* , *EntryPoint* , *CompCode* , *Reason*)

Parameters

Hconfig
Type: MQHCONFIG - input

Configuration handle. This handle represents the component that is being configured for this particular installable service. It must be the same as the component passed to the component configuration function by the queue manager on the component initialization call.

Function
Type: MQLONG - input

Function identifier. Valid values for this are defined for each installable service.

If MQZEP is called more than once for the same function, the last call made provides the entry point that is used.

EntryPoint
Type: PMQFUNC - input

Function entry point. This is the address of the entry point provided by the component to perform the function.

The value NULL is valid, and indicates that the function is not provided by this component. NULL is assumed for entry points that are not defined using MQZEP.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_FUNCTION_ERROR
(2281, X'8E9') Function identifier not valid.

MQRC_HCONFIG_ERROR
(2280, X'8E8') Configuration handle not valid.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZEP (Hconfig, Function, EntryPoint, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONFIG  Hconfig;     /* Configuration handle */
MQLONG     Function;    /* Function identifier */
PMQFUNC    EntryPoint;  /* Function entry point */
MQLONG     CompCode;    /* Completion code */
MQLONG     Reason;      /* Reason code qualifying CompCode */
```

MQZFP - Free parameters:

The MQZFP structure is used on the MQZ_FREE_USER call for the *FreeParms* parameter. This parameter specifies data related to resource to be freed.

Table 1. summarizes the fields in the structure.

Table 403. Fields in MQZFP

Field	Description
StrucId	Structure identifier
Version	Version
Reserved	Reserved field
CorrelationPtr	Correlation pointer

Fields

StrucId
Type: MQCHAR4 - input

Structure identifier. The value is as follows:

MQZIC_STRUC_ID
Identifier for identity context structure. For the C programming language, the constant MQZIC_STRUC_ID_ARRAY is also defined; this has the same value as MQZIC_STRUC_ID, but is an array of characters instead of a string.

Version
Type: MQLONG - input

Structure version number. The value is as follows:

MQZFP_VERSION_1
Version-1 free parameters structure.

The following constant specifies the version number of the current version:

MQZFP_CURRENT_VERSION
Current version of free parameters structure.

Reserved
Type: MQBYTE8 - input

Reserved field. The initial value is null.

CorrelationPtr
Type: MQPTR - input

Correlation pointer. Address of correlation data relating to the resource to be freed.

C declaration

```
typedef struct tagMQZFP MQZFP;
struct tagMQZFP {
  MQCHAR4   StrucId;          /* Structure identifier */
  MQLONG    Version;          /* Structure version number */
  MQBYTE8   Reserved;         /* Reserved field */
  MQPTR     CorrelationPtr;   /* Address of correlation data */
};
```

Fields

MQZIC - Identity context:

The MQZIC structure is used on the MQZ_AUTHENTICATE_USER call for the *IdentityContext* parameter.

The MQZIC structure contains identity context information, which identifies the user of the application that first put the message on a queue:

- The queue manager fills the *UserIdentifier* field with a name that identifies the user, the way that the queue manager can do this depends on the environment in which the application is running.
- The queue manager fills the *AccountingToken* field with a token or number that it determined from the application that put the message.
- Applications can use the *ApplIdentityData* field for any extra information that they want to include about the user (for example, an encrypted password).

Suitably authorized applications can set the identity context using the MQZ_AUTHENTICATE_USER function.

A Windows systems security identifier (SID) is stored in the *AccountingToken* field when a message is created under IBM MQ for Windows. The SID can be used to supplement the *UserIdentifier* field and to establish the credentials of a user.

Table 1. summarizes the fields in the structure.

Table 404. Fields in MQZIC

Field	Description
StrucId	Structure identifier
Version	Version
UserIdentifier	User identifier
AccountingToken	Accounting token
ApplIdentityData	Application identity data

Fields

StrucId

Type: MQCHAR4 - input

Structure identifier. The value is as follows:

MQZIC_STRUC_ID
Identifier for identity context structure. For the C programming language, the constant MQZIC_STRUC_ID_ARRAY is also defined; this has the same value as MQZIC_STRUC_ID, but is an array of characters instead of a string.

Version

Type: MQLONG - input

Structure version number. The value is as follows:

MQZIC_VERSION_1
Version-1 identity context structure.

The following constant specifies the version number of the current version:

MQZIC_CURRENT_VERSION
Current version of identity context structure.

UserIdentifier

Type: MQCHAR12 - input

User identifier. This is part of the identity context of the message. *UserIdentifier* specifies the user identifier of the application that originated the message. The queue manager treats this information as character data, but does not define the format of it. For more information on the *UserIdentifier* field, see "UserIdentifier (MQCHAR12)" on page 2436.

AccountingToken

Type: MQBYTE32 - input

Accounting token. This is part of the identity context of the message. *AccountingToken* allows an application to cause work done as a result of the message to be appropriately charged. The queue manager treats this information as a string of bits and does not check its content. For more information on the *AccountingToken* field, see "AccountingToken (MQBYTE32)" on page 2390.

ApplIdentityData

Type: MQCHAR32 - input

Application data relating to identity. This is part of the identity context of the message. ApplIdentityData is information that is defined by the application suite that can be used to provide additional information about the origin of the message. For example, it could be set by applications running with suitable user authority to indicate whether the identity data is trusted. For more information on the ApplIdentityData field, see "ApplIdentityData (MQCHAR32)" on page 2392.

C declaration

```
typedef struct tagMQZED MQZED;
struct tagMQZED {
  MQCHAR4   StrucId;          /* Structure identifier */
  MQLONG    Version;          /* Structure version number */
  MQCHAR12  UserIdentifier;   /* User identifier */
  MQBYTE32  AccountingToken;  /* Accounting token */
  MQCHAR32  ApplIdentityData; /* Application data relating to identity */
};
```

Fields

Installable services interface reference information on IBM i

IBM i

Use this information to understand the reference information for the installable services for IBM i.

For each function there is a description, including the function identifier (for MQZEP).

The *parameters* are shown listed in the order they must occur. They must all be present.

Each parameter name is followed by its data type in parentheses. These are the elementary data types described in "Elementary data types" on page 3000.

The C language invocation is also given, after the description of the parameters.

Related information:

IBM i Installable services and components for IBM i

ULW Installable services and components for UNIX, Linux and Windows

Installable services reference information for UNIX, Linux and Windows
This collection of topics provides reference information for the installable services.

MQZEP (Add component entry point) on IBM i: IBM i

This function is invoked by a service component, during initialization, to add an entry point to the entry point vector for that service component.

Syntax

```
MQZEP (Hconfig, Function, EntryPoint, CompCode, Reason)
```

Parameters

The MQZEP call has the following parameters.

Hconfig (MQHCONFIG) - input
Configuration handle.

This handle represents the component which is being configured for this particular installable service. It must be the same as the one passed to the component configuration function by the queue manager on the component initialization call.

Function (MQLONG) - input
Function identifier.

Valid values for this are defined for each installable service. If MQZEP is called more than once for the same function, the last call made provides the entry point which is used.

EntryPoint (PMQFUNC) - input
Function entry point.

This is the address of the entry point provided by the component to perform the function. The value NULL is valid, and indicates that the function is not provided by this component. NULL is assumed for entry points which are not defined using MQZEP.

CompCode (MQLONG) - output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:
MQRC_FUNCTION_ERROR
(2281, X'8E9') Function identifier not valid.
MQRC_HCONFIG_ERROR
(2280, X'8E8') Configuration handle not valid.

For more information on these reason codes, see Reason codes.

C invocation

```
MQZEP (Hconfig, Function, EntryPoint, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONFIG  Hconfig;     /* Configuration handle */
MQLONG     Function;    /* Function identifier */
PMQFUNC    EntryPoint;  /* Function entry point */
MQLONG     CompCode;    /* Completion code */
MQLONG     Reason;      /* Reason code qualifying CompCode */
```

MQHCONFIG (Configuration handle) on IBM i: IBM i

The MQHCONFIG data type represents a configuration handle, that is, the component that is being configured for a particular installable service. A configuration handle must be aligned on its natural boundary.

Applications must test variables of this type for equality only.

C declaration

```
typedef void MQPOINTER MQHCONFIG;
```

PMQFUNC (Pointer to function) on IBM i: IBM i

Pointer to a function.

C declaration

```
typedef void MQPOINTER PMQFUNC;
```

MQZ_AUTHENTICATE_USER (Authenticate user) on IBM i: IBM i

This function is provided by a MQZAS_VERSION_5 authorization service component. It is invoked by the queue manager to authenticate a user, or to set identity context fields.

It is invoked when an IBM MQ user application context is established. This happens during connect calls at the point where the application's user context is initialized, and at each point where the application's user context is changed. Each time a connect call is made, the application's user context information is reacquired in the *IdentityContext* field.

The function identifier for this function (for MQZEP) is MQZID_AUTHENTICATE_USER.

Syntax

MQZ_AUTHENTICATE_USER *(QMgrName, SecurityParms, ApplicationContext, IdentityContext, CorrelationPtr, ComponentData, Continuation, CompCode, Reason)*

Parameters

The MQZ_AUTHENTICATE_USER call has the following parameters.

QMgrName (MQCHAR48) - input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character. The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

SecurityParms (MQCSP) - input
Security parameters.

Data relating to the user ID, password, and authentication type.

During an MQCONN MQI call this parameter contains null, or default values.

ApplicationContext (MQZAC) - input
Application context.

Data relating to the calling application. See "MQZAC (Application context) on IBM i" on page 3814 for details. During every MQCONN or MQCONNX MQI call, the user context information in the MQZAC structure is reacquired.

IdentityContext (MQZIC) - input/output
Identity context.

On input to the authenticate user function, this identifies the current identity context. The authenticate user function can change this, at which point the queue manager adopts the new identity context. See "MQZIC (Identity context) on IBM i" on page 3822 for more details on the MQZIC structure.

CorrelationPtr (MQPTR) - output
Correlation pointer.

Specifies the address of any correlation data. This pointer is then passed on to other OAM calls.

ComponentData (MQBYTE x ComponentDataLength) - input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this components functions is called. The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) - output
Continuation flag.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on other components.

MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) - output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

For more information about these reason codes, see Reason codes.

C invocation

```
MQZ_AUTHENTICATE_USER (QMgrName, SecurityParms, ApplicationContext,
                  IdentityContext, &CorrelationPtr, ComponentData,
                  &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;           /* Queue manager name */
MQCSP     SecurityParms;      /* Security parameters */
MQZAC     ApplicationContext; /* Application context */
MQZIC     IdentityContext;    /* Identity context */
MQPTR     CorrelationPtr;     /* Correlation pointer */
MQBYTE    ComponentData[n];   /* Component data */
MQLONG    Continuation;       /* Continuation indicator set by
                                 component */
MQLONG    CompCode;           /* Completion code */
MQLONG    Reason;             /* Reason code qualifying CompCode */
```

MQZ_CHECK_AUTHORITY (Check authority) on IBM i: 

This function is provided by a MQZAS_VERSION_1 authorization service component, and is invoked by the queue manager to check whether an entity has authority to perform a particular action, or actions, on a specified object.

The function identifier for this function (for MQZEP) is MQZID_CHECK_AUTHORITY.

Syntax

```
MQZ_CHECK_AUTHORITY (QMgrName, EntityName, EntityType,
        ObjectName, ObjectType, Authority, ComponentData, Continuation, CompCode, Reason)
```

Parameters

The MQZ_CHECK_AUTHORITY call has the following parameters.

QMgrName (MQCHAR48) - input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character. The queue manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityName (MQCHAR12) - input
Entity name.

The name of the entity whose authorization to the object is to be checked. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

It is not essential for this entity to be known to the underlying security service. If it is not known, then the authorizations of the special **nobody** group (to which all entities are assumed to belong) are used for the check. An all-blank name is valid and can be used in this way.

EntityType (MQLONG) - input
Entity type.

The type of entity specified by *EntityName*. It is one of the following:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName (MQCHAR48) - input
Object name.

The name of the object to which access is required. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) - input
Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

Authority (MQLONG) - input
Authority to be checked.

If one authorization is being checked, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authorization is being checked, it is the bitwise OR of the corresponding MQZAO_* constants.

The following authorizations apply to use of the MQI calls:

MQZAO_CONNECT
Ability to use the MQCONN call.

MQZAO_BROWSE
Ability to use the MQGET call with a browse option.

This allows the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_MSG_UNDER_CURSOR, or MQGMO_BROWSE_NEXT option to be specified on the MQGET call.

MQZAO_INPUT
Ability to use the MQGET call with an input option.

This allows the MQOO_INPUT_SHARED, MQOO_INPUT_EXCLUSIVE, or MQOO_INPUT_AS_Q_DEF option to be specified on the MQOPEN call.

MQZAO_OUTPUT
Ability to use the MQPUT call.

This allows the MQOO_OUTPUT option to be specified on the MQOPEN call.

MQZAO_INQUIRE
Ability to use the MQINQ call.

This allows the MQOO_INQUIRE option to be specified on the MQOPEN call.

MQZAO_SET
Ability to use the MQSET call.

This allows the MQOO_SET option to be specified on the MQOPEN call.

MQZAO_PASS_IDENTITY_CONTEXT
Ability to pass identity context.

This allows the MQOO_PASS_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_PASS_ALL_CONTEXT
Ability to pass all context.

This allows the MQOO_PASS_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_IDENTITY_CONTEXT
Ability to set identity context.

This allows the MQOO_SET_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_ALL_CONTEXT
Ability to set all context.

This allows the MQOO_SET_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_ALTERNATE_USER_AUTHORITY
Ability to use alternate user authority.

This allows the MQOO_ALTERNATE_USER_AUTHORITY option to be specified on the MQOPEN call, and the MQPMO_ALTERNATE_USER_AUTHORITY option to be specified on the MQPUT1 call.

MQZAO_ALL_MQI
All of the MQI authorizations.

This enables all of the authorizations described previously.

The following authorizations apply to administration of a queue manager:

MQZAO_CREATE
Ability to create objects of a specified type.

MQZAO_DELETE
Ability to delete a specified object.

MQZAO_DISPLAY
Ability to display the attributes of a specified object.

MQZAO_CHANGE
Ability to change the attributes of a specified object.

MQZAO_CLEAR
Ability to delete all messages from a specified queue.

MQZAO_AUTHORIZE
Ability to authorize other users for a specified object.

MQZAO_CONTROL
Ability to start, stop, or ping a non-client channel object.

MQZAO_CONTROL_EXTENDED
Ability to reset a sequence number, or resolve an indoubt message on a non-client channel object.

MQZAO_ALL_ADMIN
All of the administration authorizations, other than MQZAO_CREATE.

The following authorizations apply to both use of the MQI and to administration of a queue manager:

MQZAO_ALL
All authorizations, other than MQZAO_CREATE.

MQZAO_NONE
No authorizations.

ComponentData (MQBYTE x ComponentDataLength) - input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) - output
Continuation indicator set by component.

The following values can be specified:
MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY this has the same effect as MQZCI_STOP.
MQZCI_CONTINUE
Continue with next component.
MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) - output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:
MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.
MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see Reason codes.

C invocation

```
MQZ_CHECK_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                     ObjectType, Authority, ComponentData,
                     &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority to be checked */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_CHECK_PRIVILEGED - Check if user is privileged:

This function is provided by an MQZAS_VERSION_6 authorization service component, and is invoked by the queue manager to determine whether a specified user is a privileged user.

The function identifier for this function (for MQZEP) is MQZID_CHECK_PRIVILEGED.

Syntax

```
MQZ_CHECK_PRIVILEGED( QMgrName , EntityData , EntityType , ComponentData , Continuation ,
 CompCode , Reason )
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityData
Type: MQZED - input

Entity data. Data relating to the entity that is to be checked. For more information, see "MQZED - Entity descriptor" on page 3777.

EntityType
Type: MQLONG - input

Entity type. The type of entity specified by EntityData. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ComponentData
Type: MQBYTExComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

If the call to a component fails (that is, *CompCode* returns MQCC_FAILED), and the *Continuation* parameter is MQZCI_DEFAULT or MQZCI_CONTINUE, the queue manager continues to call other components if there are any.

If the call succeeds (that is, *CompCode* returns MQCC_OK) no other components are called no matter what the setting of *Continuation* is.

If the call fails and the *Continuation* parameter is MQZCI_STOP then no other components are called and the error is returned to the queue manager. Components have no knowledge of previous calls, so the *Continuation* parameter is always set to MQZCI_DEFAULT before the call.

CompCode

Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_PRIVILEGED
(2584, X'A18') This user is not a privileged user ID.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_CHECK_PRIVILEGED (QMgrName, &EntityData, EntityType,
                      ComponentData, &Continuation,
                      &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQZED     EntityData;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_COPY_ALL_AUTHORITY (Copy all authority) on IBM i: IBM i

This function is provided by an authorization service component. It is invoked by the queue manager to copy all of the authorizations that are currently in force for a reference object to another object.

The function identifier for this function (for MQZEP) is MQZID_COPY_ALL_AUTHORITY.

Syntax

```
MQZ_COPY_ALL_AUTHORITY (QMgrName, RefObjectName, ObjectName,
        ObjectType, ComponentData, Continuation, CompCode, Reason)
```

Parameters

The MQZ_COPY_ALL_AUTHORITY call has the following parameters.

QMgrName (MQCHAR48) - input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

RefObjectName (MQCHAR48) - input
Reference object name.

The name of the reference object, the authorizations for which are to be copied. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

ObjectName (MQCHAR48) - input
Object name.

The name of the object for which accesses are to be set. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

ObjectType (MQLONG) - input
Object type.

The type of object specified by *RefObjectName* and *ObjectName*. It is one of the following:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

ComponentData (MQBYTE x ComponentDataLength) - input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) - output
Continuation indicator set by component.

The following values can be specified:
MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_COPY_ALL_AUTHORITY this has the same effect as MQZCI_STOP.
MQZCI_CONTINUE
Continue with next component.
MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) - output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:
MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.
MQRC_UNKNOWN_REF_OBJECT
(2294, X'8F6') Reference object unknown.

For more information on these reason codes, see Reason codes.

C invocation

```
MQZ_COPY_ALL_AUTHORITY (QMgrName, RefObjectName, ObjectName, ObjectType,
                        ComponentData, &Continuation, &CompCode,
                        &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  RefObjectName;     /* Reference object name */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_DELETE_AUTHORITY (Delete authority) on IBM i: IBM i

This function is provided by an authorization service component, and is invoked by the queue manager to delete all of the authorizations associated with the specified object.

The function identifier for this function (for MQZEP) is MQZID_DELETE_AUTHORITY.

Syntax

MQZ_DELETE_AUTHORITY *(QMgrName, ObjectName, ObjectType, ComponentData, Continuation, CompCode, Reason)*

Parameters

The MQZ_DELETE_AUTHORITY call has the following parameters.

QMgrName (MQCHAR48) - input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

ObjectName (MQCHAR48) - input
Object name.

The name of the object for which accesses are to be deleted. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) - input
Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

ComponentData (MQBYTE x ComponentDataLength) - input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) - output
Continuation indicator set by component.

The following values can be specified:
MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_DELETE_AUTHORITY this has the same effect as MQZCI_STOP.
MQZCI_CONTINUE
Continue with next component.
MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) - output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:
MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see Reason codes.

C invocation

```
MQZ_DELETE_AUTHORITY (QMgrName, ObjectName, ObjectType, ComponentData,
                      &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_ENUMERATE_AUTHORITY_DATA (Enumerate authority data) on IBM i: IBM i

This function is provided by an MQZAS_VERSION_4 authorization service component, and is invoked repeatedly by the queue manager to retrieve all of the authority data that matches the selection criteria specified on the first invocation.

The function identifier for this function (for MQZEP) is MQZID_ENUMERATE_AUTHORITY_DATA.

Syntax

MQZ_ENUMERATE_AUTHORITY_DATA *(QMgrName, StartEnumeration, Filter, AuthorityBufferLength, AuthorityBuffer, AuthorityDataLength, ComponentData, Continuation, CompCode, Reason)*

Parameters

The MQZ_ENUMERATE_AUTHORITY_DATA call has the following parameters.

QMgrName (MQCHAR48) - input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

StartEnumeration (MQLONG) - input
Flag indicating whether call should start enumeration.

This indicates whether the call should start the enumeration of authority data, or continue the enumeration of authority data started by a previous call to MQZ_ENUMERATE_AUTHORITY_DATA. The value is one of the following:

MQZSE_START
Start enumeration.

The call is invoked with this value to start the enumeration of authority data. The **Filter** parameter specifies the selection criteria to be used to select the authority data returned by this and successive calls.

MQZSE_CONTINUE
Continue enumeration.

The call is invoked with this value to continue the enumeration of authority data. The **Filter** parameter is ignored in this case, and can be specified as the null pointer (the selection criteria are determined by the **Filter** parameter specified by the call that had *StartEnumeration* set to MQZSE_START).

Filter (MQZAD) - input
Filter.

If *StartEnumeration* is MQZSE_START, *Filter* specifies the selection criteria to be used to select the authority data to return. If *Filter* is the null pointer, no selection criteria are used, that is, all authority data is returned. See "MQZAD (Authority data) on IBM i" on page 3817 for details of the selection criteria that can be used.

If *StartEnumeration* is MQZSE_CONTINUE, *Filter* is ignored, and can be specified as the null pointer.

AuthorityBufferLength (MQLONG) - input
Length of *AuthorityBuffer*.

This is the length in bytes of the **AuthorityBuffer** parameter. The authority buffer must be big enough to accommodate the data to be returned.

AuthorityBuffer (MQZAD) - output
Authority data.

This is the buffer in which the authority data is returned. The buffer must be big enough to accommodate an MQZAD structure, an MQZED structure, plus the longest entity name and longest domain name defined.

Note: This parameter is defined as an MQZAD, as the MQZAD always occurs at the start of the buffer. However, if the buffer is actually declared as an MQZAD, the buffer will be too small - it needs to be bigger than an MQZAD so that it can accommodate the MQZAD, MQZED, plus entity and domain names.

AuthorityDataLength (MQLONG) - output
Length of data returned in *AuthorityBuffer*.

This is the length of the data returned in *AuthorityBuffer*. If the authority buffer is too small, *AuthorityDataLength* is set to the length of the buffer required, and the call returns completion code MQCC_FAILED and reason code MQRC_BUFFER_LENGTH_ERROR.

ComponentData (MQBYTE x ComponentDataLength) - input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) - output
Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_ENUMERATE_AUTHORITY_DATA this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) - output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:
MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.
MQRC_NO_DATA_AVAILABLE
(2379, X'94B') No data available.
MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

For more information on these reason codes, see Reason codes.

C invocation

```
MQZ_ENUMERATE_AUTHORITY_DATA (QMgrName, StartEnumeration, &Filter,
                              AuthorityBufferLength,
                              &AuthorityBuffer,
                              &AuthorityDataLength, ComponentData,
                              &Continuation, &CompCode,
                              &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;               /* Queue manager name */
MQLONG    StartEnumeration;       /* Flag indicating whether call should
                                     start enumeration */
MQZAD     Filter;                 /* Filter */
MQLONG    AuthorityBufferLength;  /* Length of AuthorityBuffer */
MQZAD     AuthorityBuffer;        /* Authority data */
MQLONG    AuthorityDataLength;    /* Length of data returned in
                                     AuthorityBuffer */
MQBYTE    ComponentData[n];       /* Component data */
MQLONG    Continuation;           /* Continuation indicator set by
                                     component */
MQLONG    CompCode;               /* Completion code */
MQLONG    Reason;                 /* Reason code qualifying CompCode */
```

MQZ_FREE_USER - Free user:

This function is provided by a MQZAS_VERSION_5 authorization service component, and is invoked by the queue manager to free associated allocated resource. It is invoked when an application has finished running under all user contexts, for example during an MQDISC MQI call.

The function identifier for this function (for MQZEP) is MQZID_FREE_USER.

MQZ_GET_AUTHORITY (Get authority) on IBM i: IBM i

This function is provided by a MQZAS_VERSION_1 authorization service component, and is invoked by the queue manager to retrieve the authority that an entity has to access the specified object.

The function identifier for this function (for MQZEP) is MQZID_GET_AUTHORITY.

Syntax

```
MQZ_GET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
      ObjectType, Authority, ComponentData, Continuation, CompCode, Reason)
```

Parameters

The MQZ_GET_AUTHORITY call has the following parameters.

QMgrName (MQCHAR48) - input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityName (MQCHAR12) - input
Entity name.

The name of the entity whose access to the object is to be retrieved. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

EntityType (MQLONG) - input
Entity type.

The type of entity specified by *EntityName*. The following value can be specified:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName (MQCHAR48) - input
Object name.

The name of the object for which the entity's authority is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) - input
Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

Authority (MQLONG) - output
Authority of entity.

If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData (MQBYTE x ComponentDataLength) - input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) - output
Continuation indicator set by component.

The following values can be specified:
MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_GET_AUTHORITY this has the same effect as MQZCI_CONTINUE.
MQZCI_CONTINUE
Continue with next component.
MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) - output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:
MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.
MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.
MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information on these reason codes, see Reason codes.

C invocation

```
MQZ_GET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                   ObjectType, &Authority, ComponentData,
                   &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority of entity */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_GET_EXPLICIT_AUTHORITY (Get explicit authority) on IBM i: IBM i

This function is provided by a MQZAS_VERSION_1 authorization service component, and is invoked by the queue manager to retrieve the authority that a named group has to access a specified object (but without the additional authority of the **nobody** group), or the authority that the primary group of the named principal has to access a specified object.

The function identifier for this function (for MQZEP) is MQZID_GET_EXPLICIT_AUTHORITY.

Syntax

MQZ_GET_EXPLICIT_AUTHORITY *(QMgrName, EntityName, EntityType, ObjectName, ObjectType, Authority, ComponentData, Continuation, CompCode, Reason)*

Parameters

The MQZ_GET_EXPLICIT_AUTHORITY call has the following parameters.

QMgrName (MQCHAR48) - input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityName (MQCHAR12) - input
Entity name.

The name of the entity from which access to the object is to be retrieved. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

EntityType (MQLONG) - input
Entity type.

The type of entity specified by *EntityName*. The following value can be specified:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName (MQCHAR48) - input
Object name.

The name of the object for which the entity's authority is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) - input
Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

Authority (MQLONG) - output
Authority of entity.

If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData (MQBYTE x ComponentDataLength) - input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) - output
Continuation indicator set by component.

The following values can be specified:
MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_GET_EXPLICIT_AUTHORITY this has the same effect as MQZCI_CONTINUE.
MQZCI_CONTINUE
Continue with next component.
MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) - output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:
MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.
MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information on these reason codes, see Reason codes.

C invocation

```
MQZ_GET_EXPLICIT_AUTHORITY (QMgrName, EntityName, EntityType,
                            ObjectName, ObjectType, &Authority,
                            ComponentData, &Continuation,
                            &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority of entity */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_INIT_AUTHORITY (Initialize authorization service) on IBM i: 

This function is provided by an authorization service component, and is invoked by the queue manager during configuration of the component. It is expected to call MQZEP in order to provide information to the queue manager.

The function identifier for this function (for MQZEP) is MQZID_INIT_AUTHORITY.

Syntax

MQZ_INIT_AUTHORITY *(Hconfig, Options, QMgrName, ComponentDataLength, ComponentData, Version, CompCode, Reason)*

Parameters

The MQZ_INIT_AUTHORITY call has the following parameters.

Hconfig (MQHCONFIG) - input
Configuration handle.

This handle represents the particular component being initialized. It is to be used by the component when calling the queue manager with the MQZEP function.

Options (MQLONG) - input
Initialization options.

It is one of the following:
MQZIO_PRIMARY
Primary initialization.
MQZIO_SECONDARY
Secondary initialization.

QMgrName (MQCHAR48) - input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

ComponentDataLength (MQLONG) - input
Length of component data.

Length in bytes of the *ComponentData* area. This length is defined in the component configuration data.

ComponentData (MQBYTE x ComponentDataLength) - input/output
Component data.

This is initialized to all zeros before calling the component's primary initialization function. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions (including the initialization function) provided by this component are preserved, and presented the next time one of this component's functions is called.

Version (MQLONG) - input/output
Version number.

On input to the initialization function, this identifies the *highest* version number that the queue manager supports. The initialization function must change this, if necessary, to the version of the interface which *it* supports. If on return the queue manager does not support the version returned by the component, it calls the component's MQZ_TERM_AUTHORITY function and makes no further use of this component.

The following values are supported:
MQZAS_VERSION_1
Version 1.
MQZAS_VERSION_2
Version 2.
MQZAS_VERSION_3
Version 3.
MQZAS_VERSION_4
Version 4.
MQZAS_VERSION_5
Version 5.
MQZAS_VERSION_6
Version 6.

CompCode (MQLONG) - output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:
MQRC_INITIALIZATION_FAILED
(2286, X'8EE') Initialization failed for an undefined reason.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see Reason codes.

C invocation

```
MQZ_INIT_AUTHORITY (Hconfig, Options, QMgrName, ComponentDataLength,
                    ComponentData, &Version, &CompCode,
                    &Reason);
```

The parameters passed to the service are declared as follows:

```
MQHCONFIG  Hconfig;              /* Configuration handle */
MQLONG     Options;              /* Initialization options */
MQCHAR48   QMgrName;             /* Queue manager name */
MQLONG     ComponentDataLength;  /* Length of component data */
MQBYTE     ComponentData[n];     /* Component data */
MQLONG     Version;              /* Version number */
MQLONG     CompCode;             /* Completion code */
MQLONG     Reason;               /* Reason code qualifying CompCode */
```

MQZ_INQUIRE (Inquire authorization service) on IBM i: IBM i

This function is provided by a MQZAS_VERSION_5 authorization service component, and is invoked by the queue manager to query the supported functionality. Where multiple service components are used, service components are called in reverse order to the order they were installed in.

The function identifier for this function (for MQZEP) is MQZID_INQUIRE.

Syntax

MQZ_INQUIRE
(QMgrName, SelectorCount, Selectors, IntAttrCount, IntAttrs, CharAttrLength, CharAttrs, SelectorReturned, ComponentData, Continuation, CompCode, Reason)

Parameters

The MQZ_INQUIRE call has the following parameters.

QMgrName (MQCHAR48) - input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

SelectorCount (MQLONG) - input
Number of selectors.

The number of selectors supplied in the Selectors parameter.

The value must be between zero and 256.

Selectors (MQLONG x SelectorCount) - input
Selectors.

Array of selectors. Each selector identifies a required attribute and must be of one of the following types:

- MQIACF_* (integer)

- MQCACF_* (character)

Selectors can be specified in any order. The number of selectors in the array is indicated by the SelectorCount parameter.

Integer attributes identified by selectors are returned in the IntAttrs parameter in the same order as they appear in Selectors.

Character attributes identified by selectors are returned in the CharAttrs parameter in the same order as they in appear Selectors.

IntAttrCount (MQLONG) - input
Number of integer attributes.

The number of integer attributes supplied in the IntAttrs parameter.

The value must be in the range 0 through 256.

IntAttrs (MQLONG x IntAttrCount) - output
Integer attributes.

Array of integer attributes. The integer attributes are returned in the same order as the corresponding integer selectors in the Selectors array.

CharAttrCount (MQLONG) - input
Length of the character attributes buffer.

The length in bytes of the CharAttrs parameter.

The value must at least sum of the lengths of the requested character attributes. If no character attributes are requested, zero is a valid value.

CharAttrs (MQLONG x CharAttrCount) - output
Character attributes buffer.

Buffer containing character attributes, concatenated together. The character attributes are returned in the same order as the corresponding character selectors in the Selectors array.

The length of the buffer is given by the CharAttrCount parameter.

SelectorReturned (MQLONGxSelectorCount) - input
Selector returned.

Array of values identifying which attributes have been returned from the set requested for by the selectors in the Selectors parameter. The number of values in this array is indicated by the SelectorCount parameter. Each value in the array relates to the selector from the corresponding position in the Selectors array. Each value is one of the following:

MQZSL_RETURNED
The attribute requested by the corresponding selector in the Selectors parameter has been returned.

MQZSL_NOT_RETURNED
The attribute requested by the corresponding selector in the Selectors parameter has not been returned.

The array is initialized with all values as *MQZSL_NOT_RETURNED*. When an authorization service component returns an attribute, it sets the appropriate value in the array to *MQZSL_RETURNED*. This allows any other authorization service components, to which the inquire call is made, to identify which attributes have already been returned.

ComponentData (MQBYTE x ComponentDataLength) - input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) - output
Continuation flag.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on other components.

MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) - output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_CHAR_ATTRS_TOO_SHORT
Not enough space for character attributes.

MQRC_INT_COUNT_TOO_SMALL
Not enough space for integer attributes.

If *CompCode* is MQCC_FAILED:

MQRC_SELECTOR_COUNT_ERROR
Number of selectors is not valid.

MQRC_SELECTOR_ERROR
Attribute selector not valid.

MQRC_SELECTOR_LIMIT_EXCEEDED
Too many selectors specified.

MQRC_INT_ATTR_COUNT_ERROR
Number of integer attributes is not valid.

MQRC_INT_ATTRS_ARRAY_ERROR
Integer attributes array not valid.

MQRC_CHAR_ATTR_LENGTH_ERROR
Number of character attributes is not valid.

MQRC_CHAR_ATTRS_ERROR
Character attributes string is not valid.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

C invocation

```
MQZ_INQUIRE (QMgrName, SelectorCount, Selectors, IntAttrCount,
                  &IntAttrs, CharAttrLength, &CharAttrs,
                  SelectorReturned, ComponentData, &Continuation,
                  &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48   QMgrName;              /* Queue manager name */
MQLONG     SelectorCount;         /* Selector count */
MQLONG     Selectors[n];          /* Selectors */
MQLONG     IntAttrCount;          /* IntAttrs count */
MQLONG     IntAttrs[n];           /* Integer attributes */
MQLONG     CharAttrCount;         /* CharAttrs count */
MQLONG     CharAttrs[n];          /* Chatacter attributes */
MQLONG     SelectorReturned[n];   /* Selector returned */
MQBYTE     ComponentData[n];      /* Component data */
MQLONG     Continuation;          /* Continuation indicator set by
                                     component */
MQLONG     CompCode;              /* Completion code */
MQLONG     Reason;                /* Reason code qualifying CompCode */
```

MQZ_REFRESH_CACHE (Refresh all authorizations) on IBM i: IBM i

This function is provided by an MQZAS_VERSION_3 authorization service component. It is invoked by the queue manager to refresh the list of authorizations held internally by the component.

The function identifier for this function (for MQZEP) is MQZID_REFRESH_CACHE (8L).

Syntax

MQZ_REFRESH_CACHE
(QMgrName, ComponentData, Continuation, CompCode, Reason)

Parameters

QMgrName **(MQCHAR48) - input**
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

ComponentData **(MQBYTE x *ComponentDataLength*) - input/output**
Component data.

This data is kept by the queue manager on behalf of this particular component. Any changes made to it by any of the functions provided by this component are preserved, and presented the next time a function of the component is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation **(MQLONG) - output**
Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_REFRESH_CACHE, this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode **(MQLONG) - output**
Completion code.

It is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

C invocation

```
MQZ_REFRESH_CACHE (QMgrName, ComponentData,
                   &Continuation, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_SET_AUTHORITY (Set authority) on IBM i: IBM i

This function is provided by a MQZAS_VERSION_1 authorization service component, and is invoked by the queue manager to set the authority that an entity has to access the specified object.

The function identifier for this function (for MQZEP) is MQZID_SET_AUTHORITY.

Note: This function overrides any existing authorities. To preserve any existing authorities you must set them again with this function.

Syntax

MQZ_SET_AUTHORITY *(QMgrName, EntityName, EntityType, ObjectName, ObjectType, Authority, ComponentData, Continuation, CompCode, Reason)*

Parameters

The MQZ_SET_AUTHORITY call has the following parameters.

QMgrName (MQCHAR48) - input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityName (MQCHAR12) - input
Entity name.

The name of the entity for which access to the object is to be set. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

EntityType (MQLONG) - input
Entity type.

The type of entity specified by *EntityName*. The following value can be specified:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName (MQCHAR48) - input
Object name.

The name of the object to which access is required. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) - input
Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

Authority (MQLONG) - input
Authority to be checked.

If one authorization is being set, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authorization is being set, it is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData (MQBYTE x ComponentDataLength) - input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) - output
Continuation indicator set by component.

The following values can be specified:
MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_SET_AUTHORITY this has the same effect as MQZCI_STOP.
MQZCI_CONTINUE
Continue with next component.
MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) - output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:
MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.
MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.
MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

C invocation

```
MQZ_SET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                   ObjectType, Authority, ComponentData,
                   &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority to be checked */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_TERM_AUTHORITY - Terminate authorization service:

This function is provided by an authorization service component, and is invoked by the queue manager when it no longer requires the services of this component. The function must perform any cleanup required by the component.

The function identifier for this function (for MQZEP) is MQZID_TERM_AUTHORITY.

Syntax

MQZ_TERM_AUTHORITY *(Hconfig, Options, QMgrName, ComponentData, CompCode, Reason)*

Parameters

The MQZ_TERM_AUTHORITY call has the following parameters.

Hconfig (MQHCONFIG) - input
Configuration handle.

This handle represents the particular component being terminated.

Options (MQLONG) - input
Termination options.

It is one of the following:
MQZTO_PRIMARY
Primary termination.
MQZTO_SECONDARY
Secondary termination.

QMgrName (MQCHAR48) - input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

ComponentData (MQBYTE x ComponentDataLength) - input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the **ComponentDataLength** parameter on the MQZ_INIT_AUTHORITY call.

When the MQZ_TERM_AUTHORITY call has completed, the queue manager discards this data.

CompCode (MQLONG) - output
Completion code.

It is one of the following:
MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) - output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.
MQRC_TERMINATION_FAILED
(2287, X'8FF') Termination failed for an undefined reason.

For more information on these reason codes, see Reason codes.

C invocation

```
MQZ_TERM_AUTHORITY (Hconfig, Options, QMgrName, ComponentData,
                    &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQHCONFIG  Hconfig;           /* Configuration handle */
MQLONG     Options;           /* Termination options */
MQCHAR48   QMgrName;          /* Queue manager name */
MQBYTE     ComponentData[n];  /* Component data */
MQLONG     CompCode;          /* Completion code */
MQLONG     Reason;            /* Reason code qualifying CompCode */
```

MQZAC (Application context) on IBM i: IBM i

This parameter specifies data related to the calling application.

The MQZAC structure is used on the MQZ_AUTHENTICATE_USER call for the **ApplicationContext** parameter.

Fields

StrucId (MQCHAR4)
Structure identifier.

The value is:

MQZAC_STRUC_ID
Identifier for application context structure.

For the C programming language, the constant MQZAC_STRUC_ID_ARRAY is also defined; this has the same value as MQZAC_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the service.

Version (MQLONG
Structure version number.

The value is:

MQZAC_VERSION_1
Version-1 application context structure.

The following constant specifies the version number of the current version:

MQZAC_CURRENT_VERSION
Current version of application context structure.

This is an input field to the service.

ProcessId (MQPID)
Process identifier.

The process identifier of the application.

ThreadId (MQTID)
Thread identifier.

The thread identifier of the application.

ApplName (MQCHAR28)
Application name.

The application name.

UserID (MQCHAR12)
User identifier.

For IBM i systems the user profile that the application job was created under. (On IBM i, when a profile swap is done with the QWTSETP API in the application job, the current user profile is returned).

EffectiveUserID (MQCHAR12)
Effective user identifier.

For IBM i systems the application job's current user profile.

Environment (MQLONG)
Environment.

This field specifies the environment from which the call was made.

This can have one of the following values:

MQXE_COMMAND_SERVER
Command server.

MQXE_MQSC
`runmqsc` command interpreter.

MQXE_MCA
Message channel agent

MQXE_OTHER
Undefined environment

CallerType (MQLONG)
Caller Type.

This field specifies the type of program that made the call.

This can have one of the following values:

MQXACT_EXTERNAL
The call is external to the queue manager.

MQXACT_INTERNAL
The call is internal to the queue manager.

AuthenticationType (MQLONG)
Authentication Type.

This field specifies the type of authentication being performed.

This can have one of the following values:

MQZAT_INITIAL_CONTEXT
The authentication call is due to user context being initialized. This value is used during an MQCONN or MQCONNX call.

MQZAT_CHANGE_CONTEXT
The authentication call is due to the user context being changed. This value is used when the MCA changes the user context.

v

BindType (MQLONG
Bind Type.

This field specifies the type of binding in use.

This can have one of the following values:

MQCNO_FASTPATH_BINDING
Fastpath binding.

MQCNO_SHARED_BINDING
Shared binding.

MQCNO_ISOLATED_BINDING
Isolated binding.

C declaration

```
typedef struct tagMQZAC MQZAC;
struct tagMQZAC {
  MQCHAR4   StrucId;            /* Structure identifier */
  MQLONG    Version;            /* Structure version number */
  MQPID     ProcessId;          /* Process identifier */
  MQTID     ThreadId;           /* Thread identifier */
  MQCHAR28  ApplName;           /* Application name */
  MQCHAR12  UserID;             /* User identifier */
  MQCHAR12  EffectiveUserID;    /* Effective user identifier */
  MQLONG    Environment;        /* Environment */
  MQLONG    CallerType;         /* Caller type */
  MQLONG    AuthenticationType; /* Authentication type */
  MQLONG    BindType;           /* Bind type */
};
```

MQZAD (Authority data) on IBM i: IBM i

The MQZAD structure is used on the MQZ_ENUMERATE_AUTHORITY_DATA call for two parameters.

The two parameters are:

- MQZAD is used for the **Filter** parameter which is input to the call. This parameter specifies the selection criteria that are to be used to select the authority data returned by the call.
- MQZAD is also used for the **AuthorityBuffer** parameter which is output from the call. This parameter specifies the authorizations for one combination of profile name, object type, and entity.

Fields

StrucId (MQCHAR4)
Structure identifier.

The value is:

MQZAD_STRUC_ID
Identifier for authority data structure.

For the C programming language, the constant MQZAD_STRUC_ID_ARRAY is also defined; this has the same value as MQZAD_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the service.

Version (MQLONG)
Structure version number.

The value is:

MQZAD_VERSION_1
Version-1 authority data structure.

The following constant specifies the version number of the current version:

MQZAD_CURRENT_VERSION
Current version of authority data structure.

This is an input field to the service.

ProfileName (MQCHAR48)
Profile name.

For the **Filter** parameter, this field is the profile name from which authority data is required. If the name is entirely blank up to the end of the field or the first null character, authority data for all profile names is returned.

For the **AuthorityBuffer** parameter, this field is the name of a profile that matches the specified selection criteria.

ObjectType (MQLONG)
Object type.

For the **Filter** parameter, this field is the object type for which authority data is required. If the value is MQOT_ALL, authority data for all object types is returned.

For the **AuthorityBuffer** parameter, this field is the object type to which the profile identified by *ProfileName* applies.

The value is one of the following; for the **Filter** parameter, the value MQOT_ALL is also valid:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

Authority (MQLONG)
Authority.

For the **Filter** parameter, this field is ignored.

For the **AuthorityBuffer** parameter, this field represents the authorizations that the entity has to the objects identified by *ProfileName* and *ObjectType*. If the entity has only one authority, the field is equal to the appropriate authorization value (MQZAO_* constant). If the entity has more than one authority, the field is the bitwise OR of the corresponding MQZAO_* constants.

EntityDataPtr (PMQZED)
Address of MQZED structure identifying an entity.

For the **Filter** parameter, this field points to an MQZED structure that identifies the entity from which authority data is required. If *EntityDataPtr* is the null pointer, authority data for all entities is returned.

For the **AuthorityBuffer** parameter, this field points to an MQZED structure that identifies the entity that the returned authority data came from.

EntityType (MQLONG)
Entity type.

For the **Filter** parameter, this field specifies the entity type for which authority data is required. If the value is MQZAET_NONE, authority data for all entity types is returned.

For the **AuthorityBuffer** parameter, this field specifies the type of the entity identified by the MQZED structure pointed to by *EntityDataPtr*.

The value is one of the following; for the **Filter** parameter, the value MQZAET_NONE is also valid:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

Options (MQAUTHOPT)
Options.

This field specifies options that give control over the profiles that are displayed.

One of the following must be specified:

MQAUTHOPT_NAME_ALL_MATCHING
Displays all profiles

MQAUTHOPT_NAME_EXPLICIT
Displays profiles that have exactly the same name as specified in the *ProfileName* field.

In addition, one of the following must also be specified:

MQAUTHOPT_ENTITY_SET
Display all profiles used to calculate the cumulative authority that the entity has to the object specified by *ProfileName*. The *ProfileName* field must not contain any wildcard characters.

- If the specified entity is a principal, for each member of the set {entity, groups} the most applicable profile that applies to the object is displayed.
- If the specified entity is a group, the most applicable profile from the group that applies to the object is displayed.
- If this value is specified, then the values of *ProfileName*, *ObjectType*, *EntityType*, and the entity name specified in the *EntityDataPtr* MQZED structure, must all be non-blank.

If you have specified *MQAUTHOPT_NAME_ALL_MATCHING*, you can also specify the following:

MQAUTHOPT_ENTITY_EXPLICIT
Displays profiles that have exactly the same entity name as the entity name specified in the *EntityDataPtr* MQZED structure.

C declaration

```
typedef struct tagMQZAD MQZAD;
struct tagMQZAD {
  MQCHAR4   StrucId;        /* Structure identifier */
  MQLONG    Version;        /* Structure version number */
  MQCHAR48  ProfileName;    /* Profile name */
  MQLONG    ObjectType;     /* Object type */
  MQLONG    Authority;      /* Authority */
  PMQZED    EntityDataPtr;  /* Address of MQZED structure identifying an
                               entity */
  MQLONG    EntityType;     /* Entity type */
  MQAUTHOPT Options;        /* Options */
};
```

MQZED (Entity descriptor) on IBM i: IBM i

The MQZED structure is used in a number of authorization service calls to specify the entity for which authorization is to be checked.

Fields

StrucId (MQCHAR4)
Structure identifier.

The value is:

MQZED_STRUC_ID
Identifier for entity descriptor structure.

For the C programming language, the constant MQZED_STRUC_ID_ARRAY is also defined; this has the same value as MQZED_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the service.

Version (MQLONG)
Structure version number.

The value is:

MQZED_VERSION_1
Version-1 entity descriptor structure.

The following constant specifies the version number of the current version:

MQZED_CURRENT_VERSION
Current version of entity descriptor structure.

This is an input field to the service.

EntityNamePtr (PMQCHAR)
Address of entity name.

This is a pointer to the name of the entity whose authorization is to be checked.

EntityDomainPtr (PMQCHAR)
Address of entity domain name.

This is a pointer to the name of the domain containing the definition of the entity whose authorization is to be checked.

SecurityId (MQBYTE40)
Security identifier.

This is the security identifier whose authorization is to be checked.

CorrelationPtr (MQPTR)
Correlation pointer.

This facilitates the passing of correlational data between the authenticate user function and other appropriate OAM functions.

C declaration

```
typedef struct tagMQZED MQZED;
struct tagMQZED {
  MQCHAR4   StrucId;          /* Structure identifier */
  MQLONG    Version;          /* Structure version number */
  PMQCHAR   EntityNamePtr;    /* Address of entity name */
```

```
  PMQCHAR   EntityDomainPtr;  /* Address of entity domain name */
  MQBYTE40  SecurityId;       /* Security identifier */
  MQPTR     CorrelationPtr;   /* Address of correlation data */
```

MQZFP (Free parameters) on IBM i: 

This parameter specifies data related to resource to be freed.

The MQZFP structure is used on the MQZ_FREE_USER call for the **FreeParms** parameter.

Fields

StrucId (MQCHAR4)
Structure identifier.

The value is:

MQZFP_STRUC_ID
Identifier for free parameters structure.

For the C programming language, the constant MQZFP_STRUC_ID_ARRAY is also defined; this has the same value as MQZFP_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the service.

Version (MQLONG)
Structure version number.

The value is:

MQZFP_VERSION_1
Version-1 free parameters structure.

The following constant specifies the version number of the current version:

MQZFP_CURRENT_VERSION
Current version of free parameters structure.

This is an input field to the service.

Reserved (MQBYTE8)
Reserved field.

The initial value is null.

CorrelationPtr (MQPTR)
Correlation pointer.

Address of correlation data relating to the resource to be freed.

C declaration

```
typedef struct tagMQZFP MQZFP;
struct tagMQZFP {
  MQCHAR4   StrucId;          /* Structure identifier */
  MQLONG    Version;          /* Structure version number */
  MQBYTE8   Reserved;         /* Reserved field */
  MQPTR     CorrelationPtr;   /* Address of correlation data */
};
```

MQZIC (Identity context) on IBM i: 

The MQZIC structure is used on the MQZ_AUTHENTICATE_USER call for the **IdentityContext** parameter.

The MQZIC structure contains identity context information, that identifies the user of the application that first put the message on a queue:

- The queue manager fills the UserIdentifier field with a name that identifies the user, the way that the queue manager can do this depends on the environment in which the application is running.
- The queue manager fills the AccountingToken field with a token or number that it determined from the application that put the message.
- Applications can use the ApplIdentityData field for any extra information that they want to include about the user (for example, an encrypted password).

Suitably authorized applications may set the identity context using the MQZ_AUTHENTICATE_USER function.

A Windows systems security identifier (SID) is stored in the AccountingToken field when a message is created under IBM MQ for Windows. The SID can be used to supplement the UserIdentifier field and to establish the credentials of a user.

Fields

StrucId (MQCHAR4)
Structure identifier.

The value is:

MQZIC_STRUC_ID
Identifier for identity context structure.

For the C programming language, the constant MQZIC_STRUC_ID_ARRAY is also defined; this has the same value as MQZIC_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the service.

Version (MQLONG)
Structure version number.

The value is:

MQZIC_VERSION_1
Version-1 identity context structure.

The following constant specifies the version number of the current version:

MQZIC_CURRENT_VERSION
Current version of identity context structure.

This is an input field to the service.

UserIdentifier (MQCHAR12)
User identifier.

This is part of the **identity context** of the message.

UserIdentifier specifies the user identifier of the application that originated the message. The queue manager treats this information as character data, but does not define the format of it. For more information on the *UserIdentifier* field, see "UserIdentifier (MQCHAR12)" on page 2436.

AccountingToken (MQBYTE32)
Accounting token.

This is part of the **identity context** of the message.

AccountingToken allows an application to cause work done as a result of the message to be appropriately charged. The queue manager treats this information as a string of bits and does not check its content. For more information on the *AccountingToken* field, see "AccountingToken (MQBYTE32)" on page 2390.

ApplIdentityData (MQCHAR32)
Application data relating to identity.

This is part of the **identity context** of the message.

ApplIdentityData is information that is defined by the application suite that can be used to provide additional information about the origin of the message. For example, it could be set by applications running with suitable user authority to indicate whether the identity data is trusted. For more information on the *ApplIdentityData* field, see "ApplIdentityData (MQCHAR32)" on page 2392.

C declaration

```
typedef struct tagMQZED MQZED;
struct tagMQZED {
  MQCHAR4   StrucId;          /* Structure identifier */
  MQLONG    Version;          /* Structure version number */
  MQCHAR12  UserIdentifier;   /* User identifier */
  MQBYTE32  AccountingToken;  /* Accounting token */
  MQCHAR32  ApplIdentityData; /* Application data relating to identity */
};
```

Reference material for IBM MQ bridge for HTTP

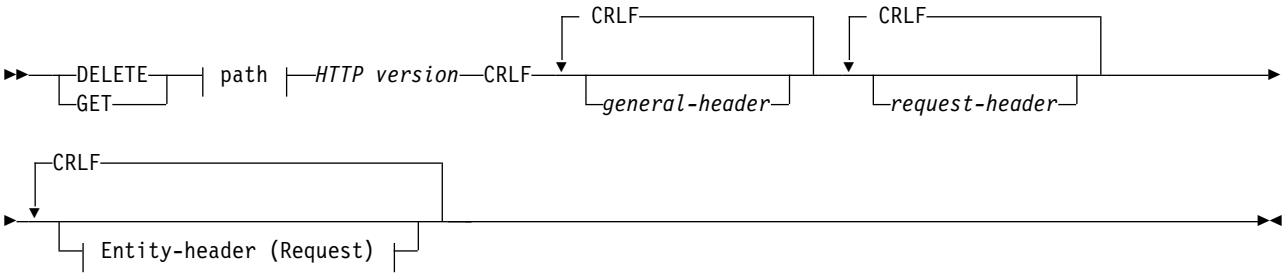
Reference topics for IBM MQ bridge for HTTP, arranged alphabetically

HTTP DELETE: IBM MQ bridge for HTTP command

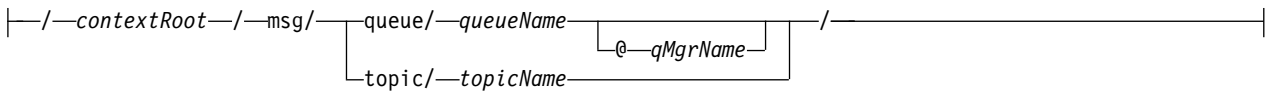
The HTTP **DELETE** operation gets a message from an IBM MQ queue, or retrieves a publication from a topic. The message is removed from the queue. If the publication is retained, it is not removed. A response message is sent back to the client including information about the message.

Syntax

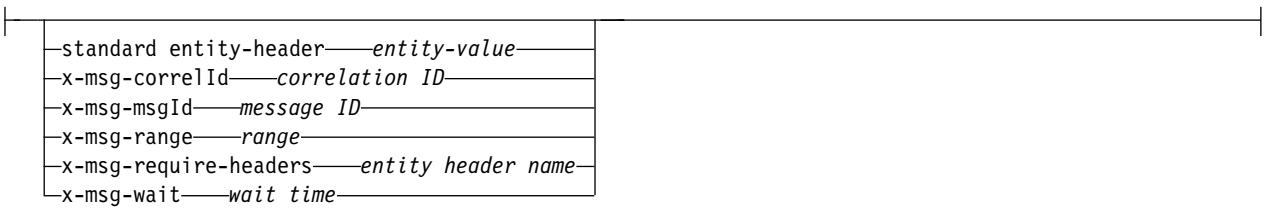
Request



Path:



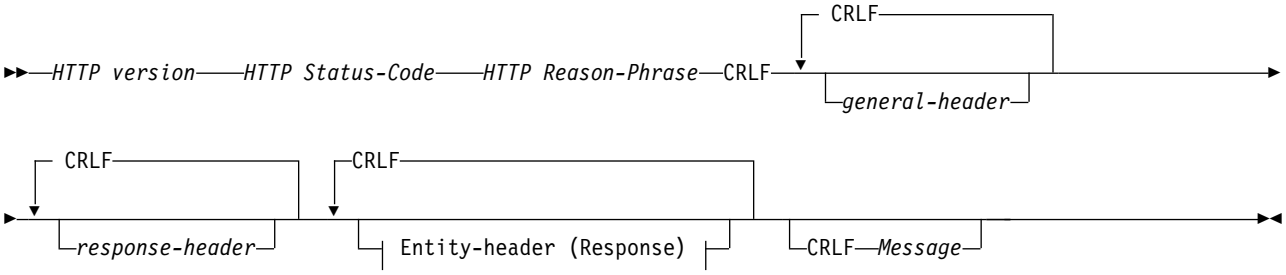
entity-header (Request):



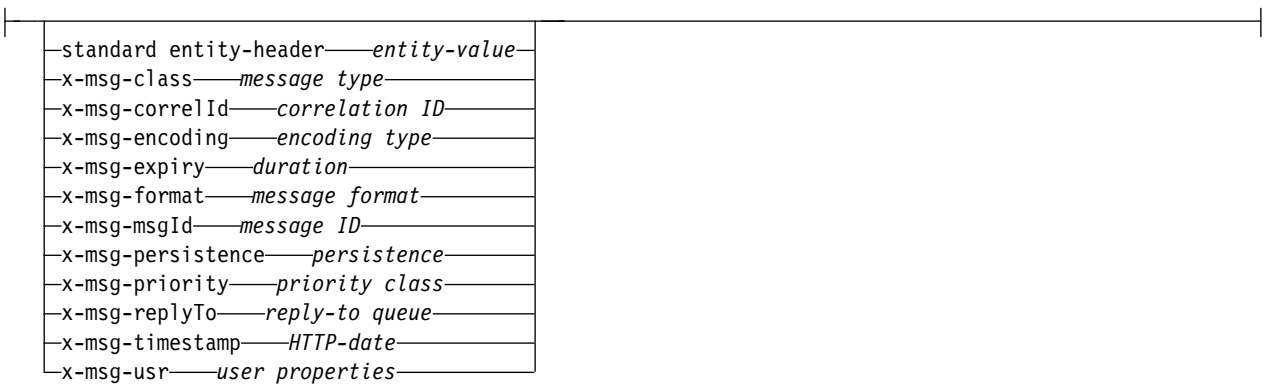
Note:

1. If a question mark (?) is used it must be substituted with `%3f`. For example, `orange?topic` should be specified as `orange%3ftopic`.
2. *@ qMgrName* is only valid on an HTTP **POST**

Response



entity-header (Response):



Request parameters

Path
See "URI Format" on page 3859.

HTTP version
HTTP version; for example, `HTTP/1.1`

general-header
See HTTP/1.1 - 4.5 General Header Fields.

request-header
See HTTP/1.1 - 5.3 Request Header Fields. The Host field is mandatory on an HTTP/1.1 request. It is often automatically inserted by the tool you use to create a client request.

entity-header (Request)
See HTTP/1.1 - 7.1 Entity Header Fields. One of the entity headers listed in the Request syntax diagram.

Response parameters

Path
See "URI Format" on page 3859.

HTTP version
HTTP version; for example, `HTTP/1.1`

general-header
See HTTP/1.1 - 4.5 General Header Fields.

response-header
See HTTP/1.1 - 6.2 Response Header Fields.

entity-header (Response)
See HTTP/1.1 - 7.1 Entity Header Fields. One of the entity or response headers listed in the Response syntax diagram. The `Content-Length` is always present in a response. It is set to zero if there is no message body.

Message
Message body.

Description

If the HTTP **DELETE** request is successful, the response message contains the data retrieved from the IBM MQ queue. The number of bytes in the body of the message is returned in the HTTP `Content-Length` header. The status code for the HTTP response is set to `200 OK`. If `x-msg-range` is specified as `0`, or `0-0`, then the status code of the HTTP response is `204 No Content`.

If the HTTP **DELETE** request is unsuccessful, the response includes an IBM MQ bridge for HTTP error message and an HTTP status code.

HTTP `DELETE` example

HTTP **DELETE** gets a message from a queue and deletes the message, or retrieves and deletes a publication. The **HTTPDELETE** Java sample is an example an HTTP **DELETE** request reading a message from a queue. Instead of using Java, you could create an HTTP **DELETE** request using a browser form, or an AJAX toolkit instead.

The following figure shows an HTTP request to delete the next message on queue called `myQueue`. In response, the message body is returned to the client. In IBM MQ terms, HTTP **DELETE** is a destructive get.

The request contains the HTTP request header `x-msg-wait`, which instructs IBM MQ bridge for HTTP how long to wait for a message to arrive on the queue. The request also contains the `x-msg-require-headers` request header, which specifies that the client is to receive the message correlation ID in the response.

```
DELETE /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

Figure 89. Example of an HTTP **DELETE** *request*

The following figure shows the response returned to the client. The correlation ID is returned to the client, as requested in `x-msg-require-headers` of the request.

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here is my message body that is retrieved from the queue.
```

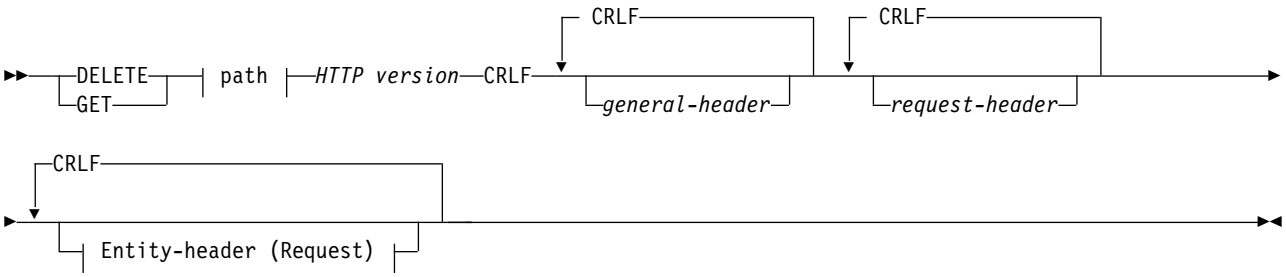
Figure 90. Example of an HTTP **DELETE** *response*

HTTP GET: IBM MQ bridge for HTTP command

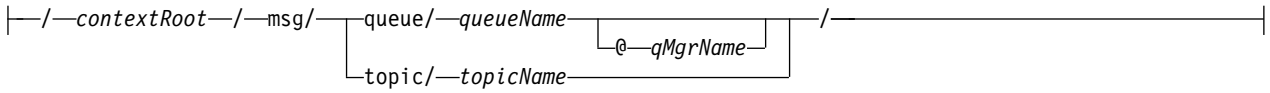
The HTTP **GET** operation gets a message from an IBM MQ queue. The message remains on the queue. The HTTP **GET** operation is equivalent to browsing an IBM MQ queue.

Syntax

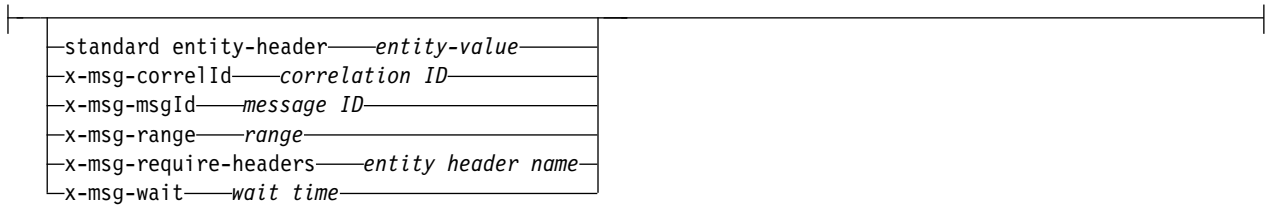
Request



Path:



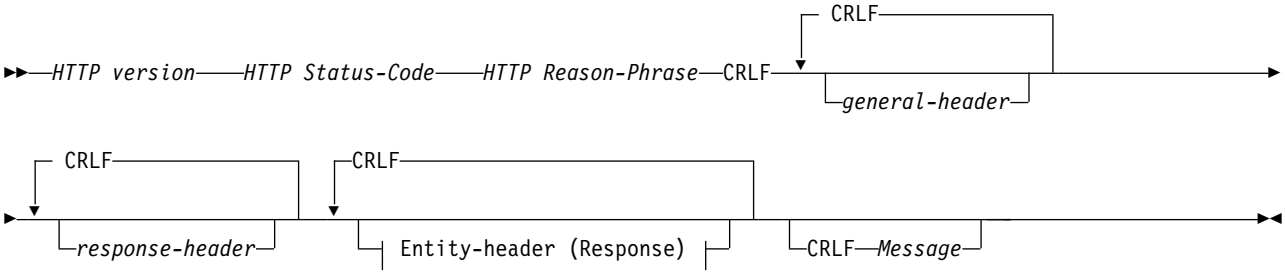
entity-header (Request):



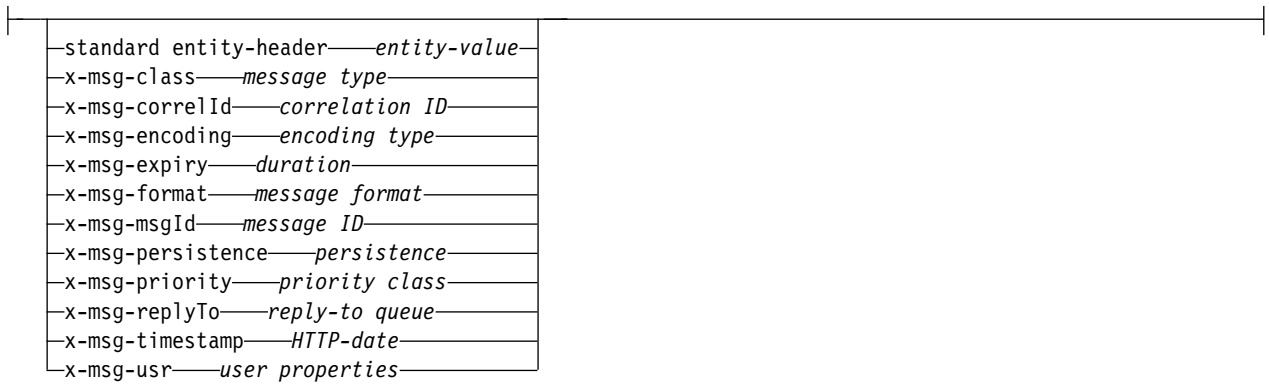
Note:

1. If a question mark (?) is used it must be substituted with `%3f`. For example, `orange?topic` should be specified as `orange%3ftopic`.
2. @ *qMgrName* is only valid on an HTTP **POST**

Response



entity-header (Response):



Request parameters

Path
See "URI Format" on page 3859.

HTTP version
HTTP version; for example, `HTTP/1.1`

general-header
See HTTP/1.1 - 4.5 General Header Fields.

request-header
See HTTP/1.1 - 5.3 Request Header Fields. The Host field is mandatory on an HTTP/1.1 request. It is often automatically inserted by the tool you use to create a client request.

entity-header (Request)
See HTTP/1.1 - 7.1 Entity Header Fields. One of the entity headers listed in the Request syntax diagram.

Response parameters

Path
See "URI Format" on page 3859.

HTTP version
HTTP version; for example, `HTTP/1.1`

general-header
See HTTP/1.1 - 4.5 General Header Fields.

response-header
See HTTP/1.1 - 6.2 Response Header Fields.

entity-header (Response)
See HTTP/1.1 - 7.1 Entity Header Fields. One of the entity or response headers listed in the Response syntax diagram. The `Content-Length` is always present in a response. It is set to zero if there is no message body.

Message
Message body.

Description

If the HTTP **GET** request is successful, the response message contains the data retrieved from the IBM MQ queue. The number of bytes in the body of the message is returned in the HTTP `Content-Length` header. The status code for the HTTP response is set to `200 OK`. If `x-msg-range` is specified as `0`, or `0-0`, then the status code of the HTTP response is `204 No Content`.

If the HTTP **GET** request is unsuccessful, the response includes an IBM MQ bridge for HTTP error message and an HTTP status code.

HTTP `GET` example

HTTP **GET** gets a message from a queue. The message remains on the queue. In IBM MQ terms, HTTP **GET** is a browse request. You could create an HTTP **GET** request using a Java client, a browser form, or an AJAX toolkit.

The following figure shows an HTTP request to browse the next message on queue called `myQueue`.

The request contains the HTTP request header `x-msg-wait`, which instructs IBM MQ bridge for HTTP how long to wait for a message to arrive on the queue. The request also contains the `x-msg-require-headers` request header, which specifies that the client is to receive the message correlation ID in the response.

```
GET /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

Figure 91. Example of an HTTP **GET** *request*

The following figure shows the response returned to the client. The correlation ID is returned to the client, as requested in `x-msg-require-headers` of the request.

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here is my message body that appears on the queue.
```

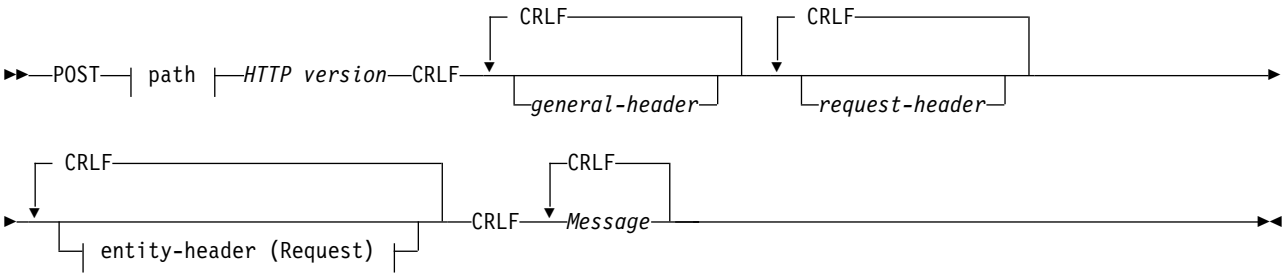
Figure 92. Example of an HTTP **GET** *response*

HTTP POST: IBM MQ bridge for HTTP command

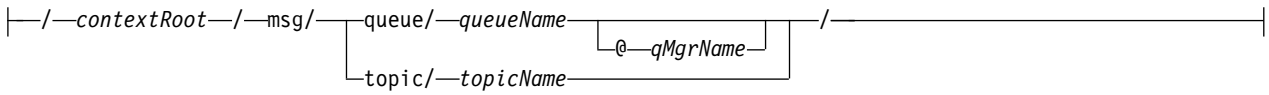
The HTTP **POST** operation puts a message on an IBM MQ queue, or publishes a message to a topic.

Syntax

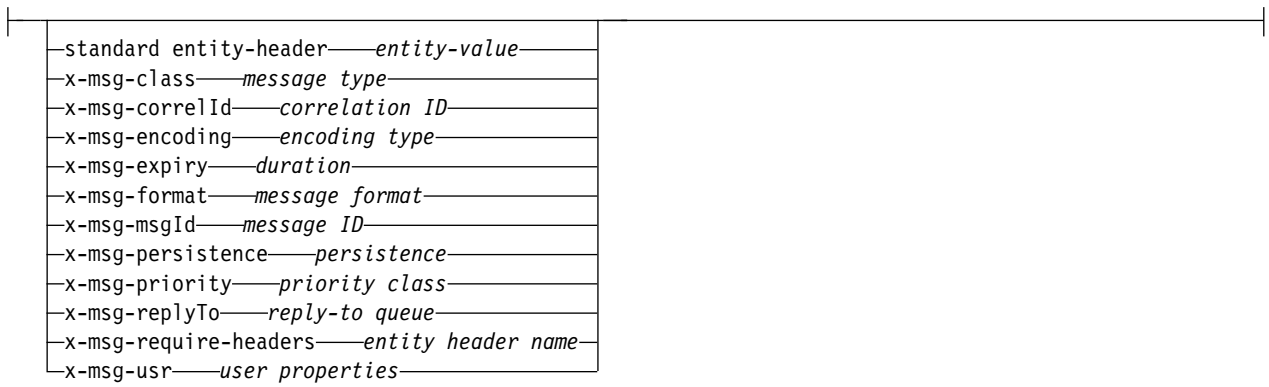
Request



Path:



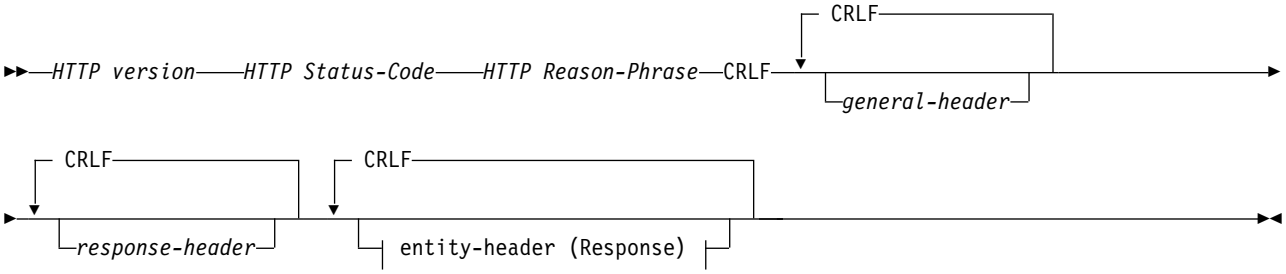
entity-header (Request):



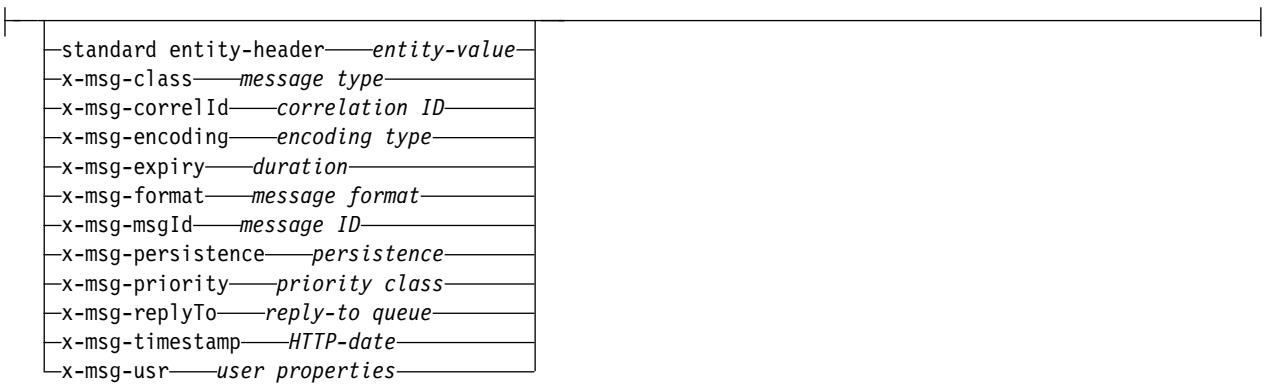
Note:

1. If a question mark (?) is used it must be substituted with `%3f`. For example, `orange?topic` should be specified as `orange%3ftopic`.
2. *@ qMgrName* is only valid on an HTTP **POST**

Response



entity-header (Response):



Request parameters

Path
See "URI Format" on page 3859.

HTTP version
HTTP version; for example, `HTTP/1.1`

general-header
See HTTP/1.1 - 4.5 General Header Fields.

request-header
See HTTP/1.1 - 5.3 Request Header Fields. The Host field is mandatory on an HTTP/1.1 request. It is often automatically inserted by the tool you use to create a client request.

entity-header (Request)
See HTTP/1.1 - 7.1 Entity Header Fields. One of the entity headers listed in the Request syntax diagram. The `Content-Length` and `Content-Type` should be inserted in a request, and are often inserted automatically by the tool you use to create a client request. The `Content-Type` must match the type defined in the `x-msg-class` custom entity-header, if it is specified.

Message
Message to put onto the queue, or publication to post to a topic.

Response parameters

Path
See "URI Format" on page 3859.

HTTP version
HTTP version; for example, `HTTP/1.1`

general-header
See HTTP/1.1 - 4.5 General Header Fields.

response-header
See HTTP/1.1 - 6.2 Response Header Fields.

entity-header (Response)
See HTTP/1.1 - 7.1 Entity Header Fields. One of the entity or response headers listed in the Response syntax diagram. The `Content-Length` is always present in a response. It is set to zero if there is no message body.

Description

If no `x-msg-usr` header is included, and message class is `BYTES` or `TEXT`, the message put on the queue does not have an `MQRFH2`.

Use HTTP entity and request headers in the HTTP **POST** request to set the properties of the message that is put onto the queue. You can also use `x-msg-require-headers` to request which headers are returned in the response message.

If the HTTP **POST** request is successful, the entity of the response message is empty and its Content-Length is zero. The HTTP status code is `200 OK`.

If the HTTP **POST** request is unsuccessful, the response includes an IBM MQ bridge for HTTP error message and an HTTP status code. The IBM MQ message is not put on the queue or topic.

HTTP `POST` example

HTTP **POST** puts a message to a queue, or a publication to a topic. The **HTTPPOST** Java sample is an example an HTTP **POST** request of a message to a queue. Instead of using Java, you could create an HTTP **POST** request using a browser form, or an AJAX toolkit instead.

The following figure shows an HTTP request to put a message on a queue called `myQueue`. This request contains the HTTP header `x-msg-correlId` to set the correlation ID of the IBM MQ message.

```
POST /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
Content-Type: text/plain
x-msg-correlID: 1234567890
Content-Length: 50

Here is my message body that is posted on the queue.
```

Figure 93. Example of an HTTP **POST** *request to a queue*

The following figure shows the response sent back to the client. There is no response content.

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 0
```

Figure 94. Example of an HTTP POST response

HTTP headers

The IBM MQ bridge for HTTP supports custom request HTTP headers, custom entity HTTP headers, and a subset of standard HTTP headers.

HTTP practice is to prefix all custom headers with `x-`, so the IBM MQ Bridge for HTTP headers are prefixed with `x-msg-`. For example, to set the priority header use `x-msg-priority`.

Note:

- Most header values are case sensitive. For example, when using the `msgId` header, `NONE` is a keyword, whereas `none` is a `msgID`.
- Misspelled headers are ignored.

Custom entity HTTP headers

The custom entity HTTP headers contain information about IBM MQ messages. Using entity headers, you can set values in the message descriptor (`MQMD`), or query values in the `MQMD`. An additional entity header, `x-msg-usr`, sets and returns any user property information you want to associate with a request.

You can use entity headers in different HTTP request contexts:

DELETE You can only use the `x-msg-correlId`, or `x-msg-msgId`, or both, entity headers with a **DELETE** HTTP request. The effect of the headers is to select a particular message by `MsgId` and `CorrelId` in an MQGET, and to delete the message from its queue.

GET You can only use the `x-msg-correlId`, or `x-msg-msgId`, or both, entity headers with a **GET** HTTP request. The effect of the headers is to select a particular message by `MsgId` and `CorrelId` in an MQGET for browse.

POST You can use any entity header in a **POST** HTTP request, except `x-msg-timestamp`.

x-msg-require-headers
On any **GET**, **POST** or **DELETE** HTTP request, you can add multiple entity headers inside the `x-msg-require-headers` request header, separated by commas. The effect is to return the specified entity headers in the HTTP response message, containing the value of the associated message property.

The description of each header lists in which contexts the header is processed by IBM MQ bridge for HTTP. For example, in the header **POST**, `x-msg-require-headers`, the header is processed by IBM MQ bridge for HTTP in an HTTP **POST** request, or in the `x-msg-require-headers` request header in either an HTTP **POST**, **GET**, or **DELETE** request. If the header is included in a context it is not allowed in, the header is ignored. No error is reported.

You can put any standard HTTP headers into requests to be processed by the Web server, or other request handlers. Similarly, the response might contain other standard HTTP headers inserted by the Web server or other response handlers.

Custom request HTTP headers

The three custom request HTTP headers, `x-msg-range`, `x-msg-require-headers`, and `x-msg-wait`, pass additional information about the HTTP request to the server. They act as request modifiers. With `x-msg-range`, you can restrict the amount of message data returned in a response. With `x-msg-require-headers`, you can request the response to contain information about the result of the request. With `x-msg-wait`, you can modify the time the client waits for an HTTP response.

Standard HTTP headers

The `Host` standard HTTP request-header must be specified in an HTTP/1.1 request.

The `Content-Length` and `Content-Type` standard HTTP entity headers can be specified in a request.

The `Content-Length`, `Content-Location`, `Content-Range`, `Content-Type`, and `Server` standard HTTP entity headers can be returned in response to a request. Specify one or more of the standard HTTP headers in the `x-msg-request-header` header in the request message.

Alphabetic list of HTTP headers

`class`: HTTP `x-msg-class` entity-header:

Set or return the message type.

Type	Description
HTTP header name	`x-msg-class`
HTTP header type	Entity-header
Valid in HTTP request message	POST, `x-msg-require-headers`
Allowed values	**`BYTES`** **`MAP`** **`STREAM`** **`TEXT`**
Default value	`BYTES`

Description

- In an HTTP **`POST`** request, sets the type of the message created.
- Specifying the class header on a **`GET`** or **`DELETE`** returns a `400 Bad Request` with entity body of `MQHTTP40007`.
- Specified in `x-msg-require-headers`, sets `x-msg-class` in the HTTP response message to the type of a message.
- If an invalid value is specified for this header a `MQHTTP40005` message is returned.
- If the `x-msg-class` header is not specified and the content-type of the message is `application/x-www-form-urlencoded`, the data is assumed to be a JMS map object.

`Content-Length`: HTTP entity-header:

Set or return the length, in bytes, of the body of the message.

Type	Description
HTTP header name	`Content-Length`
HTTP header type	Entity-header
Valid in HTTP request message	`x-msg-require-headers`
Allowed and returned value	*`Integer value`* Length in bytes of the message body.

Description

- The `Content-Length` is optional in an HTTP request. For a **`GET`** or **`DELETE`** the length must be zero. For **`POST`**, if `Content-Length` is specified and it does not match the length of the message-line, the message is either truncated, or padded with nulls to the specified length.
- The `Content-Length` is always returned in the HTTP response even when there is no content, in which case the value is zero.

`Content-Location`: HTTP entity-header:

Returns the queue or topic referenced in the request, in the standard `Content-Location` header in the HTTP response message.

Type	Description
HTTP header name	`Content-Location`
HTTP header type	Entity-header
Valid in HTTP request message	`x-msg-require-headers`
Returned value	URI in the format, *`/msg/queue/queuename`* or *`/msg/topic/topicname`*

Description

- When requested in `x-msg-require-headers`, the `Content-Location` entity-header returns the queue or topic referenced in the HTTP request.

Content-Range: HTTP entity-header:

Return the range of bytes selected from an IBM MQ message in the `Content-Range` header in an HTTP response.

Type	Description
HTTP header name	`Content-Range`
HTTP header type	Entity-header
Valid in HTTP request message	`x-msg-require-headers`
Returned value	*String* Returns the lower limit, *m* and upper limit, *n* of the returned substring, and *length* of the whole message. For example, *m - n/length*

Description

•

- The `Content-Range` is only returned in the HTTP response when `Content-Range` is specified in a **GET** or **DELETE** request that contains an `x-msg-range` request header.
- If `x-msg-range` is specified on a **GET** or **DELETE** request, the range of bytes specified in the Content-Range header are returned in the response. For example, if `x-msg-range: 0-60` is used in a request for a message containing `100` bytes, the content-range header holds the string `0-60/100`
- An `x-msg-range` request also returns the content range in the `x-msg-range` header in the HTTP response.

Content-Type: HTTP entity-header:

Set or return the class of the JMS message in an IBM MQ message according the to HTTP content-type.

Type	Description
HTTP header name	`Content-Type`
HTTP header type	Entity-header
Valid in HTTP request message	**POST**, `x-msg-require-headers`
Allowed or returned value	*media-type* For media-types that are supported see Table 405.

Table 405. Mapping between `x-msg-class` and HTTP `Content-Type`

x-msg-class	HTTP Content-Type
BYTES	application/octet-stream application/xml
TEXT	text/*
MAP	application/x-www-form-urlencoded application/xml (optional)
STREAM	application/xml (optional)

Description

- On an HTTP **POST** request, specify either the `Content-Type` or the `x-msg-class`. If you specify both, they must be consistent or an `HTTP Bad Request` exception, `Status code 400` is returned. If you omit both, the `Content-Type` and the `x-msg-class`, a `Content-Type` of `text/*` is assumed.
- The `Content-Type` is always set in the response to an HTTP **GET** or **DELETE** that has a message body. The `Content-Type` is set according to the rules in Table 406.

Table 406. Mapping message types to `x-msg-class` and `Content-Type`

Message format	JMS Message type	x-msg-class	Content-Type
Anything except `MQFMT_STRING`	None	`BYTES`	`application/octet-stream`
`MQFMT_STRING`	None	`TEXT`	`text/plain`
`MQFMT_NONE`	`jms_bytes`	`BYTES`	`application/octet-stream`
`MQFMT_NONE`	`jms_text`	`TEXT`	`text/plain`
`MQFMT_NONE`	`jms_map`	`MAP`	`application/xml`
`MQFMT_NONE`	`jms_stream`	`STREAM`	`application/xml`

correlId: HTTP x-msg-correlId entity-header:

Set or return the correlation identifier.

Type	Description
HTTP header name	`x-msg-correlId`
HTTP header type	Entity-header
Valid in HTTP request message	**DELETE**, **GET**, **POST**, `x-msg-require-headers`
Allowed values	*String value* For example: `x-msg-correlId: mycorrelationid` Strings enclosed in quotation marks are permitted; for example: `x-msg-correlId: "my id"` *Hex value* A hex value prefixed with `0x:` ; for example: `x-msg-correlId: 0x:43c1d23a` The hex value following 0x: is limited to 48 characters representing 24 bytes. Additional data is ignored.
Default value	Not applicable

Description

- On an HTTP **POST** request, sets the correlation ID of the message created.
- On an HTTP **GET** or **DELETE** request, selects the message from the queue or topic. If no message exists with the specified correlation ID, an `HTTP 504 Gateway Timeout` response is returned. `x-msg-correlId` can be used with `x-msg-msgID` to select a message from a queue or topic that matches both selectors.
- Specified in `x-msg-require-headers`, sets `x-msg-corelId` in the HTTP response message to the correlation ID of a message.
- Horizontal white space is allowed after the `0x:` prefix.

Note:

- Specifying `x-msg-correlId` without a value on an HTTP **`GET`** or **`DELETE`** request; for example, `"x-msg-correlId:"`, returns the next message on the queue or topic regardless of its correlation ID.
- If you specify a selector of 24 characters or fewer, or 0x: followed by 48 characters or fewer, IBM MQ bridge for HTTP uses an optimized selector for improved performance.
- A JMS message selector containing JMSCorrelationID is used when selecting messages from the queue. This selector behaves as described in Selection behavior.

`encoding`: HTTP `x-msg-encoding` entity-header:

Set or return the message encoding.

Type	Description
HTTP header name	`x-msg-encoding`
HTTP header type	Entity-header
Valid in HTTP request message	**`POST`**, `x-msg-require-headers`
Allowed values	A comma-separated list of the following values: **`DECIMAL_NORMAL`** **`DECIMAL_REVERSED`** **`FLOAT_IEEE_NORMAL`** **`FLOAT_IEEE_REVERSED`** **`FLOAT_S390`** **`INTEGER_NORMAL`** **`INTEGER_REVERSED`** For example, `x-msg-encoding: INTEGER_NORMAL,DECIMAL_NORMAL,FLOAT_IEEE_NORMAL` **Note:** The value is case-sensitive
Default value	`DECIMAL_NORMAL, FLOAT_IEEE_NORMAL, INTEGER_NORMAL`

Description

- On an HTTP **`POST`** request, specifies the encoding of the message created.
- On an HTTP **`GET`** or **`DELETE`** request, the `x-msg-encoding` header is ignored.
- Specified in `x-msg-require-headers`, sets `x-msg-encoding` in the HTTP response message to the encoding property of a message.

expiry: HTTP x-msg-expiry entity-header:

Set or return the message expiry duration.

Type	Description
HTTP header name	x-msg-expiry
HTTP header type	Entity-header
Valid in HTTP request message	**POST**, x-msg-require-headers
Allowed values	**UNLIMITED** For example; x-msg-expiry: UNLIMITED *Integer value* Milliseconds before expiry. For example; x-msg-expiry: 10000
Default value	UNLIMITED

Description

- When set on an HTTP **POST** request, the request message expires in the time specified.
- On an HTTP **GET** or **DELETE** request, the x-msg-expiry header is ignored.
- Specified in x-msg-require-headers, sets x-msg-expiry in the HTTP response message to the expiry time of a message.
- UNLIMITED specifies that the message never expires.
- The expiry of a message starts from the time the message arrives on the queue, as a result network latency is ignored.
- The maximum value is limited by IBM MQ to 214748364700 milliseconds. If a value greater than this is specified then the maximum possible expiry time is assumed.

format: HTTP x-msg-format entity-header:

Set or return the IBM MQ message format.

Type	Description
HTTP header name	x-msg-format
HTTP header type	Entity-header
Valid in HTTP request message	**POST**, x-msg-require-headers
Allowed values	**NONE** For example, x-msg-format: NONE *String value* Any user-defined value of up to eight characters. For example, x-msg-format: myformat
Default value	None

Description

- When set on an HTTP **POST** request, set the request message format.
- On an HTTP **GET** or **DELETE** request, the `x-msg-format` header is ignored.
- Specified in `x-msg-require-headers`, sets `x-msg-format` in the HTTP response message to the format of a message.
- `NONE` is case sensitive, and indicates that the message format is blank.
- The value of `x-msg-format` is used, even if it contradicts the media-type of the HTTP request. See Table 407.

Table 407. Mapping content-type and `x-msg-class` to message format

x-msg-class	**Content-type**	**Message format on queue/topic**
`BYTES`	• `application/octet-stream` • application/xml	IBM MQ message: `MQFMT` set to `MQC.MQFMT_NONE`
`TEXT`	• `text/*`	IBM MQ message: `MQFMT` set to `MQC.MQFMT_STRING`
`MAP`	• `application/x-www-form-urlencoded` • `application/xml` (optional)	`JMSMap`
`STREAM`	• `application/xml` (optional)	`JMSStream`

msgId: HTTP x-msg-msgId entity-header:

Set or return the message identifier.

Type	**Description**
HTTP header name	`x-msg-msgId`
HTTP header type	Entity-header
Valid in HTTP request message	**DELETE**, **GET**, **POST**, `x-msg-require-headers`
Allowed values	*String value* For example, `x-msg-msgId: mymsgid` Strings enclosed in quotation marks for example, `x-msg-msgId: "my id"` *Hex value* A hex value prefixed with `0x:` ; for example, `x-msg-msgId: 0x:43c1d23a`
Default value	Not applicable

Description

- On an HTTP **POST** request, sets the message ID of the message created.
- On an HTTP **GET** or **DELETE** request, selects the message from the queue or topic. If no message exists with the specified message ID, an `HTTP 504 Gateway Timeout` response is returned. `x-msg-msgId` can be used with `x-msg-correlID` to select a message from a queue or topic that matches both selectors.
- Specified in `x-msg-require-headers`, returns `x-msg-msgId` in the HTTP response to the message ID of a message.
- Horizontal white space is allowed after the `0x:` prefix.

Note: Specifying `x-msg-msgId` without a value on an HTTP **GET** or **DELETE** request; for example, `"x-msg-msgId:"`, returns the next message on the queue or topic regardless of its message ID.

A JMS message selector containing JMSMessageID is used when selecting messages from the queue. This selector behaves as described in Selection behavior.

persistence: HTTP x-msg-persistence entity-header:

Set or return the message persistence.

Type	Description
HTTP header name	`x-msg-persistence`
HTTP header type	Entity-header
Valid in HTTP request message	**POST**, `x-msg-require-headers`
Allowed values	**NON_PERSISTENT** The message does not survive system failures or queue manager restarts. For example, `x-msg-persistence: NON_PERSISTENT` **PERSISTENT** The message survives system failures and restarts of the queue manager. For example, `x-msg-persistence: PERSISTENT` **AS_DESTINATION** Applies to **POST** only. Use the default persistence of the destination as determined by the message provider. **Note:** Case sensitive
Default value	`NON_PERSISTENT`

Description

- When set on an HTTP **POST** request, set the request message persistence.
- On an HTTP **GET** or **DELETE** request, the `x-msg-persistence` header is ignored.
- Specified in `x-msg-require-headers`, sets `x-msg-persistence` in the HTTP response message to the persistence of a message.

priority: HTTP x-msg-priority entity-header:

Set or return the message priority.

Type	Description
HTTP header name	`x-msg-priority`
HTTP header type	Entity-header
Valid in HTTP request message	**POST**, `x-msg-require-headers`
Allowed values	**LOW** For example, `x-msg-priority: LOW` **MEDIUM** This priority is equal to an IBM MQ priority level of 4. For example, `x-msg-priority: MEDIUM` **HIGH** For example, `x-msg-priority: HIGH` *Integer value* A string representation of an integer in the range 0 through 9; for example, `x-msg-priority: 3` **AS_DESTINATION** Applies to **POST** only. Use the default priority of the destination as determined by the message provider. **Note:** Case sensitive
Default value	`MEDIUM`

Description

- When set on an HTTP **POST** request, set the request message priority.
- On an HTTP **GET** or **DELETE** request, the `x-msg-priority` header is ignored.
- Specified in `x-msg-require-headers`, sets `x-msg-priority` in the HTTP response message to the priority of a message.

priority: HTTP x-msg-priority entity-header:

Set or return the message priority.

Type	Description
HTTP header name	`x-msg-priority`
HTTP header type	Entity-header
Valid in HTTP request message	**POST**, `x-msg-require-headers`

Type	Description
Allowed values	**LOW** For example, `x-msg-priority: LOW` **MEDIUM** This priority is equal to an IBM MQ priority level of 4. For example, `x-msg-priority: MEDIUM` **HIGH** For example, `x-msg-priority: HIGH` *Integer value* A string representation of an integer in the range 0 through 9; for example, `x-msg-priority: 3` **AS_DESTINATION** Applies to **POST** only. Use the default priority of the destination as determined by the message provider. **Note:** Case sensitive
Default value	`MEDIUM`

Description

- When set on an HTTP **POST** request, set the request message priority.
- On an HTTP **GET** or **DELETE** request, the `x-msg-priority` header is ignored.
- Specified in `x-msg-require-headers`, sets `x-msg-priority` in the HTTP response message to the priority of a message.

`replyTo`: HTTP `x-msg-replyTo` entity-header:

Set or return the message reply-to queue and queue manager name.

Type	Description
HTTP header name	`x-msg-replyTo`
HTTP header type	Entity-header
Valid in HTTP request message	**POST**, `x-msg-require-headers`
Allowed values	*URI* A point-to-point URI; for example, `x-msg-replyTo: /msg/queue/myReplyQueue` `x-msg-replyTo: /msg/queue/myReplyQueue@myReplyQueueManager` **Note:** Case sensitive
Default value	`MEDIUM`

Description

- When set on an HTTP **POST** request, set the request message replyTo destination.
- On an HTTP **GET** or **DELETE** request, the `x-msg-replyTo` header is ignored.
- Specified in `x-msg-require-headers`, sets `x-msg-replyTo` in the HTTP response message to the reply-to queue and queue manager name of a message.

Note: The URI in the HTTP response can include the name of the queue manager to which the IBM MQ bridge for HTTP is connected.

Server: HTTP response-header:

Returns information about the server and protocol the client is connected to.

Type	Description
HTTP header name	`Server`
HTTP header type	Response-header
Valid in HTTP request message	`x-msg-require-headers`
Returned value	`WMQ-HTTP/1.1 JEE-Bridge/1.1` or `Server: Product-token WMQ-HTTP/1.1 JEE-Bridge/1.1`

Description

- If IBM MQ Bridge for HTTP is deployed to an application server, the IBM MQ bridge for HTTP details is appended to the server response header. For example, the IBM MQ bridge for HTTP deployed to WebSphere Application Server Community Edition, called Apache-Coyote, gives the response:

  ```
  Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
  ```

require-headers: HTTP x-msg-require-headers request-header:

Set which headers to return in the HTTP response message.

Type	Description
HTTP header name	`x-msg-require-headers`
HTTP header type	Request-header
Valid in HTTP request message	**POST**, **GET**, **DELETE**

Type	Description
Allowed values	A comma-separated list of the entity header names: **ALL** **ALL-USR** **class** **content-location** **correlId** **encoding** **expiry** **format** **msgId** **NO_require-headers** **persistence** **priority** **replyTo** **server** **timestamp** **usr-** *property name* For example, `x-msg-require-headers: msgId` or, `x-msg-require-headers: expiry,correlId,timestamp` To request a specific property: `x-msg-require-headers: usr-myCustomProperty` To request all properties: `x-msg-require-headers: ALL-USR, ALL`
Default value	`NO_require-headers`

Description

- The value of `x-msg-require-headers` is not case-sensitive, except in the cases of the `ALL`, `NO_require-headers`, and `ALL-USR` constants, and the *property-name* variable.

timestamp: HTTP x-msg-timestamp entity-header:

Return the message time stamp.

Type	Description
HTTP header name	x-msg-timestamp
HTTP header type	Entity-header
Valid in HTTP request message	x-msg-require-headers
Returned value	*HTTP-date* A date in the format; day, date month year time time-zone; for example, Sun, 06 Nov 1994 08:49:37 GMT Defined by RFC 822, and updated in RFC 1123.
Default value	Not applicable

Description

- On an HTTP **POST**, **GET** or **DELETE** request, the x-msg-timestamp header is ignored.
- Specified in x-msg-require-headers, sets x-msg-timestamp in the HTTP response message to the timestamp of a message.

usr: HTTP x-msg-usr entity-header:

Set or return the user properties.

Type	Description
HTTP header name	x-msg-usr
HTTP header type	Entity-header
Valid in HTTP request message	**POST**, x-msg-require-headers
Allowed values	See "Syntax" ; for example, x-msg-usr: myProp1;5;i1, x-msg-usr: myProp2;"My String";string
Default value	Not applicable

Description

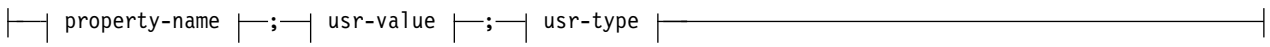
- When set on an HTTP **POST** request, set the request message user properties.
- On an HTTP **GET** or **DELETE** request, the x-msg-usr header is ignored.
- Specified in x-msg-require-headers, sets x-msg-usr in the HTTP response message to user properties of a message.
- Multiple properties can be set on a message. Specify multiple comma-separated properties in a single x-msg-usr header, or by using two or more separate instances of the x-msg-usr header.
- You can request a specific property to be returned in the response to a **GET** or **DELETE** request. Specify the name of the property in the x-msg-require-headers header of the request, using the prefix usr-. For example,

  x-msg-require-headers: usr-myProp1
- To request that all user properties are returned in a response, use the ALL-USR constant. For example,

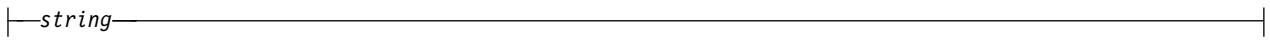
  x-msg-require-headers: ALL-USR

Syntax

usr-property-value:



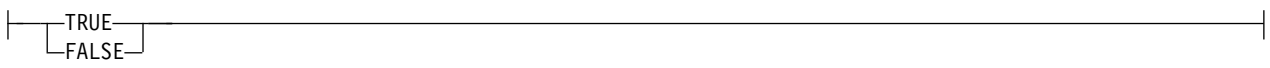
property-name:



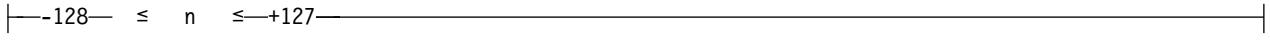
usr-value:

usr-type:

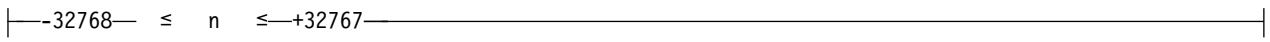
boolean



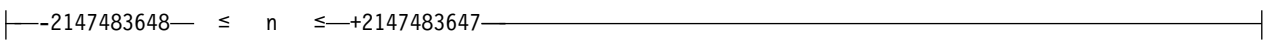
i1



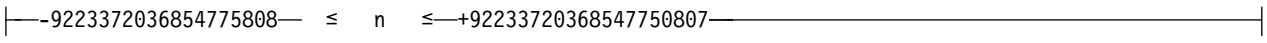
i2



i4

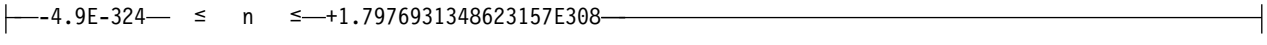


i8

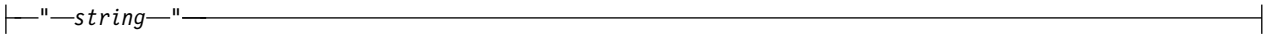


r4

r8



qstring



wait: HTTP x-msg-wait request-header:

Set the period of time to wait for a message to arrive, before returning an `HTTP 504 Gateway Timeout` response message.

Type	Description
HTTP header name	`x-msg-wait`
HTTP header type	Request-header
Valid in HTTP request message	**GET**, **DELETE**
Allowed value	**NO_WAIT** For example, `x-msg-wait: NO_WAIT` *Integer value* The number of milliseconds that the IBM MQ bridge for HTTP waits for a message to arrive; for example, `x-msg-wait: 8`
Default value	`NO_WAIT`

Description

- On an HTTP **POST** request, the `x-msg-wait` header is ignored.
- On an HTTP **GET** or **DELETE** request, `x-msg-wait` specifies time to wait for a message to arrive before returning an `HTTP 504 Gateway Timeout` response.
- `NO_WAIT` is case-sensitive.
- The default maximum wait time is 35000. You can change the default by setting the **maximum_wait_time** parameter of the servlet.
- If you set a value greater than `maximum_wait_time`, `maximum_wait_time` is used instead.

HTTP return codes

List of return codes from the IBM MQ bridge for HTTP

The IBM MQ bridge for HTTP returns four types of error:

Servlet errors
MQHTTP0001 and MQHTTP0002 are servlet errors. They are logged, but not returned to the HTTP client.

Successful operations
An HTTP status code in the range 200 - 299 indicates a successful operation.

Client errors
An HTTP status code in the range 400 - 499 indicates a client error. IBM MQ Bridge for HTTP return codes in the range MQHTTP40001 - MQHTTP49999 correspond to client errors.

Server errors

An HTTP status code in the range 500 - 599 indicates a client error. IBM MQ Bridge for HTTP return codes in the range MQHTTP50001 - MQHTTP59999 correspond to server errors.

If a server error occurs, a complete stack trace is output to the application server error log. The stack trace is also returned to the HTTP client in the HTTP response. Handle the stack trace in the client application or refer it to the application server administrator to resolve the problem.

If the stack trace contains resource adapter errors, refer to the documentation for your resource adapter.

Alphabetic list of return codes

HTTP 200: OK:

This class of status code indicates that the request was successfully received, understood and accepted.

HTTP status code

```
200 OK
```

HTTP 204: No content:

Sent following a successful HTTP **GET** or **DELETE** and `x-msg-range: 0` was sent in the request.

HTTP status code

```
204 No Content
```

MQHTTP0001: No connection factory specified in the Servlet context:

Servlet error

Explanation

Servlet error

HTTP status code

None

Programmer response

Where these errors are logged is specific to your application server. Refer to the documentation for your application server.

MQHTTP0002: Could not get connection manager for *queueOrTopic* using the JNDI name of *jndiNameTried*:

Servlet error

Explanation

Servlet error

HTTP status code

None

Programmer response

Where these errors are logged is specific to your application server. Refer to the documentation for your application server.

MQHTTP40001: Reserved:

Reserved

HTTP status code

`400 Bad Request`

MQHTTP40002: URI is not valid for IBM MQ transport for HTTP:

The URI specified in the HTTP request is not valid.

Explanation

The URI specified in the HTTP request is not valid.

HTTP status code

`400 Bad Request`

Programmer response

Confirm that the format and syntax of the specified URI are correct.

MQHTTP40003: URI is not valid. `@qmgr` is only valid on POST:

The `@qmgr` URI option has been specified in a URI for an HTTP request that is not a **`POST`** request.

Explanation

The `@qmgr` URI option has been specified in a URI for an HTTP request that is not a **`POST`** request.

HTTP status code

`400 Bad Request`

Programmer response

If you are attempting to put a message by using the **`POST`** verb, change the HTTP request to a **`POST`** request. If you are attempting to get a message by using the **`DELETE`** or **`GET`** verbs, remove `@qmgr` from the URI.

MQHTTP40004: Invalid `Content-Type` specified:

The `Content-Type` header field specified on a **`POST`** request is not compatible with the `x-msg-class` header value.

Explanation

The `Content-Type` header field specified on a **`POST`** request is not compatible with the `x-msg-class` header value.

HTTP status code

`400 Bad Request`

Programmer response

Change the `Content-Type` header field to one that is supported. The `Content-Type` header must be compatible with the specified `x-msg-class` header field.

MQHTTP40005: Bad message header value:

A supported header field has been specified with a value that is not valid for the specified request.

Explanation

A supported header field has been specified with a value that is not valid for the specified request.

HTTP status code

`400 Bad Request`

Programmer response

Change the value specified for the given header field to a value which is valid. Check the case of the value specified, as some header fields have case-sensitive values.

MQHTTP40006: *Header_name* is not a valid request header:

A header that is valid only in an HTTP response message has been specified in an HTTP request message.

Explanation

A header which is valid only in an HTTP response message has been specified in an HTTP request message.

HTTP status code

`400 Bad Request`

Programmer response

Remove any headers from the HTTP request which are only valid in an HTTP response; for example, `x-msg-timestamp`.

MQHTTP40007: *Header_name* is only valid on ...:

A header has been specified in an HTTP request, but the header field is not valid for the given request verb.

Explanation

A header has been specified in an HTTP request, but the header field is not valid for the given request verb.

HTTP status code

`400 Bad Request`

Programmer response

Remove any headers from the HTTP request which are not valid for the given request verb. For example, `x-msg-encoding` is valid for HTTP **`POST`** requests, but not valid for HTTP **`GET`** or HTTP **`DELETE`** requests.

MQHTTP40008: *Header_name* maximum length is ...:

The maximum length for the given header field has been exceeded.

Explanation

The maximum length for the given header field has been exceeded.

HTTP status code

`400 Bad Request`

Programmer response

Change the value of the header field to a value which is within the range permitted for the header field.

MQHTTP40009: Header field *header_field* is not valid for ...:

A header field specified in an HTTP request is not supported by the messaging provider to which the IBM MQ bridge for HTTP is connected.

Explanation

A header field specified in an HTTP request is not supported by the messaging provider to which the IBM MQ bridge for HTTP is connected. The error occurs if a messaging provider is used that cannot support all the features of the IBM MQ bridge for HTTP.

HTTP status code

`400 Bad Request`

Programmer response

Remove the unsupported header from the HTTP request.

MQHTTP40010: Message with `Content-Type` *content_type* could not be parsed:

The content of the HTTP request is not compatible with the `Content-Type` of the request.

Explanation

The content of the HTTP request is not compatible with the `Content-Type` of the request. A common cause is badly formed `application/x-www-form-urlencoded` or `application/xml data`.

HTTP status code

`400 Bad Request`

Programmer response

Correct the content of the HTTP request so that it is in the correct format for the `Content-Type` of the request.

MQHTTP40301: You are forbidden from accessing ...:

The IBM MQ bridge for HTTP has been unable to authenticate itself for the specified destination.

Explanation

The IBM MQ bridge for HTTP has been unable to authenticate itself for the specified destination.

HTTP status code

```
403 Forbidden
```

Programmer response

Change the authentication properties of the destination so that the IBM MQ Bridge for HTTP is authorized to connect to it. Alternatively, specify a destination to which the IBM MQ bridge for HTTP is authorized to connect.

MQHTTP40302: You are forbidden from ...:

The IBM MQ bridge for HTTP has been unable to connect to the queue manager.

Explanation

The IBM MQ bridge for HTTP has been unable to connect to the queue manager. The IBM MQ bridge for HTTP security configuration is incorrect.

HTTP status code

```
403 Forbidden
```

Programmer response

Change the authentication configuration of the queue manager so that the IBM MQ Bridge for HTTP is authorized to connect to it. Alternatively, configure the IBM MQ bridge for HTTP to connect to a queue manager to which it is authorized to connect.

MQHTTP40401: The destination *destination_name* could not be found:

The destination specified in the HTTP request URI cannot be found by the IBM MQ bridge for HTTP.

Explanation

The destination specified in the HTTP request URI cannot be found by the IBM MQ bridge for HTTP.

HTTP status code

```
404 Not found
```

Programmer response

Check that the destination specified in the HTTP request URI exists, or specify an alternative destination.

MQHTTP40501: Method *method_name* not allowed:

The method specified in the HTTP request is not supported by the IBM MQ bridge for HTTP.

Explanation

The method specified in the HTTP request is not supported by the IBM MQ bridge for HTTP.

HTTP status code

```
405 Method not allowed
```

Programmer response

Change the method specified in the HTTP request to one which is supported by the IBM MQ bridge for HTTP.

MQHTTP41301: The message being posted was too large for the destination:

The destination specified in the HTTP POST request URI cannot accept messages that are as long as the message specified in the HTTP request.

Explanation

The destination specified in the HTTP POST request URI cannot accept messages that are as long as the message specified in the HTTP request.

HTTP status code

```
413 Request entity too large
```

Programmer response

Reduce the size of the message specified in the HTTP request. Alternatively, specify a destination which can support messages of the wanted length.

MQHTTP41501: The media type character set is unsupported:

The character set specified in the `Content-Type` header field is not supported by the IBM MQ bridge for HTTP.

Explanation

The character set specified in the `Content-Type` header field is not supported by the IBM MQ bridge for HTTP.

HTTP status code

```
415 Unsupported media type
```

Programmer response

Change the character set of the `Content-Type` header field to one that is supported by the IBM MQ bridge for HTTP.

MQHTTP41502: Media-type *media-type* is not supported ...:

The media-type specified in the HTTP request is not supported by the IBM MQ bridge for HTTP for the specified HTTP verb.

Explanation

The media-type specified in the HTTP request is not supported by the IBM MQ bridge for HTTP for the specified HTTP verb.

HTTP status code

```
415 Unsupported media type
```

Programmer response

Change the media-type specified in the HTTP request to one that is supported by the IBM MQ Bridge for HTTP for the specified HTTP verb.

MQHTTP41503: Media-type *media-type* is not supported ...:

The media-type specified in the HTTP request is not supported by the IBM MQ bridge for HTTP for the specified `x-msg-class` header field.

Explanation

The media-type specified in the HTTP request is not supported by the IBM MQ bridge for HTTP for the specified `x-msg-class` header field.

HTTP status code

```
415 Unsupported media type
```

Programmer response

Change the media-type specified in the HTTP request to one that is supported by the IBM MQ Bridge for HTTP for the specified `x-msg-class` header field.

MQHTTP41701: The HTTP header `Expect` is not supported:

The IBM MQ bridge for HTTP does not support the `Expect` header field.

Explanation

The `Expect` header has been specified in an HTTP request. The IBM MQ bridge for HTTP does not support the `Expect` header field.

HTTP status code

```
417 Expectation failed
```

Programmer response

Remove the `Expect` header from the HTTP request.

MQHTTP50001: There has been an unexpected problem ...:

An error has occurred in the IBM MQ bridge for HTTP.

Explanation

An error has occurred in the IBM MQ bridge for HTTP.

HTTP status code

`500 Internal server error`

Programmer response

Contact the system administrator of the IBM MQ Bridge for HTTP.

MQHTTP50201: An error has occurred between the IBM MQ bridge for HTTP and the queue manager:

An error has occurred between the IBM MQ bridge for HTTP and the queue manager

Explanation

An error has occurred between the IBM MQ bridge for HTTP and the queue manager

HTTP status code

`502 Bad Gateway`

Programmer response

Contact the system administrator of the IBM MQ Bridge for HTTP.

MQHTTP50401: Message retrieval timed out:

No message matching the specified request parameters in an HTTP **`GET`** or HTTP **`DELETE`** was returned in the timeout period.

Explanation

No message matching the specified request parameters in an HTTP **`GET`** or HTTP **`DELETE`** was returned in the timeout period. The return code indicates that no suitable message was available at any time during the life of the HTTP request.

HTTP status code

`504 Gateway timeout`

Programmer response

If a message was expected, check the header fields of the HTTP request such as `x-msg-correlId` and `x-msg-msgid`. Check that the destination specified in the HTTP request URI is correct. Try extending the wait time of the HTTP request using the `x-msg-wait` header field.

MQHTTP50501: HTTP 1.1 and upwards ...:

The HTTP protocol used in the HTTP request is not supported by the IBM MQ bridge for HTTP.

Explanation

The HTTP protocol used in the HTTP request is not supported by the IBM MQ bridge for HTTP.

HTTP status code

```
505 HTTP version not supported
```

Programmer response

Change the HTTP request to use HTTP protocol V1.1 or higher.

Message types and message mappings for WebSphere Bridge for HTTP

IBM MQ bridge for HTTP supports four message classes, `TEXT`, `BYTES`, `STREAM` and `MAP`. The message classes are mapped to JMS message types and HTTP `Content-Type`.

HTTP `POST`

The message type that arrives at the destination depends on the value of the `x-msg-class` header or the `Content-Type` of the HTTP request. Table 408 shows the HTTP `Content-Type` type that corresponds to each `x-msg-class`. Either field can be used to set the message type and message format. If both fields are set, and are set inconsistently, then a `Bad Request exception` is returned (`HTTP 400, MQHTTP20004`).

Table 408. Mapping between `x-msg-class` and HTTP `Content-Type`

x-msg-class	**HTTP Content-Type**
BYTES	application/octet-stream application/xml
TEXT	text/*
MAP	application/x-www-form-urlencoded application/xml (optional)
STREAM	application/xml (optional)

If the JMS message type is set in the `MQRFH2` header, it is mapped according to Table 409.

Table 409. Mapping between `x-msg-class` and JMS message types.

x-msg-class	**JMS message type**
BYTES	jms_bytes
TEXT	jms_text
MAP	jms_map
STREAM	jms_stream

The JMS message type is always set for a message class of `MAP` or `STREAM`. It is not always set for a message class of `BYTES` or `TEXT`. If a `MQRFH2` is to be created for the request, the JMS message type is always set. Otherwise, if no `MQRFH2` is created, no JMS message type is set. An `MQRFH2` is created if user properties are set in the request, using the `x-msg-usr` header.

If the JMS message type is set, then the message format is set to `MQFMT_NONE`, see Table 411 on page 3858:

Table 410. Mapping between x-msg-class and IBM MQ message format

x-msg-class	Message format with MQRFH2 present in message	Message format with *no* MQRFH2 present in message
BYTES	MQFMT_NONE	MQFMT_NONE
TEXT	MQFMT_NONE	MQFMT_STRING
MAP	MQFMT_NONE	Not possible
STREAM	MQFMT_NONE	Not possible

HTTP GET or DELETE

The message type or format retrieved determines the value of the x-msg-class header and the Content-Type of the HTTP response. The x-msg-class header is returned only if requested in a x-msg-headers request.

Table 411 describes the mappings between x-msg-class and Content-Type, and the message type retrieved from the queue or topic.

Table 411. Mapping message types to x-msg-class and Content-Type

Message format	JMS Message type	x-msg-class	Content-Type
Anything except MQFMT_STRING	None	BYTES	application/octet-stream
MQFMT_STRING	None	TEXT	text/plain
MQFMT_NONE	jms_bytes	BYTES	application/octet-stream
MQFMT_NONE	jms_text	TEXT	text/plain
MQFMT_NONE	jms_map	MAP	application/xml
MQFMT_NONE	jms_stream	STREAM	application/xml

MAP and STREAM message class serialization

MAP and STREAM message classes are serialized back to the client in the HTTP response in the same way as a message is serialized to a queue.

For MAP, the XML name, type, and value triplets are encoded as:

```
<map>
 <elt name="elementname1" dt="datatype1">value1</elt>
 <elt name="elementname2" dt="datatype2">value2</elt>
 ...
</map>
```

STREAM is like MAP, but it does not have element names:

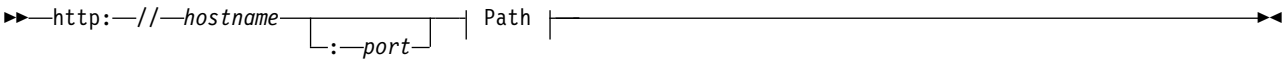
```
<stream>
 <elt dt="datatype1">value1</elt>
 <elt dt="datatype2">value2</elt>
 ...
</stream>
```

Note: datatype is one of the data types defined for defining user-defined properties and listed in "usr: HTTP x-msg-usr entity-header" on page 3845. The attribute dt="string" is omitted for string elements because the default data type is string.

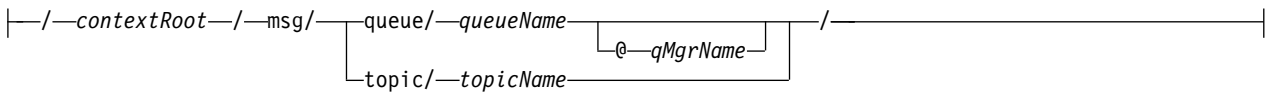
URI Format

URIs intercepted by IBM MQ bridge for HTTP.

Syntax



Path:



Note:

1. If a question mark (?) is used it must be substituted with `%3f`. For example, `orange?topic` should be specified as `orange%3ftopic`.
2. @ *qMgrName* is only valid on an HTTP **POST**

Description

Deploy the IBM MQ bridge for HTTP servlet to your JEE application server with a context root of *contextRoot*. Requests to

```
http://hostname: port/context_root/msg/queue/queueName @ qMgrName
```

and

```
http://hostname: port/context_root/msg/topic/topicString
```

are intercepted by IBM MQ bridge for HTTP.

The IBM MQ .NET classes and interfaces

IBM MQ .NET classes and interfaces are listed alphabetically. The properties, methods and constructors are described.

MQAsyncStatus.NET class

Use MQAsyncStatus to inquire on the status of previous MQI activity; for example inquiring on the success of previous asynchronous put operations. MQAsyncStatus encapsulates features of the MQSTS data structure.

Class

```
System.Object
 |
 ≪— IBM.WMQ.MQBase
  |
  ≪— IBM.WMQ.MQBaseObject
   |
   ≪— IBM.WMQ.MQAsyncStatus
```

```
public class IBM.WMQ.MQAsyncStatus extends IBM.WMQ.MQBaseObject;
```


- “Properties” on page 3860
- “Constructors” on page 3860

Properties

Test for MQException being thrown when getting properties.

public static int CompCode {get;}

The completion code from the first error or warning.

public static int Reason {get;}

The reason code from the first error or warning.

public static int PutSuccessCount {get;}

The number of successful asynchronous MQI put calls.

public static int PutWarningCount {get;}

The number of asynchronous MQI put calls that succeeded with a warning.

public static int PutFailureCount {get;}

The number of failed asynchronous MQI put calls.

public static int ObjectType {get;}

The object type for the first error. The following values are possible:

- MQC.MQOT_ALIAS_Q
- MQC.MQOT_LOCAL_Q
- MQC.MQOT_MODEL_Q
- MQC.MQOT_Q
- MQC.MQOT_REMOTE_Q
- MQC.MQOT_TOPIC
- 0, meaning that no object is returned

public static string ObjectName {get;}

The object name.

public static string ObjectQMgrName {get;}

The object queue manager name.

public static string ResolvedObjectName {get;}

The resolved object name.

public static string ResolvedObjectQMgrName {get;}

The resolved object queue manager name.

Constructors

public MQAsyncStatus() throws MQException;

Constructor method, constructs an object with fields initialized to zero or blank as appropriate.

MQAuthenticationInformationRecord.NET class

Use MQAuthenticationInformationRecord to specify information about an authenticator that is to be used in an IBM MQ TLS client connection. MQAuthenticationInformationRecord encapsulates an authentication information record, MQAIR.

Class

```
System.Object
 |
 ≪— IBM.WMQ.MQAuthenticationInformationRecord

public class IBM.WMQ.MQAuthenticationInformationRecord extends System.Object;
```

- "Properties"
- "Constructors"

Properties

Test for MQException being thrown when getting properties.

public long Version {get; set;}

Structure version number.

public long AuthInfoType {get; set;}

The type of authentication information. This attribute must be set to one of the following values:

- `OCSP` - Certificate revocation status checking is done using OCSP.
- `CRLLDAP` - Certificate revocation status checking is done using Certificate Revocation Lists on LDAP servers.

public string AuthInfoConnName {get; set;}

The DNS name or IP address of the host on which the LDAP server is running, with an optional port number. This keyword is required.

public string LDAPPassword {get; set;}

The password associated with the distinguished name of the user who is accessing the LDAP server. This property applies only when **AuthInfoType** is set to `CRLLDAP`.

public string LDAPUserName {get; set;}

The distinguished name of the user who is accessing the LDAP server. When you set this property, `LDAPUserNameLength` and `LDAPUserNamePtr` are automatically set correctly. This property applies only when `AuthInfoType` is set to `CRLLDAP`.

public string OCSPResponderURL {get; set;}

The URL at which the OCSP responder can be contacted. This property applies only when `AuthInfoType` is set to `OCSP`

This field is case sensitive. It must start with the string http:// in lowercase. The rest of the URL might be case sensitive, depending on the OCSP server implementation.

Constructors

MQAuthenticationInformationRecord();

MQDestination.NET class

Use MQDestination to access methods that are common to MQQueue and MQTopic. MQDestination is an abstract base class and cannot be instantiated.

Class

```
public class IBM.WMQ.MQDestination extends IBM.WMQ.MQManagedObject;
System.Object
        |
        ≪— IBM.WMQ.MQBase
                |
                ≪— IBM.WMQ.MQBaseObject
                        |
                        ≪— IBM.WMQ.MQManagedObject
                                |
                                ≪— IBM.WMQ.MQDestination
                                        |
                                        ≪— IBM.WMQ.MQQueue
                                        |
                                        ≪— IBM.WMQ.MQTopic
```

- "Properties"
- "Methods"
- "Constructors" on page 3864

Properties

Test for MQException being thrown when getting properties.

public DateTime CreationDateTime {get;}

The date and time that the queue or topic was created. Originally contained within MQQueue, this property has been moved into the base MQDestination class.

There is no default value.

public int DestinationType {get;}

Integer value describing the type of destination being used. Initialized from the sub classes constructor, MQQueue or MQTopic, this value can take one of these values:

- MQOT_Q
- MQOT_TOPIC

There is no default value.

Methods

public void Get(MQMessage *message***);**
public void Get(MQMessage *message***, MQGetMessageOptions** *getMessageOptions***);**
public void Get(MQMessage *message***, MQGetMessageOptions** *getMessageOptions***, int** *MaxMsgSize***);**

Throws MQException.

Gets a message from a queue if the destination is an MQQueue object or from a topic if the destination is an MQTopic object, using a default instance of MQGetMessageOptions to do the get.

If the get fails, the MQMessage object is unchanged. If it succeeds, the message descriptor and message data portions of the MQMessage are replaced with the message descriptor and message data from the incoming message.

All calls to IBM MQ from a particular MQQueueManager are synchronous. Therefore, if you perform a get with wait, all other threads using the same MQQueueManager are blocked from making further IBM MQ calls until the Get call is accomplished. If you need multiple threads to access IBM MQ simultaneously, each thread must create its own MQQueueManager object.

message
Contains the message descriptor and the returned message data. Some of the fields in the message descriptor are input parameters. It is important to ensure that the `MessageId` and `CorrelationId` input parameters are set as required.

A reconnectable client returns the reason code `MQRC_BACKED_OUT` after successful reconnection, for messages received under `MQGM_SYNCPOINT`.

getMessageOptions
Options controlling the action of the get.

Using option `MQC.MQGMO_CONVERT` might result in an exception with reason code `MQC.MQRC_CONVERTED_STRING_TOO_BIG` when converting from single-byte character codes to double byte codes. In this case, the message is copied into the buffer without conversion.

If *getMessageOptions* is not specified, the message option used is `MQGMO_NOWAIT`.

If you use the `MQGMO_LOGICAL_ORDER` option in a reconnectable client, the `MQRC_RECONNECT_INCOMPATIBLE` reason code is returned.

MaxMsgSize
The largest message this message object is to receive. If the message on the queue is larger than this size, one of two things occurs:

- If the `MQGMO_ACCEPT_TRUNCATED_MSG` flag is set in the MQGetMessageOptions object, the message is filled with as much of the message data as possible. An exception is thrown with the `MQCC_WARNING` completion code and `MQRC_TRUNCATED_MSG_ACCEPTED` reason code.
- If the `MQGMO_ACCEPT_TRUNCATED_MSG` flag is not set, the message remains on the queue. An exception is thrown with the `MQCC_WARNING` completion code and `MQRC_TRUNCATED_MSG_FAILED` reason code.

If *MaxMsgSize* is not specified, the whole message is retrieved.

```
public void Put(MQMessage message);
public void Put(MQMessage message, MQPutMessageOptions putMessageOptions);
```

Throws MQException.

Puts a message to a queue if the destination is an MQQueue object or publishes a message to a topic if the destination is an MQTopic object.

Modifications to the MQMessage object after the Put call has been accomplished do not affect the actual message on the IBM MQ queue or publication topic.

Put updates the `MessageId` and `CorrelationId` properties of the MQMessage object and does not clear message data. Further Put or Get calls refer to the updated information in the MQMessage object. For example, in the following code snippet, the first message contains `a` and the second `ab`.

```
msg.WriteString("a");
q.Put(msg,pmo);
msg.WriteString("b");
q.Put(msg,pmo);
```

message
An MQMessage object containing the message descriptor data, and message to be sent. The message descriptor can be altered as a consequence of this method. The values in the message descriptor immediately after the completion of this method are the values that were put to the queue or published to the topic.

The following reason codes are returned to a reconnectable client:

- MQRC_CALL_INTERRUPTED if the connection is broken while running a Put call on a persistent message and the reconnection is successful.
- MQRC_NONE if the connection is successful while running a Put call on a non-persistent message (see Application Recovery).

putMessageOptions
Options controlling the action of the put.

If *putMessageOptions* is not specified the default instance of MQPutMessageOptons is used.

If you use the MQPMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned.

Note: For simplicity and performance, if you want to put a single message to a queue, use MQQueueManager.Put object. You should have an MQQueue object for this.

Constructors

MQDestination is an abstract base class and cannot be instantiated. Access destinations using MQQueue and MQTopic constructors, or using MQQueueManager.AccessQueue and MQQueueManager.AccessTopic methods.

MQEnvironment.NET class

Use MQEnvironment to control how the MQQueueManager constructor is called and to select an IBM MQ MQI client connection. The MQEnvironment class contains properties that control the behavior of the IBM MQ.

Class

```
System.Object
        |
        ≪— IBM.WMQ.MQEnvironment
```

```
public class IBM.WMQ.MQEnvironment extends System.Object;
```

- "Properties - client only"
- "Properties" on page 3865
- "Constructors" on page 3866

Properties - client only

Test for MQException being thrown when getting properties.

public static int CertificateValPolicy {get; set;}

Set which TLS certificate validation policy is used to validate digital certificates received from remote partner systems. Valid values are:

- MQC.CERTIFICATE_VALIDATION_POLICY_ANY
- MQC.CERTIFICATE_VALIDATION_POLICY_RFC5280

public static ArrayList EncryptionPolicySuiteB {get; set;}

Set the level of Suite B compliant cryptography. Valid values are:

- MQC.MQ_SUITE_B_NONE - This is the default value.
- MQC.MQ_SUITE_B_128_BIT
- MQC.MQ_SUITE_B_192_BIT

public static string Channel {get; set;}
The name of the channel to connect to the target queue manager. You must set the channel property before instantiating an MQQueueManager instance in client mode.

public static int FipsRequired {get; set;}

Specify MQC.MQSSL_FIPS_YES to use only FIPS-certified algorithms if cryptography is carried out in IBM MQ. The default is MQC.MQSSL_FIPS_NO.

If cryptographic hardware is configured, the cryptographic modules used are those provided by the hardware product. Depending on the hardware in use, these might not be FIPS-certified to a particular level.

public static string Hostname {get; set;}
The TCP/IP host name of the computer on which the IBM MQ server resides. If the host name is not set, and no overriding properties are set, server bindings mode is used to connect to the local queue manager.

public static int Port {get; set;}
The port to connect to. This is the port on which the IBM MQ server is listening for incoming connection requests. The default value is 1414.

public static string SSLCipherSpec {get; set;}
Set SSLCipherSpec to the value of the CipherSpec set on the SVRCONN channel to enable TLS for the connection. The default is Null, and TLS is not enabled for the connection.

public static string sslPeerName {get; set;}

A distinguished name pattern. If sslCipherSpec is set, this variable can be used to ensure that the correct queue manager is used. If set to null (default), the DN of the queue manager is not performed. sslPeerName is ignored if sslCipherSpec is null.

Properties

Test for MQException being thrown when getting properties.

public static ArrayList HdrCompList {get; set;}
Header Data Compression List

public static int KeyResetCount {get; set;}
Indicates the number of unencrypted bytes sent and received within an TLS conversation before the secret key is renegotiated.

public static ArrayList MQAIRArray {get; set;}
An array of MQAuthenticationInformationRecord objects.

public static ArrayList MsgCompList {get; set;}
Message Data Compression List

public static string Password {get; set;}
The password to be authenticated. The password referenced from the MQCSP structure gets populated by setting this Password property.

public static string ReceiveExit {get; set;}
A receive exit allows you to examine and alter data received from a queue manager. It is normally used with a corresponding send exit at the queue manager. If ReceiveExit is set to null, no receive exit is called.

public static string ReceiveUserData {get; set;}
The user data associated with a receive exit. Limited to 32 characters.

public static string SecurityExit {get; set;}
A security exit allows you to customize the security flows that occur when an attempt is made to connect to a queue manager. If SecurityExit is set to null, no security exit is called.

public static string SecurityUserData {get; set;}
The user data associated with a security exit. Limited to 32 characters.

public static string SendExit {get; set;}
A send exit allows you to examine or alter the data sent to a queue manager. It is normally used with a corresponding receive exit at the queue manager. If `SendExit` is set to null, no send exit is called.

public static string SendUserData {get; set;}
The user data associated with a send exit. Limited to 32 characters.

public static string SharingConversations {get; set;}

The `SharingConversations` field is used on connections from .NET applications, when these applications are not using a client channel definition table (CCDT).

`SharingConversations` determines the maximum number of conversations that can be shared on a socket associated with this connection.

A value of 0 means that the channel operates as it did before IBM WebSphere MQ Version 7.0, with regard to conversation sharing, read ahead, and heartbeat.

The field is passed in the hash table of properties as a `SHARING_CONVERSATIONS_PROPERTY`, when instantiating an IBM MQ queue manager.

If you do not specify `SharingConversations`, a default value of 10 is used.

public static string SSLCryptoHardware {get; set;}

Sets the name of the parameter string required to configure the cryptographic hardware present on the system. `SSLCryptoHardware` is ignored if `sslCipherSpec` is null.

public static string SSLKeyRepository {get; set;}

Set the fully qualified file name of the key repository.

If `SSLKeyRepository` is set to null (default), the certificate `MQSSLKEYR` environment variable is used to locate the key repository. `SSLCryptoHardware` is ignored if `sslCipherSpec` is null.

Note: The `.kdb` extension is a mandatory part of the file name, but is not included as part of the value of the parameter. The directory you specify must exist. IBM MQ creates the file the first time it accesses the new key repository, unless the file already exists.

public static string UserId {get; set;}

The user ID to be authenticated. The user ID referenced from the `MQCSP` structure gets populated by setting `UserId`. Authenticate `UserId` using an API or Security exit.

Constructors

public MQEnvironment()

MQException.NET class

Use MQException to find out the completion and reason code of a failed IBM MQ function. An MQException is thrown whenever an IBM MQ error occurs.

Class

```
System.Object
        |
        ≪— System.Exception
                |
                ≪— System.ApplicationException
                        |
                        ≪— IBM.WMQ.MQException
```

```
public class IBM.WMQ.MQException extends System.ApplicationException ;
```

- "Properties"
- "Constructors"

Properties

public int CompletionCode {get; set;}

The IBM MQ completion code associated with the error. The possible values are:

- MQException.MQCC_OK
- MQException.MQCC_WARNING
- MQException.MQCC_FAILED

public int ReasonCode {get; set;}

IBM MQ reason code describing the error.

Constructors

public MQException(int *completionCode*, **int** *reasonCode*)

completionCode
The IBM MQ completion code.

reasonCode
The IBM MQ completion code.

MQGetMessageOptions.NET class

Use MQGetMessageOptions to specify how messages are retrieved. It modifies the behavior of MQDestination.Get.

Class

```
System.Object
        |
        ≪— IBM.WMQ.MQBase
                |
                ≪— IBM.WMQ.MQBaseObject
                        |
                        ≪— IBM.WMQ.MQGetMessageOptions
```

```
public class IBM.WMQ.MQGetMessageOptions extends IBM.WMQ.MQBaseObject;
```

- "Properties" on page 3868
- "Constructors" on page 3870

Properties

Note: The behavior of some of the options available in this class depends on the environment in which they are used. These elements are marked with an asterisk *.

Test for MQException being thrown when getting properties.

public int GroupStatus {get;}*
GroupStatus indicates whether the retrieved message is in a group and if it is the last in the group. Possible values are:

MQC.MQGS_LAST_MSG_IN_GROUP
Message is the last or only message in the group.

MQC.MQGS_MSG_IN_GROUP
Message is in a group, but is not the last in the group.

MQC.MQGS_NOT_IN_GROUP
Message is not in a group.

public int MatchOptions {get; set;}*
MatchOptions determines how a message is selected. The following match options can be set:

MQC.MQMO_MATCH_CORREL_ID
Correlation ID to be matched.

MQC.MQMO_MATCH_GROUP_ID
Group ID to be matched.

MQC.MQMO_MATCH_MSG_ID
Message ID to be matched.

MQC.MQMO_MATCH_MSG_SEQ_NUMBER
Match message sequence number.

MQC.MQMO_NONE
No matching required.

public int Options {get; set;}
Options control the action of MQQueue.get. Any of the following values can be specified. If more than one option is required, the values can be added, or combined using the bitwise OR operator.

MQC.MQGMO_ACCEPT_TRUNCATED_MSG
Allow truncation of message data.

MQC.MQGMO_ALL_MSGS_AVAILABLE*
Retrieve messages from a group only when all the messages in the group are available.

MQC.MQGMO_ALL_SEGMENTS_AVAILABLE*
Retrieve the segments of a logical message only when all the segments in the group are available.

MQC.MQGMO_BROWSE_FIRST
Browse from start of queue.

MQC.MQGMO_BROWSE_MSG_UNDER_CURSOR*
Browse message under browse cursor.

MQC.MQGMO_BROWSE_NEXT
Browse from the current position in the queue.

MQC.MQGMO_COMPLETE_MSG*
Retrieve only complete logical messages.

MQC.MQGMO_CONVERT
Request the application data to be converted, to conform to the `CharacterSet` and `Encoding` attributes of the MQMessage, before the data is copied into the message buffer. Because data conversion is also applied when the data is retrieved from the message buffer, applications do not set this option.

Using this option can cause problems when converting from single-byte character sets to double-byte character sets. Instead, do the conversion using the readString, readLine, and writeString methods after the message has been delivered.

MQC.MQGMO_FAIL_IF_QUIESCING
Fail if the queue manager is quiescing.

MQC.MQGMO_LOCK*
Lock the message that is browsed.

MQC.MQGMO_LOGICAL_ORDER*
Return messages in groups, and segments of logical messages, in logical order.

If you use the `MQGMO_LOGICAL_ORDER` option in a reconnectable client, the `MQRC_RECONNECT_INCOMPATIBLE` reason code is returned to the application.

MQC.MQGMO_MARK_SKIP_BACKOUT*
Allow a unit of work to be backed out without reinstating the message on the queue.

MQC.MQGMO_MSG_UNDER_CURSOR
Get message under browse cursor.

MQC.MQGMO_NONE
No other options have been specified; all options assume their default values.

MQC.MQGMO_NO_PROPERTIES
No properties of the message, except properties contained in the message descriptor (or extension) are retrieved.

MQC.MQGMO_NO_SYNCPOINT
Get message without sync point control.

MQC.MQGMO_NO_WAIT
Return immediately if there is no suitable message.

MQC.MQGMO_PROPERTIES_AS_Q_DEF
Retrieve message properties as defined by the `PropertyControl` attribute of MQQueue. Access to the message properties in the message descriptor, or extension, are not affected by the `PropertyControl` attribute.

MQC.MQGMO_PROPERTIES_COMPATIBILITY
Retrieve message properties with a prefix of `mcd`, `jms`, `usr`, or `mqext`, in `MQRFH2` headers. Other properties of the message, except properties contained in the message descriptor, or extension, are discarded.

MQC.MQGMO_PROPERTIES_FORCE_MQRFH2
Retrieve message properties, except properties contained in the message descriptor, or extension, in `MQRFH2` headers. Use `MQC.MQGMO_PROPERTIES_FORCE_MQRFH2` in applications that are expecting to retrieve properties but cannot be changed to use message handles.

MQC.MQGMO_PROPERTIES_IN_HANDLE
Retrieve message properties using a `MsgHandle`.

MQC.MQGMO_SYNCPOINT
Get the message under sync point control. The message is marked as being unavailable to other applications, but it is deleted from the queue only when the unit of work is committed. The message is made available again if the unit of work is backed out.

MQC.MQGMO_SYNCPOINT_IF_PERSISTENT*
Get message with sync point control if message is persistent.

MQC.MQGMO_UNLOCK*
Unlock a previously locked message.

MQC.MQGMO_WAIT
Wait for a message to arrive.

public string ResolvedQueueName {get;}
The queue manager sets `ResolvedQueueName` to the local name of the queue from which the message was retrieved. `ResolvedQueueName` is different from the name used to open the queue if an alias queue or model queue was opened.

public char Segmentation {get;}*
`Segmentation` indicates whether you can allow segmentation for the retrieved message. Possible values are:

MQC.MQSEG_INHIBITED
Do not allow segmentation.

MQC.MQSEG_ALLOWED
Allow segmentation

public byte SegmentStatus {get;}*
`SegmentStatus` is an output field that indicates whether the retrieved message is a segment of a logical message. If the message is a segment, the flag indicates whether it is the last segment. Possible values are:

MQC.MQSS_LAST_SEGMENT
Message is the last or only segment of the logical message.

MQC.MQSS_NOT_A_SEGMENT
Message is not a segment.

MQC.MQSS_SEGMENT
Message is a segment, but is not the last segment of the logical message.

public int WaitInterval {get; set;}
`WaitInterval` is the maximum time in milliseconds that an `MQQueue.get` call waits for a suitable message to arrive. Use `WaitInterval` with `MQC.MQGMO_WAIT`. Set a value of `MQC.MQWI_UNLIMITED` to wait an unlimited time for a message.

Constructors

public MQGetMessageOptions()
Construct a new MQGetMessageOptions object with `Options` set to `MQC.MQGMO_NO_WAIT`, `WaitInterval` set to zero, and `ResolvedQueueName` set to blank.

MQManagedObject.NET class

Use MQManagedObject to inquire and set attributes of MQDestination, MQProcess, MQQueueManager, and MQSubscription. MQManagedObject is a superclass of these classes.

Classes

```
System.Object
        |
        ≪— IBM.WMQ.MQBase
                |
                ≪— IBM.WMQ.MQBaseObject
                        |
                        ≪— IBM.WMQ.MQManagedObject
                                |
                                ≪— IBM.WMQ.MQProcess
                                |
                                ≪— IBM.WMQ.MQQueueManager
                                |
                                ≪— IBM.WMQ.MQSubscription
                                |
                                ≪— IBM.WMQ.MQDestination
```

```
public class IBM.WMQ.MQManagedObject extends IBM.WMQ.MQBaseObject;
```

- "Properties"
- "Methods" on page 3872
- "Constructors" on page 3873

Properties

Test for MQException being thrown when getting properties.

public string AlternateUserId {get; set;}
The alternate user ID, if any, set when the resource was opened. `AlternateUserID.set` is ignored when issued for an object that is opened. `AlternateUserId` is not valid for subscriptions.

public int CloseOptions {get; set;}

Set this attribute to control the way the resource is closed. The default value is `MQC.MQCO_NONE`. `MQC.MQCO_NONE` s the only permissible value for all resources other than permanent dynamic queues, temporary dynamic queues, subscriptions, and topics that are being accessed by the objects that created them.

For queues and topics, the following additional values are permissible:

MQC.MQCO_DELETE
Delete the queue if there are no messages.

MQC.MQCO_DELETE_PURGE
Delete the queue, purging any messages on it.

MQC.MQCO_QUIESCE
Request the queue be closed, receiving a warning if any messages remain (allowing them to be retrieved before final closing).

For subscriptions, the following additional values are permissible:

MQC.MQCO_KEEP_SUB
The subscription is not deleted. This option is valid only if the original subscription is durable. `MQC.MQCO_KEEP_SUB` is the default value for a durable topic.

MQC.MQCO_REMOVE_SUB
The subscription is deleted. `MQC.MQCO_REMOVE_SUB` is the default value for a non-durable, unmanaged topic.

MQC.MQCO_PURGE_SUB
The subscription is deleted. MQC.MQCO_PURGE_SUB is the default value for a non-durable, managed topic.

public MQQueueManager ConnectionReference {get;}
The queue manager to which this resource belongs.

public string MQDescription {get;}
The description of the resource as held by the queue manager. MQDescription returns an empty string for subscriptions and topics.

public boolean IsOpen {get;}
Indicates whether the resource is currently open.

public string Name {get;}
The name of the resource. The name is either the supplied on the access method, or the allocated by the queue manager for a dynamic queue.

public int OpenOptions {get; set;}
OpenOptions are set when an IBM MQ object is opened. The OpenOptions.set method is ignored and does not cause an error. Subscriptions have no OpenOptions.

Methods

public virtual void Close();
Throws MQException.

Closes the object. No further operations against this resource are permitted after calling Close. To change the behavior of the Close method, set the closeOptions attribute.

public string GetAttributeString(int *selector*, int *length*);
Throws MQException.

Gets an attribute string.

selector
Integer indicating which attribute is being queried.

length
Integer indicating the length of the string required.

public void Inquire(int[] *selectors*, int[] *intAttrs*, byte[] *charAttrs*);
Throws MQException.

Returns an array of integers and a set of character strings containing the attributes of a queue, process, or queue manager. The attributes to be queried are specified in the selectors array.

Note: Many of the more common attributes can be queried using the Get methods defined in MQManagedObject, MQQueue and MQQueueManager.

selectors
Integer array identifying the attributes with values to be inquired on.

intAttrs
The array in which the integer attribute values are returned. Integer attribute values are returned in the same order as the integer attribute selectors in the selectors array.

charAttrs
The buffer in which the character attributes are returned, concatenated. Character attributes are returned in the same order as the character attribute selectors in the selectors array. The length of each attribute string is fixed for each attribute.

public void Set(int[] *selectors*, int[] *intAttrs*, byte[] *charAttrs*);
Throws MQException.

Sets the attributes defined in the vector of selectors. The attributes to be set are specified in the selectors array.

selectors
Integer array identifying the attributes with values to be set.

intAttrs
The array of integer attribute values to be set. These values must be in the same order as the integer attribute selectors in the selectors array.

charAttrs
The buffer in which the character attributes to be set are concatenated. These values must be in the same order as the character attribute selectors in the selectors array. The length of each character attribute is fixed.

public void SetAttributeString(int *selector*, **string** *value*, **int** *length*);
Throws MQException.

Sets an attribute string.

selector
Integer indicating which attribute is being set.

value
The string to set as the attribute value.

length
Integer indicating the length of the string required.

Constructors

protected MQManagedObject()
Constructor method. This object is an abstract base class which cannot be instantiated by itself.

MQMessage.NET class

Use MQMessage to access the message descriptor and data for an IBM MQ message. MQMessage encapsulates an IBM MQ message.

Class

```
System.Object
        |
        ≪— IBM.WMQ.MQBase
                |
                ≪— IBM.WMQ.MQBaseObject
                        |
                        ≪— IBM.WMQ.MQMessage

public class IBM.WMQ.MQMessage extends IBM.WMQ.MQBaseObject;
```

Create an MQMessage object and then use the Read and Write methods to transfer data between the message and other objects in your application. Send and receive MQMessage objects using the Put and Get methods of the MQDestination, MQQueue and MQTopic classes.

Get and set the properties of the message descriptor using the properties of MQMessage. Set and Get extended message properties using the SetProperty and GetProperty methods.

- "Properties" on page 3874
- "Read and Write message methods" on page 3879
- "Buffer methods" on page 3881
- "Property methods" on page 3882

• "Constructors" on page 3883

Properties

Test for MQException being thrown when getting properties.

public string AccountingToken {get; set;}
Part of the identity context of the message; it helps an application to charge for work done as a result of the message. The default value is `MQC.MQACT_NONE`.

public string ApplicationIdData {get; set;}
Part of the identity context of the message. `ApplicationIdData` is information that is defined by the application suite, and can be used to provide additional information about the message or its originator. The default value is `""`.

public string ApplicationOriginData {get; set;}
Information defined by the application that can be used to provide additional information about the origin of the message. The default value is `""`.

public int BackoutCount {get;}
A count of the number of times the message has previously been returned and backed out by an MQQueue.Get call as part of a unit of work. The default value is zero.

public int CharacterSet {get; set;}

The coded character set identifier of character data in the message.

Set `CharacterSet` to identify the character set of character data in the message. Get `CharacterSet` to find out in what character set has been used to encode character data in the message.

.NET applications always run in Unicode, whereas in other environments applications run in the same character set as the queue manager is running under.

The ReadString and ReadLine methods convert the character data in the message to Unicode for you.

The WriteString method converts from Unicode to the character set encoded in `CharacterSet`. If `CharacterSet` is set to its default value, `MQC.MQCCSI_Q_MGR`, which is 0, no conversion takes place and `CharacterSet` is set to 1200. If you set `CharacterSet` to some other value, WriteString converts from Unicode to the alternate value.

Note: Other read and write methods do not use `CharacterSet`.

- ReadChar and WriteChar read and write a Unicode character to and from the message buffer without conversion.
- ReadUTF and WriteUTF convert between a Unicode string in the application, and a UTF-8 string, prefixed by a 2-byte length field, in the message buffer.
- Byte methods transfer bytes between the application and the message buffer without alteration.

public byte[] CorrelationId {get; set;}

- For an MQQueue.Get call, the correlation identifier of the message to be retrieved. The queue manager returns the first message with a message identifier and a correlation identifier that match the message descriptor fields. The default value, `MQC.MQCI_NONE`, helps any correlation identifier to match.
- For an MQQueue.Put call, the correlation identifier to set.

public int DataLength {get;}
The number of bytes of message data remaining to be read.

public int DataOffset {get; set;}
The current cursor position within the message data. Reads and writes take effect at the current position.

public int Encoding {get; set;}

The representation used for numeric values in the application message data. Encoding applies to binary, packed decimal, and floating point data. The behavior of the read and write methods for these numeric formats is altered accordingly. Construct a value for the encoding field by adding one value from each of these three sections. Alternatively, construct the value combining the values from each of the three sections using the bitwise OR operator.

1. Binary integer

   MQC.MQENC_INTEGER_NORMAL
   Big-endian integers.

   MQC.MQENC_INTEGER_REVERSED
   Little-endian integers, as used in Intel architecture.

2. Packed-decimal

   MQC.MQENC_DECIMAL_NORMAL
   Big-endian packed-decimal, as used by z/OS.

   MQC.MQENC_DECIMAL_REVERSED
   Little-endian packed-decimal.

3. Floating-point

   MQC.MQENC_FLOAT_IEEE_NORMAL
   Big-endian IEEE floats.

   MQC.MQENC_FLOAT_IEEE_REVERSED
   Little-endian IEEE floats, as used Intel architecture.

   MQC.MQENC_FLOAT_S390
   z/OS format floating points.

The default value is:

```
MQC.MQENC_INTEGER_REVERSED |
MQC.MQENC_DECIMAL_REVERSED |
MQC.MQENC_FLOAT_IEEE_REVERSED
```

The default setting causes WriteInt to write a little-endian integer, and ReadInt to read a little-endian integer. If you set the flag MQC.MQENC_INTEGER_NORMAL flag instead, WriteInt writes a big-endian integer, and ReadInt reads a big-endian integer.

Note: A loss in precision can occur when converting from IEEE format floating points to zSeries format floating points.

public int Expiry {get; set;}

An expiry time expressed in tenths of a second, set by the application that puts the message. After the expiry time of a message has elapsed, it is eligible to be discarded by the queue manager. If the message specified one of the MQC.MQRO_EXPIRATION flags, a report is generated when the message is discarded. The default value is MQC.MQEI_UNLIMITED, meaning that the message never expires.

public int Feedback {get; set;}

Use Feedback with a message of type MQC.MQMT_REPORT to indicate the nature of the report. The following feedback codes are defined by the system:

- MQC.MQFB_EXPIRATION
- MQC.MQFB_COA
- MQC.MQFB_COD
- MQC.MQFB_QUIT
- MQC.MQFB_PAN
- MQC.MQFB_NAN

- MQC.MQFB_DATA_LENGTH_ZERO
- MQC.MQFB_DATA_LENGTH_NEGATIVE
- MQC.MQFB_DATA_LENGTH_TOO_BIG
- MQC.MQFB_BUFFER_OVERFLOW
- MQC.MQFB_LENGTH_OFF_BY_ONE
- MQC.MQFB_IIH_ERROR

Application-defined feedback values in the range MQC.MQFB_APPL_FIRST to MQC.MQFB_APPL_LAST can also be used. The default value of this field is MQC.MQFB_NONE, indicating that no feedback is provided.

public string Format {get; set;}
A format name used by the sender of the message to indicate the nature of the data in the message to the receiver. You can use your own format names, but names beginning with the letters MQ have meanings that are defined by the queue manager. The queue manager built-in formats are:

MQC.MQFMT_ADMIN
Command server request/reply message.

MQC.MQFMT_COMMAND_1
Type 1 command reply message.

MQC.MQFMT_COMMAND_2
Type 2 command reply message.

MQC.MQFMT_DEAD_LETTER_HEADER
Dead-letter header.

MQC.MQFMT_EVENT
Event message.

MQC.MQFMT_NONE
No format name.

MQC.MQFMT_PCF
User-defined message in programmable command format.

MQC.MQFMT_STRING
Message consisting entirely of characters.

MQC.MQFMT_TRIGGER
Trigger message

MQC.MQFMT_XMIT_Q_HEADER
Transmission queue header.

The default value is MQC.MQFMT_NONE.

public byte[] GroupId {get; set;}
A byte string that identifies the message group to which the physical message belongs. The default value is MQC.MQGI_NONE.

public int MessageFlags {get; set;}
Flags controlling the segmentation and status of a message.

public byte[] MessageId {get; set;}

For an MQQueue.Get call, this field specifies the message identifier of the message to be retrieved. Normally, the queue manager returns the first message with a message identifier and correlation identifier that match the message descriptor fields. Allow any message identifier to match using the special value MQC.MQMI_NONE.

For an MQQueue.Put call, this field specifies the message identifier to use. If MQC.MQMI_NONE s specified, the queue manager generates a unique message identifier when the message is put. The value of this member variable is updated after the put, to indicate the message identifier that was used. The default value is MQC.MQMI_NONE.

public int MessageLength {get;}
The number of bytes of message data in the MQMessage object.

public int MessageSequenceNumber {get; set;}
The sequence number of a logical message within a group.

public int MessageType {get; set;}
Indicates the type of the message. The following values are currently defined by the system:

- MQC.MQMT_DATAGRAM
- MQC.MQMT_REPLY
- MQC.MQMT_REPORT
- MQC.MQMT_REQUEST

Application-defined values can also be used, in the range MQC.MQMT_APPL_FIRST to MQC.MQMT_APPL_LAST. The default value of this field is MQC.MQMT_DATAGRAM.

public int Offset {get; set;}
In a segmented message, the offset of data in a physical message from the start of a logical message.

public int OriginalLength {get; set;}
The original length of a segmented message.

public int Persistence {get; set;}

Message persistence. The following values are defined:

- MQC.MQPER_NOT_PERSISTENT

  If you set this option in a reconnectable client, the MQRC_NONE reason code is returned to the application when the connection is successful.
- MQC.MQPER_PERSISTENT

  If you set this option in a reconnectable client, the MQRC_CALL_INTERRUPTED reason code is returned to the application after the connection is successful.
- MQC.MQPER_PERSISTENCE_AS_Q_DEF

The default value is MQC.MQPER_PERSISTENCE_AS_Q_DEF, which takes the persistence for the message from the default persistence attribute of the destination queue.

public int Priority {get; set;}
The message priority. The special value MQC.MQPRI_PRIORITY_AS_Q_DEF can also be set in outbound message. The priority for the message is then taken from the default priority attribute of the destination queue. The default value is MQC.MQPRI_PRIORITY_AS_Q_DEF.

public int PropertyValidation {get; set;}
Specifies whether validation of properties takes place when a property of the message is set. Possible values are:

- MQCMHO_DEFAULT_VALIDATION
- MQCMHO_VALIDATE
- MQCMHO_NO_VALIDATION

The default value is MQCMHO_DEFAULT_VALIDATION.

public string PutApplicationName {get; set;}
The name of the application that put the message. The default value is "".

public int PutApplicationType {get; set;}

The type of application that put the message. PutApplicationType can be a system-defined or user-defined value. The following values are defined by the system:

- MQC.MQAT_AIX
- MQC.MQAT_CICS
- MQC.MQAT_DOS
- MQC.MQAT_IMS
- MQC.MQAT_MVS
- MQC.MQAT_OS2
- MQC.MQAT_OS400
- MQC.MQAT_QMGR
- MQC.MQAT_UNIX
- MQC.MQAT_WINDOWS
- MQC.MQAT_JAVA

The default value is MQC.MQAT_NO_CONTEXT, which indicates that no context information is present in the message.

public DateTime PutDateTime {get; set;}

The time and date that the message was put.

public string ReplyToQueueManagerName {get; set;}

The name of the queue manager to send reply or report messages. The default value is "", and the queue manager provides the ReplyToQueueManagerName.

public string ReplyToQueueName {get; set;}

The name of the message queue to which the application that issued the get request for the message sends MQC.MQMT_REPLY and MQC.MQMT_REPORT messages. The default ReplyToQueueName is "".

public int Report {get; set;}

Use Report to specify options about report and reply messages:

- Whether reports are required.
- Whether the application message data is to be included in the reports.
- How to set the message and correlation identifiers in the report or reply.

Any combination of the four report types can be requested:

- Specify any combination of the four report types. Selecting any of the three options for each report type, depending on whether the application message data is to be included in the report message.
  1. Confirm on arrival
     - MQC.MQRO_COA
     - MQC.MQRO_COA_WITH_DATA
     - MQC.MQRO_COA_WITH_FULL_DATA **
  2. Confirm on delivery
     - MQC.MQRO_COD
     - MQC.MQRO_COD_WITH_DATA
     - MQC.MQRO_COD_WITH_FULL_DATA **
  3. Exception
     - MQC.MQRO_EXCEPTION
     - MQC.MQRO_EXCEPTION_WITH_DATA
     - MQC.MQRO_EXCEPTION_WITH_FULL_DATA **

4. Expiration
   - MQC.MQRO_EXPIRATION
   - MQC.MQRO_EXPIRATION_WITH_DATA
   - MQC.MQRO_EXPIRATION_WITH_FULL_DATA **

   Note: Values marked with ** in the list are not supported by z/OS queue managers. Do not use them if your application is likely to access a z/OS queue manager, regardless of the platform on which the application is running.

- Specify one of the following to control how the message ID is generated for the report or reply message:
  - MQC.MQRO_NEW_MSG_ID
  - MQC.MQRO_PASS_MSG_ID
- Specify one of the following to control how the correlation ID of the report or reply message is to be set:
  - MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID
  - MQC.MQRO_PASS_CORREL_ID
- Specify one of the following to control the disposition of the original message when it cannot be delivered to the destination queue:
  - MQC.MQRO_DEAD_LETTER_Q
  - MQC.MQRO_DISCARD_MSG **
- If no report options are specified, the default is:

  ```
  MQC.MQRO_NEW_MSG_ID |
  MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID |
  MQC.MQRO_DEAD_LETTER_Q
  ```
- You can specify one or both of the following to request that the receiving application sends a positive action or negative action report message.
  - MQC.MQRO_PAN
  - MQC.MQRO_NAN

public int TotalMessageLength {get;}
The total number of bytes in the message as stored on the message queue from which this message was received.

public string UserId {get; set;}
UserId is part of the identity context of the message. The queue manager generally provides the value. You can override the value if you have authority to set the identity context.

public int Version {get; set;}
The version of the MQMD structure in use.

Read and Write message methods

The Read and Write methods perform the same functions as the members of the BinaryReader and BinaryWriter classes in the .NET System.IO namespace. See MSDN for the full language syntax and usage examples. The methods read or write from the current position in the message buffer. They move the current position forward by the number of bytes read or written.

Note: If the message data contains an MQRFH or MQRFH2 header, you must use the ReadBytes method to read the data.

- All the methods throw IOException.
- The ReadFully methods automatically resize the target byte or sbyte array to fit the message exactly. A null array is also resized.
- Read methods throw EndOfStreamException.

- WriteDecimal methods throw MQException.
- ReadString, ReadLine and WriteString methods convert between Unicode and the character set of the message; see `CharacterSet` .
- The Decimal methods read and write packed decimal numbers encoded either in big-endian, `MQC.MQENC_DECIMAL_NORMAL`, or little-endian `MQC.MQENC_DECIMAL_REVERSE` format, according to the value of `Encoding`. Decimal ranges and corresponding .NET types are as follows:

  `Decimal2/short`
  -999 to 999

  `Decimal4/int`
  -9999999 to 9999999

  `Decimal8/long`
  -999999999999999 to 999999999999999
- The Double and Float methods read and write floating values encoded in IEE big-endian and little-endian formats, `MQC.MQENC_FLOAT_IEEE_NORMAL` and `MQC.MQENC_FLOAT_IEEE_REVERSED`, or in S/390 format, `MQC.MQENC_FLOAT_S390`, according to the value of `Encoding`.
- The Int methods read and write integer values encoded in big-endian, `MQC.MQENC_INTEGER_NORMAL`, or little-endian, `MQC.MQENC_INTEGER_REVERSED`, format, according to the value of `Encoding`. The integers are all signed, except for the addition of an unsigned 2-byte integer type. The integer sizes, and .NET and IBM MQ types are as follows:

  2 byte `short`, `Int2`, `ushort`, `UInt2`

  4 byte `int`, `Int4`

  8 byte `long`, `Int8`
- WriteObject transfers the class of an object, the values of its non-transient and non-static fields, and the fields of its supertypes, to the message buffer.
- ReadObject creates an object from the class of the object, the signature of the class, and the values of its non-transient and non-static fields, and the fields of its supertypes.

Table 412. Read and Write message methods

Target type	Method signatures
`Boolean`	`public bool ReadBoolean();` `public void WriteBoolean(bool value);`
`Byte`	`public byte ReadByte()` `public byte ReadUnsignedByte()` `public void Write(int value)` `public void WriteByte(int value)` `public void WriteByte(byte value)` `public void WriteByte(sbyte value)`
`Bytes`	`public byte[] ReadBytes(int count)` `public void ReadFully(ref byte[] value)` `public void ReadFully(ref sbyte[] value)` `public void ReadFully(ref byte[] value, int offset,int length)` `public void ReadFully(ref sbyte[] value, int offset,int length)` `public void Write(byte[] value)` `public void Write(sbyte[] value)` `public void Write(byte[] value, int offset,int length)` `public void Write(sbyte[] value, int offset,int length)` `public void WriteBytes(string value)`
`Decimal2`	`public void WriteDecimal2(short value)`
`Decimal4`	`public void WriteDecimal4(short value)`
`Decimal8`	`public void WriteDecimal8(short value)`

Table 412. Read and Write message methods (continued)

Target type	Method signatures
Double	public double ReadDouble() public void WriteDouble(double *value*)
Float	public float ReadFloat() public void WriteFloat(float *value*)
Int2	public void WriteInt2(int *value*)
Int4	public int readDecimal4() public int ReadInt() public int ReadInt4() public void WriteInt(int *value*) public void WriteInt4(int *value*)
Int8	public void WriteInt8(long *value*)
Long	public long ReadDecimal8() public long ReadLong() public long ReadInt8() public void WriteLong(long *value*)
Object	public Object ReadObject() public void WriteObject(Object *object*)
Short	public short ReadShort() public short ReadDecimal2() public short ReadInt2() public void WriteShort(int *value*)
string	public string ReadString(int *length*) public void WriteString(string *string*)
Unsigned Short	public ushort ReadUnsignedShort() public ushort ReadUInt2()
Unicode	public string ReadLine() public char ReadChar() public void WriteChar(int *value*) public void WriteChars(string *string*)
UTF	public string ReadUTF() public void WriteUTF(string *string*)

Buffer methods

public void ClearMessage();

Throws IOException.

Discards any data in the message buffer and sets the data offset back to zero.

public void ResizeBuffer(int *size*)

Throws IOException.

A hint to the MQMessage object about the size of buffer that might be required for subsequent get operations. If the message currently contains message data, and the new size is less than the current size, the message data is truncated.

public void Seek(int *pos*)

Throws IOException, ArgumentOutOfRangeException, ArgumentException.

Moves the cursor to the absolute position in the message buffer given by *pos*. Subsequent reads and writes act at this position in the buffer.

public int SkipBytes(int *i*)

Throws IOException, EndOfStreamException.

Moves forward `n` bytes in the message buffer and returns `n`, the number of bytes skipped.

SkipBytes method blocks until one of the following events occurs:

- All the bytes are skipped
- The end of message buffer is detected
- An exception is thrown

Property methods

public void DeleteProperty(string *name*);

Throws MQException.

Deletes a property with the specified name from the message.

name
The name of the property to delete.

public System.Collections.IEnumerator GetPropertyNames(string *name*)

Throws MQException.

Returns an IEnumerator of all the property names matching the specified name. The percent sign '`%`' can be used at the end of the name as a wildcard character to filter the properties of the message, matching on zero, or more characters, including the period.

name
The name of the property to match on.

- All the SetProperty and GetProperty methods throw MQException

Table 413. SetProperty and GetProperty methods

Type	Method signatures
Boolean	`public boolean GetBooleanProperty(string name);` `public boolean GetBooleanProperty(string name, MQPropertyDescriptor pd);` `public void SetBooleanProperty(string name, boolean value);` `public void SetBooleanProperty(string name, MQPropertyDescriptor pd, boolean value);`
Byte	`public sbyte GetByteProperty(string name);` `public sbyte GetByteProperty(string name, MQPropertyDescriptor pd);` `public void SetByteProperty(string name, sbyte value);` `public void SetByteProperty(string name, MQPropertyDescriptor pd, sbyte value);`
Bytes	`public sbyte[] GetBytesProperty(string name);` `public sbyte[] GetBytesProperty(string name, MQPropertyDescriptor pd);` `public void SetBytesProperty(string name, sbyte[] value);` `public void SetBytesProperty(string name, MQPropertyDescriptor pd, sbyte[] value);`
Double	`public double GetDoubleProperty(string name);` `public double GetDoubleProperty(string name, MQPropertyDescriptor pd);` `public void SetDoubleProperty(string name, double value);` `public void SetDoubleProperty(string name, MQPropertyDescriptor pd, double value);`
Float	`public float GetFloatProperty(string name);` `public float GetFloatProperty(string name, MQPropertyDescriptor pd);` `public void SetFloatProperty(string name, float value);` `public void SetFloatProperty(string name, MQPropertyDescriptor pd, float value);`

Table 413. SetProperty and GetProperty methods (continued)

Type	Method signatures
Int2	public short GetInt2Property(string *name*); public short GetInt2Property(string *name*, MQPropertyDescriptor *pd*); public void SetInt2Property(string *name*, short *value*); public void SetInt2Property(string *name*, MQPropertyDescriptor *pd*, short *value*);
Int4	public int GetInt4Property(string *name*); public int GetInt4Property(string *name*, MQPropertyDescriptor *pd*); public void SetInt4Property(string *name*, int *value*); public void SetInt4Property(string *name*, MQPropertyDescriptor *pd*, int *value*);
Int8	public long GetInt8Property(string *name*); public long GetInt8Property(string *name*, MQPropertyDescriptor *pd*); public void SetInt8Property(string *name*, long *value*); public void SetInt8Property(string *name*, MQPropertyDescriptor *pd*, long *value*);
Long	public long GetLongProperty(string *name*); public long GetLongProperty(string *name*, MQPropertyDescriptor *pd*); public void SetLongProperty(string *name*, long *value*); public void SetLongProperty(string *name*, MQPropertyDescriptor *pd*, long *value*);
Object	public Object GetObjectProperty(string *name*); public Object GetObjectProperty(string *name*, MQPropertyDescriptor *pd*); public void SetObjectProperty(string *name*, Object *value*); public void SetObjectProperty(string *name*, MQPropertyDescriptor *pd*, Object *value*);
Short	public short GetShortProperty(string *name*); public short GetShortProperty(string *name*, MQPropertyDescriptor *pd*); public void SetShortProperty(string *name*, short *value*); public void SetShortProperty(string *name*, MQPropertyDescriptor *pd*, short *value*);
string	public string GetStringProperty(string *name*); public string GetStringProperty(string *name*, MQPropertyDescriptor *pd*); public void SetStringProperty(string *name*, string *value*); public void SetStringProperty(string *name*, MQPropertyDescriptor *pd*, string *value*);

Constructors

public MQMessage();

Creates an MQMessage object with default message descriptor information and an empty message buffer.

MQProcess.NET class

Use MQProcess to query the attributes of an IBM MQ process. Create an MQProcess object using a constructor, or an MQQueueManager AccessProcess method.

Class

```
System.Object
        |
        ≪— IBM.WMQ.MQBase
                |
                ≪— IBM.WMQ.MQBaseObject
                        |
                        ≪— IBM.WMQ.MQManagedObject
                                |
                                ≪— IBM.WMQ.MQProcess
```

```
public class IBM.WMQ.MQProcess extends IBM.WMQ.MQManagedObject;
```

- "Properties"
- "Constructors" on page 3885

Properties

Test for MQException being thrown when getting properties.

public string ApplicationId {get;}

Gets the character string that identifies the application to be started. `ApplicationId` is used by a trigger monitor application. `ApplicationId` is sent to the initiation queue as part of the trigger message.

The default value is null.

public int ApplicationType {get;}

Identifies the type of the process to be started by a trigger monitor application. Standard types are defined, but others can be used:

- `MQAT_AIX`
- `MQAT_CICS`
- `MQAT_IMS`
- `MQAT_MVS`
- `MQAT_NATIVE`
- `MQAT_OS400`
- `MQAT_UNIX`
- `MQAT_WINDOWS`
- `MQAT_JAVA`
- `MQAT_USER_FIRST`
- `MQAT_USER_LAST`

The default value is `MQAT_NATIVE`.

public string EnvironmentData {get;}

Gets information about the environment of the application that is to be started.

The default value is null.

public string UserData {get;}

Gets information the user has provided about the application to be started.

The default value is null.

Constructors

public MQProcess(MQQueueManager *queueManager*, **string** *processName*, **int** *openOptions*);
public MQProcess(MQQueueManager *qMgr*, **string** *processName*, **int** *openOptions*, **string** *queueManagerName*, **string** *alternateUserId*);

Throws MQException.

Access an IBM MQ process on queue manager *qMgr* to inquire on process attributes.

qMgr
Queue manager to access.

processName
The name of the process to open.

openOptions
Options that control the opening of the process. The valid options that can be added, or combined using a bitwise OR, are:

- MQC.MQOO_FAIL_IF_QUIESCING
- MQC.MQOO_INQUIRE
- MQC.MQOO_SET
- MQC.MQOO_ALTERNATE_USER_AUTHORITY

queueManagerName
The name of the queue manager on which the process is defined. You can leave a blank or null queue manager name if the queue manager is the same as the one the process is accessing.

alternateUserId

If MQC.MQOO_ALTERNATE_USER_AUTHORITY is specified in the **openOptions** parameter, *alternateUserId* specifies the alternative user ID used to check the authorization for the action. If MQOO_ALTERNATE_USER_AUTHORITY is not specified, *alternateUserId* can be blank or null.

Default user authority is used for connection to the queue manager if MQC.MQOO_ALTERNATE_USER_AUTHORITY is not specified.

public MQProcess MQQueueManager.AccessProcess(string *processName*, **int** *openOptions*);
public MQProcess MQQueueManager.AccessProcess(string *processName*, **int** *openOptions*, **string** *queueManagerName*, **string** *alternateUserId*);

Throws MQException.

Access an IBM MQ process on this queue manager to inquire on process attributes.

processName
The name of the process to open.

openOptions
Options that control the opening of the process. The valid options that can be added, or combined using a bitwise OR, are:

- MQC.MQOO_FAIL_IF_QUIESCING
- MQC.MQOO_INQUIRE
- MQC.MQOO_SET
- MQC.MQOO_ALTERNATE_USER_AUTHORITY

queueManagerName
The name of the queue manager on which the process is defined. You can leave a blank or null queue manager name if the queue manager is the same as the one the process is accessing.

alternateUserId

If MQC.MQOO_ALTERNATE_USER_AUTHORITY is specified in the **openOptions** parameter, *alternateUserId* specifies the alternative user ID used to check the authorization for the action. If MQOO_ALTERNATE_USER_AUTHORITY is not specified, *alternateUserId* can be blank or null.

Default user authority is used for connection to the queue manager if MQC.MQOO_ALTERNATE_USER_AUTHORITY is not specified.

MQPropertyDescriptor.NET class

Use MQPropertyDescriptor as a parameter to MQMessage GetProperty and SetProperty methods. MQPropertyDescriptor describes an MQMessage property.

Class

```
System.Object
        |
        ≪— IBM.WMQ.MQPropertyDescriptor

public class IBM.WMQ.MQPropertyDescriptor extends System.Object;
```

- "Properties"
- "Constructors" on page 3887

Properties

Test for MQException being thrown when getting properties.

public int Context {get; set;}

The message context the property belongs to. Possible values are:

MQC.MQPD_NO_CONTEXT
The property is not associated with a message context.

MQC.MQPD_USER_CONTEXT
The property is associated with the user context.

If the user is authorized, a property associated with the user context is saved when a message is retrieved. A subsequent Put method referencing the saved context, can pass the property into the new message.

public int CopyOptions {get; set;}

CopyOptions describes which type of message the property can be copied into.

When a queue manager receives a message containing an IBM MQ defined property that the queue manager recognizes as being incorrect, the queue manager corrects the value of the CopyOptions field.

Any combination of the following options can be specified. Combine the options by adding the values, or using bitwise OR.

MQC.MQCOPY_ALL
The property is copied into all types of subsequent messages.

MQC.MQCOPY_FORWARD
The property is copied into a message being forwarded.

MQC.MQCOPY_PUBLISH
The property is copied into the message received by a subscriber when a message is being published.

MQC.MQCOPY_REPLY
The property is copied into a reply message.

MQC.MQCOPY_REPORT
The property is copied into a report message.

MQC.MQCOPY_DEFAULT
The value indicated no other copy options have been specified. No relationship exists between the property and subsequent messages. MQC.MQCOPY_DEFAULT is always returned for message descriptor properties.

MQC.MQCOPY_NONE
The same as MQC.MQCOPY_DEFAULT

public int Options { set; }
Options defaults to CMQC.MQPD_NONE. You cannot set any other value.

public int Support { get; set; }

Set Support to specify the level of support required for IBM MQ-defined message properties. Support for all other properties is optional. Any or none of the following values can be specified

MQC.MQPD_SUPPORT_OPTIONAL
The property is accepted by a queue manager even if it is not supported. The property can be discarded in order for the message to flow to a queue manager that does not support message properties. This value is also assigned to properties that are not IBM MQ defined.

MQC.MQPD_SUPPORT_REQUIRED
Support for the property is required. If you put the message to a queue manager that does not support the IBM MQ-defined property, the method fails. It returns completion code MQC.MQCC_FAILED and reason code MQC.MQRC_UNSUPPORTED_PROPERTY.

MQC.MQPD_SUPPORT_REQUIRED_IF_LOCAL

Support for the property is required, if the message is destined for a local queue. If you put the message to a local queue on a queue manager that does not support the IBM MQ-defined property, the method fails. It returns completion code MQC.MQCC_FAILED and reason code MQC.MQRC_UNSUPPORTED_PROPERTY.

No check is made if the message is put to a remote queue manager.

Constructors

PropertyDescriptor();
Create a property descriptor.

MQPutMessageOptions.NET class

Use MQPutMessageOptions to specify how messages are sent. It modifies the behavior of MQDestination.Put.

Class

```
System.Object
        |
        ≪— IBM.WMQ.MQBase
                |
                ≪— IBM.WMQ.MQBaseObject
                        |
                        ≪— IBM.WMQ.MQPutMessageOptions
```

```
public class IBM.WMQ.MQPutMessageOptions extends IBM.WMQ.MQBaseObject;
```

• "Properties" "Constructors" on page 3890

Properties

Test for MQException being thrown when getting properties.

Note: The behavior of some of the options available in this class depends on the environment in which they are used. These elements are marked with an asterisk, *.

public MQQueue ContextReference {get; set;}

If the options field includes MQC.MQPMO_PASS_IDENTITY_CONTEXT or MQC.MQPMO_PASS_ALL_CONTEXT, set this field to refer to the MQQueue from which to take the context information.

The initial value of this field is null.

public int InvalidDestCount {get;} *

Generally, used for distribution lists, InvalidDestCount indicates the number of messages that could not be sent to queues in a distribution list. The count includes queues that failed to open and also the queues that were opened successfully, but for which the put operation had failed.

.NET does not support distribution lists, but InvalidDestCount is set when opening a single queue.

public int KnownDestCount {get;} *

Generally used for distribution lists, KnownDestCount indicates the number of messages that the current call has sent successfully to queues that resolve to local queues.

.NET does not support distribution lists, but InvalidDestCount is set when opening a single queue.

public int Options {get; set;}

Options that control the action of MQDestination.put and MQQueueManager.put. Any or none of the following values can be specified. If more than one option is required, the values can be added or combined using the bitwise OR operator.

MQC.MQPMO_ASYNC_RESPONSE
This option causes the MQDestination.put call to be made asynchronously, with some response data.

MQC.MQPMO_DEFAULT_CONTEXT
Associate default context with the message.

MQC.MQPMO_FAIL_IF_QUIESCING
Fail if the queue manager is quiescing.

MQC.MQPMO_LOGICAL_ORDER *
Put logical messages and segments in message groups into their logical order.

If you use the MQPMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned to the application.

MQC.MQPMO_NEW_CORREL_ID *
Generate a new correlation ID for each sent message.

MQC.MQPMO_NEW_MSG_ID *
Generate a new message ID for each sent message.

MQC.MQPMO_NONE
No options specified. Do not use with other options.

MQC.MQPMO_NO_CONTEXT
No context is to be associated with the message.

MQC.MQPMO_NO_SYNCPOINT
Put a message without sync point control. If the sync point control option is not specified, a default of no sync point is assumed.

MQC.MQPMO_PASS_ALL_CONTEXT
Pass all context from an input queue handle.

MQC.MQPMO_PASS_IDENTITY_CONTEXT
Pass identity context from an input queue handle.

MQC.MQPMO_RESPONSE_AS_Q_DEF

For an MQDestination.put call, this option takes the put response type from DEFPRESP attribute of the queue.

For an MQQueueManager.put call, this option causes the call to be made synchronously.

MQC.MQPMO_RESPONSE_AS_TOPIC_DEF
MQC.MQPMO_RESPONSE_AS_TOPIC_DEF is a synonym for MQC.MQPMO_RESPONSE_AS_Q_DEF for use with topic objects.

MQC.MQPMO_RETAIN

The publication being sent is to be retained by the queue manager. If this option is used and the publication cannot be retained, the message is not published and the call fails with MQC.MQRC_PUT_NOT_RETAINED.

Request a copy of this publication after the time it was published, by calling the MQSubscription.RequestPublicationUpdate method. The saved publication is sent to applications that create a subscription without setting the MQC.MQSO_NEW_PUBLICATIONS_ONLY option. Check the MQIsRetained message property of a publication, when it is received, to find out if it was the retained publication.

When retained publications are requested by a subscriber, the subscription used might contain a wildcard in the topic string. If there are multiple retained publications in the topic tree that match the subscription, they are all sent.

MQC.MQPMO_SET_ALL_CONTEXT
Set all context from the application.

MQC.MQPMO_SET_IDENTITY_CONTEXT
Set identity context from the application.

MQC.MQPMO_SYNC_RESPONSE
This option causes the MQDestination.put or MQQueueManager.put call to be made synchronously, with full response data.

MQC.MQPMO_SUPPRESS_REPLYTO
Any information filled into the ReplyToQueueName and ReplyToQueueManagerName fields of the publication is not passed on to subscribers. If this option is used in combination with a report option that requires a ReplyToQueueName, the call fails with MQC.MQRC_MISSING_REPLY_TO_Q.

MQC.MQPMO_SYNCPOINT
Put a message with sync point control. The message is not visible outside the unit of work until the unit of work is committed. If the unit of work is backed out, the message is deleted.

public int RecordFields {get; set;} *

Information about distribution lists. Distribution lists are not supporting in .NET.

public string ResolvedQueueManagerName {get;}

An output field set by the queue manager to the name of the queue manager that owns the queue specified by the remote queue name. ResolvedQueueManagerName might be different from the name of the queue manager from which the queue was accessed if the queue is a remote queue.

A nonblank value is returned only if the object is a single queue. If the object is a distribution list or a topic, the value returned is undefined.

public string ResolvedQueueName {get;}

An output field that is set by the queue manager to the name of the queue on which the message is placed. ResolvedQueueName might be different from the name used to open the queue if the opened queue was an alias or model queue.

A non-blank value is returned only if the object is a single queue. If the object is a distribution list or a topic, the value returned is undefined.

public int UnknownDestCount {get;} *

Generally used for distribution lists, UnknownDestCount is an output field set by the queue manager. It reports the number of messages that the current call has sent successfully to queues that resolve to remote queues.

.NET does not support distribution lists, but InvalidDestCount is set when opening a single queue.

Constructors

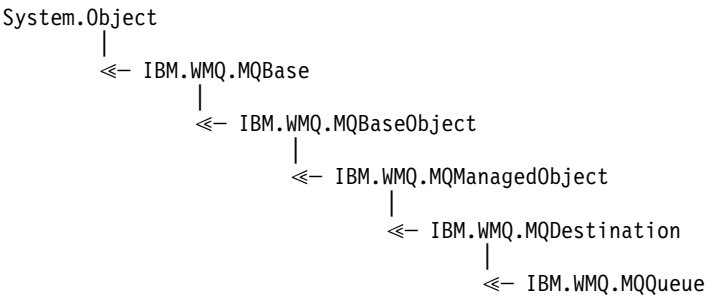
public MQPutMessageOptions();

Construct a new MQPutMessageOptions object with no options set, and a blank ResolvedQueueName and ResolvedQueueManagerName.

MQQueue.NET class

Use MQQueue to send and receive messages, and query attributes of an IBM MQ queue. Create an MQQueue object using a constructor, or an MQQueueManager.AccessProcess method.

Class




```
public class IBM.WMQ.MQQueue extends IBM.WMQ.MQDestination;
```

- "Properties"
- "Methods" on page 3893
- "Constructors" on page 3896

Properties

Test for MQException being thrown when getting properties.

public int ClusterWorkLoadPriority {get;}
Specifies the priority of the queue. This parameter is valid only for local, remote, and alias queues.

public int ClusterWorkLoadRank {get;}
Specifies the rank of the queue. This parameter is valid only for local, remote, and alias queues.

public int ClusterWorkLoadUseQ {get;}
Specifies the behavior of an MQPUT operation when the target queue has a local instance and at least one remote cluster instance. This parameter does not apply if the MQPUT originates from a cluster channel. This parameter is valid only for local queues.

public DateTime CreationDateTime {get;}
The date and time that this queue was created.

public int CurrentDepth {get;}
Gets the number of messages currently on the queue. This value is incremented during a put call, and during backout of a get call. It is decremented during a non-browse get and during backout of a put call.

public int DefinitionType {get;}
How the queue was defined. The possible values are:

- MQC.MQQDT_PREDEFINED
- MQC.MQQDT_PERMANENT_DYNAMIC
- MQC.MQQDT_TEMPORARY_DYNAMIC

public int InhibitGet {get; set;}
Controls whether you can get messages on this queue or for this topic. The possible values are:

- MQC.MQQA_GET_INHIBITED
- MQC.MQQA_GET_ALLOWED

public int InhibitPut {get; set;}
Controls whether you can put messages on this queue or for this topic. The possible values are:

- MQQA_PUT_INHIBITED
- MQQA_PUT_ALLOWED

public int MaximumDepth {get;}
The maximum number of messages that can exist on the queue at any one time. An attempt to put a message to a queue that already contains this many messages fails with reason code MQC.MQRC_Q_FULL.

public int MaximumMessageLength {get;}
The maximum length of the application data that can exist in each message on this queue. An attempt to put a message larger than this value fails with reason code MQC.MQRC_MSG_TOO_BIG_FOR_Q.

public int NonPersistentMessageClass {get;}
The level of reliability for non-persistent messages put to this queue.

public int OpenInputCount {get;}
The number of handles that are currently valid for removing messages from the queue. OpenInputCount is the total number of valid input handles known to the local queue manager, not just handles created by the application.

public int OpenOutputCount {get;}
The number of handles that are currently valid for adding messages to the queue. OpenOutputCount is the total number of valid output handles known to the local queue manager, not just handles created by the application.

public int QueueAccounting {get;}
Specifies whether you can enable the collection of accounting information for the queue.

public int QueueMonitoring {get;}
Specifies whether you can enable the monitoring for the queue.

public int QueueStatistics {get;}
Specifies whether you can enable the collection of statistics for the queue.

public int QueueType {get;}
The type of this queue with one of the following values:

- MQC.MQQT_ALIAS
- MQC.MQQT_LOCAL
- MQC.MQQT_REMOTE
- MQC.MQQT_CLUSTER

public int Shareability {get;}
Whether the queue can be opened for input multiple times. The possible values are:

- MQC.MQQA_SHAREABLE
- MQC.MQQA_NOT_SHAREABLE

public string TPIPE {get;}
The TPIPE name used for communication with OTMA using the IBM MQ IMS bridge.

public int TriggerControl {get; set;}
Whether trigger messages are written to an initiation queue, to start an application to service the queue. The possible values are:

- MQC.MQTC_OFF
- MQC.MQTC_ON

public string TriggerData {get; set;}
The free-format data that the queue manager inserts into the trigger message. It inserts

TriggerData when a message arriving on this queue causes a trigger message to be written to the initiation queue. The maximum permissible length of the string is given by MQC.MQ_TRIGGER_DATA_LENGTH.

public int TriggerDepth {get; set;}
The number of messages that must be on the queue before a trigger message is written when trigger type is set to MQC.MQTT_DEPTH.

public int TriggerMessagePriority {get; set;}
The message priority under which messages do not contribute to the generation of trigger messages. That is, the queue manager ignores these messages when deciding whether to generate a trigger. A value of zero causes all messages to contribute to the generation of trigger messages.

public int TriggerType {get; set;}
The conditions under which trigger messages are written as a result of messages arriving on this queue. The possible values are:

- MQC.MQTT_NONE
- MQC.MQTT_FIRST
- MQC.MQTT_EVERY
- MQC.MQTT_DEPTH

Methods

```
public void Get(MQMessage message);
public void Get(MQMessage message, MQGetMessageOptions getMessageOptions);
public void Get(MQMessage message, MQGetMessageOptions getMessageOptions, int MaxMsgSize);
```

Throws MQException.

Gets a message from a queue.

If the get fails, the MQMessage object is unchanged. If it succeeds, the message descriptor and message data portions of the MQMessage are replaced with the message descriptor and message data from the incoming message.

All calls to IBM MQ from a particular MQQueueManager are synchronous. Therefore, if you perform a get with wait, all other threads using the same MQQueueManager are blocked from making further IBM MQ calls until the Get call is accomplished. If you need multiple threads to access IBM MQ simultaneously, each thread must create its own MQQueueManager object.

message
Contains the message descriptor and the returned message data. Some of the fields in the message descriptor are input parameters. It is important to ensure that the MessageId and CorrelationId input parameters are set as required.

A reconnectable client returns the reason code MQRC_BACKED_OUT after successful reconnection, for messages received under MQGM_SYNCPOINT.

getMessageOptions
Options controlling the action of the get.

Using option MQC.MQGMO_CONVERT might result in an exception with reason code MQC.MQRC_CONVERTED_STRING_TOO_BIG when converting from single-byte character codes to double byte codes. In this case, the message is copied into the buffer without conversion.

If *getMessageOptions* is not specified, the message option used is MQGMO_NOWAIT.

If you use the MQGMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned.

MaxMsgSize
The largest message this message object is to receive. If the message on the queue is larger than this size, one of two things occurs:

- If the MQGMO_ACCEPT_TRUNCATED_MSG flag is set in the MQGetMessageOptions object, the message is filled with as much of the message data as possible. An exception is thrown with the MQCC_WARNING completion code and MQRC_TRUNCATED_MSG_ACCEPTED reason code.
- If the MQGMO_ACCEPT_TRUNCATED_MSG flag is not set, the message remains on the queue. An exception is thrown with the MQCC_WARNING completion code and MQRC_TRUNCATED_MSG_FAILED reason code.

If *MaxMsgSize* is not specified, the whole message is retrieved.

```
public void Put(MQMessage message);
public void Put(MQMessage message, MQPutMessageOptions putMessageOptions);
```

Throws MQException.

Puts a message to a queue.

Modifications to the MQMessage object after the Put call has been accomplished do not affect the actual message on the IBM MQ queue or publication topic.

Put updates the MessageId and CorrelationId properties of the MQMessage object and does not clear message data. Further Put or Get calls refer to the updated information in the MQMessage object. For example, in the following code snippet, the first message contains a and the second ab.

```
msg.WriteString("a");
q.Put(msg,pmo);
msg.WriteString("b");
q.Put(msg,pmo);
```

message
An MQMessage object containing the message descriptor data, and message to be sent. The message descriptor can be altered as a consequence of this method. The values in the message descriptor immediately after the completion of this method are the values that were put to the queue or published to the topic.

The following reason codes are returned to a reconnectable client:

- MQRC_CALL_INTERRUPTED if the connection is broken while running a Put call on a persistent message and the reconnection is successful.
- MQRC_NONE if the connection is successful while running a Put call on a non-persistent message (see Application Recovery).

putMessageOptions
Options controlling the action of the put.

If *putMessageOptions* is not specified the default instance of MQPutMessageOptons is used.

If you use the MQPMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned.

Note: For simplicity and performance, if you want to put a single message to a queue, use MQQueueManager.Put object. You should have an MQQueue object for this.

```
public void PutForwardMessage(MQMessage message);
public void PutForwardMessage(MQMessage message, MQPutMessageOptions putMessageOptions);
```

Throws MQException

Put a message being forwarded onto the queue, where *message* is the original message.

message
An MQMessage object containing the message descriptor data, and message to be sent. The

message descriptor can be altered as a consequence of this method. The values in the message descriptor immediately after the completion of this method are the values that were put to the queue or published to the topic.

The following reason codes are returned to a reconnectable client:

- MQRC_CALL_INTERRUPTED if the connection is broken while running a Put call on a persistent message and the reconnection is successful.
- MQRC_NONE if the connection is successful while running a Put call on a non-persistent message (see Application Recovery).

putMessageOptions
Options controlling the action of the put.

If *putMessageOptions* is not specified the default instance of MQPutMessageOptons is used.

If you use the MQPMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned.

```
public void PutReplyMessage(MQMessage message)
public void PutReplyMessage(MQMessage message, MQPutMessageOptions putMessageOptions)
```

Throws MQException.

Put a reply message onto the queue, where *message* is the original message.

message
Contains the message descriptor and the returned message data. Some of the fields in the message descriptor are input parameters. It is important to ensure that the MessageId and CorrelationId input parameters are set as required.

A reconnectable client returns the reason code MQRC_BACKED_OUT after successful reconnection, for messages received under MQGM_SYNCPOINT.

putMessageOptions
Options controlling the action of the put.

If *putMessageOptions* is not specified the default instance of MQPutMessageOptons is used.

If you use the MQPMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned.

```
public void PutReportMessage(MQMessage message)
public void PutReportMessage(MQMessage message, MQPutMessageOptions putMessageOptions)
```

Throws MQException.

Put a report message onto the queue, where *message* is the original message.

message
Contains the message descriptor and the returned message data. Some of the fields in the message descriptor are input parameters. It is important to ensure that the MessageId and CorrelationId input parameters are set as required.

A reconnectable client returns the reason code MQRC_BACKED_OUT after successful reconnection, for messages received under MQGM_SYNCPOINT.

putMessageOptions
Options controlling the action of the put.

If *putMessageOptions* is not specified the default instance of MQPutMessageOptons is used.

If you use the MQPMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned.

Constructors

public MQQueue MQQueueManager.AccessQueue(string *queueName*, **int** *openOptions*);
public MQQueue MQQueueManager.AccessQueue(string *queueName*, **int** *openOptions*, **string** *queueManagerName*, **string** *dynamicQueueName*, **string** *alternateUserId*);

Throws MQException.

Accesses a queue on this queue manager.

You can get or browse messages, put messages, inquire about the attributes of the queue or set the attributes of the queue. If the queue named is a model queue, a dynamic local queue is created. Query the `name` attribute of the resultant MQQueue object to find out the name of the dynamic queue.

queueName
Name of queue to open.

openOptions
Options that control the opening of the queue.

MQC.MQOO_ALTERNATE_USER_AUTHORITY
Validate with the specified user identifier.

MQC.MQOO_BIND_AS_QDEF
Use default binding for queue.

MQC.MQOO_BIND_NOT_FIXED
Do not bind to a specific destination.

MQC.MQOO_BIND_ON_OPEN
Bind handle to destination when queue is opened.

MQC.MQOO_BROWSE
Open to browse message.

MQC.MQOO_FAIL_IF_QUIESCING
Fail if the queue manager is quiescing.

MQC.MQOO_INPUT_AS_Q_DEF
Open to get messages using queue-defined default.

MQC.MQOO_INPUT_SHARED
Open to get messages with shared access.

MQC.MQOO_INPUT_EXCLUSIVE
Open to get messages with exclusive access.

MQC.MQOO_INQUIRE
Open for inquiry - required if you want to query properties.

MQC.MQOO_OUTPUT
Open to put messages.

MQC.MQOO_PASS_ALL_CONTEXT
Allow all context to be passed.

MQC.MQOO_PASS_IDENTITY_CONTEXT
Allow identity context to be passed.

MQC.MQOO_SAVE_ALL_CONTEXT
Save context when message retrieved.

MQC.MQOO_SET
Open to set attributes - required if you want to set properties.

MQC.MQOO_SET_ALL_CONTEXT
Allows all context to be set.

MQC.MQOO_SET_IDENTITY_CONTEXT
Allows identity context to be set.

queueManagerName
Name of the queue manager on which the queue is defined. A name that is entirely blank or null denotes the queue manager to which the MQQueueManager object is connected.

dynamicQueueName
dynamicQueueName is ignored unless queueName specifies the name of a model queue. If it does, *dynamicQueueName* specifies the name of the dynamic queue to be created. A blank or null name is not valid if queueName specifies the name of a model queue. If the last nonblank character in the name is an asterisk, *, the queue manager replaces the asterisk with a string of characters. The characters guarantee that the name generated for the queue is unique on this queue manager.

alternateUserId
If MQC.MQOO_ALTERNATE_USER_AUTHORITY is specified in the openOptions parameter, *alternateUserId* specifies the alternate user identifier that is used to check the authorization for the open. If MQC.MQOO_ALTERNATE_USER_AUTHORITY is not specified, *alternateUserId* can be left blank, or null.

public MQQueue(MQQueueManager *queueManager*, **string** *queueName*, **int** *openOptions*, **string** *queueManagerName*, **string** *dynamicQueueName*, **string** *alternateUserId*);

Throws MQException.

Accesses a queue on queueManager.

You can get or browse messages, put messages, inquire about the attributes of the queue or set the attributes of the queue. If the queue named is a model queue, a dynamic local queue is created. Query the name attribute of the resultant MQQueue object to find out the name of the dynamic queue.

queueManager
Queue manager to access queue on.

queueName
Name of queue to open.

openOptions
Options that control the opening of the queue.

MQC.MQOO_ALTERNATE_USER_AUTHORITY
Validate with the specified user identifier.

MQC.MQOO_BIND_AS_QDEF
Use default binding for queue.

MQC.MQOO_BIND_NOT_FIXED
Do not bind to a specific destination.

MQC.MQOO_BIND_ON_OPEN
Bind handle to destination when queue is opened.

MQC.MQOO_BROWSE
Open to browse message.

MQC.MQOO_FAIL_IF_QUIESCING
Fail if the queue manager is quiescing.

MQC.MQOO_INPUT_AS_Q_DEF
Open to get messages using queue-defined default.

MQC.MQOO_INPUT_SHARED
Open to get messages with shared access.

MQC.MQOO_INPUT_EXCLUSIVE
Open to get messages with exclusive access.

MQC.MQOO_INQUIRE
Open for inquiry - required if you want to query properties.

MQC.MQOO_OUTPUT
Open to put messages.

MQC.MQOO_PASS_ALL_CONTEXT
Allow all context to be passed.

MQC.MQOO_PASS_IDENTITY_CONTEXT
Allow identity context to be passed.

MQC.MQOO_SAVE_ALL_CONTEXT
Save context when message retrieved.

MQC.MQOO_SET
Open to set attributes - required if you want to set properties.

MQC.MQOO_SET_ALL_CONTEXT
Allows all context to be set.

MQC.MQOO_SET_IDENTITY_CONTEXT
Allows identity context to be set.

queueManagerName
Name of the queue manager on which the queue is defined. A name that is entirely blank or null denotes the queue manager to which the MQQueueManager object is connected.

dynamicQueueName
dynamicQueueName is ignored unless `queueName` specifies the name of a model queue. If it does, *dynamicQueueName* specifies the name of the dynamic queue to be created. A blank or null name is not valid if queueName specifies the name of a model queue. If the last nonblank character in the name is an asterisk, `*`, the queue manager replaces the asterisk with a string of characters. The characters guarantee that the name generated for the queue is unique on this queue manager.

alternateUserId
If `MQC.MQOO_ALTERNATE_USER_AUTHORITY` is specified in the `openOptions` parameter, *alternateUserId* specifies the alternate user identifier that is used to check the authorization for the open. If `MQC.MQOO_ALTERNATE_USER_AUTHORITY` is not specified, *alternateUserId* can be left blank, or null.

MQQueueManager.NET class

Use MQQueueManager to connect to a queue manager and access queue manager objects. It also controls transactions. The MQQueueManager constructor creates either a client or server connection.

Class

```
System.Object
        |
        ≪— IBM.WMQ.MQBase
                |
                ≪— IBM.WMQ.MQBaseObject
                        |
                        ≪— IBM.WMQ.ManagedObject
                                |
                                ≪— IBM.WMQ.MQQueueManager
```

```
public class IBM.WMQ.MQQueueManager extends IBM.WMQ.MQManagedObject;
```

- "Properties"
- "Methods" on page 3902
- "Constructors" on page 3908

Properties

Test for MQException being thrown when getting properties.

public int AccountingConnOverride {get;}
Whether applications can override the setting of the MQI accounting and queue accounting values.

public int AccountingInterval {get;}
How long before intermediate accounting records are written (in seconds).

public int ActivityRecording {get;}
Controls the generation of activity reports.

public int AdoptNewMCACheck {get;}
Specifies which elements are checked to determine whether the MCA is adopted when a new inbound channel is detected. To be adopted, the MCA name must match the name of an active MCA.

public int AdoptNewMCAInterval {get;}
The amount of time, in seconds, that the new channel waits for the orphaned channel to end.

public int AdoptNewMCAType {get;}
Whether an orphaned MCA instance is to be adopted (restarted) when a new inbound channel request is detected matching the AdoptNewMCACheck value.

public int BridgeEvent {get;}
Whether IMS bridge events are generated.

public int ChannelEvent {get;}
Whether channel events are generated.

public int ChannelInitiatorControl {get;}
Whether the channel initiator starts automatically when the queue manager starts.

public int ChannelInitiatorAdapters {get;}
The number of adapter subtasks to process IBM MQ calls.

public int ChannelInitiatorDispatchers {get;}
The number of dispatchers to use for the channel initiator.

public int ChannelInitiatorTraceAutoStart {get;}
Specifies whether the channel initiator trace starts automatically.

public int ChannelInitiatorTraceTableSize {get;}
The size, in megabytes, of the trace data space of a channel initiator.

public int ChannelMonitoring {get;}
Whether channel monitoring is used.

public int ChannelStatistics {get;}
Controls the collection of statistics data for channels.

public int CharacterSet {get;}
Returns the coded character set identifier (CCSID) of the queue manager. `CharacterSet` is used by the queue manager for all character string fields in the application programming interface.

public int ClusterSenderMonitoring {get;}
Controls the collection of online monitoring data for automatically defined cluster sender channels.

public int ClusterSenderStatistics {get;}
Controls the collection of statistics data for automatically defined cluster sender channels.

public int ClusterWorkLoadMRU {get;}
The maximum number of outbound cluster channels.

public int ClusterWorkLoadUseQ {get;}
The default value of the MQQueue property, `ClusterWorkLoadUseQ`, if it specifies a value of `QMGR`.

public int CommandEvent {get;}
Specifies whether command events are generated.

public string CommandInputQueueName {get;}
Returns the name of the command input queue defined on the queue manager. Applications can send commands to this queue, if authorized to do so.

public int CommandLevel {get;}
Indicates the function level of the queue manager. The set of functions that correspond to a particular function level depends on the platform. On a particular platform, you can rely on every queue manager supporting the functions at the lowest functional level common to all the queue managers.

public int CommandLevel {get;}
Whether the command server starts automatically when the queue manager starts.

public string DNSGroup {get;}
No longer used.

public int DNSWLM {get;}
No longer used.

public int IPAddressVersion {get;}
Which IP protocol (IPv4 or IPv6) to use for a channel connection.

public boolean IsConnected {get;}
Returns the value of the `isConnected`.

If true, a connection to the queue manager has been made, and is not known to be broken. Any calls to IsConnected do not actively attempt to reach the queue manager, so it is possible that physical connectivity can break, but IsConnected can still return true. The IsConnected state is only updated when activity, for example, putting a message, getting a message, is performed on the queue manager.

If false, a connection to the queue manager has not been made, or has been broken, or has been disconnected.

public int KeepAlive {get;}
Specifies whether the TCP KEEPALIVE facility is to be used to check that the other end of the connection is still available. If it is unavailable, the channel is closed.

public int ListenerTimer {get;}
The time interval, in seconds, between attempts by IBM MQ to restart the listener after an APPC or TCP/IP failure.

public int LoggerEvent {get;}
Whether logger events are generated.

public string LU62ARMSuffix {get;}
The suffix of the APPCPM member of SYS1.PARMLIB. This suffix nominates the LUADD for this channel initiator. When automatic restart manager (ARM) restarts the channel initiator, the z/OS command SET APPC=xx is issued.

public string LUGroupName {get; z/os}
The generic LU name to be used by the LU 6.2 listener that handles inbound transmissions for the queue-sharing group.

public string LUName {get;}
The name of the LU to use for outbound LU 6.2 transmissions.

public int MaximumActiveChannels {get;}
The maximum number of channels that can be active at any time.

public int MaximumCurrentChannels {get;}
The maximum number of channels that can be current at any time (including server-connection channels with connected clients).

public int MaximumLU62Channels {get;}
The maximum number of channels that can be current, or clients that can be connected, that use the LU 6.2 transmission protocol.

public int MaximumMessageLength {get;}
Returns the maximum length of a message (in bytes) that can be handled by the queue manager. No queue can be defined with a maximum message length greater than `MaximumMessageLength`.

public int MaximumPriority {get;}
Returns the maximum message priority supported by the queue manager. Priorities range from zero (lowest) to this value. Throws MQException if you call this method after disconnecting from the queue manager.

public int MaximumTCPChannels {get;}
The maximum number of channels that can be current, or clients that can be connected, that use the TCP/IP transmission protocol.

public int MQIAccounting {get;}
Controls the collection of accounting information for MQI data.

public int MQIStatistics {get;}
Controls the collection of statistics monitoring information for the queue manager.

public int OutboundPortMax {get;}
The maximum value in the range of port numbers to be used when binding outgoing channels.

public int OutboundPortMin {get;}
The minimum value in the range of port numbers to be used when binding outgoing channels.

public int QueueAccounting {get;}
Whether class 3 accounting (thread-level and queue-level accounting) data is to be used for all queues.

public int QueueMonitoring {get;}
Controls the collection of online monitoring data for queues.

public int QueueStatistics {get;}
Controls the collection of statistics data for queues.

public int ReceiveTimeout {get;}
The length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to the inactive state.

public int ReceiveTimeoutMin {get;}
The minimum length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to an inactive state.

public int ReceiveTimeoutType {get;}
The qualifier to apply to the value in `ReceiveTimeout`.

public int SharedQueueQueueManagerName {get;}
Specifies how to deliver messages to a shared queue. If the put specifies a different queue manager from the same queue-sharing group as the target queue manager, the message is delivered in two ways:

MQC.MQSQQM_USE
Messages are delivered to the object queue manager before being put on the shared queue.

MQCMQSQQM_IGNORE
Messages are put directly on the shared queue.

public int SSLEvent {get;}
Whether TLS events are generated.

public int SSLFips {get;}
Whether only FIPS-certified algorithms are to be used if cryptography is performed in IBM MQ, rather than cryptographic hardware.

public int SSLKeyResetCount {get;}
Indicates the number of unencrypted bytes sent and received within a TLS conversation before the secret key is renegotiated.

public int ClusterSenderStatistics {get;}
Specifies the interval, in minutes, between consecutive gatherings of statistics.

public int SyncpointAvailability {get;}
Indicates whether the queue manager supports units of work and sync points with the MQQueue.get and MQQueue.put methods.

public string TCPName {get;}
The name of either the only, or default, TCP/IP system to be used, depending on the value of TCPStackType.

public int TCPStackType {get;}
Specifies whether the channel initiator uses only the TCP/IP address space specified in `TCPName`. Alternatively, the channel initiator can bind to any TCP/IP address.

public int TraceRouteRecording {get;}
Controls the recording of route tracing information.

Methods

```
public MQProcess AccessProcess(string processName, int openOptions);
public MQProcess AccessProcess(string processName, int openOptions, string queueManagerName,
string alternateUserId);
```

Throws MQException.

Access an IBM MQ process on this queue manager to inquire on process attributes.

processName
The name of the process to open.

openOptions
Options that control the opening of the process. The valid options that can be added, or combined using a bitwise OR, are:

- MQC.MQOO_FAIL_IF_QUIESCING
- MQC.MQOO_INQUIRE
- MQC.MQOO_SET
- MQC.MQOO_ALTERNATE_USER_AUTHORITY

queueManagerName
The name of the queue manager on which the process is defined. You can leave a blank or null queue manager name if the queue manager is the same as the one the process is accessing.

alternateUserId

If MQC.MQOO_ALTERNATE_USER_AUTHORITY is specified in the **openOptions** parameter, *alternateUserId* specifies the alternative user ID used to check the authorization for the action. If MQOO_ALTERNATE_USER_AUTHORITY is not specified, *alternateUserId* can be blank or null.

Default user authority is used for connection to the queue manager if MQC.MQOO_ALTERNATE_USER_AUTHORITY is not specified.

```
public MQQueue AccessQueue(string queueName, int openOptions);
public MQQueue AccessQueue(string queueName, int openOptions, string queueManagerName, string
dynamicQueueName, string alternateUserId);
```

Throws MQException.

Accesses a queue on this queue manager.

You can get or browse messages, put messages, inquire about the attributes of the queue or set the attributes of the queue. If the queue named is a model queue, a dynamic local queue is created. Query the name attribute of the resultant MQQueue object to find out the name of the dynamic queue.

queueName
Name of queue to open.

openOptions
Options that control the opening of the queue.

MQC.MQOO_ALTERNATE_USER_AUTHORITY
Validate with the specified user identifier.

MQC.MQOO_BIND_AS_QDEF
Use default binding for queue.

MQC.MQOO_BIND_NOT_FIXED
Do not bind to a specific destination.

MQC.MQOO_BIND_ON_OPEN
Bind handle to destination when queue is opened.

MQC.MQOO_BROWSE
Open to browse message.

MQC.MQOO_FAIL_IF_QUIESCING
Fail if the queue manager is quiescing.

MQC.MQOO_INPUT_AS_Q_DEF
Open to get messages using queue-defined default.

MQC.MQOO_INPUT_SHARED
Open to get messages with shared access.

MQC.MQOO_INPUT_EXCLUSIVE
Open to get messages with exclusive access.

MQC.MQOO_INQUIRE
Open for inquiry - required if you want to query properties.

MQC.MQOO_OUTPUT
Open to put messages.

MQC.MQOO_PASS_ALL_CONTEXT
Allow all context to be passed.

MQC.MQOO_PASS_IDENTITY_CONTEXT
Allow identity context to be passed.

MQC.MQOO_SAVE_ALL_CONTEXT
Save context when message retrieved.

MQC.MQOO_SET
Open to set attributes - required if you want to set properties.

MQC.MQOO_SET_ALL_CONTEXT
Allows all context to be set.

MQC.MQOO_SET_IDENTITY_CONTEXT
Allows identity context to be set.

queueManagerName
Name of the queue manager on which the queue is defined. A name that is entirely blank or null denotes the queue manager to which the MQQueueManager object is connected.

dynamicQueueName
dynamicQueueName is ignored unless queueName specifies the name of a model queue. If it does, *dynamicQueueName* specifies the name of the dynamic queue to be created. A blank or null name is not valid if queueName specifies the name of a model queue. If the last nonblank character in the name is an asterisk, *, the queue manager replaces the asterisk with a string of characters. The characters guarantee that the name generated for the queue is unique on this queue manager.

alternateUserId
If MQC.MQOO_ALTERNATE_USER_AUTHORITY is specified in the openOptions parameter, *alternateUserId* specifies the alternate user identifier that is used to check the authorization for the open. If MQC.MQOO_ALTERNATE_USER_AUTHORITY is not specified, *alternateUserId* can be left blank, or null.

```
public MQTopic AccessTopic( MQDestination destination, string topicName, string topicObject, int
options);
public MQTopic AccessTopic( MQDestination destination, string topicName, string topicObject, int
options, string alternateUserId);
public MQTopic AccessTopic( MQDestination destination, string topicName, string topicObject, int
options, string alternateUserId, string subscriptionName);
public MQTopic AccessTopic( MQDestination destination, string topicName, string topicObject, int
options, string alternateUserId, string subscriptionName, System.Collections.Hashtable
properties);
```

```
public MQTopic AccessTopic(string topicName, string topicObject, int openAs, int options);
public MQTopic AccessTopic(string topicName, string topicObject, int openAs, int options, string
alternateUserId);
public MQTopic AccessTopic(string topicName, string topicObject, int options, string
alternateUserId, string subscriptionName);
public MQTopic AccessTopic(string topicName, string topicObject, int options, string
alternateUserId, string subscriptionName, System.Collections.Hashtable properties);
```

Access a topic on this queue manager.

MQTopic objects are closely related to administrative topic objects, which are sometimes called topic objects. On input, `topicObject` points to an administrative topic object. The MQTopic constructor obtains a topic string from the topic object and combines it with `topicName` to create a topic name. Either or both `topicObject` or `topicName` can be null. The topic name is matched to the topic tree, and the name of the closest matching administrative topic object is returned in `topicObject`.

The topics that are associated with the MQTopic object are the result of combining two topic strings. The first topic string is defined by the administrative topic object identified by *topicObject*. The second topic string is *topicString*. The resulting topic string associated with the MQTopic object can identify multiple topics by including wildcards.

Depending on whether the topic is opened for publishing or subscribing, you can use the MQTopic.Put methods to publish on topics, or MQTopic.Get methods to receive publications on topics. If you want to publish and subscribe to the same topic, you must access the topic twice, once for publish and once for subscribe.

If you create an MQTopic object for subscription, without providing an MQDestination object, a managed subscription is assumed. If you pass a queue as an MQDestination object, an unmanaged subscription is assumed. You must ensure the subscription options you set are consistent with the subscription being managed or unmanaged.

destination
destination is an MQQueue instance. By providing *destination*, MQTopic is opened as an unmanaged subscription. Publications on the topic are delivered to the queue accessed as *destination*.

topicName
A topic string that is the second part of the topic name. *topicName* is concatenated with the topic string defined in the *topicObject* administrative topic object. You can set *topicName* to null, in which case the topic name is defined by the topic string in *topicObject*.

topicObject
On input, *topicObject* is the name of the topic object that contains the topic string that forms the first part of the topic name. The topic string in *topicObject* is concatenated with *topicName*. The rules for constructing topic strings are defined in Combining topic strings.

On output, *topicObject* contains the name of the administrative topic object that is the closest match in the topic tree to the topic identified by the topic string.

openAs
Access the topic to publish or subscribe. The parameter can contain only one of these options:

- `MQC.MQTOPIC_OPEN_AS_SUBSCRIPTION`
- `MQC.MQTOPIC_OPEN_AS_PUBLICATION`

options

Combine the options that control the opening of the topic for either publication or subscription. Use `MQC.MQSO_*` constants to access a topic for subscription and `MQC.MQOO_*` constants to access a topic for publication.

If more than one option is required, add the values together, or combine the option values using the bitwise OR operator.

alternateUserId
Specify the alternate user ID that is used to check for the required authorization to finish the operation. You must specify *alternateUserId*, if either MQC.MQOO_ALTERNATE_USER_AUTHORITY or MQC.MQSO_ALTERNATE_USER_AUTHORITY is set in the options parameter.

subscriptionName
subscriptionName is required if the options MQC.MQSO_DURABLE or MQC.MQSO_ALTER are provided. In both cases, MQTopic is implicitly opened for subscription. An exception is thrown if the MQC.MQSO_DURABLE is set, and the subscription exists, or if MQC.MQSO_ALTER is set, and the subscription does not exist.

properties
Set any of the special subscription properties listed using a hash table. Specified entries in the hash table are updated with output values. Entries are not added to the hash table to report output values.

- MQC.MQSUB_PROP_ALTERNATE_SECURITY_ID
- MQC.MQSUB_PROP_SUBSCRIPTION_EXPIRY
- MQC.MQSUB_PROP_SUBSCRIPTION_USER_DATA
- MQC.MQSUB_PROP_SUBSCRIPTION_CORRELATION_ID
- MQC.MQSUB_PROP_PUBLICATION_PRIORITY
- MQC.MQSUB_PROP_PUBLICATION_ACCOUNTING_TOKEN
- MQC.MQSUB_PROP_PUBLICATION_APPLICATIONID_DATA

public MQAsyncStatus GetAsyncStatus();

Throws MQException

Returns an MQAsyncStatus object, which represents the asynchronous activity for the queue manager connection.

public void Backout();

Throws MQException.

Backout any messages that were read or written within sync point since the last sync point.

Messages that were written with the MQC.MQPMO_SYNCPOINT flag set are removed from queues. Messages read with the MQC.MQGMO_SYNCPOINT flag are reinstated on the queues they came from. If the messages are persistent, the changes are logged.

For reconnectable clients, the MQRC_NONE reason code is returned to a client after reconnection is successful.

public void Begin();

Throws MQException.

Begin is supported only in server bindings mode. It starts a global unit of work.

public void Commit();

Throws MQException.

Commit any messages that were read or written within sync point since the last sync point.

Messages written with the MQC.MQPMO_SYNCPOINT flag set are made available to other applications. Messages retrieved with the MQC.MQGMO_SYNCPOINT flag set are deleted. If the messages are persistent, the changes are logged.

The following reason codes are returned to a reconnectable client:

- MQRC_CALL_INTERRUPTED if connection is lost while carrying out the commit call.
- MQRC_BACKED_OUT if the commit call is issued after reconnection.

```
Disconnect();
```

Throws MQException.

Close the connection to the queue manager. All objects accessed on this queue manager are not longer accessible to this application. To reaccess the objects, create a MQQueueManager object.

Generally, any work performed as part of a unit of work is committed. However, if the unit of work is managed by .NET, the unit of work might be rolled back.

```
public void Put(int type, string destinationName, MQMessage message);
public void Put(int type, string destinationName, MQMessage message MQPutMessageOptions
putMessageOptions);
public void Put(int type, string destinationName, string queueManagerName, string topicString,
MQMessage message);
public void Put(string queueName, MQMessage message);
public void Put(string queueName, MQMessage message, MQPutMessageOptions putMessageOptions);
public void Put(string queueName, string queueManagerName, MQMessage message);
public void Put(string queueName, string queueManagerName, MQMessage message,
MQPutMessageOptions putMessageOptions);
public void Put(string queueName, string queueManagerName, MQMessage message,
MQPutMessageOptions putMessageOptions, string alternateUserId);
```

Throws MQException.

Places a single message onto a queue or topic without creating an MQQueue or MQTopic object first.

queueName
The name of the queue onto which to place the message.

destinationName
The name of a destination object. It is either a queue or a topic depending on the value of *type*.

type
The type of destination object. You must not combine the options.

MQC.MQOT_Q
Queue

MQC.MQOT_TOPIC
Topic

queueManagerName

The name of the queue manager or queue manager alias, on which the queue is defined. If type MQC.MQOT_TOPIC is specified this parameter is ignored.

If the queue is a model queue, and the resolved queue manager name is not this queue manager, an MQException is thrown.

topicString

topicString is combined with the topic name in the *destinationName* topic object.

topicString is ignored if *destinationName* is a queue.

message
The message to send. Message is an input/output object.

The following reason codes are returned to a reconnectable client:

- MQRC_CALL_INTERRUPTED if the connection is broken while performing a Put call on a persistent message.
- MQRC_NONE if the connection is successful while performing a Put call on a non-persistent message (see Application Recovery).

putMessageOptions

Options controlling the actions of the put.

If you omit *putMessageOptions*, a default instance of *putMessageOptions* is created. *putMessageOptions* is an input/output object.

If you use the MQPMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned.

alternateUserId

Specifies an alternate user identifier used to check authorization when placing the message on a queue.

You can omit *alternateUserId* if you do not set MQC.MQOO_ALTERNATE_USER_AUTHORITY in *putMessageOptions*. If you set MQC.MQOO_ALTERNATE_USER_AUTHORITY, you must also set *alternateUserId*. *alternateUserId* has not effect unless you also set MQC.MQOO_ALTERNATE_USER_AUTHORITY.

Constructors

```
public MQQueueManager();
public MQQueueManager(string queueManagerName);
public MQQueueManager(string queueManagerName, Int options);
public MQQueueManager(string queueManagerName, Int options, string channel, string connName);
public MQQueueManager(string queueManagerName, string channel, string connName);
public MQQueueManager(string queueManagerName, System.Collections.Hashtable properties);
```

Throws MQException.

Creates a connection to a queue manager. Select between creating a client connection or a server connection.

You must have inquire (inq) authority on the queue manager when attempting to connect to the queue manager. Without inquire authority, the connection attempt fails.

A client connection is created if one of the following conditions is true:

1. *channel* or *connName* are specified in the constructor.
2. *HostName*, *Port*, or *Channel* are specified in *properties*.
3. *MQEnvironment.HostName*, *MQEnvironment.Port*, or *MQEnvironment.Channel* are specified.

The values of the connection properties are defaulted in the order shown. The *channel* and *connName* in the constructor take precedence over the property values in the constructor. The constructor property values take precedence of the MQEnvironment properties.

The host name, channel name, and port are defined in the MQEnvironment class.

queueManagerName

Name of the queue manager, or queue manager group to connect to.

Omit the parameter, or leave it null, or blank to make a default queue manager selection. The default queue manager connection on a server is to the default queue manager on the server. The default queue manager connection on a client connection is to the queue manager the listener is connected to.

options

Specify MQCNO connection options. The values must be applicable to the type of connection being made. For example, if you specify the following server connection properties for a client connection an MQException is thrown.

- MQC.MQCNO_FASTPATH_BINDING
- MQC.MQCNO_STANDARD_BINDING

properties

The properties parameter takes a series of key/value pairs that override the properties set by MQEnvironment ; see the example, "Override MQEnvironment properties" on page 3911. The following properties can be overridden:

- MQC.CONNECT_OPTIONS_PROPERTY
- MQC.CONNNAME_PROPERTY
- MQC.ENCRYPTION_POLICY_SUITE_B
- MQC.HOST_NAME_PROPERTY
- MQC.PORT_PROPERTY
- MQC.CHANNEL_PROPERTY
- MQC.SSL_CIPHER_SPEC_PROPERTY
- MQC.SSL_PEER_NAME_PROPERTY
- MQC.SSL_CERT_STORE_PROPERTY
- MQC.SSL_CRYPTO_HARDWARE_PROPERTY
- MQC.SECURITY_EXIT_PROPERTY
- MQC.SECURITY_USERDATA_PROPERTY
- MQC.SEND_EXIT_PROPERTY
- MQC.SEND_USERDATA_PROPERTY
- MQC.RECEIVE_EXIT_PROPERTY
- MQC.RECEIVE_USERDATA_PROPERTY
- MQC.USER_ID_PROPERTY
- MQC.PASSWORD_PROPERTY
- MQC.MQAIR_ARRAY
- MQC.KEY_RESET_COUNT
- MQC.FIPS_REQUIRED
- MQC.HDR_CMP_LIST
- MQC.MSG_CMP_LIST
- MQC.TRANSPORT_PROPERTY

channel

Name of a server connection channel

connName

Connection name in the format *HostName* (*Port*.

You can supply a list of *hostnames* and *ports* as an argument to the constructor MQQueueManager (String queueManagerName, Hashtable properties) using CONNECTION_NAME_PROPERTY.
For example:

```
ConnectionName = "fred.mq.com(2344),nick.mq.com(3746),tom.mq.com(4288)";
Hashtable Properties-new Hashtable();
properties.Add(MQC.CONNECTION_NAME_PROPERTY,ConnectionName);
MQQueueManager qmgr=new MQQueue Manager("qmgrname",properties);
```

When a connection attempt is made, the connection name list is processed in order. If the

connection attempt to the first host name and port fails, then connection to the second pair of attributes is attempted. The client repeats this process until either a successful connection is made or the list is exhausted. If the list is exhausted, an appropriate reason code and completion code is returned to the client application.
When a port number is not provided for the connection name, the default port (configured in `mqclient.ini`) is used.

Set the Connection List

You can set the connection list by using the following methods when the automatic client reconnection options are set:

Set the connection list through MQSERVER
You can set the connection list through the command prompt.

At the command prompt, set the following command:

```
MQSERVER=SYSTEM.DEF.SVRCONN/TCP/Hostname1(Port1),Hostname2(Por2),Hostname3(Port3)
```

For example:

```
MQSERVER=SYSTEM.DEF.SVRCONN/TCP/fred.mq.com(5266),nick.mq.com(6566),jack.mq.com(8413)
```

If you set the connection in the MQSERVER, do not set it in the application.

If you set the connection list in the application, the application overwrites whatever is set in the MQSERVER environment variable.

Set the connection list through the application
You can set the connection list in the application by specifying the host name and port properties.

```
String connName = "fred.mq.com(2344), nick.mq.com(3746), chris.mq.com(4288)";
MQQueueManager qm = new MQQueueManager("QM1", "TestChannel", connName);
```

Set the connection list through `app.config`
`App.config` is an XML file where you specify the key-value pairs.

In the connection list specify

```
<app.Settings>
<add key="Connection1" value="Hostname1(Port1)"/>
<add key="Connection2" value="Hostname2(Port2)"/>
<app.Settings>
```

For example:

```
<app.Settings>
<add key>="Connection1" value="fred.mq.com(2966)"/>
<add key>="Connection2" value="alex.mq.com(6533)"/>
<app.Settings>
```

You can directly change the connection list in the `app.config` file.

Set the connection list through MQEnvironment
To set the Connection list through the MQEnvironment, use the *ConnectionName* property.

```
MQEnvironment.ConnectionName = "fred.mq.com(4288),"alex.mq.com(5211);
```

The *ConnectionName* property overwrites the host name and port properties set in the MQEnvironment.

Create a client connection

The following example shows you how to create a client connection to a queue manager. You can create a client connection by setting the MQEnvironment variables before creating a new MQQueueManager Object.

```
MQEnvironment.Hostname = "fred.mq.com";  // host to connect to
MQEnvironment.Port     = 1414;           // port to connect to
                                         //If not explicitly set,
                                         // defaults to 1414
                                         // (the default IBM MQ port)
MQEnvironment.Channel  = "channel.name"; // the case sensitive
                                         //  name of the
                                         // SVR CONN channel on
                                         // the queue manager
MQQueueManager qMgr    = new MQQueueManager("MYQM");
```

Figure 95. Client connection

Override MQEnvironment properties

The following example shows you how to create a queue manager with its user ID and password defined in a hash table.

```
Hashtable properties = new Hashtable();

properties.Add( MQC.USER_ID_PROPERTY, "ExampleUserId" );
properties.Add( MQC.PASSWORD_PROPERTY, "ExamplePassword" );

try
{
    MQQueueManager qMgr = new MQQueueManager("qmgrname", properties);
}
catch (MQException mqe)
{
    System.Console.WriteLine("Connect failed with " + mqe.Message);
    return((int)mqe.Reason);
}
```

Figure 96. Overriding MQEnvironment properties

Create a reconnectable connection

The following example shows you how to automatically reconnect a client to a Queue Manager.

```
Hashtable properties = new Hashtable(); // The queue manager name and the
                                // properties how it has to be connected

properties.Add(MQC.CONNECT_OPTIONS_PROPERTY, MQC.MQCNO_RECONNECT); // Options
                                // through which reconnection happens

properties.Add(MQC.CONNECTION_NAME_PROPERTY,"fred.mq.com(4789),nick.mq.com(4790)"); // The list
                                // of queue managers through which reconnection happens

MQ QueueManager qmgr = new MQQueueManager("qmgrname", properties);
```

Figure 97. Automatically reconnecting a client to a queue manager

MQSubscription.NET class

Use MQSubscription to request that retained publications are sent to the subscriber. MQSubscription is a property of an MQTopic object opened for subscription.

Class

```
System.Object
        |
        ≪— IBM.WMQ.MQBase
                |
                ≪— IBM.WMQ.MQBaseObject
                        |
                        ≪— IBM.WMQ.MQManagedObject
                                |
                                ≪— IBM.WMQ.MQSubscription
```

```
public class IBM.WMQ.MQSubscription extends IBM.WMQ.MQManagedObject;
```

- "Properties"
- "Methods"
- "Constructors"

Properties

Access subscription properties using the MQManagedObject class; see "Properties" on page 3871.

Methods

Access subscription Inquire, Set and Get methods using the MQManagedObject class; see "Methods" on page 3872.

public int RequestPublicationUpdate(int *options***);**

Throws MQException.

Request an updated publication for the current topic. If the queue manager has a retained publications for the topic, they are sent to the subscriber.

Before calling RequestPublicationUpdate, open a topic for subscription to obtain an MQSubscription object.

Typically, open the subscription with the MQC.MQSO_PUBLICATIONS_ON_REQUEST option. If no wildcards are present in the topic string, then only one publication is sent as a result of this call. If the topic string contains wildcards, many publications might be sent. The method returns the number of retained publications that are sent to the subscription queue. There is no guarantee that this many publications are received, especially if they are non-persistent messages.

options

MQC.MQSRO_FAIL_IF_QUIESCING
The method fails if the queue manager is in a quiescent state. On z/OS, for a CICS or IMS application, MQC.MQSRO_FAIL_IF_QUIESCING also forces the method to fail if the connection is in a quiescent state.

MQC.MQSRO_NONE
No options are specified.

Constructors

No Public constructor.

An MQSubscription object is returned in the SubscriptionReference property of an MQTopic object that is opened for subscription,

Call the RequestPublicationUpdate method. MQSubscription is a subclass of MQManagedObject. Use the reference to access the properties and methods of MQManagedObject.

MQTopic.NET class

Use MQTopic to publish or subscribe messages on a topic, or to query or set attributes of a topic. Create an MQTopic object for publishing or subscribing by using a constructor or the MQQueueManager.AccessTopic method.

Class

```
System.Object
        |
        ≪— IBM.WMQ.MQBase
                |
                ≪— IBM.WMQ.MQBaseObject
                        |
                        ≪— IBM.WMQ.MQManagedObject
                                |
                                ≪— IBM.WMQ.MQDestination
                                        |
                                        ≪— IBM.WMQ.MQTopic
```

```
public class IBM.WMQ.MQTopic extends IBM.WMQ.MQDestination;
```


- "Properties"
- "Methods" on page 3914
- "Constructors" on page 3915

Properties

Test for MQException being thrown when getting properties.

public Boolean IsDurable {get;}
Read only property that returns True if the subscription is durable or False otherwise. If the topic was opened for publication, the property is ignored and would always return False.

public Boolean IsManaged {get;};
Read only property that returns True if the subscription is managed by the queue manager, or False otherwise. If the topic was opened for publication, the property is ignored and would always return False.

public Boolean IsSubscribed {get;};
Read only property that returns True if the topic was opened for subscription and False if the topic was opened for publication.

public MQSubscription SubscriptionReference {get;};
Read only property that returns the MQSubscription object associated with a topic object opened for subscription. The reference is available if you want to modify the close options or start any of the objects methods.

public MQDestination UnmanagedDestinationReference {get;};
Read only property that returns the MQQueue associated with an unmanaged subscription. It is the destination specified when the topic object was created. The property returns null for any topic objects opened for publication or with a managed subscription.

Methods

public void Put(MQMessage *message*);
public void Put(MQMessage *message*, **MQPutMessageOptions** *putMessageOptions*);
Throws MQException.

Publishes a message to the topic.

Modifications to the MQMessage object after the Put call has been accomplished do not affect the actual message on the IBM MQ queue or publication topic.

Put updates the `MessageId` and `CorrelationId` properties of the MQMessage object and does not clear message data. Further Put or Get calls refer to the updated information in the MQMessage object. For example, in the following code snippet, the first message contains `a` and the second `ab`.

```
msg.WriteString("a");
q.Put(msg,pmo);
msg.WriteString("b");
q.Put(msg,pmo);
```

message
An MQMessage object containing the message descriptor data, and message to be sent. The message descriptor can be altered as a consequence of this method. The values in the message descriptor immediately after the completion of this method are the values that were put to the queue or published to the topic.

The following reason codes are returned to a reconnectable client:

- `MQRC_CALL_INTERRUPTED` if the connection is broken while running a Put call on a persistent message and the reconnection is successful.
- `MQRC_NONE` if the connection is successful while running a Put call on a non-persistent message (see Application Recovery).

putMessageOptions
Options controlling the action of the put.

If *putMessageOptions* is not specified the default instance of MQPutMessageOptons is used.

If you use the `MQPMO_LOGICAL_ORDER` option in a reconnectable client, the `MQRC_RECONNECT_INCOMPATIBLE` reason code is returned.

Note: For simplicity and performance, if you want to put a single message to a queue, use MQQueueManager.Put object. You should have an MQQueue object for this.

public void Get(MQMessage *message*);
public void Get(MQMessage *message*, **MQGetMessageOptions** *getMessageOptions*);
public void Get(MQMessage *message*, **MQGetMessageOptions** *getMessageOptions*, **int** *MaxMsgSize*);
Throws MQException.

Retrieves a message from the topic.

This method uses a default instance of MQGetMessageOptions to do the get. The message option used is `MQGMO_NOWAIT`.

If the get fails, the MQMessage object is unchanged. If it succeeds, the message descriptor and message data portions of the MQMessage are replaced with the message descriptor and message data from the incoming message.

All calls to IBM MQ from a particular MQQueueManager are synchronous. Therefore, if you perform a get with wait, all other threads using the same MQQueueManager are blocked from making further IBM MQ calls until the Get call is accomplished. If you need multiple threads to access IBM MQ simultaneously, each thread must create its own MQQueueManager object.

message
Contains the message descriptor and the returned message data. Some of the fields in the

message descriptor are input parameters. It is important to ensure that the MessageId and CorrelationId input parameters are set as required.

A reconnectable client returns the reason code MQRC_BACKED_OUT after successful reconnection, for messages received under MQGM_SYNCPOINT.

getMessageOptions
Options controlling the action of the get.

Using option MQC.MQGMO_CONVERT might result in an exception with reason code MQC.MQRC_CONVERTED_STRING_TOO_BIG when converting from single-byte character codes to double byte codes. In this case, the message is copied into the buffer without conversion.

If *getMessageOptions* is not specified, the message option used is MQGMO_NOWAIT.

If you use the MQGMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned.

MaxMsgSize
The largest message this message object is to receive. If the message on the queue is larger than this size, one of two things occurs:

- If the MQGMO_ACCEPT_TRUNCATED_MSG flag is set in the MQGetMessageOptions object, the message is filled with as much of the message data as possible. An exception is thrown with the MQCC_WARNING completion code and MQRC_TRUNCATED_MSG_ACCEPTED reason code.
- If the MQGMO_ACCEPT_TRUNCATED_MSG flag is not set, the message remains on the queue. An exception is thrown with the MQCC_WARNING completion code and MQRC_TRUNCATED_MSG_FAILED reason code.

If *MaxMsgSize* is not specified, the whole message is retrieved.

Constructors

```
public MQTopic(MQQueueManager queueManager, MQDestination destination, string topicName, string
topicObject, int options);
public MQTopic(MQQueueManager queueManager, MQDestination destination, string topicName, string
topicObject, int options, string alternateUserId);
public MQTopic(MQQueueManager queueManager, MQDestination destination, string topicName, string
topicObject, int options, string alternateUserId, string subscriptionName);
public MQTopic(MQQueueManager queueManager, MQDestination destination, string topicName, string
topicObject, int options, string alternateUserId, string subscriptionName,
System.Collections.Hashtable properties);
public MQTopic(MQQueueManager queueManager, string topicName, string topicObject, int openAs,
int options);
public MQTopic(MQQueueManager queueManager, string topicName, string topicObject, int openAs,
int options, string alternateUserId);
public MQTopic(MQQueueManager queueManager, string topicName, string topicObject, int options,
string alternateUserId, string subscriptionName);
public MQTopic(MQQueueManager queueManager, string topicName, string topicObject, int options,
string alternateUserId, string subscriptionName, System.Collections.Hashtable properties);
```

Access a topic on *queueManager*.

MQTopic objects are closely related to administrative topic objects, which are sometimes called topic objects. On input, topicObject points to an administrative topic object. The MQTopic constructor obtains a topic string from the topic object and combines it with topicName to create a topic name. Either or both topicObject or topicName can be null. The topic name is matched to the topic tree, and the name of the closest matching administrative topic object is returned in topicObject.

The topics that are associated with the MQTopic object are the result of combining two topic strings. The first topic string is defined by the administrative topic object identified by

topicObject. The second topic string is *topicString*. The resulting topic string associated with the MQTopic object can identify multiple topics by including wildcards.

Depending on whether the topic is opened for publishing or subscribing, you can use the MQTopic.Put methods to publish on topics, or MQTopic.Get methods to receive publications on topics. If you want to publish and subscribe to the same topic, you must access the topic twice, once for publish and once for subscribe.

If you create an MQTopic object for subscription, without providing an MQDestination object, a managed subscription is assumed. If you pass a queue as an MQDestination object, an unmanaged subscription is assumed. You must ensure the subscription options you set are consistent with the subscription being managed or unmanaged.

queueManager
Queue manager to access a topic on.

destination
destination is an MQQueue instance. By providing *destination*, MQTopic is opened as an unmanaged subscription. Publications on the topic are delivered to the queue accessed as *destination*.

topicName
A topic string that is the second part of the topic name. *topicName* is concatenated with the topic string defined in the *topicObject* administrative topic object. You can set *topicName* to null, in which case the topic name is defined by the topic string in *topicObject*.

topicObject
On input, *topicObject* is the name of the topic object that contains the topic string that forms the first part of the topic name. The topic string in *topicObject* is concatenated with *topicName*. The rules for constructing topic strings are defined in Combining topic strings.

On output, *topicObject* contains the name of the administrative topic object that is the closest match in the topic tree to the topic identified by the topic string.

openAs
Access the topic to publish or subscribe. The parameter can contain only one of these options:

- MQC.MQTOPIC_OPEN_AS_SUBSCRIPTION
- MQC.MQTOPIC_OPEN_AS_PUBLICATION

options

Combine the options that control the opening of the topic for either publication or subscription. Use MQC.MQSO_* constants to access a topic for subscription and MQC.MQOO_* constants to access a topic for publication.

If more than one option is required, add the values together, or combine the option values using the bitwise OR operator.

alternateUserId
Specify the alternate user ID that is used to check for the required authorization to finish the operation. You must specify *alternateUserId*, if either MQC.MQOO_ALTERNATE_USER_AUTHORITY or MQC.MQSO_ALTERNATE_USER_AUTHORITY is set in the options parameter.

subscriptionName
subscriptionName is required if the options MQC.MQSO_DURABLE or MQC.MQSO_ALTER are provided. In both cases, MQTopic is implicitly opened for subscription. An exception is thrown if the MQC.MQSO_DURABLE is set, and the subscription exists, or if MQC.MQSO_ALTER is set, and the subscription does not exist.

properties
Set any of the special subscription properties listed using a hash table. Specified entries in the hash table are updated with output values. Entries are not added to the hash table to report output values.

- MQC.MQSUB_PROP_ALTERNATE_SECURITY_ID
- MQC.MQSUB_PROP_SUBSCRIPTION_EXPIRY
- MQC.MQSUB_PROP_SUBSCRIPTION_USER_DATA
- MQC.MQSUB_PROP_SUBSCRIPTION_CORRELATION_ID
- MQC.MQSUB_PROP_PUBLICATION_PRIORITY
- MQC.MQSUB_PROP_PUBLICATION_ACCOUNTING_TOKEN
- MQC.MQSUB_PROP_PUBLICATION_APPLICATIONID_DATA

```
public MQTopic MQQueueManager.AccessTopic(MQDestination destination, string topicName, string
topicObject, int options);
public MQTopic MQQueueManager.AccessTopic(MQDestination destination, string topicName, string
topicObject, int options, string alternateUserId);
public MQTopic MQQueueManager.AccessTopic(MQDestination destination, string topicName, string
topicObject, int options, string alternateUserId, string subscriptionName);
public MQTopic MQQueueManager.AccessTopic(MQDestination destination, string topicName, string
topicObject, int options, string alternateUserId, string subscriptionName,
System.Collections.Hashtable properties);
public MQTopic MQQueueManager.AccessTopic(string topicName, string topicObject, int openAs, int
options);
public MQTopic MQQueueManager.AccessTopic(string topicName, string topicObject, int openAs, int
options, string alternateUserId);
public MQTopic MQQueueManager.AccessTopic(string topicName, string topicObject, int options,
string alternateUserId, string subscriptionName);
public MQTopic MQQueueManager.AccessTopic(string topicName, string topicObject, int options,
string alternateUserId, string subscriptionName, System.Collections.Hashtable properties);
```

Access a topic on this queue manager.

MQTopic objects are closely related to administrative topic objects, which are sometimes called topic objects. On input, `topicObject` points to an administrative topic object. The MQTopic constructor obtains a topic string from the topic object and combines it with `topicName` to create a topic name. Either or both `topicObject` or `topicName` can be null. The topic name is matched to the topic tree, and the name of the closest matching administrative topic object is returned in `topicObject`.

The topics that are associated with the MQTopic object are the result of combining two topic strings. The first topic string is defined by the administrative topic object identified by *topicObject*. The second topic string is *topicString*. The resulting topic string associated with the MQTopic object can identify multiple topics by including wildcards.

Depending on whether the topic is opened for publishing or subscribing, you can use the MQTopic.Put methods to publish on topics, or MQTopic.Get methods to receive publications on topics. If you want to publish and subscribe to the same topic, you must access the topic twice, once for publish and once for subscribe.

If you create an MQTopic object for subscription, without providing an MQDestination object, a managed subscription is assumed. If you pass a queue as an MQDestination object, an unmanaged subscription is assumed. You must ensure the subscription options you set are consistent with the subscription being managed or unmanaged.

destination
destination is an MQQueue instance. By providing *destination*, MQTopic is opened as an unmanaged subscription. Publications on the topic are delivered to the queue accessed as *destination*.

topicName
A topic string that is the second part of the topic name. *topicName* is concatenated with the topic string defined in the *topicObject* administrative topic object. You can set *topicName* to null, in which case the topic name is defined by the topic string in *topicObject*.

topicObject
On input, *topicObject* is the name of the topic object that contains the topic string that forms the first part of the topic name. The topic string in *topicObject* is concatenated with *topicName*. The rules for constructing topic strings are defined in Combining topic strings.

On output, *topicObject* contains the name of the administrative topic object that is the closest match in the topic tree to the topic identified by the topic string.

openAs
Access the topic to publish or subscribe. The parameter can contain only one of these options:

- MQC.MQTOPIC_OPEN_AS_SUBSCRIPTION
- MQC.MQTOPIC_OPEN_AS_PUBLICATION

options

Combine the options that control the opening of the topic for either publication or subscription. Use MQC.MQSO_* constants to access a topic for subscription and MQC.MQOO_* constants to access a topic for publication.

If more than one option is required, add the values together, or combine the option values using the bitwise OR operator.

alternateUserId
Specify the alternate user ID that is used to check for the required authorization to finish the operation. You must specify *alternateUserId*, if either MQC.MQOO_ALTERNATE_USER_AUTHORITY or MQC.MQSO_ALTERNATE_USER_AUTHORITY is set in the options parameter.

subscriptionName
subscriptionName is required if the options MQC.MQSO_DURABLE or MQC.MQSO_ALTER are provided. In both cases, MQTopic is implicitly opened for subscription. An exception is thrown if the MQC.MQSO_DURABLE is set, and the subscription exists, or if MQC.MQSO_ALTER is set, and the subscription does not exist.

properties
Set any of the special subscription properties listed using a hash table. Specified entries in the hash table are updated with output values. Entries are not added to the hash table to report output values.

- MQC.MQSUB_PROP_ALTERNATE_SECURITY_ID
- MQC.MQSUB_PROP_SUBSCRIPTION_EXPIRY
- MQC.MQSUB_PROP_SUBSCRIPTION_USER_DATA
- MQC.MQSUB_PROP_SUBSCRIPTION_CORRELATION_ID
- MQC.MQSUB_PROP_PUBLICATION_PRIORITY
- MQC.MQSUB_PROP_PUBLICATION_ACCOUNTING_TOKEN
- MQC.MQSUB_PROP_PUBLICATION_APPLICATIONID_DATA

IMQObjectTrigger.NET interface

Implement IMQObjectTrigger to process messages passed by the **runmqdnm**.NET monitor.

Interface

```
public interface IBM.WMQMonitor.IMQObjectTrigger();
```

Depending on whether sync point control is specified in the **runmqdnm** command the message is removed from the queue before or after the Execute method returns.

Methods

void Execute (MQQueueManager *queueManager*, **MQQueue** *queue*, **MQMessage** *message*, **string** *param*);

queueManager
Queue manager hosting the queue being monitored.

queue
Queue being monitored.

message
Message read from the queue.

param
Data passed from `UserParameter`.

MQC.NET interface

Refer to an `MQI` constant by prefixing the constant name with `MQC.`. MQC defines all the constants used by the `MQI`.

Interface

```
System.Object
        |
        ≪— IBM.WMQ.MQC

public interface IBM.WMQ.MQC extends System.Object;
```

Example

```
MQQueue queue;
queue.closeOptions = MQC.MQCO_DELETE;
```

Character set identifiers for .NET applications

Descriptions of the character sets you can select to encode .NET IBM MQ messages

Character set	Description
37	ibm037
437	ibm437 / PC Original
500	ibm500
819	iso-8859-1 / latin1 / ibm819
1200	Unicode
1208	UTF-8
273	ibm273
277	ibm277
278	ibm278
280	ibm280

Character set	Description
284	ibm284
285	ibm285
297	ibm297
420	ibm420
424	ibm424
737	ibm737 / PC Greek
775	ibm775 / PC Baltic
813	iso-8859-7 / greek / ibm813
838	ibm838
850	ibm850 / PC Latin 1
852	ibm852 / PC Latin 2
855	ibm855 / PC Cyrillic
856	ibm856
857	ibm857 / PC Turkish
860	ibm860 / PC Portuguese
861	ibm861 / PC Icelandic
862	ibm862 / PC Hebrew
863	ibm863 / PC Canadian French
864	ibm864 / PC Arabic
865	ibm865 / PC Nordic
866	ibm866 / PC Russian
868	ibm868
869	ibm869 / PC Modern Greek
870	ibm870
871	ibm871
874	ibm874
875	ibm875
912	iso-8859-2 / latin2 / ibm912
913	iso-8859-3 / latin3 / ibm913
914	iso-8859-4 / latin4 / ibm914
915	iso-8859-5 / cyrillic / ibm915
916	iso-8859-8 / hebrew / ibm916
918	ibm918
920	iso-8859-9 / latin5 / ibm920
921	ibm921
922	ibm922
930	ibm930
932	PC Japanese
933	ibm933
935	ibm935
937	ibm937

Character set	Description
939	ibm939
942	ibm942
943	ibm943
948	ibm948
949	ibm949
950	ibm950 / Big 5 Traditional Chinese
954	EUCJIS
964	ibm964 / CNS 11643 Traditional Chinese
970	ibm970
1006	ibm1006
1025	ibm1025
1026	ibm1026
1089	iso-8859-6 / arabic / ibm1089
1097	ibm1097
1098	ibm1098
1112	ibm1112
1122	ibm1122
1123	ibm1123
1124	ibm1124
1250	Windows Latin 2
1251	Windows Cyrillic
1252	Windows Latin 1
1253	Windows Greek
1254	Windows Turkish
1255	Windows Hebrew
1256	Windows Arabic
1257	Windows Baltic
1258	Windows Vietnamese
1381	ibm1381
1383	ibm1383
2022	JIS
5601	ksc-5601 Korean
33722	ibm33722

IBM MQ C++ classes

The IBM MQ C++ classes encapsulate the IBM MQ Message Queue Interface (MQI). There is a single C++ header file, **imqi.hpp**, which covers all of these classes.

For each class, the following information is shown:

Class hierarchy diagram
A class diagram showing the class in its inheritance relation to its immediate parent classes, if any.

Other relevant classes
Document links to other relevant classes, such as parent classes, and the classes of objects used in method signatures.

Object attributes
Attributes of the class. These are in addition to those attributes defined for any parent classes. Many attributes reflect IBM MQ data-structure members (see "C++ and MQI cross-reference" on page 3923). For detailed descriptions, see "Attributes of objects" on page 2792.

Constructors
Signatures of the special methods used to create an object of the class.

Object methods (public)
Signatures of methods that require an instance of the class for their operation, and that have no usage restrictions.

Where it applies, the following information is also shown:

Class methods (public)
Signatures of methods that do not require an instance of the class for their operation, and that have no usage restrictions.

Overloaded (parent class) methods
Signatures of those virtual methods that are defined in parent classes, but exhibit different, polymorphic, behavior for this class.

Object methods (protected)
Signatures of methods that require an instance of the class for their operation, and are reserved for use by the implementations of derived classes. This section is of interest only to class writers, as opposed to class users.

Object data (protected)
Implementation details for object instance data available to the implementations of derived classes. This section is of interest only to class writers, as opposed to class users.

Reason codes
MQRC_* values (see API reason codes) that can be expected from those methods that fail. For an exhaustive list of reason codes that can occur for an object of a class, consult the parent class documentation. The documented list of reason codes for a class does not include the reason codes for parent classes.

Note:

1. Objects of these classes are not thread-safe. This ensures optimal performance, but take care not to access any object from more than one thread.
2. It is recommended that, for a multithreaded program, a separate ImqQueueManager object is used for each thread. Each manager object must have its own independent collection of other objects, ensuring that objects in different threads are isolated from one another.

The classes are:

- "ImqAuthenticationRecord C++ class" on page 3937

C++ and MQI cross-reference

This collection of topics contains information relating C++ to the MQI.

Read this information together with "Data types used in the MQI" on page 2196.

This table relates MQI data structures to the C++ classes and include files. The following topics show cross-reference information for each C++ class. These cross-references relate to the use of the underlying IBM MQ procedural interfaces. The classes ImqBinary, ImqDistributionList, and ImqString have no attributes that fall into this category and are excluded.

Table 414. Data structure, class, and include-file cross-reference

Data structure	Class	Include file
MQAIR	ImqAuthenticationRecord	imqair.hpp
	ImqBinary	imqbin.hpp
	ImqCache	imqcac.hpp
MQCD	ImqChannel	imqchl.hpp
MQCIH	ImqCICSBridgeHeader	imqcih.hpp
MQDLH	ImqDeadLetterHeader	imqdlh.hpp
MQOR	ImqDistributionList	imqdst.hpp
	ImqError	imqerr.hpp
MQGMO	ImqGetMessageOptions	imqgmo.hpp
	ImqHeader	imqhdr.hpp
MQIIH	ImqIMSBridgeHeader	imqiih.hpp

Table 414. Data structure, class, and include-file cross-reference (continued)

Data structure	Class	Include file
	ImqItem	imqitm.hpp
MQMD	ImqMessage	imqmsg.hpp
	ImqMessageTracker	imqmtr.hpp
	ImqNamelist	imqnml.hpp
MQOD, MQRR	ImqObject	imqobj.hpp
MQPMO, MQPMR, MQRR	ImqPutMessageOptions	imqpmo.hpp
	ImqProcess	imqpro.hpp
	ImqQueue	imqque.hpp
MQBO, MQCNO, MQCSP	ImqQueueManager	imqmgr.hpp
MQRMH	ImqReferenceHeader	imqrfh.hpp
	ImqString	imqstr.hpp
MQTM	ImqTrigger	imqtrg.hpp
MQTMC		
MQTMC2	ImqTrigger	imqtrg.hpp
MQXQH		
MQWIH	ImqWorkHeader	imqwih.hpp

ImqAuthenticationRecord cross-reference:

Cross-reference of attributes, data structures, fields, and calls for the ImqAuthenticationRecord C++ class.

Attribute	Data structure	Field	Call
connection name	MQAIR	AuthInfoConnName	MQCONNX
password	MQAIR	LDAPPassword	MQCONNX
type	MQAIR	AuthInfoType	MQCONNX
user name	MQAIR	LDAPUserNamePtr	MQCONNX
	MQAIR	LDAPUserNameOffset	MQCONNX
	MQAIR	LDAPUserNameLength	MQCONNX

ImqCache cross-reference:

Cross-reference of attributes and calls for the ImqCache C++ class.

Attribute	Call
automatic buffer	MQGET
buffer length	MQGET
buffer pointer	MQGET, MQPUT
data length	MQGET
data offset	MQGET
data pointer	MQGET
message length	MQGET, MQPUT

ImqChannel cross-reference:

Cross-reference of attributes, data structures, fields, and calls for the ImqChannel C++ class.

Attribute	Data structure	Field	Call
batch heart-beat	MQCD	BatchHeartbeat	MQCONNX
channel name	MQCD	ChannelName	MQCONNX
connection name	MQCD	ConnectionName	MQCONNX
	MQCD	ShortConnectionName	MQCONNX
header compression	MQCD	HdrCompList	MQCONNX
heart-beat interval	MQCD	HeartbeatInterval	MQCONNX
keep alive interval	MQCD	KeepAliveInterval	MQCONNX
local address	MQCD	LocalAddress	MQCONNX
maximum message length	MQCD	MaxMsgLength	MQCONNX
message compression	MQCD	MsgCompList	MQCONNX
mode name	MQCD	ModeName	MQCONNX
password	MQCD	Password	MQCONNX
receive exit count	MQCD		MQCONNX
receive exit names	MQCD	ReceiveExit	MQCONNX
	MQCD	ReceiveExitsDefined	MQCONNX
	MQCD	ReceiveExitPtr	MQCONNX
receive user data	MQCD	ReceiveUserData	MQCONNX
	MQCD	ReceiveUserDataPtr	MQCONNX
security exit name	MQCD	SecurityExit	MQCONNX
security user data	MQCD	SecurityUserData	MQCONNX
send exit count	MQCD		MQCONNX
send exit names	MQCD	SendExit	MQCONNX
	MQCD	SendExitsDefined	MQCONNX
	MQCD	SendExitPtr	MQCONNX
send user data	MQCD	SendUserData	MQCONNX
	MQCD	SendUserDataPtr	MQCONNX

Attribute	Data structure	Field	Call
SSL CipherSpec	MQCD	sslCipherSpecification	MQCONNX
SSL client authentication type	MQCD	sslClientAuthentication	MQCONNX
SSL peer name	MQCD	sslPeerName	MQCONNX
transaction program name	MQCD	TpName	MQCONNX
transport type	MQCD	TransportType	MQCONNX
user id	MQCD	UserIdentifier	MQCONNX

ImqCICSBridgeHeader cross-reference:

Cross-reference of attributes, data structures, and fields for the ImqCICSBridgeHeader C++ class.

Attribute	Data structure	Field
bridge abend code	MQCIH	AbendCode
ADS descriptor	MQCIH	AdsDescriptor
attention identifier	MQCIH	AttentionId
authenticator	MQCIH	Authenticator
bridge completion code	MQCIH	BridgeCompletionCode
bridge error offset	MQCIH	ErrorOffset
bridge reason code	MQCIH	BridgeReason
bridge cancel code	MQCIH	CancelCode
conversational task	MQCIH	ConversationalTask
cursor position	MQCIH	CursorPosition
facility token	MQCIH	Facility
facility keep time	MQCIH	FacilityKeepTime
facility like	MQCIH	FacilityLike
function	MQCIH	Function
get wait interval	MQCIH	GetWaitInterval
link type	MQCIH	LinkType
next transaction identifier	MQCIH	NextTransactionId
output data length	MQCIH	OutputDataLength
reply-to format	MQCIH	ReplyToFormat
bridge return code	MQCIH	ReturnCode
start code	MQCIH	StartCode
task end status	MQCIH	TaskEndStatus
transaction identifier	MQCIH	TransactionId
uow control	MQCIH	UowControl
version	MQCIH	Version

ImqDeadLetterHeader cross-reference:

Cross-reference of attributes, data structures, and fields for the ImqDeadLetterHeader C++ class.

Attribute	Data structure	Field
dead-letter reason code	MQDLH	Reason
destination queue manager name	MQDLH	DestQMgrName
destination queue name	MQDLH	DestQName
put application name	MQDLH	PutApplName
put application type	MQDLH	PutApplType
put date	MQDLH	PutDate
put time	MQDLH	PutTime

ImqError cross-reference:

Cross-reference of attributes and calls for the ImqError C++ class.

Attribute	Call
completion code	MQBACK, MQBEGIN, MQCLOSE, MQCMIT, MQCONN, MQCONNX, MQDISC, MQGET, MQINQ, MQOPEN, MQPUT, MQSET
reason code	MQBACK, MQBEGIN, MQCLOSE, MQCMIT, MQCONN, MQCONNX, MQDISC, MQGET, MQINQ, MQOPEN, MQPUT, MQSET

ImqGetMessageOptions cross-reference:

Cross-reference of attributes, data structures, and fields for the ImqGetMessageOptions C++ class.

Attribute	Data structure	Field
group status	MQGMO	GroupStatus
match options	MQGMO	MatchOptions
message token	MQGMO	MessageToken
options	MQGMO	Options
resolved queue name	MQGMO	ResolvedQName
returned length	MQGMO	ReturnedLength
segmentation	MQGMO	Segmentation
segment status	MQGMO	SegmentStatus
	MQGMO	Signal1
	MQGMO	Signal2
syncpoint participation	MQGMO	Options
wait interval	MQGMO	WaitInterval

ImqHeader cross-reference:

Cross-reference of attributes, data structures, and fields for the ImqHeader C++ class.

Attribute	Data structure	Field
character set	MQDLH, MQIIH	CodedCharSetId
encoding	MQDLH, MQIIH	Encoding
format	MQDLH, MQIIH	Format
header flags	MQIIH, MQRMH	Flags

ImqIMSBridgeHeader cross-reference:

Cross-reference of attributes, data structures, and fields for the ImqAuthenticationRecord C++ class.

Attribute	Data structure	Field
authenticator	MQIIH	Authenticator
commit mode	MQIIH	CommitMode
logical terminal override	MQIIH	LTermOverride
message format services map name	MQIIH	MFSMapName
reply-to format	MQIIH	ReplyToFormat
security scope	MQIIH	SecurityScope
transaction instance id	MQIIH	TranInstanceId
transaction state	MQIIH	TranState

ImqItem cross-reference:

Cross-reference of attributes and calls for the ImqItem C++ class.

Attribute	Call
structure id	MQGET

ImqMessage cross-reference:

Cross-reference of attributes, data structures, fields, and calls for the ImqMessage C++ class.

Attribute	Data structure	Field	Call
application ID data	MQMD	ApplIdentityData	
application origin data	MQMD	ApplOriginData	
backout count	MQMD	BackoutCount	
character set	MQMD	CodedCharSetId	
encoding	MQMD	Encoding	
expiry	MQMD	Expiry	
format	MQMD	Format	
message flags	MQMD	MsgFlags	
message type	MQMD	MsgType	
offset	MQMD	Offset	

Attribute	Data structure	Field	Call
original length	MQMD	OriginalLength	
persistence	MQMD	Persistence	
priority	MQMD	Priority	
put application name	MQMD	PutApplName	
put application type	MQMD	PutApplType	
put date	MQMD	PutDate	
put time	MQMD	PutTime	
reply-to queue manager name	MQMD	ReplyToQMgr	
reply-to queue name	MQMD	ReplyToQ	
report	MQMD	Report	
sequence number	MQMD	MsgSeqNumber	
total message length		DataLength	MQGET
user id	MQMD	UserIdentifier	

ImqMessageTracker cross-reference:

Cross-reference of attributes, data structures, and fields for the ImqMessageTracker C++ class.

Attribute	Data structure	Field
accounting token	MQMD	AccountingToken
correlation id	MQMD	CorrelId
feedback	MQMD	Feedback
group id	MQMD	GroupId
message id	MQMD	MsgId

ImqNamelist cross-reference:

Cross-reference of attributes, inquiries, and calls for the ImqNamelist C++ class.

Attribute	Inquiry	Call
name count	MQIA_NAME_COUNT	MQINQ
namelist name	MQCA_NAMELIST_NAME	MQINQ

ImqObject cross-reference:

Cross-reference of attributes, data structures, fields, inquiries, and calls for the ImqObject C++ class.

Attribute	Data structure	Field	Inquiry	Call
alteration date			MQCA_ALTERATION_DATE	MQINQ
alteration time			MQCA_ALTERATION_TIME	MQINQ
alternate user id	MQOD	AlternateUserId		
alternate security id				
close options				MQCLOSE
description			MQCA_Q_DESC, MQCA_Q_MGR_DESC, MQCA_PROCESS_DESC	MQINQ
name	MQOD	ObjectName	MQCA_Q_MGR_NAME, MQCQ_Q_NAME, MQCA_PROCESS_NAME	MQINQ
open options				MQOPEN
open status				MQOPEN, MQCLOSE
queue manager identifier	queue manager identifier		MQCA_Q_MGR_IDENTIFIER	MQINQ

ImqProcess cross-reference:

Cross-reference of attributes, inquiries, and calls for the ImqAuthenticationRecord C++ class.

Attribute	Inquiry	Call
application id	MQCA_APPL_ID	MQINQ
application type	MQIA_APPL_TYPE	MQINQ
environment data	MQCA_ENV_DATA	MQINQ
user data	MQCA_USER_DATA	MQINQ

ImqPutMessageOptions cross-reference:

Cross-reference of attributes, data structures, and fields for the ImqAuthenticationRecord C++ class.

Table 415. ImqPutMessageOptions cross-reference

Attribute	Data structure	Field
context reference	MQPMO	Context
	MQPMO	InvalidDestCount
	MQPMO	KnownDestCount
options	MQPMO	Options
record fields	MQPMO	PutMsgRecFields
resolved queue manager name	MQPMO	ResolvedQMgrName
resolved queue name	MQPMO	ResolvedQName
	MQPMO	Timeout
	MQPMO	UnknownDestCount
syncpoint participation	MQPMO	Options

ImqQueue cross-reference:

Cross-reference of attributes, data structures, fields, inquiries, and calls for the ImqQueue C++ class.

Table 416. ImqQueue cross-reference

Attribute	Data structure	Field	Inquiry	Call
backout requeue name			MQCA_BACKOUT_REQ_Q_NAME	MQINQ
backout threshold			MQIA_BACKOUT_THRESHOLD	MQINQ
base queue name			MQCA_BASE_Q_NAME	MQINQ
cluster name			MQCA_CLUSTER_NAME	MQINQ
cluster namelist name			MQCA_CLUSTER_NAMELIST	MQINQ
cluster workload rank			MQIA_CLWL_Q_RANK	MQINQ
cluster workload priority			MQIA_CLWL_Q_PRIORITY	MQINQ
cluster workload use queue			MQIA_CLWL_USEQ	MQINQ
creation date			MQCA_CREATION_DATE	MQINQ
creation time			MQCA_CREATION_TIME	MQINQ
current depth			MQIA_CURRENT_Q_DEPTH	MQINQ
default bind			MQIA_DEF_BIND	MQINQ
default input open option			MQIA_DEF_INPUT_OPEN_OPTION	MQINQ
default persistence			MQIA_DEF_PERSISTENCE	MQINQ
default priority			MQIA_DEF_PRIORITY	MQINQ
definition type			MQIA_DEFINITION_TYPE	MQINQ
depth high event			MQIA_Q_DEPTH_HIGH_EVENT	MQINQ
depth high limit			MQIA_Q_DEPTH_HIGH_LIMIT	MQINQ
depth low event			MQIA_Q_DEPTH_LOW_EVENT	MQINQ

Table 416. ImqQueue cross-reference (continued)

Attribute	Data structure	Field	Inquiry	Call
depth low limit			MQIA_Q_DEPTH_LOW_LIMIT	MQINQ
depth maximum event			MQIA_Q_DEPTH_MAX_LIMIT	MQINQ
distribution lists			MQIA_DIST_LISTS	MQINQ, MQSET
dynamic queue name	MQOD	DynamicQName		
harden get backout			MQIA_HARDEN_GET_BACKOUT	MQINQ
index type			MQIA_INDEX_TYPE	MQINQ
inhibit get			MQIA_INHIBIT_GET	MQINQ, MQSET
inhibit put			MQIA_INHIBIT_PUT	MQINQ, MQSET
initiation queue name			MQCA_INITIATION_Q_NAME	MQINQ
maximum depth			MQIA_MAX_Q_DEPTH	MQINQ
maximum message length			MQIA_MAX_MSG_LENGTH	MQINQ
message delivery sequence			MQIA_MSG_DELIVERY_SEQUENCE	MQINQ
next distributed queue				
non persistent message class			MQIA_NPM_CLASS	MQINQ
open input count			MQIA_OPEN_INPUT_COUNT	MQINQ
open output count			MQIA_OPEN_OUTPUT_COUNT	MQINQ
previous distributed queue				
process name			MQCA_PROCESS_NAME	MQINQ
queue accounting			MQIA_ACCOUNTING_Q	MQINQ
queue manager name	MQOD	ObjectQMgrName		
queue monitoring			MQIA_MONITORING_Q	MQINQ
queue statistics			MQIA_STATISTICS_Q	MQINQ
queue type			MQIA_Q_TYPE	MQINQ
remote queue manager name			MQCA_REMOTE_Q_MGR_NAME	MQINQ
remote queue name			MQCA_REMOTE_Q_NAME	MQINQ
resolved queue manager name	MQOD	ResolvedQMgrName		
resolved queue name	MQOD	ResolvedQName		
retention interval			MQIA_RETENTION_INTERVAL	MQINQ
scope			MQIA_SCOPE	MQINQ
service interval			MQIA_Q_SERVICE_INTERVAL	MQINQ
service interval event			MQIA_Q_SERVICE_INTERVAL_EVENT	MQINQ
shareability			MQIA_SHAREABILITY	MQINQ
storage class			MQCA_STORAGE_CLASS	MQINQ

Table 416. ImqQueue cross-reference (continued)

Attribute	Data structure	Field	Inquiry	Call
transmission queue name			MQCA_XMIT_Q_NAME	MQINQ
trigger control			MQIA_TRIGGER_CONTROL	MQINQ, MQSET
trigger data			MQCA_TRIGGER_DATA	MQINQ, MQSET
trigger depth			MQIA_TRIGGER_DEPTH	MQINQ, MQSET
trigger message priority			MQIA_TRIGGER_MSG_PRIORITY	MQINQ, MQSET
trigger type			MQIA_TRIGGER_TYPE	MQINQ, MQSET
usage			MQIA_USAGE	MQINQ

ImqQueueManager cross-reference:

Cross-reference of attributes, data structures, fields, inquiries, and calls for the ImqQueueManager C++ class.

Attribute	Data structure	Field	Inquiry	Call
accounting connections override			MQIA_ACCOUNTING_CONN_OVERRIDE	MQINQ
accounting interval			MQIA_ACCOUNTING_INTERVAL	MQINQ
activity recording			MQIA_ACTIVITY_RECORDING	MQINQ
adopt new mca check			MQIA_ADOPTNEWMCA_CHECK	MQINQ
adopt new mca type			MQIA_ADOPTNEWMCA_TYPE	MQINQ
authentication type	MQCSP	AuthenticationType		MQCONNX
authority event			MQIA_AUTHORITY_EVENT	MQINQ
begin options	MQBO	Options		MQBEGIN
bridge event			MQIA_BRIDGE_EVENT	MQINQ
channel auto definition			MQIA_CHANNEL_AUTO_DEF	MQINQ
channel auto definition event			MQIA_CHANNEL_AUTO_EVENT	MQIA
channel auto definition exit			MQIA_CHANNEL_AUTO_EXIT	MQIA
channel event			MQIA_CHANNEL_EVENT	MQINQ
channel initiator adapters			MQIA_CHINIT_ADAPTERS	MQINQ
channel initiator control			MQIA_CHINIT_CONTROL	MQINQ
channel initiator dispatchers			MQIA_CHINIT_DISPATCHERS	MQINQ

Attribute	Data structure	Field	Inquiry	Call
channel initiator trace auto start			MQIA_CHINIT_TRACE_AUTO_START	MQINQ
channel initiator trace table size			MQIA_CHINIT_TRACE_TABLE_SIZE	MQINQ
channel monitoring			MQIA_MONITORING_CHANNEL	MQINQ
channel reference	MQCD	ChannelType		MQCONNX
channel statistics			MQIA_STATISTICS_CHANNEL	MQINQ
character set			MQIA_CODED_CHAR_SET_ID	MQINQ
cluster sender monitoring			MQIA_MONITORING_AUTO_CLUSSDR	MQINQ
cluster sender statistics			MQIA_STATISTICS_AUTO_CLUSSDR	MQINQ
cluster workload data			MQCA_CLUSTER_WORKLOAD_DATA	MQINQ
cluster workload exit			MQCA_CLUSTER_WORKLOAD_EXIT	MQINQ
cluster workload length			MQIA_CLUSTER_WORKLOAD_LENGTH	MQINQ
cluster workload mru			MQIA_CLWL_MRU_CHANNELS	MQINQ
cluster workload use queue			MQIA_CLWL_USEQ	MQINQ
command event			MQIA_COMMAND_EVENT	MQINQ
command input queue name			MQCA_COMMAND_INPUT_Q_NAME	MQINQ
command level			MQIA_COMMAND_LEVEL	MQINQ
command server control			MQIA_CMD_SERVER_CONTROL	MQINQ
connect options	MQCNO	Options		MQCONN, MQCONNX
connection id	MQCNO	ConnectionId		MQCONNX
connection status				MQCONN, MQCONNX, MQDISC
connection tag	MQCD	ConnTag		MQCONNX
cryptographic hardware	MQSCO	CryptoHardware		MQCONNX
dead-letter queue name			MQCA_DEAD_LETTER_Q_NAME	MQINQ
default transmission queue name			MQCA_DEF_XMIT_Q_NAME	MQINQ
distribution lists			MQIA_DIST_LISTS	MQINQ
dns group			MQCA_DNS_GROUP	MQINQ
dns wlm			MQIA_DNS_WLM	MQINQ
first authentication record	MQSCO	AuthInfoRecOffset		MQCONNX

Attribute	Data structure	Field	Inquiry	Call
	MQSCO	AuthInfoRecPtr		MQCONNX
inhibit event			MQIA_INHIBIT_EVENT	MQINQ
ip address version			MQIA_IP_ADDRESS_VERSION	MQINQ
key repository	MQSCO	KeyRepository		MQCONNX
key reset count	MQSCO	KeyResetCount		MQCONNX
listener timer			MQIA_LISTENER_TIMER	MQINQ
local event			MQIA_LOCAL_EVENT	MQINQ
logger event			MQIA_LOGGER_EVENT	MQINQ
lu group name			MQCA_LU_GROUP_NAME	MQINQ
lu name			MQCA_LU_NAME	MQINQ
lu62 arm suffix			MQCA_LU62_ARM_SUFFIX	MQINQ
lu62 channels			MQIA_LU62_CHANNELS	MQINQ
maximum active channels			MQIA_ACTIVE_CHANNELS	MQINQ
maximum channels			MQIA_MAX_CHANNELS	MQINQ
maximum handles			MQIA_MAX_HANDLES	MQINQ
maximum message length			MQIA_MAX_MSG_LENGTH	MQINQ
maximum priority			MQIA_MAX_PRIORITY	MQINQ
maximum uncommitted messages			MQIA_MAX_UNCOMMITTED_MSGS	MQINQ
mqi accounting			MQIA_ACCOUNTING_MQI	MQINQ
mqi statistics			MQIA_STATISTICS_MQI	MQINQ
outbound port maximum			MQIA_OUTBOUND_PORT_MAX	MQINQ
outbound port minimum			MQIA_OUTBOUND_PORT_MIN	MQINQ
password	MQCSP	CSPPasswordPtr		MQCONNX
	MQCSP	CSPPasswordOffset		MQCONNX
	MQCSP	CSPPasswordLength		MQCONNX
performance event			MQIA_PERFORMANCE_EVENT	MQINQ
platform			MQIA_PLATFORM	MQINQ
queue accounting			MQIA_ACCOUNTING_Q	MQINQ
queue monitoring			MQIA_MONITORING_Q	MQINQ
queue statistics			MQIA_STATISTICS_Q	MQINQ
receive timeout			MQIA_RECEIVE_TIMEOUT	MQINQ
receive timeout minimum			MQIA_RECEIVE_TIMEOUT_MIN	MQINQ
receive timeout type			MQIA_RECEIVE_TIMEOUT_TYPE	MQINQ
remote event			MQIA_REMOTE_EVENT	MQINQ
repository name			MQCA_REPOSITORY_NAME	MQINQ

Attribute	Data structure	Field	Inquiry	Call
repository namelist			MQCA_REPOSITORY_NAMELIST	MQINQ
shared queue queue manager name			MQIA_SHARED_Q_Q_MGR_NAME	MQINQ
ssl event			MQIA_SSL_EVENT	MQINQ
ssl fips			MQIA_SSL_FIPS_REQUIRED	MQINQ
ssl key reset count			MQIA_SSL_RESET_COUNT	MQINQ
start-stop event			MQIA_START_STOP_EVENT	MQINQ
statistics interval			MQIA_STATISTICS_INTERVAL	MQINQ
syncpoint availability			MQIA_SYNCPOINT	MQINQ
tcp channels			MQIA_TCP_CHANNELS	MQINQ
tcp keep alive			MQIA_TCP_KEEP_ALIVE	MQINQ
tcp name			MQCA_TCP_NAME	MQINQ
tcp stack type			MQIA_TCP_STACK_TYPE	MQINQ
trace route recording			MQIA_TRACE_ROUTE_RECORDING	MQINQ
trigger interval			MQIA_TRIGGER_INTERVAL	MQINQ
user id	MQCSP	CSPUserIdPtr		MQCONNX
	MQCSP	CSPUserIdOffset		MQCONNX
	MQCSP	CSPUserIdLength		MQCONNX

ImqReferenceHeader cross-reference:

Cross-reference of attributes, data structures, and fields for the ImqAuthenticationRecord C++ class.

Attribute	Data structure	Field
destination environment	MQRMH	DestEnvLength, DestEnvOffset
destination name	MQRMH	DestNameLength, DestNameOffset
instance id	MQRMH	ObjectInstanceId
logical length	MQRMH	DataLogicalLength
logical offset	MQRMH	DataLogicalOffset
logical offset 2	MQRMH	DataLogicalOffset2
reference type	MQRMH	ObjectType
source environment	MQRMH	SrcEnvLength, SrcEnvOffset
source name	MQRMH	SrcNameLength, SrcNameOffset

ImqTrigger cross-reference:

Cross-reference of attributes, data structures, and fields for the ImqAuthenticationRecord C++ class.

Table 417. ImqTrigger cross-reference

Attribute	Data structure	Field
application id	MQTM	ApplId
application type	MQTM	ApplType
environment data	MQTM	EnvData
process name	MQTM	ProcessName
queue name	MQTM	QName
trigger data	MQTM	TriggerData
user data	MQTM	UserData

ImqWorkHeader cross-reference:

Cross-reference of attributes, data structures, and fields for the ImqAuthenticationRecord C++ class.

Attribute	Data structure	Field
message token	MQWIH	MessageToken
service name	MQWIH	ServiceName
service step	MQWIH	ServiceStep

ImqAuthenticationRecord C++ class

This class encapsulates an authentication information record (MQAIR) for use during execution of the ImqQueueManager::connect method, for custom TLS client connections.

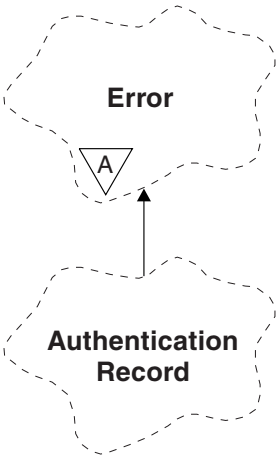


Figure 98. ImqAuthenticationRecord class

See the description of the ImqQueueManager::connect method for more details. This class is not available on the z/OS platform.

- "Object attributes" on page 3938
- "Constructors" on page 3938
- "Object methods (public)" on page 3938

• "Object methods (protected)" on page 3939

Object attributes

connection name
The name of the connection to the LDAP CRL server. This is the IP address or DNS name, followed optionally by the port number, in parentheses.

connection reference
A reference to an ImqQueueManager object that provides the required connection to a (local) queue manager. The initial value is zero. Do not confuse this with the queue manager name that identifies a queue manager (possibly remote) for a named queue.

next authentication record
Next object of this class, in no particular order, having the same **connection reference** as this object. The initial value is zero.

password
A password supplied for connection authentication to the LDAP CRL server.

previous authentication record
Previous object of this class, in no particular order, having the same **connection reference** as this object. The initial value is zero.

type The type of authentication information contained in the record.

user name
A user identifier supplied for authorization to the LDAP CRL server.

Constructors

ImqAuthenticationRecord( );
The default constructor.

Object methods (public)

void operator = (const ImqAuthenticationRecord & *air*);
Copies instance data from *air*, replacing the existing instance data.

const ImqString & connectionName ( ) const ;
Returns the **connection name**.

void setConnectionName (const ImqString & *name*);
Sets the **connection name**.

void setConnectionName (const char * *name* = 0);
Sets the **connection name**.

ImqQueueManager * connectionReference ( ) const ;
Returns the **connection reference**.

void setConnectionReference (ImqQueueManager & *manager*);
Sets the **connection reference**.

void setConnectionReference (ImqQueueManager * *manager* = 0);
Sets the **connection reference**.

void copyOut (MQAIR * *pAir*);
Copies instance data to *pAir*, replacing the existing instance data. This might involve allocating dependent storage.

void clear (MQAIR * *pAir*);
Clears the structure and releases dependent storage referenced by *pAir*.

ImqAuthenticationRecord * nextAuthenticationRecord ( ) const ;
Returns the **next authentication record**.

const ImqString & password ( ) const ;
Returns the **password**.

void setPassword (const ImqString & *password* **);**
Sets the **password**.

void setPassword (const char * *password* **= 0);**
Sets the **password**.

ImqAuthenticationRecord * previousAuthenticationRecord ( ) const ;
Returns the **previous authentication record**.

MQLONG type ( ) const ;
Returns the **type**.

void setType (const MQLONG *type* **);**
Sets the **type**.

const ImqString & userName ( ) const ;
Returns the **user name**.

void setUserName (const ImqString & *name* **);**
Sets the **user name**.

void setUserName (const char * *name* **= 0);**
Sets the **user name**.

Object methods (protected)

void setNextAuthenticationRecord (ImqAuthenticationRecord * *pAir* **= 0);**
Sets the **next authentication record**.

Attention: Use this function only if you are sure that it will not break the authentication record list.

void setPreviousAuthenticationRecord (ImqAuthenticationRecord * *pAir* **= 0);**
Sets the **previous authentication record**.

Attention: Use this function only if you are sure that it will not break the authentication record list.

ImqBinary C++ class

This class encapsulates a binary byte array that can be used for ImqMessage **accounting token**, **correlation id**, and **message id** values. It allows easy assignment, copying, and comparison.

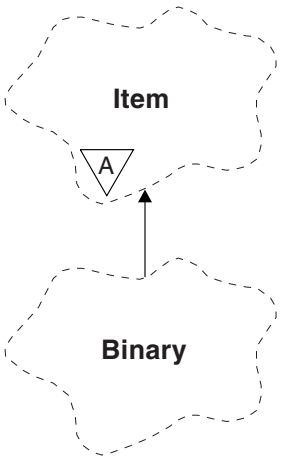


Figure 99. ImqBinary class

Object attributes

data An array of bytes of binary data. The initial value is null.

data length
The number of bytes. The initial value is zero.

data pointer
The address of the first byte of the **data**. The initial value is zero.

Constructors

ImqBinary( );
The default constructor.

ImqBinary(const ImqBinary & ***binary*** **);**
The copy constructor.

ImqBinary(const void * ***data*****, const size_t** ***length*** **);**
Copies *length* bytes from *data*.

Overloaded ImqItem methods

virtual ImqBoolean copyOut (ImqMessage & ***msg*** **);**
Copies the **data** to the message buffer, replacing any existing content. Sets the *msg* **format** to MQFMT_NONE.

See the ImqItem class method description for further details.

virtual ImqBoolean pasteIn (ImqMessage & ***msg*** **);**
Sets the **data** by transferring the remaining data from the message buffer, replacing the existing **data**.

To be successful, the ImqMessage **format** must be MQFMT_NONE.

See the ImqItem class method description for further details.

Object methods (public)

void operator = (const ImqBinary & *binary*);
Copies bytes from *binary*.

ImqBoolean operator == (const ImqBinary & *binary*);
Compares this object with *binary*. It returns FALSE if not equal and TRUE otherwise. The objects are equal if they have the same **data length** and the bytes match.

ImqBoolean copyOut (void * *buffer*, const size_t *length*, const char *pad* = 0);
Copies up to *length* bytes from the **data pointer** to *buffer*. If the **data length** is insufficient, the remaining space in *buffer* is filled with *pad* bytes. *buffer* can be zero if *length* is also zero. *length* must not be negative. It returns TRUE if successful.

size_t dataLength ( ) const ;
Returns the **data length**.

ImqBoolean setDataLength (const size_t *length*);
Sets the **data length**. If the **data length** is changed as a result of this method, the data in the object is uninitialized. It returns TRUE if successful.

void * dataPointer ( ) const ;
Returns the **data pointer**.

ImqBoolean isNull ( ) const ;
Returns TRUE if the **data length** is zero, or if all the **data** bytes are zero. Otherwise it returns FALSE.

ImqBoolean set (const void * *buffer*, const size_t *length*);
Copies *length* bytes from *buffer*. It returns TRUE if successful.

Object methods (protected)

void clear ( );
Reduces the **data length** to zero.

Reason codes

- MQRC_NO_BUFFER
- MQRC_STORAGE_NOT_AVAILABLE
- MQRC_INCONSISTENT_FORMAT

ImqCache C++ class

Use this class to hold or marshal data in memory.

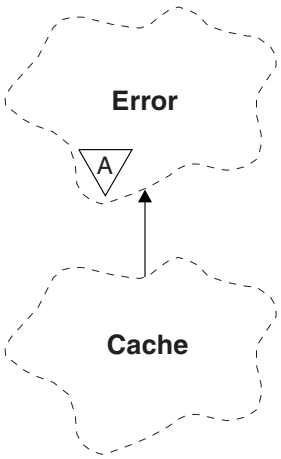


Figure 100. ImqCache class

Use this class to hold or marshal data in memory. You can nominate a buffer of memory of fixed size, or the system can provide a flexible amount of memory automatically. This class relates to the MQI calls listed in "ImqCache cross-reference" on page 3925.

Object attributes

automatic buffer
Indicates whether buffer memory is managed automatically by the system (TRUE) or is supplied by the user (FALSE). It is initially set to TRUE.

This attribute is not set directly. It is set indirectly using either the **useEmptyBuffer** or the **useFullBuffer** method.

If user storage is supplied, this attribute is FALSE, buffer memory cannot grow, and buffer overflow errors can occur. The address and length of the buffer remain constant.

If user storage is not supplied, this attribute is TRUE, and buffer memory can grow incrementally to accommodate an arbitrary amount of message data. However, when the buffer grows, the address of the buffer might change, so be careful when using the **buffer pointer** and **data pointer**.

buffer length
The number of bytes of memory in the buffer. The initial value is zero.

buffer pointer
The address of the buffer memory. The initial value is null.

data length
The number of bytes succeeding the **data pointer**. This must be equal to or less than the **message length**. The initial value is zero.

data offset
The number of bytes preceding the **data pointer**. This must be equal to or less than the **message length**. The initial value is zero.

data pointer
The address of the part of the buffer that is to be written to or read from next. The initial value is null.

message length
The number of bytes of significant data in the buffer. The initial value is zero.

Constructors

ImqCache( );
The default constructor.

ImqCache(const ImqCache & *cache*);
The copy constructor.

Object methods (public)

void operator = (const ImqCache & *cache*);
Copies up to **message length** bytes of data from the *cache* object to the object. If **automatic buffer** is FALSE, the **buffer length** must already be sufficient to accommodate the copied data.

ImqBoolean automaticBuffer ( ) const ;
Returns the **automatic buffer** value.

size_t bufferLength ( ) const ;
Returns the **buffer length**.

char * bufferPointer ( ) const ;
Returns the **buffer pointer**.

void clearMessage ( );
Sets the **message length** and **data offset** to zero.

size_t dataLength ( ) const ;
Returns the **data length**.

size_t dataOffset ( ) const ;
Returns the **data offset**.

ImqBoolean setDataOffset (const size_t *offset*);
Sets the **data offset**. The **message length** is increased if necessary to ensure that it is no less than the **data offset**. This method returns TRUE if successful.

char * dataPointer ( ) const ;
Returns a copy of the **data pointer**.

size_t messageLength ( ) const ;
Returns the **message length**.

ImqBoolean setMessageLength (const size_t *length*);
Sets the **message length**. Increases the **buffer length** if necessary to ensure that the **message length** is no greater than the **buffer length**. Reduces the **data offset** if necessary to ensure that it is no greater than the **message length**. It returns TRUE if successful.

ImqBoolean moreBytes (const size_t *bytes-required*);
Assures that *bytes-required* more bytes are available (for writing) between the **data pointer** and the end of the buffer. It returns TRUE if successful.

If **automatic buffer** is TRUE, more memory is acquired as required; otherwise, the **buffer length** must already be adequate.

ImqBoolean read (const size_t *length*, char * & *external-buffer*);
Copies *length* bytes, from the buffer starting at the **data pointer** position, into the *external-buffer*. After the data has been copied, the **data offset** is increased by *length*. This method returns TRUE if successful.

ImqBoolean resizeBuffer (const size_t *length*);
Varies the **buffer length**, provided that **automatic buffer** is TRUE. This is achieved by

reallocating the buffer memory. Up to **message length** bytes of data from the existing buffer are copied to the new one. The maximum number copied is *length* bytes. The **buffer pointer** is changed. The **message length** and **data offset** are preserved as closely as possible within the confines of the new buffer. It returns TRUE if successful, and FALSE if **automatic buffer** is FALSE.

Note: This method can fail with MQRC_STORAGE_NOT_AVAILABLE if there is any problem with system resources.

ImqBoolean useEmptyBuffer (const char * *external-buffer*, const size_t *length*);
Identifies an empty user buffer, setting the **buffer pointer** to point to *external-buffer*, the **buffer length** to *length*, and the **message length** to zero. Performs a **clearMessage**. If the buffer is fully primed with data, use the **useFullBuffer** method instead. If the buffer is partially primed with data, use the **setMessageLength** method to indicate the correct amount. This method returns TRUE if successful.

This method can be used to identify a fixed amount of memory, as described previously (*external-buffer* is not null and *length* is nonzero), in which case **automatic buffer** is set to FALSE, or it can be used to revert to system-managed flexible memory (*external-buffer* is null and *length* is zero), in which case **automatic buffer** is set to TRUE.

ImqBoolean useFullBuffer (const char * *externalBuffer*, const size_t *length*);
As for **useEmptyBuffer**, except that the **message length** is set to *length*. It returns TRUE if successful.

ImqBoolean write (const size_t *length*, const char * *external-buffer*);
Copies *length* bytes, from the *external-buffer*, into the buffer starting at the **data pointer** position. After the data has been copied, the **data offset** is increased by *length*, and the **message length** is increased if necessary to ensure that it is no less than the new **data offset** value. This method returns TRUE if successful.

If **automatic buffer** is TRUE, an adequate amount of memory is guaranteed; otherwise, the ultimate **data offset** must not exceed the **buffer length**.

Reason codes

- MQRC_BUFFER_NOT_AUTOMATIC
- MQRC_DATA_TRUNCATED
- MQRC_INSUFFICIENT_BUFFER
- MQRC_INSUFFICIENT_DATA
- MQRC_NULL_POINTER
- MQRC_STORAGE_NOT_AVAILABLE
- MQRC_ZERO_LENGTH

ImqChannel C++ class

This class encapsulates a channel definition (MQCD) for use during execution of the Manager::connect method, for custom client connections.

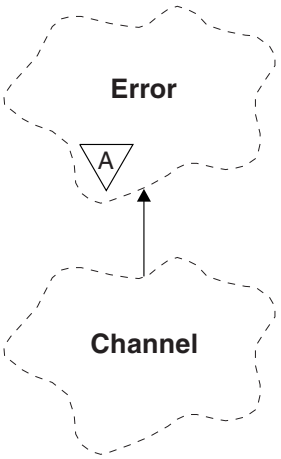


Figure 101. ImqChannel class

See the description of the Manager::connect method, and Sample program HELLO WORLD (imqwrld.cpp), for more details.

Not all the listed methods are applicable to all platforms. See the descriptions of the DEFINE CHANNEL and ALTER CHANNEL commands for more information.

The ImqChannel class is not supported on z/OS.

Object attributes

batch heart-beat
The number of milliseconds between checks that a remote channel is active. The initial value is 0.

channel name
The name of the channel. The initial value is null.

connection name
The name of the connection. For example, the IP address of a host computer. The initial value is null.

header compression
The list of header data compression techniques supported by the channel. The initial values are all set to MQCOMPRESS_NOT_AVAILABLE.

heart-beat interval
The number of seconds between checks that a connection is still working. The initial value is 300.

keep alive interval
The number of seconds passed to the communications stack specifying the keep alive timing for the channel. The initial value is MQKAI_AUTO.

local address
The local communications address for the channel.

maximum message length
The maximum length of message supported by the channel in a single communication. The initial value is 4 194 304.

message compression
The list of message data compression techniques supported by the channel. The initial values are all set to MQCOMPRESS_NOT_AVAILABLE.

mode name
The name of the mode. The initial value is null.

password
A password supplied for connection authentication. The initial value is null.

receive exit count
The number of receive exits. The initial value is zero. This attribute is read-only.

receive exit names
The names of receive exits.

receive user data
Data associated with receive exits.

security exit name
The name of a security exit to be invoked on the server side of the connection. The initial value is null.

security user data
Data to be passed to the security exit. The initial value is null.

send exit count
The number of send exits. The initial value is zero. This attribute is read-only.

send exit names
The names of send exits.

send user data
Data associated with send exits.

SSL CipherSpec
CipherSpec for use with TLS.

SSL client authentication type
Client authentication type for use with TLS.

SSL peer name
Peer name for use with TLS.

transaction program name
The name of the transaction program. The initial value is null.

transport type
The transport type of the connection. The initial value is MQXPT_LU62.

user id
A user identifier supplied for authorization. The initial value is null.

Constructors

ImqChannel( ) ;
The default constructor.

ImqChannel(const ImqChannel & *channel*);
The copy constructor.

Object methods (public)

void operator = (const ImqChannel & *channel*);
Copies instance data from *channel*, replacing any existing instance data.

MQLONG batchHeartBeat( ) const ;
Returns the **batch heart-beat**.

ImqBoolean setBatchHeartBeat(const MQLONG *heartbeat* = 0L);
Sets the **batch heart-beat**. This method returns TRUE if successful.

ImqString channelName( ) const ;
Returns the **channel name**.

ImqBoolean setChannelName(const char * *name* = 0);
Sets the **channel name**. This method returns TRUE if successful.

ImqString connectionName( ) const ;
Returns the **connection name**.

ImqBoolean setConnectionName(const char * *name* = 0);
Sets the **connection name**. This method returns TRUE if successful.

size_t headerCompressionCount( ) const ;
Returns the supported header data compression techniques count.

ImqBoolean headerCompression(const size_t count, MQLONG compress [ ]) const ;
Returns copies of the supported header data compression techniques in **compress**. This method returns TRUE if successful.

ImqBoolean setHeaderCompression(const size_t count, const MQLONG compress [ ]);

Sets the supported header data compression techniques to **compress**.

Sets the supported header data compression techniques count to **count**.

This method returns TRUE if successful.

MQLONG heartBeatInterval( ) const ;
Returns the **heart-beat interval**.

ImqBoolean setHeartBeatInterval(const MQLONG *interval* = 300L);
Sets the **heart-beat interval**. This method returns TRUE if successful.

MQLONG keepAliveInterval( ) const ;
Returns the **keep alive interval**.

ImqBoolean setKeepAliveInterval(const MQLONG *interval* = MQKAI_AUTO);
Sets the **keep alive interval**. This method returns TRUE if successful.

ImqString localAddress() const ;
Returns the **local address**.

ImqBoolean setLocalAddress (const char * *address* = 0);
Sets the **local address**. This method returns TRUE if successful.

MQLONG maximumMessageLength( ) const ;
Returns the **maximum message length**.

ImqBoolean setMaximumMessageLength(const MQLONG *length* = 4194304L);
Sets the **maximum message length**. This method returns TRUE if successful.

size_t messageCompressionCount( ) const ;
Returns the supported message data compression techniques count.

ImqBoolean messageCompression(const size_t count, MQLONG compress [ ]) const ;
Returns copies of the supported message data compression techniques in **compress**. This method returns TRUE if successful.

ImqBoolean setMessageCompression(const size_t count, const MQLONG compress [ ]);

Sets the supported message data compression techniques to compress.

Sets the supported message data compression techniques count to count.

This method returns TRUE if successful.

ImqString modeName( ) const ;
Returns the **mode name**.

ImqBoolean setModeName(const char * *name* = 0);
Sets the **mode name**. This method returns TRUE if successful.

ImqString password( ) const ;
Returns the **password**.

ImqBoolean setPassword(const char * *password* = 0);
Sets the **password**. This method returns TRUE if successful.

size_t receiveExitCount( ) const ;
Returns the **receive exit count**.

ImqString receiveExitName( );
Returns the first of the **receive exit names**, if any. If the **receive exit count** is zero, it returns an empty string.

ImqBoolean receiveExitNames(const size_t *count*, ImqString * *names* [ ]);
Returns copies of the **receive exit names** in *names*. Sets any *names* in excess of **receive exit count** to null strings. This method returns TRUE if successful.

ImqBoolean setReceiveExitName(const char * *name* = 0);
Sets the **receive exit names** to the single *name*. *name* can be blank or null. Sets the **receive exit count** to either 1 or zero. Clears the **receive user data**. This method returns TRUE if successful.

ImqBoolean setReceiveExitNames(const size_t *count*, const char * *names* [ ]);
Sets the **receive exit names** to *names*. Individual *names* values must not be blank or null. Sets the **receive exit count** to *count*. Clears the **receive user data**. This method returns TRUE if successful.

ImqBoolean setReceiveExitNames(const size_t *count*, const ImqString * *names* [ ]);
Sets the **receive exit names** to *names*. Individual *names* values must not be blank or null. Sets the **receive exit count** to *count*. Clears the **receive user data**. This method returns TRUE if successful.

ImqString receiveUserData( );
Returns the first of the **receive user data** items, if any. If the **receive exit count** is zero, returns an empty string.

ImqBoolean receiveUserData(const size_t *count*, ImqString * *data* [ ]);
Returns copies of the **receive user data** items in *data*. Sets any *data* in excess of **receive exit count** to null strings. This method returns TRUE if successful.

ImqBoolean setReceiveUserData(const char * *data* = 0);
Sets the **receive user data** to the single item *data*. If *data* is not null, **receive exit count** must be at least 1. This method returns TRUE if successful.

ImqBoolean setReceiveUserData(const size_t *count*, const char * *data* [ ]);
Sets the **receive user data** to *data*. *count* must be no greater than the **receive exit count**. This method returns TRUE if successful.

ImqBoolean setReceiveUserData(const size_t *count*, const ImqString * *data* [ ]);
Sets the **receive user data** to *data*. *count* must be no greater than the **receive exit count**. This method returns TRUE if successful.

ImqString securityExitName( ) const ;
Returns the **security exit name**.

ImqBoolean setSecurityExitName(const char * *name* = 0);
Sets the **security exit name**. This method returns TRUE if successful.

ImqString securityUserData( ) const ;
Returns the **security user data**.

ImqBoolean setSecurityUserData(const char * *data* = 0);
Sets the **security user data**. This method returns TRUE if successful.

size_t sendExitCount( ) const ;
Returns the **send exit count**.

ImqString sendExitName( );
Returns the first of the **send exit names**, if any. Returns an empty string if the **send exit count** is zero.

ImqBoolean sendExitNames(const size_t *count*, ImqString * *names* [ ]);
Returns copies of the **send exit names** in *names*. Sets any *names* in excess of **send exit count** to null strings. This method returns TRUE if successful.

ImqBoolean setSendExitName(const char * *name* = 0);
Sets the **send exit names** to the single *name*. *name* can be blank or null. Sets the **send exit count** to either 1 or zero. Clears the **send user data**. This method returns TRUE if successful

ImqBoolean setSendExitNames(const size_t *count*, const char * *names* [ ]);
Sets the **send exit names** to *names*. Individual *names* values must not be blank or null. Sets the **send exit count** to *count*. Clears the **send user data**. This method returns TRUE if successful.

ImqBoolean setSendExitNames(const size_t *count*, const ImqString * *names* [ ]);
Sets the **send exit names** to *names*. Individual *names* values must not be blank or null. Sets the **send exit count** to *count*. Clears the **send user data**. This method returns TRUE if successful.

ImqString sendUserData( );
Returns the first of the **send user data** items, if any. , Returns an empty string if the **send exit count** is zero.

ImqBoolean sendUserData(const size_t *count*, ImqString * *data* [ ]);
Returns copies of the **send user data** items in *data*. Sets any *data* in excess of **send exit count** to null strings. This method returns TRUE if successful.

ImqBoolean setSendUserData(const char * *data* = 0);
Sets the **send user data** to the single item *data*. If *data* is not null, **send exit count** must be at least 1. This method returns TRUE if successful.

ImqBoolean setSendUserData(const size_t *count*, const char * *data* [ ]);
Sets the **send user data** to *data*. *count* must be no greater than the **send exit count**. This method returns TRUE if successful.

ImqBoolean setSendUserData(const size_t *count*, const ImqString * *data* [ ]);
Sets the **send user data** to *data*. *count* must be no greater than the **send exit count**. This method returns TRUE if successful.

ImqString sslCipherSpecification( ) const ;
Returns the TLS cipher specification.

ImqBoolean setSslCipherSpecification(const char * *name* = 0);
Sets the TLS cipher specification. This method returns TRUE if successful.

MQLONG sslClientAuthentication( ) const ;
Returns the TLS client authentication type.

ImqBoolean setSslClientAuthentication(const MQLONG *auth* = MQSCA_REQUIRED);
Sets the TLS client authentication type. This method returns TRUE if successful.

ImqString sslPeerName( ) const ;
Returns the TLS peer name.

ImqBoolean setSslPeerName(const char * *name* = 0);
Sets the TLS peer name. This method returns TRUE if successful.

ImqString transactionProgramName( ) const ;
Returns the **transaction program name**.

ImqBoolean setTransactionProgramName(const char * *name* = 0);
Sets the **transaction program name**. This method returns TRUE if successful.

MQLONG transportType( ) const ;
Returns the **transport type**.

ImqBoolean setTransportType(const MQLONG *type* = MQXPT_LU62);
Sets the **transport type**. This method returns TRUE if successful.

ImqString userId( ) const ;
Returns the **user id**.

ImqBoolean setUserId(const char * *id* = 0);
Sets the **user id**. This method returns TRUE if successful.

Reason codes

- MQRC_DATA_LENGTH_ERROR
- MQRC_ITEM_COUNT_ERROR
- MQRC_NULL_POINTER
- MQRC_SOURCE_BUFFER_ERROR

ImqCICSBridgeHeader C++ class

This class encapsulates specific features of the MQCIH data structure.

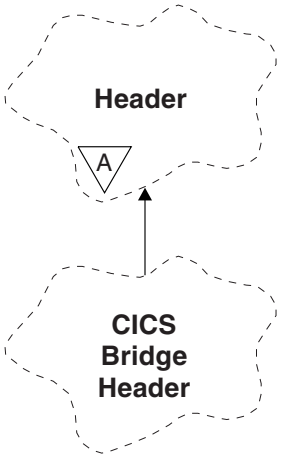


Figure 102. ImqCICSBridgeHeader class

Objects of this class are used by applications that send messages to the CICS bridge through IBM MQ for z/OS.

Object attributes

ADS descriptor
Send/receive ADS descriptor. This is set using MQCADSD_NONE. The initial value is MQCADSD_NONE. The following additional values are possible:

- MQCADSD_NONE
- MQCADSD_SEND
- MQCADSD_RECV
- MQCADSD_MSGFORMAT

attention identifier
AID key. The field must be of length MQ_ATTENTION_ID_LENGTH.

authenticator
RACF password or passticket. The initial value contains blanks, of length MQ_AUTHENTICATOR_LENGTH.

bridge abend code
Bridge abend code, of length MQ_ABEND_CODE_LENGTH. The initial value is four blank characters. The value returned in this field is dependent on the return code. See Table 418 on page 3956 for more details.

bridge cancel code
Bridge abend transaction code. The field is reserved, must contain blanks, and be of length MQ_CANCEL_CODE_LENGTH.

bridge completion code
Completion code, which can contain either the IBM MQ completion code or the CICS EIBRESP value. The field has the initial value of MQCC_OK. The value returned in this field is dependent on the return code. See Table 418 on page 3956 for more details.

bridge error offset
Bridge error offset. The initial value is zero. This attribute is read-only.

bridge reason code
Reason code. This field can contain either the IBM MQ reason or the CICS EIBRESP2 value. The field has the initial value of MQRC_NONE. The value returned in this field is dependent on the return code. See Table 418 on page 3956 for more details.

bridge return code
Return code from the CICS bridge. The initial value is MQCRC_OK.

conversational task
Whether the task can be conversational. The initial value is MQCCT_NO. The following additional values are possible:

- MQCCT_YES
- MQCCT_NO

cursor position
Cursor position. The initial value is zero.

facility keep time
CICS bridge facility release time.

facility like
Terminal emulated attribute. The field must be of length MQ_FACILITY_LIKE_LENGTH.

facility token
BVT token value. The field must be of length MQ_FACILITY_LENGTH. The initial value is MQCFAC_NONE.

function
Function, which can contain either the IBM MQ call name or the CICS EIBFN function. The field has the initial value of MQCFUNC_NONE, with length MQ_FUNCTION_LENGTH. The value returned in this field is dependent on the return code. See Table 418 on page 3956 for more details.

The following additional values are possible when **function** contains an IBM MQ call name:

- MQCFUNC_MQCONN
- MQCFUNC_MQGET
- MQCFUNC_MQINQ
- MQCFUNC_NONE
- MQCFUNC_MQOPEN
- MQCFUNC_PUT
- MQCFUNC_MQPUT1

get wait interval
Wait interval for an MQGET call issued by the CICS bridge task. The initial value is MQCGWI_DEFAULT. The field applies only when **uow control** has the value MQCUOWC_FIRST. The following additional values are possible:

- MQCGWI_DEFAULT
- MQWI_UNLIMITED

link type
Link type. The initial value is MQCLT_PROGRAM. The following additional values are possible:

- MQCLT_PROGRAM
- MQCLT_TRANSACTION

next transaction identifier
ID of the next transaction to attach. The field must be of length MQ_TRANSACTION_ID_LENGTH.

output data length
COMMAREA data length. The initial value is MQCODL_AS_INPUT.

reply-to format
Format name of the reply message. The initial value is MQFMT_NONE with length MQ_FORMAT_LENGTH.

start code
Transaction start code. The field must be of length MQ_START_CODE_LENGTH. The initial value is MQCSC_NONE. The following additional values are possible:

- MQCSC_START
- MQCSC_STARTDATA
- MQCSC_TERMINPUT
- MQCSC_NONE

task end status
Task end status. The initial value is MQCTES_NOSYNC. The following additional values are possible:

- MQCTES_COMMIT
- MQCTES_BACKOUT
- MQCTES_ENDTASK
- MQCTES_NOSYNC

transaction identifier
ID of the transaction to attach. The initial value must contain blanks, and must be of length MQ_TRANSACTION_ID_LENGTH. The field applies only when **uow control** has the value MQCUOWC_FIRST or MQCUOWC_ONLY.

UOW control
UOW control. The initial value is MQCUOWC_ONLY. The following additional values are possible:

- MQCUOWC_FIRST
- MQCUOWC_MIDDLE
- MQCUOWC_LAST
- MQCUOWC_ONLY
- MQCUOWC_COMMIT
- MQCUOWC_BACKOUT
- MQCUOWC_CONTINUE

version
The MQCIH version number. The initial value is MQCIH_VERSION_2. The only other supported value is MQCIH_VERSION_1.

Constructors

ImqCICSBridgeHeader( );
The default constructor.

ImqCICSBridgeHeader(const ImqCICSBridgeHeader & *header*);
The copy constructor.

Overloaded ImqItem methods

virtual ImqBoolean copyOut(ImqMessage & *msg*);
Inserts an MQCIH data structure into the message buffer at the beginning, moving existing message data further along, and sets the message format to MQFMT_CICS.

See the parent class method description for more details.

virtual ImqBoolean pasteIn(ImqMessage & *msg*);
Reads an MQCIH data structure from the message buffer. To be successful, the encoding of the *msg* object must be MQENC_NATIVE. Retrieve messages with MQGMO_CONVERT to MQENC_NATIVE. To be successful, the ImqMessage format must be MQFMT_CICS.

See the parent class method description for more details.

Object methods (public)

void operator = (const ImqCICSBridgeHeader & *header*);
Copies instance data from the *header*, replacing the existing instance data.

MQLONG ADSDescriptor( ) const;
Returns a copy of the **ADS descriptor**.

void setADSDescriptor(const MQLONG *descriptor* = MQCADSD_NONE);
Sets the **ADS descriptor**.

ImqString attentionIdentifier( ) const;
Returns a copy of the **attention identifier**, padded with trailing blanks to length MQ_ATTENTION_ID_LENGTH.

void setAttentionIdentifier(const char * *data* = 0);
Sets the **attention identifier**, padded with trailing blanks to length MQ_ATTENTION_ID_LENGTH. If no *data* is supplied, resets **attention identifier** to the initial value.

ImqString authenticator( ) const;
Returns a copy of the **authenticator**, padded with trailing blanks to length MQ_AUTHENTICATOR_LENGTH.

void setAuthenticator(const char * *data* = 0);
Sets the **authenticator**, padded with trailing blanks to length MQ_AUTHENTICATOR_LENGTH. If no *data* is supplied, resets **authenticator** to the initial value.

ImqString bridgeAbendCode( ) const;
Returns a copy of the **bridge abend code**, padded with trailing blanks to length MQ_ABEND_CODE_LENGTH.

ImqString bridgeCancelCode( ) const;
Returns a copy of the **bridge cancel code**, padded with trailing blanks to length MQ_CANCEL_CODE_LENGTH.

void setBridgeCancelCode(const char * *data* = 0);
Sets the **bridge cancel code**, padded with trailing blanks to length MQ_CANCEL_CODE_LENGTH. If no *data* is supplied, resets the **bridge cancel code** to the initial value.

MQLONG bridgeCompletionCode( ) const;
Returns a copy of the **bridge completion code**.

MQLONG bridgeErrorOffset( ) const ;
Returns a copy of the **bridge error offset**.

MQLONG bridgeReasonCode( ) const;
Returns a copy of the **bridge reason code**.

MQLONG bridgeReturnCode( ) const;
Returns the **bridge return code**.

MQLONG conversationalTask( ) const;
Returns a copy of the **conversational task**.

void setConversationalTask(const MQLONG *task* = MQCCT_NO);
Sets the **conversational task**.

MQLONG cursorPosition( ) const ;
Returns a copy of the **cursor position**.

void setCursorPosition(const MQLONG *position* = 0);
Sets the **cursor position**.

MQLONG facilityKeepTime( ) const;
Returns a copy of the **facility keep time**.

void setFacilityKeepTime(const MQLONG *time* = 0);
Sets the **facility keep time**.

ImqString facilityLike( ) const;
Returns a copy of the **facility like**, padded with trailing blanks to length MQ_FACILITY_LIKE_LENGTH.

void setFacilityLike(const char * *name* = 0);
Sets the **facility like**, padded with trailing blanks to length MQ_FACILITY_LIKE_LENGTH. If no *name* is supplied, resets **facility like** the initial value.

ImqBinary facilityToken( ) const;
Returns a copy of the **facility token**.

ImqBoolean setFacilityToken(const ImqBinary & *token*);
Sets the **facility token**. The **data length** of *token* must be either zero or MQ_FACILITY_LENGTH. It returns TRUE if successful.

void setFacilityToken(const MQBYTE8 *token* = 0);
Sets the **facility token**. *token* can be zero, which is the same as specifying MQCFAC_NONE. If *token* is nonzero it must address MQ_FACILITY_LENGTH bytes of binary data. When using predefined values such as MQCFAC_NONE, you might need to make a cast to ensure a signature match. For example, (MQBYTE *)MQCFAC_NONE.

ImqString function( ) const;
Returns a copy of the **function**, padded with trailing blanks to length MQ_FUNCTION_LENGTH.

MQLONG getWaitInterval( ) const;
Returns a copy of the **get wait interval**.

void setGetWaitInterval(const MQLONG *interval* = MQCGWI_DEFA
Sets the **get wait interval**.

MQLONG linkType( ) const;
Returns a copy of the **link type**.

void setLinkType(const MQLONG *type* = MQCLT_PROGRAM);
Sets the **link type**.

ImqString nextTransactionIdentifier( ) const ;
Returns a copy of the **next transaction identifier** data, padded with trailing blanks to length MQ_TRANSACTION_ID_LENGTH.

MQLONG outputDataLength( ) const;
Returns a copy of the **output data length**.

void setOutputDataLength(const MQLONG *length* = MQCODL_AS_INPUT);
Sets the **output data length**.

ImqString replyToFormat( ) const;
Returns a copy of the **reply-to format** name, padded with trailing blanks to length MQ_FORMAT_LENGTH.

void setReplyToFormat(const char * *name* = 0);
Sets the **reply-to format**, padded with trailing blanks to length MQ_FORMAT_LENGTH. If no *name* is supplied, resets **reply-to format** to the initial value.

ImqString startCode( ) const;
Returns a copy of the **start code**, padded with trailing blanks to length MQ_START_CODE_LENGTH.

void setStartCode(const char * *data* = 0);
Sets the **start code** data, padded with trailing blanks to length MQ_START_CODE_LENGTH. If no *data* is supplied, resets **start code** to the initial value.

MQLONG taskEndStatus( ) const;
Returns a copy of the **task end status**.

ImqString transactionIdentifier( ) const;
Returns a copy of the **transaction identifier** data, padded with trailing blanks to the length MQ_TRANSACTION_ID_LENGTH.

void setTransactionIdentifier(const char * *data* = 0);
Sets the **transaction identifier**, padded with trailing blanks to length MQ_TRANSACTION_ID_LENGTH. If no *data* is supplied, resets **transaction identifier** to the initial value.

MQLONG UOWControl( ) const;
Returns a copy of the **UOW control**.

void setUOWControl(const MQLONG *control* = MQCUOWC_ONLY);
Sets the **UOW control**.

MQLONG version( ) const;
Returns the **version** number.

ImqBoolean setVersion(const MQLONG *version* = MQCIH_VERSION_2);
Sets the **version** number. It returns TRUE if successful.

Object data (protected)

MQLONG *olVersion*
The maximum MQCIH version number that can be accommodated in the storage allocated for *opcih*.

PMQCIH *opcih*
The address of an MQCIH data structure. The amount of storage allocated is indicated by *olVersion*.

Reason codes

- MQRC_BINARY_DATA_LENGTH_ERROR
- MQRC_WRONG_VERSION

Return codes

Table 418. ImqCICSBridgeHeader class return codes

Return Code	Function	CompCode	Reason	Abend Code
MQCRC_OK				
MQCRC_BRIDGE_ERROR			MQFB_CICS	
MQCRC_MQ_API_ERROR	IBM MQ call name	IBM MQ CompCode	IBM MQ Reason	
MQCRC_BRIDGE_TIMEOUT	IBM MQ call name	IBM MQ CompCode	IBM MQ Reason	
MQCRC_CICS_EXEC_ERROR	CICS EIBFN	CICS EIBRESP	CICS EIBRESP2	
MQCRC_SECURITY_ERROR	CICS EIBFN	CICS EIBRESP	CICS EIBRESP2	
MQCRC_PROGRAM_NOT_AVAILABLE	CICS EIBFN	CICS EIBRESP	CICS EIBRESP2	
MQCRC_TRANSID_NOT_AVAILABLE	CICS EIBFN	CICS EIBRESP	CICS EIBRESP2	
MQCRC_BRIDGE_ABEND				CICS ABCODE

Table 418. ImqCICSBridgeHeader class return codes (continued)

Return Code	Function	CompCode	Reason	Abend Code
MQCRC_APPLICATION_ABEND				CICS ABCODE

ImqDeadLetterHeader C++ class

This class encapsulates features of the MQDLH data structure.

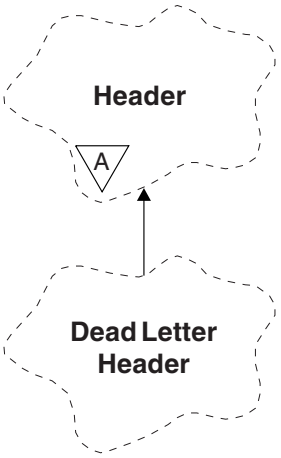


Figure 103. ImqDeadLetterHeader class

Objects of this class are typically used by an application that encounters an message that cannot be processed. A new message comprising a dead-letter header and the message content is placed on the dead-letter queue, and the message is discarded.

Object attributes

dead-letter reason code
The reason the message arrived on the dead-letter queue. The initial value is MQRC_NONE.

destination queue manager name
The name of the original destination queue manager. The name is a string of length MQ_Q_MGR_NAME_LENGTH. Its initial value is null.

destination queue name
The name of the original destination queue. The name is a string of length MQ_Q_NAME_LENGTH. Its initial value is null.

put application name
The name of the application that put the message on the dead-letter queue. The name is a string of length MQ_PUT_APPL_NAME_LENGTH. Its initial value is null.

put application type
The type of application that put the message on the dead-letter queue. The initial value is zero.

put date
The date when the message was put on the dead-letter queue. The date is a string of length MQ_PUT_DATE_LENGTH. Its initial value is a null string.

put time
The time when the message was put on the dead-letter queue. The time is a string of length MQ_PUT_TIME_LENGTH. Its initial value is a null string.

Constructors

ImqDeadLetterHeader( );
The default constructor.

ImqDeadLetterHeader(const ImqDeadLetterHeader & *header*);
The copy constructor.

Overloaded ImqItem methods

virtual ImqBoolean copyOut (ImqMessage & *msg*);
Inserts an MQDLH data structure into the message buffer at the beginning, moving existing message data further along. Sets the *msg* format to MQFMT_DEAD_LETTER_HEADER.

See the ImqHeader class method description on page "ImqHeader C++ class" on page 3965 for further details.

virtual ImqBoolean pasteIn (ImqMessage & *msg*);
Reads an MQDLH data structure from the message buffer.

To be successful, the ImqMessage format must be MQFMT_DEAD_LETTER_HEADER.

See the ImqHeader class method description on page "ImqHeader C++ class" on page 3965 for further details.

Object methods (public)

void operator = (const ImqDeadLetterHeader & *header*);
Copies instance data is copied from *header*, replacing the existing instance data.

MQLONG deadLetterReasonCode ( ) const ;
Returns the dead-letter reason code.

void setDeadLetterReasonCode (const MQLONG *reason*);
Sets the dead-letter reason code.

ImqString destinationQueueManagerName ( ) const ;
Returns the destination queue manager name, stripped of any trailing blanks.

void setDestinationQueueManagerName (const char * *name*);
Sets the destination queue manager name. Truncates data longer than MQ_Q_MGR_NAME_LENGTH (48 characters).

ImqString destinationQueueName ( ) const ;
Returns a copy of the destination queue name, stripped of any trailing blanks.

void setDestinationQueueName (const char * *name*);
Sets the destination queue name. Truncates data longer than MQ_Q_NAME_LENGTH (48 characters).

ImqString putApplicationName ( ) const ;
Returns a copy of the put application name, stripped of any trailing blanks.

void setPutApplicationName (const char * *name* = 0);
Sets the put application name. Truncates data longer than MQ_PUT_APPL_NAME_LENGTH (28 characters).

MQLONG putApplicationType ( ) const ;
Returns the put application type.

void setPutApplicationType (const MQLONG *type* = MQAT_NO_CONTEXT);
Sets the put application type.

ImqString putDate ( ) const ;
Returns a copy of the put date, stripped of any trailing blanks.

void setPutDate (const char * *date* = 0);
Sets the put date. Truncates data longer than MQ_PUT_DATE_LENGTH (8 characters).

ImqString putTime ( ) const ;
Returns a copy of the put time, stripped of any trailing blanks.

void setPutTime (const char * *time* = 0);
Sets the put time. Truncates data longer than MQ_PUT_TIME_LENGTH (8 characters).

Object data (protected)

MQDLH *omqdlh*
The MQDLH data structure.

Reason codes

- MQRC_INCONSISTENT_FORMAT
- MQRC_STRUC_ID_ERROR
- MQRC_ENCODING_ERROR

ImqDistributionList C++ class

This class encapsulates a dynamic distribution list that references one or more queues for the purpose of sending a message or messages to multiple destinations.

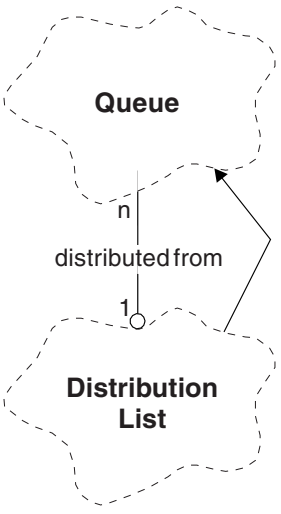


Figure 104. ImqDistributionList class

- "Object attributes" on page 3960
- "Constructors" on page 3960

- "Object methods (public)"
- "Object methods (protected)"

Object attributes

first distributed queue

The first of one or more objects of class , in no particular order, in which the **distribution list reference** addresses this object.

Initially there are no such objects. To open an ImqDistributionList successfully, there must be at least one such object.

Note: When an ImqDistributionList object is opened, any open objects that reference it are automatically closed.

Constructors

ImqDistributionList( );

The default constructor.

ImqDistributionList(const ImqDistributionList & *list*);

The copy constructor.

Object methods (public)

void operator = (const ImqDistributionList & *list*);

All objects that reference **this** object are dereferenced before copying. No objects will reference **this** object after the invocation of this method.

*** firstDistributedQueue ( ) const ;**

Returns the **first distributed queue**.

Object methods (protected)

void setFirstDistributedQueue (* *queue* = 0);

Sets the **first distributed queue**.

ImqError C++ class

This abstract class provides information on errors associated with an object.

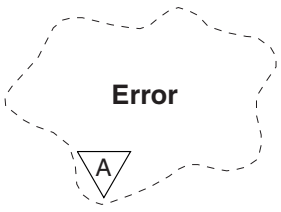


Figure 105. ImqError class

- "Object attributes" on page 3961
- "Constructors" on page 3961
- "Object methods (public)" on page 3961
- "Object methods (protected)" on page 3961
- "Reason codes" on page 3961

Object attributes

completion code
The most recent completion code. The initial value is zero. The following additional values are possible:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

reason code
The most recent reason code. The initial value is zero.

Constructors

ImqError( );
The default constructor.

ImqError(const ImqError & *error*);
The copy constructor.

Object methods (public)

void operator = (const ImqError & *error*);
Copies instance data from *error*, replacing the existing instance data.

void clearErrorCodes ( );
Sets the **completion code** and **reason code** both to zero.

MQLONG completionCode ( ) const ;
Returns the **completion code**.

MQLONG reasonCode ( ) const ;
Returns the **reason code**.

Object methods (protected)

ImqBoolean checkReadPointer (const void * *pointer*, const size_t *length*);
Verifies that the combination of pointer and length is valid for read-only access, and returns TRUE if successful.

ImqBoolean checkWritePointer (const void * *pointer*, const size_t *length*);
Verifies that the combination of pointer and length is valid for read-write access, and returns TRUE if successful.

void setCompletionCode (const MQLONG *code* = 0);
Sets the **completion code**.

void setReasonCode (const MQLONG *code* = 0);
Sets the **reason code**.

Reason codes

- MQRC_BUFFER_ERROR

ImqGetMessageOptions C++ class

This class encapsulates the MQGMO data structure

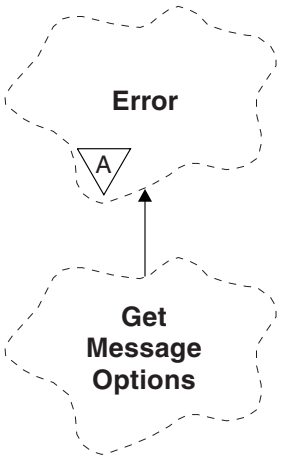


Figure 106. ImqGetMessageOptions class

Object attributes

group status

Status of a message for a group of messages. The initial value is MQGS_NOT_IN_GROUP. The following additional values are possible:

- MQGS_MSG_IN_GROUP
- MQGS_LAST_MSG_IN_GROUP

match options

Options for selecting incoming messages. The initial value is MQMO_MATCH_MSG_ID | MQMO_MATCH_CORREL_ID. The following additional values are possible:

- MQMO_GROUP_ID
- MQMO_MATCH_MSG_SEQ_NUMBER
- MQMO_MATCH_OFFSET
- MQMO_MSG_TOKEN
- MQMO_NONE

message token

Message token. A binary value (MQBYTE16) of length MQ_MSG_TOKEN_LENGTH. The initial value is MQMTOK_NONE.

options

Options applicable to a message. The initial value is MQGMO_NO_WAIT. The following additional values are possible:

- MQGMO_WAIT
- MQGMO_SYNCPOINT

- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_NO_SYNCPOINT
- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_NEXT
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_MSG_UNDER_CURSOR
- MQGMO_LOCK
- MQGMO_UNLOCK
- MQGMO_ACCEPT_TRUNCATED_MSG
- MQGMO_SET_SIGNAL
- MQGMO_FAIL_IF_QUIESCING
- MQGMO_CONVERT
- MQGMO_LOGICAL_ORDER
- MQGMO_COMPLETE_MSG
- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE
- MQGMO_NONE

resolved queue name
Resolved queue name. This attribute is read-only. Names are never longer than 48 characters and can be padded to that length with nulls. The initial value is a null string.

returned length
Returned length. The initial value is MQRL_UNDEFINED. This attribute is read-only.

segmentation
The ability to segment a message. The initial value is MQSEG_INHIBITED. The additional value, MQSEG_ALLOWED, is possible.

segment status
The segmentation status of a message. The initial value is MQSS_NOT_A_SEGMENT. The following additional values are possible:

- MQSS_SEGMENT
- MQSS_LAST_SEGMENT

syncpoint participation
TRUE when messages are retrieved under syncpoint control.

wait interval
The length of time that the class get method pauses while waiting for a suitable message to arrive, if one is not already available. The initial value is zero, which effects an indefinite wait. The additional value, MQWI_UNLIMITED, is possible. This attribute is ignored unless the options include MQGMO_WAIT.

Constructors

ImqGetMessageOptions( );
The default constructor.

ImqGetMessageOptions(const ImqGetMessageOptions & *gmo*);
The copy constructor.

Object methods (public)

void operator = (const ImqGetMessageOptions & *gmo*);
Copies instance data from *gmo*, replacing the existing instance data.

MQCHAR groupStatus ( ) const ;
Returns the group status.

void setGroupStatus (const MQCHAR *status*);
Sets the group status.

MQLONG matchOptions ( ) const ;
Returns the match options.

void setMatchOptions (const MQLONG *options*);
Sets the match options.

ImqBinary messageToken( ) const;
Returns the message token.

ImqBoolean setMessageToken(const ImqBinary & *token*);
Sets the message token. The data length of *token* must be either zero or MQ_MSG_TOKEN_LENGTH. This method returns TRUE if successful.

void setMessageToken(const MQBYTE16 *token* = 0);
Sets the message token. *token* can be zero, which is the same as specifying MQMTOK_NONE. If *token* is nonzero, then it must address MQ_MSG_TOKEN_LENGTH bytes of binary data.

When using predefined values, such as MQMTOK_NONE, you might not need to make a cast to ensure a signature match, for example (MQBYTE *)MQMTOK_NONE.

MQLONG options ( ) const ;
Returns the options.

void setOptions (const MQLONG *options*);
Sets the options, including the syncpoint participation value.

ImqString resolvedQueueName ( ) const ;
Returns a copy of the resolved queue name.

MQLONG returnedLength( ) const;
Returns the returned length.

MQCHAR segmentation ( ) const ;
Returns the segmentation.

void setSegmentation (const MQCHAR *value*);
Sets the segmentation.

MQCHAR segmentStatus ( ) const ;
Returns the segment status.

void setSegmentStatus (const MQCHAR *status*);
Sets the segment status.

ImqBoolean syncPointParticipation ( ) const ;
Returns the syncpoint participation value, which is TRUE if the options include either MQGMO_SYNCPOINT or MQGMO_SYNCPOINT_IF_PERSISTENT.

void setSyncPointParticipation (const ImqBoolean *sync*);
Sets the syncpoint participation value. If *sync* is TRUE, alters the options to include MQGMO_SYNCPOINT, and to exclude both MQGMO_NO_SYNCPOINT and MQGMO_SYNCPOINT_IF_PERSISTENT. If *sync* is FALSE, alters the options to include MQGMO_NO_SYNCPOINT, and to exclude both MQGMO_SYNCPOINT and MQGMO_SYNCPOINT_IF_PERSISTENT.

MQLONG waitInterval ( ) const ;
Returns the wait interval.

void setWaitInterval (const MQLONG *interval*);
Sets the wait interval.

Object methods (protected)

static void setVersionSupported (const MQLONG);
Sets the MQGMO version. Defaults to MQGMO_VERSION_3.

Object data (protected)

MQGMO *omqgmo*
An MQGMO Version 2 data structure. Access MQGMO fields supported for MQGMO_VERSION_2 only.

PMQGMO *opgmo*
The address of an MQGMO data structure. The version number for this address is indicated in *olVersion*. Inspect the version number before accessing MQGMO fields, to ensure that they are present.

MQLONG *olVersion*
The version number of the MQGMO data structure addressed by *opgmo*.

Reason codes

- MQRC_BINARY_DATA_LENGTH_ERROR

ImqHeader C++ class

This abstract class encapsulates common features of the MQDLH data structure.

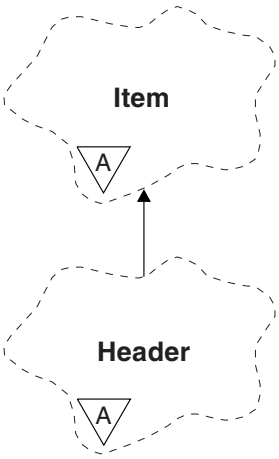


Figure 107. ImqHeader class

- "Object attributes"
- "Constructors" on page 3966
- "Object methods (public)" on page 3966

Object attributes

character set
The original coded character set identifier. Initially MQCCSI_Q_MGR.

encoding
The original encoding. Initially MQENC_NATIVE.

format
The original format. Initially MQFMT_NONE.

header flags
The initial values are:

- Zero for objects of the ImqDeadLetterHeader class
- MQIIH_NONE for objects of the ImqIMSBridgeHeader class
- MQRMHF_LAST for objects of the ImqReferenceHeader class
- MQCIH_NONE for objects of the ImqCICSBridgeHeader class
- MQWIH_NONE for objects of the ImqWorkHeader class

Constructors

ImqHeader( );
The default constructor.

ImqHeader(const ImqHeader & *header*);
The copy constructor.

Object methods (public)

void operator = (const ImqHeader & *header*);
Copies instance data from *header*, replacing the existing instance data.

virtual MQLONG characterSet ( ) const ;
Returns the **character set**.

virtual void setCharacterSet (const MQLONG *ccsid* = MQCCSI_Q_MGR);
Sets the **character set**.

virtual MQLONG encoding ( ) const ;
Returns the **encoding**.

virtual void setEncoding (const MQLONG *encoding* = MQENC_NATIVE);
Sets the **encoding**.

virtual ImqString format ( ) const ;
Returns a copy of the **format**, including trailing blanks.

virtual void setFormat (const char * *name* = 0);
Sets the **format**, padded to 8 characters with trailing blanks.

virtual MQLONG headerFlags ( ) const ;
Returns the **header flags**.

virtual void setHeaderFlags (const MQLONG *flags* = 0);
Sets the **header flags**.

ImqIMSBridgeHeader C++ class

This class encapsulates features of the MQIIH data structure.

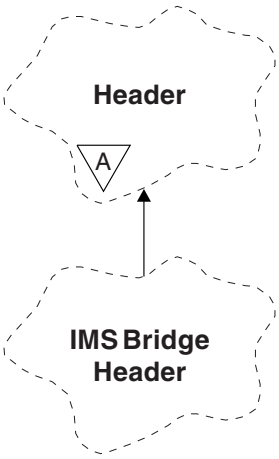


Figure 108. ImqIMSBridgeHeader class

Objects of this class are used by applications that send messages to the IMS bridge through IBM MQ for z/OS.

Note: The ImqHeader character set and encoding must have default values and must not be set to any other values.

Object attributes

authenticator
RACF password or passticket, of length MQ_AUTHENTICATOR_LENGTH. The initial value is MQIAUT_NONE.

commit mode
Commit mode. See the *OTMA User's Guide* for more information about IMS commit modes. The initial value is MQICM_COMMIT_THEN_SEND. The additional value, MQICM_SEND_THEN_COMMIT, is possible.

logical terminal override
Logical terminal override, of length MQ_LTERM_OVERRIDE_LENGTH. The initial value is a null string.

message format services map name
MFS map name, of length MQ_MFS_MAP_NAME_LENGTH. The initial value is a null string.

reply-to format
Format of any reply, of length MQ_FORMAT_LENGTH. The initial value is MQFMT_NONE.

security scope
Scope of IMS security processing. The initial value is MQISS_CHECK. The additional value, MQISS_FULL, is possible.

transaction instance id
Transaction instance identity, a binary (MQBYTE16) value of length MQ_TRAN_INSTANCE_ID_LENGTH. The initial value is MQITII_NONE.

transaction state
State of the IMS conversation. The initial value is MQITS_NOT_IN_CONVERSATION. The additional value, MQITS_IN_CONVERSATION, is possible.

Constructors

ImqIMSBridgeHeader( );
The default constructor.

ImqIMSBridgeHeader(const ImqIMSBridgeHeader & *header*);
The copy constructor.

Overloaded ImqItem methods

virtual ImqBoolean copyOut (ImqMessage & *msg*);
Inserts an MQIIH data structure into the message buffer at the beginning, moving existing message data further along. Sets the *msg* format to MQFMT_IMS.

See the parent class method description for further details.

virtual ImqBoolean pasteIn (ImqMessage & *msg*);
Reads an MQIIH data structure from the message buffer.

To be successful, the encoding of the *msg* object must be MQENC_NATIVE. Retrieve messages with MQGMO_CONVERT to MQENC_NATIVE.

To be successful, the ImqMessage format must be MQFMT_IMS.

See the parent class method description for further details.

Object methods (public)

void operator = (const ImqIMSBridgeHeader & *header*);
Copies instance data from *header*, replacing the existing instance data.

ImqString authenticator ( ) const ;
Returns a copy of the authenticator, padded with trailing blanks to length MQ_AUTHENTICATOR_LENGTH.

void setAuthenticator (const char * *name*);
Sets the authenticator.

MQCHAR commitMode ( ) const ;
Returns the commit mode.

void setCommitMode (const MQCHAR *mode*);
Sets the commit mode.

ImqString logicalTerminalOverride ( ) const ;
Returns a copy of the logical terminal override.

void setLogicalTerminalOverride (const char * *override*);
Sets the logical terminal override.

ImqString messageFormatServicesMapName ( ) const ;
Returns a copy of the message format services map name.

void setMessageFormatServicesMapName (const char * *name*);
Sets the message format services map name.

ImqString replyToFormat ( ) const ;
Returns a copy of the reply-to format, padded with trailing blanks to length MQ_FORMAT_LENGTH.

void setReplyToFormat (const char * *format*);
Sets the reply-to format, padded with trailing blanks to length MQ_FORMAT_LENGTH.

MQCHAR securityScope ( ) const ;
Returns the security scope.

void setSecurityScope (const MQCHAR *scope*);
Sets the security scope.

ImqBinary transactionInstanceId ( ) const ;
Returns a copy of the transaction instance id.

ImqBoolean setTransactionInstanceId (const ImqBinary & *id*);
Sets the transaction instance id. The data length of *token* must be either zero or MQ_TRAN_INSTANCE_ID_LENGTH. This method returns TRUE if successful.

void setTransactionInstanceId (const MQBYTE16 *id* = 0);
Sets the transaction instance id. *id* can be zero, which is the same as specifying MQITII_NONE. If *id* is nonzero, it must address MQ_TRAN_INSTANCE_ID_LENGTH bytes of binary data. When using predefined values such as MQITII_NONE, you might need to make a cast to ensure a signature match, for example (MQBYTE *)MQITII_NONE.

MQCHAR transactionState ( ) const ;
Returns the transaction state.

void setTransactionState (const MQCHAR *state*);
Sets the transaction state.

Object data (protected)

MQIIH *omqiih*
The MQIIH data structure.

Reason codes

- MQRC_BINARY_DATA_LENGTH_ERROR
- MQRC_INCONSISTENT_FORMAT
- MQRC_ENCODING_ERROR
- MQRC_STRUC_ID_ERROR

ImqItem C++ class

This abstract class represents an item, perhaps one of several, within a message.

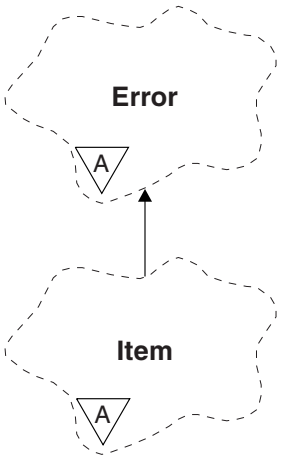


Figure 109. ImqItem class

Items are concatenated together in a message buffer. Each specialization is associated with a particular data structure that begins with a structure ID.

Polymorphic methods in this abstract class allow items to be copied to and from messages. The ImqMessage class **readItem** and **writeItem** methods provide another style of invoking these polymorphic methods that is more natural for application programs.

Object attributes

structure id
A string of four characters at the beginning of the data structure. This attribute is read-only. Consider this attribute for derived classes. It is not included automatically.

Constructors

ImqItem( );
The default constructor.

ImqItem(const ImqItem & ***item*** **);**
The copy constructor.

Class methods (public)

static ImqBoolean structureIdIs (const char * ***structure-id-to-test*****, const ImqMessage &** ***msg*** **);**
Returns TRUE if the **structure id** of the next ImqItem in the incoming *msg* is the same as *structure-id-to-test*. The next item is identified as that part of the message buffer currently addressed by the ImqCache **data pointer**. This method relies on the **structure id** and therefore is not guaranteed to work for all ImqItem derived classes.

Object methods (public)

void operator = (const ImqItem & *item*);
Copies instance data from *item*, replacing the existing instance data.

virtual ImqBoolean copyOut (ImqMessage & *msg*) = 0 ;
Writes this object as the next item in an outgoing message buffer, appending it to any existing items. If the write operation is successful, increases the ImqCache **data length**. This method returns TRUE if successful.

Override this method to work with a specific subclass.

virtual ImqBoolean pasteIn (ImqMessage & *msg*) = 0 ;
Reads this object *destructively* from the incoming message buffer. The read is destructive in that the ImqCache **data pointer** is moved on. However, the buffer content remains the same, so data can be re-read by resetting the ImqCache **data pointer**.

The (sub)class of this object must be consistent with the **structure id** found next in the message buffer of the *msg* object.

The **encoding** of the *msg* object should be MQENC_NATIVE. It is recommended that messages be retrieved with the ImqMessage **encoding** set to MQENC_NATIVE, and with the ImqGetMessageOptions **options** including MQGMO_CONVERT.

If the read operation is successful, the ImqCache **data length** is reduced. This method returns TRUE if successful.

Override this method to work with a specific subclass.

Reason codes

- MQRC_ENCODING_ERROR
- MQRC_STRUC_ID_ERROR
- MQRC_INCONSISTENT_FORMAT
- MQRC_INSUFFICIENT_BUFFER
- MQRC_INSUFFICIENT_DATA

ImqMessage C++ class

This class encapsulates an MQMD data structure and also handles the construction and reconstruction of message data.

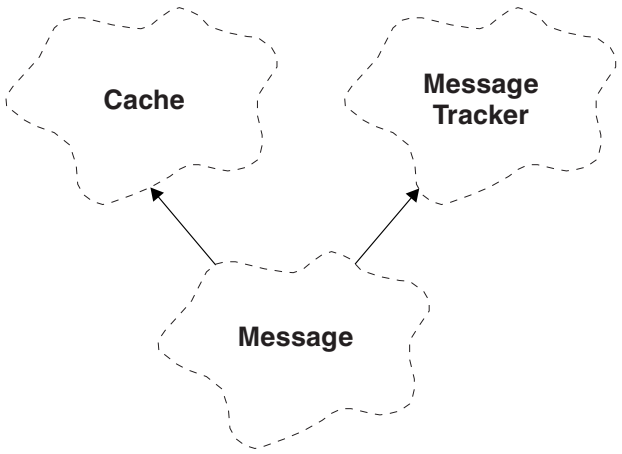


Figure 110. ImqMessage class

- "Object attributes" on page 3972

Object attributes

application ID data
Identity information associated with a message. The initial value is a null string.

application origin data
Origin information associated with a message. The initial value is a null string.

backout count
The number of times that a message has been tentatively retrieved and subsequently backed out. The initial value is zero. This attribute is read-only.

character set
Coded Character Set Id. The initial value is MQCCSI_Q_MGR. The following additional values are possible:

- MQCCSI_INHERIT
- MQCCSI_EMBEDDED

You can also use a Coded Character Set Id of your choice. For information about this, see "Code page conversion" on page 2941.

encoding
The machine encoding of the message data. The initial value is MQENC_NATIVE.

expiry A time-dependent quantity that controls how long IBM MQ retains an unretrieved message before discarding it. The initial value is MQEI_UNLIMITED.

format
The name of the format (template) that describes the layout of data in the buffer. Names longer than eight characters are truncated to eight characters. Names are always padded with blanks to eight characters. The initial constant value is MQFMT_NONE. The following additional constants are possible:

- MQFMT_ADMIN
- MQFMT_CICS
- MQFMT_COMMAND_1
- MQFMT_COMMAND_2
- MQFMT_DEAD_LETTER_HEADER
- MQFMT_DIST_HEADER
- MQFMT_EVENT
- MQFMT_IMS
- MQFMT_IMS_VAR_STRING
- MQFMT_MD_EXTENSION
- MQFMT_PCF
- MQFMT_REF_MSG_HEADER
- MQFMT_RF_HEADER
- MQFMT_STRING
- MQFMT_TRIGGER
- MQFMT_WORK_INFO_HEADER
- MQFMT_XMIT_Q_HEADER

You can also use an application-specific string of your choice. For more information about this, see the "Format (MQCHAR8)" on page 2404 field of the message descriptor (MQMD).

message flags
Segmentation control information. The initial value is MQMF_SEGMENTATION_INHIBITED. The following additional values are possible:

- MQMF_SEGMENTATION_ALLOWED
- MQMF_MSG_IN_GROUP
- MQMF_LAST_MSG_IN_GROUP
- MQMF_SEGMENT
- MQMF_LAST_SEGMENT
- MQMF_NONE

message type
The broad categorization of a message. The initial value is MQMT_DATAGRAM. The following additional values are possible:

- MQMT_SYSTEM_FIRST
- MQMT_SYSTEM_LAST
- MQMT_DATAGRAM
- MQMT_REQUEST
- MQMT_REPLY
- MQMT_REPORT
- MQMT_APPL_FIRST
- MQMT_APPL_LAST

You can also use an application-specific value of your choice. For more information about this, see the "MsgType (MQLONG)" on page 2416 field of the message descriptor (MQMD).

offset Offset information. The initial value is zero.

original length
The original length of a segmented message. The initial value is MQOL_UNDEFINED.

persistence
Indicates that the message is important and must at all times be backed up using persistent storage. This option implies a performance penalty. The initial value is MQPER_PERSISTENCE_AS_Q_DEF. The following additional values are possible:

- MQPER_PERSISTENT
- MQPER_NOT_PERSISTENT

priority
The relative priority for transmission and delivery. Messages of the same priority are usually delivered in the same sequence as they were supplied (although there are several criteria that must be satisfied to guarantee this). The initial value is MQPRI_PRIORITY_AS_Q_DEF.

property validation
Specifies whether validation of properties should take place when a property of the message is set. The initial value is **MQCMHO_DEFAULT_VALIDATION**. The following additional values are possible:

- MQCMHO_VALIDATE
- MQCMHO_NO_VALIDATION

The following methods act on **property validation**:

MQLONG propertyValidation( ) const ;
Returns the **property validation** option.

void setPropertyValidation(const MQLONG *option* **);**
Sets the **property validation** option.

put application name
The name of the application that put a message. The initial value is a null string.

put application type
The type of application that put a message. The initial value is MQAT_NO_CONTEXT. The following additional values are possible:

- MQAT_AIX
- MQAT_CICS
- MQAT_CICS_BRIDGE
- MQAT_DOS
- MQAT_IMS
- MQAT_IMS_BRIDGE
- MQAT_MVS
- MQAT_NOTES_AGENT
- MQAT_OS2
- MQAT_OS390
- MQAT_OS400
- MQAT_QMGR
- MQAT_UNIX
- MQAT_WINDOWS
- MQAT_WINDOWS_NT
- MQAT_XCF
- MQAT_DEFAULT
- MQAT_UNKNOWN
- MQAT_USER_FIRST
- MQAT_USER_LAST

You can also use an application-specific string of your choice. For more information about this, see the "PutApplType (MQLONG)" on page 2422 field of the message descriptor (MQMD).

put date
The date on which a message was put. The initial value is a null string.

put time
The time at which a message was put. The initial value is a null string.

reply-to queue manager name
The name of the queue manager to which any reply should be sent. The initial value is a null string.

reply-to queue name
The name of the queue to which any reply should be sent. The initial value is a null string.

report Feedback information associated with a message. The initial value is MQRO_NONE. The following additional values are possible:

- MQRO_EXCEPTION
- MQRO_EXCEPTION_WITH_DATA
- MQRO_EXCEPTION_WITH_FULL_DATA *
- MQRO_EXPIRATION
- MQRO_EXPIRATION_WITH_DATA

- MQRO_EXPIRATION_WITH_FULL_DATA *
- MQRO_COA
- MQRO_COA_WITH_DATA
- MQRO_COA_WITH_FULL_DATA *
- MQRO_COD
- MQRO_COD_WITH_DATA
- MQRO_COD_WITH_FULL_DATA *
- MQRO_PAN
- MQRO_NAN
- MQRO_NEW_MSG_ID
- MQRO_NEW_CORREL_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID
- MQRO_DEAD_LETTER_Q
- MQRO_DISCARD_MSG

where * indicates values that are not supported on IBM MQ for z/OS.

sequence number
Sequence information identifying a message within a group. The initial value is one.

total message length
The number of bytes that were available during the most recent attempt to read a message. This number will be greater than the ImqCache **message length** if the last message was truncated, or if the last message was not read because truncation would have occurred. This attribute is read-only. The initial value is zero.

This attribute can be useful in any situation involving truncated messages.

user id
A user identity associated with a message. The initial value is a null string.

Constructors

ImqMessage( );
The default constructor.

ImqMessage(const ImqMessage & *msg*);
The copy constructor. See the **operator =** method for details.

Object methods (public)

void operator = (const ImqMessage & *msg*);
Copies the MQMD and message data from *msg*. If a buffer has been supplied by the user for this object, the amount of data copied is restricted to the available buffer size. Otherwise, the system ensures that a buffer of adequate size is made available for the copied data.

ImqString applicationIdData ( ) const ;
Returns a copy of the **application ID data**.

void setApplicationIdData (const char * *data* = 0);
Sets the **application ID data**.

ImqString applicationOriginData ( ) const ;
Returns a copy of the **application origin data**.

void setApplicationOriginData (const char * *data* = 0);
Sets the **application origin data**.

MQLONG backoutCount ( ) const ;
Returns the **backout count**.

MQLONG characterSet ( ) const ;
Returns the **character set**.

void setCharacterSet (const MQLONG *ccsid* **= MQCCSI_Q_MGR);**
Sets the **character set**.

MQLONG encoding ( ) const ;
Returns the **encoding**.

void setEncoding (const MQLONG *encoding* **= MQENC_NATIVE);**
Sets the **encoding**.

MQLONG expiry ( ) const ;
Returns the **expiry**.

void setExpiry (const MQLONG *expiry* **);**
Sets the **expiry**.

ImqString format ( ) const ;
Returns a copy of the **format**, including trailing blanks.

ImqBoolean formatIs (const char * *format-to-test* **) const ;**
Returns TRUE if the **format** is the same as *format-to-test*.

void setFormat (const char * *name* **= 0);**
Sets the **format**, padded to eight characters with trailing blanks.

MQLONG messageFlags ( ) const ;
Returns the **message flags**.

void setMessageFlags (const MQLONG *flags* **);**
Sets the **message flags**.

MQLONG messageType ( ) const ;
Returns the **message type**.

void setMessageType (const MQLONG *type* **);**
Sets the **message type**.

MQLONG offset ( ) const ;
Returns the **offset**.

void setOffset (const MQLONG *offset* **);**
Sets the **offset**.

MQLONG originalLength ( ) const ;
Returns the **original length**.

void setOriginalLength (const MQLONG *length* **);**
Sets the **original length**.

MQLONG persistence ( ) const ;
Returns the **persistence**.

void setPersistence (const MQLONG *persistence* **);**
Sets the **persistence**.

MQLONG priority ( ) const ;
Returns the **priority**.

void setPriority (const MQLONG *priority* **);**
Sets the **priority**.

ImqString putApplicationName ( ) const ;
Returns a copy of the **put application name**.

void setPutApplicationName (const char * *name* = 0);
Sets the **put application name**.

MQLONG putApplicationType ( ) const ;
Returns the **put application type**.

void setPutApplicationType (const MQLONG *type* = MQAT_NO_CONTEXT);
Sets the **put application type**.

ImqString putDate ( ) const ;
Returns a copy of the **put date**.

void setPutDate (const char * *date* = 0);
Sets the **put date**.

ImqString putTime ( ) const ;
Returns a copy of the **put time**.

void setPutTime (const char * *time* = 0);
Sets the **put time**.

ImqBoolean readItem (ImqItem & *item*);
Reads into the *item* object from the message buffer, using the ImqItem **pasteIn** method. It returns TRUE if successful.

ImqString replyToQueueManagerName ( ) const ;
Returns a copy of the **reply-to queue manager name**.

void setReplyToQueueManagerName (const char * *name* = 0);
Sets the **reply-to queue manager name**.

ImqString replyToQueueName ( ) const ;
Returns a copy of the **reply-to queue name**.

void setReplyToQueueName (const char * *name* = 0);
Sets the **reply-to queue name**.

MQLONG report ( ) const ;
Returns the **report**.

void setReport (const MQLONG *report*);
Sets the **report**.

MQLONG sequenceNumber ( ) const ;
Returns the **sequence number**.

void setSequenceNumber (const MQLONG *number*);
Sets the **sequence number**.

size_t totalMessageLength ( ) const ;
Returns the **total message length**.

ImqString userId ( ) const ;
Returns a copy of the **user id**.

void setUserId (const char * *id* = 0);
Sets the **user id**.

ImqBoolean writeItem (ImqItem & *item*);
Writes from the *item* object into the message buffer, using the ImqItem **copyOut** method. Writing can take the form of insertion, replacement, or an append: this depends on the class of the *item* object. This method returns TRUE if successful.

Object methods (protected)

static void setVersionSupported (const MQLONG);
Sets the **MQMD version**. Defaults to **MQMD_VERSION_2**.

Object data (protected)

z/OS **MQMD1** *omqmd*
The MQMD data structure on z/OS.

Multi **MQMD2** *omqmd*
The MQMD data structure on Multiplatforms.

ImqMessageTracker C++ class

This class encapsulates those attributes of an ImqMessage or ImqQueue object that can be associated with either object.

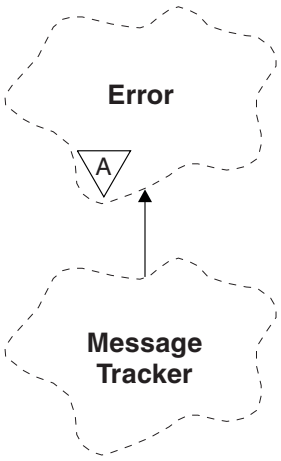


Figure 111. ImqMessageTracker class

This class relates to the MQI calls listed in "ImqMessageTracker cross-reference" on page 3929.

Object attributes

accounting token
A binary value (MQBYTE32) of length MQ_ACCOUNTING_TOKEN_LENGTH. The initial value is MQACT_NONE.

correlation id
A binary value (MQBYTE24) of length MQ_CORREL_ID_LENGTH that you assign to correlate messages. The initial value is MQCI_NONE. The additional value, MQCI_NEW_SESSION, is possible.

feedback
Feedback information to be sent with a message. The initial value is MQFB_NONE. The following additional values are possible:

- MQFB_SYSTEM_FIRST
- MQFB_SYSTEM_LAST

- MQFB_APPL_FIRST
- MQFB_APPL_LAST
- MQFB_COA
- MQFB_COD
- MQFB_EXPIRATION
- MQFB_PAN
- MQFB_NAN
- MQFB_QUIT
- MQFB_DATA_LENGTH_ZERO
- MQFB_DATA_LENGTH_NEGATIVE
- MQFB_DATA_LENGTH_TOO_BIG
- MQFB_BUFFER_OVERFLOW
- MQFB_LENGTH_OFF_BY_ONE
- MQFB_IIH_ERROR
- MQFB_NOT_AUTHORIZED_FOR_IMS
- MQFB_IMS_ERROR
- MQFB_IMS_FIRST
- MQFB_IMS_LAST
- MQFB_CICS_APPL_ABENDED
- MQFB_CICS_APPL_NOT_STARTED
- MQFB_CICS_BRIDGE_FAILURE
- MQFB_CICS_CCSID_ERROR
- MQFB_CICS_CIH_ERROR
- MQFB_CICS_COMMAREA_ERROR
- MQFB_CICS_CORREL_ID_ERROR
- MQFB_CICS_DLQ_ERROR
- MQFB_CICS_ENCODING_ERROR
- MQFB_CICS_INTERNAL_ERROR
- MQFB_CICS_NOT_AUTHORIZED
- MQFB_CICS_UOW_BACKED_OUT
- MQFB_CICS_UOW_ERROR

You can also use an application-specific string of your choice. For more information about this, see the "Feedback (MQLONG)" on page 2400 field of the message descriptor (MQMD).

group id
A binary value (MQBYTE24) of length MQ_GROUP_ID_LENGTH unique within a queue. The initial value is MQGI_NONE.

message id
A binary value (MQBYTE24) of length MQ_MSG_ID_LENGTH unique within a queue. The initial value is MQMI_NONE.

Constructors

ImqMessageTracker( );
The default constructor.

ImqMessageTracker(const ImqMessageTracker & *tracker*);
The copy constructor. See the **operator =** method for details.

Object methods (public)

void operator = (const ImqMessageTracker & *tracker*);
Copies instance data from *tracker*, replacing the existing instance data.

ImqBinary accountingToken ( ) const ;
Returns a copy of the **accounting token**.

ImqBoolean setAccountingToken (const ImqBinary & *token*);
Sets the **accounting token**. The **data length** of *token* must be either zero or MQ_ACCOUNTING_TOKEN_LENGTH. This method returns TRUE if successful.

void setAccountingToken (const MQBYTE32 *token* = 0);
Sets the **accounting token**. *token* can be zero, which is the same as specifying MQACT_NONE. If *token* is nonzero, it must address MQ_ACCOUNTING_TOKEN_LENGTH bytes of binary data. When using predefined values such as MQACT_NONE, you might need to make a cast to ensure a signature match; for example, (MQBYTE *)MQACT_NONE.

ImqBinary correlationId ( ) const ;
Returns a copy of the **correlation id**.

ImqBoolean setCorrelationId (const ImqBinary & *token*);
Sets the **correlation id**. The **data length** of *token* must be either zero or MQ_CORREL_ID_LENGTH. This method returns TRUE if successful.

void setCorrelationId (const MQBYTE24 *id* = 0);
Sets the **correlation id**. *id* can be zero, which is the same as specifying MQCI_NONE. If *id* is nonzero, it must address MQ_CORREL_ID_LENGTH bytes of binary data. When using predefined values such as MQCI_NONE, you might need to make a cast to ensure a signature match; for example, (MQBYTE *)MQCI_NONE.

MQLONG feedback ( ) const ;
Returns the **feedback**.

void setFeedback (const MQLONG *feedback*);
Sets the **feedback**.

ImqBinary groupId ( ) const ;
Returns a copy of the **group id**.

ImqBoolean setGroupId (const ImqBinary & *token*);
Sets the **group id**. The **data length** of *token* must be either zero or MQ_GROUP_ID_LENGTH. This method returns TRUE if successful.

void setGroupId (const MQBYTE24 *id* = 0);
Sets the **group id**. *id* can be zero, which is the same as specifying MQGI_NONE. If *id* is nonzero, it must address MQ_GROUP_ID_LENGTH bytes of binary data. When using predefined values such as MQGI_NONE, you might need to make a cast to ensure a signature match, for example (MQBYTE *)MQGI_NONE.

ImqBinary messageId ( ) const ;
Returns a copy of the **message id**.

ImqBoolean setMessageId (const ImqBinary & *token*);
Sets the **message id**. The **data length** of *token* must be either zero or MQ_MSG_ID_LENGTH. This method returns TRUE if successful.

void setMessageId (const MQBYTE24 *id* = 0);
Sets the **message id**. *id* can be zero, which is the same as specifying MQMI_NONE. If *id* is nonzero, it must address MQ_MSG_ID_LENGTH bytes of binary data. When using predefined values such as MQMI_NONE, you might need to make a cast to ensure a signature match, for example (MQBYTE *)MQMI_NONE.

Reason codes

- MQRC_BINARY_DATA_LENGTH_ERROR

ImqNamelist C++ class

This class encapsulates a namelist.

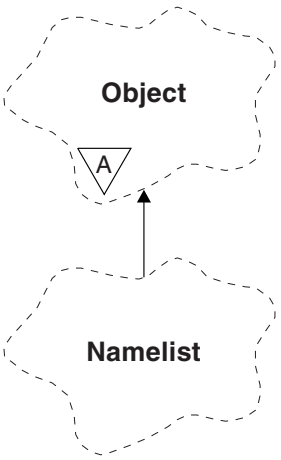


Figure 112. ImqNamelist class

This class relates to the MQI calls listed in "ImqNamelist cross-reference" on page 3929.

- "Object attributes"
- "Constructors"
- "Object methods (public)"
- "Reason codes" on page 3982

Object attributes

name count
The number of object names in **namelist names**. This attribute is read-only.

namelist names
Object names, the number of which is indicated by the **name count**. This attribute is read-only.

Constructors

ImqNamelist( );
The default constructor.

ImqNamelist(const ImqNamelist & *list*);
The copy constructor. The ImqObject **open status** is false.

ImqNamelist(const char * *name*);
Sets the ImqObject name to **name**.

Object methods (public)

void operator = (const ImqNamelist & *list*);
Copies instance data from *list*, replacing the existing instance data. The ImqObject **open status** is false.

ImqBoolean nameCount(MQLONG & *count*);
Provides a copy of the **name count**. It returns TRUE if successful.

MQLONG nameCount ( );
Returns the **name count** without any indication of possible errors.

ImqBoolean namelistName (const MQLONG *index*, ImqString & *name*);
Provides a copy of one the **namelist names** by zero based index. It returns TRUE if successful.

ImqString namelistName (const MQLONG *index*);
Returns one of the **namelist names** by zero-based index without any indication of possible errors.

Reason codes

- MQRC_INDEX_ERROR
- MQRC_INDEX_NOT_PRESENT

ImqObject C++ class

This class is abstract. When an object of this class is destroyed, it is automatically closed, and its ImqQueueManager connection severed.

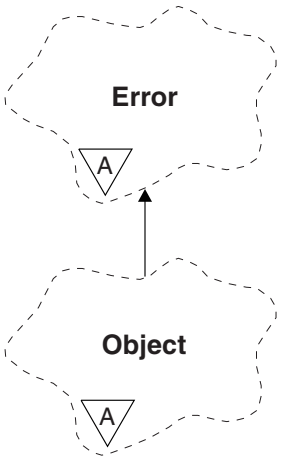


Figure 113. ImqObject class

This class relates to the MQI calls listed in "ImqObject cross-reference" on page 3930.

Class attributes

behavior
Controls the behavior of implicit opening.

IMQ_IMPL_OPEN (8L)
Implicit opening is allowed. This is the default.

Object attributes

alteration date
The alteration date. This attribute is read-only.

alteration time
The alteration time. This attribute is read-only.

alternate user id
The alternate user ID, up to MQ_USER_ID_LENGTH characters. The initial value is a null string.

alternate security id
The alternate security ID. A binary value (MQBYTE40) of length MQ_SECURITY_ID_LENGTH. The initial value is MQSID_NONE.

close options
Options that apply when an object is closed. The initial value is MQCO_NONE. This attribute is ignored during implicit reopen operations, where a value of MQCO_NONE is always used.

connection reference
A reference to an ImqQueueManager object that provides the required connection to a (local) queue manager. For an ImqQueueManager object, it is the object itself. The initial value is zero.

Note: Do not confuse this with the queue manager name that identifies a queue manager (possibly remote) for a named queue.

description
The descriptive name (up to 64 characters) of the queue manager, queue, namelist, or process. This attribute is read-only.

name The name (up to 48 characters) of the queue manager, queue, namelist, or process. The initial value is a null string. The name of a model queue changes after an **open** to the name of the resulting dynamic queue.

Note: An ImqQueueManager can have a null name, representing the default queue manager. The name changes to the actual queue manager after a successful open. An ImqDistributionList is dynamic and must have a null name.

next managed object
This is the next object of this class, in no particular order, having the same connection reference as this object. The initial value is zero.

open options
Options that apply when an object is opened. The initial value is MQOO_INQUIRE. There are two ways to set appropriate values:

1. Do not set the open options and do not use the open method. IBM MQ automatically adjusts the open options and automatically opens, reopens, and closes objects as required. This can result in unnecessary reopen operations, because IBM MQ uses the openFor method, and this adds open options incrementally only.
2. Set the open options before using any methods that result in an MQI call (see "C++ and MQI cross-reference" on page 3923). This ensures that unnecessary reopen operations do not occur. Set open options explicitly if any of the potential reopen problems are likely to occur (see Reopen).

   If you use the open method, you must ensure that the open options are appropriate first. However, using the open method is not mandatory; IBM MQ still exhibits the same behavior as in case 1, but in this circumstance, the behavior is efficient.

Zero is not a valid value; set the appropriate value before attempting to open the object. This can be done using either **setOpenOptions** (*lOpenOptions*) followed by **open** ( ), or **openFor** (*lRequiredOpenOption*).

Note:

1. MQOO_OUTPUT is substituted for MQOO_INQUIRE during the **open** method for a distribution list, as MQOO_OUTPUT is the only valid **open option** at this time. However, it is good practice always to set MQOO_OUTPUT explicitly in application programs that use the **open** method.
2. Specify MQOO_RESOLVE_NAMES if you want to use the **resolved queue manager name** and **resolved queue name** attributes of the class.

open status
Whether the object is open (TRUE) or closed (FALSE). The initial value is FALSE. This attribute is read-only.

previous managed object
The previous object of this class, in no particular order, having the same connection reference as this object. The initial value is zero.

queue-manager-identifier
The queue manager identifier. This attribute is read-only.

Constructors

ImqObject( );
The default constructor.

ImqObject(const ImqObject & *object*);
The copy constructor. The open status will be FALSE.

Class methods (public)

static MQLONG behavior( );
Returns the behavior.

void setBehavior(const MQLONG *behavior* = 0);
Sets the behavior.

Object methods (public)

void operator = (const ImqObject & *object*);
Performs a close if necessary, and copies the instance data from *object*. The open status will be FALSE.

ImqBoolean alterationDate(ImqString & *date*);
Provides a copy of the alteration date. It returns TRUE if successful.

ImqString alterationDate( );
Returns the alteration date without any indication of possible errors.

ImqBoolean alterationTime(ImqString & *time*);
Provides a copy of the alteration time. It returns TRUE if successful.

ImqString alterationTime( );
Returns the alteration time without any indication of possible errors.

ImqString alternateUserId ( ) const ;
Returns a copy of the alternate user id.

ImqBoolean setAlternateUserId (const char * *id*);
Sets the alternate user id. The alternate user id can be set only while the open status is FALSE. This method returns TRUE if successful.

ImqBinary alternateSecurityId( ) const ;
Returns a copy of the alternate security ID.

ImqBoolean setAlternateSecurityId(const ImqBinary & *token*);
Sets the alternate security id. The alternate security id can be set only while the open status is FALSE. The data length of *token* must be either zero or MQ_SECURITY_ID_LENGTH. It returns TRUE if successful.

ImqBoolean setAlternateSecurityId(const MQBYTE* *token* = 0);
Sets the alternate security id. *token* can be zero, which is the same as specifying MQSID_NONE. If *token* is nonzero, it must address MQ_SECURITY_ID_LENGTH bytes of binary data. When using predefined values such as MQSID_NONE, you might need to make a cast to ensure signature match; for example, (MQBYTE *)MQSID_NONE.

The alternate security id can be set only while the open status is TRUE. It returns TRUE if successful.

ImqBoolean setAlternateSecurityId(const unsigned char * *id* = 0);
Sets the alternate security id.

ImqBoolean close ( );
Sets the open status to FALSE. It returns TRUE if successful.

MQLONG closeOptions ( ) const ;
Returns the close options.

void setCloseOptions (const MQLONG *options*);
Sets the close options.

ImqQueueManager * connectionReference ( ) const ;
Returns the connection reference.

void setConnectionReference (ImqQueueManager & *manager*);
Sets the connection reference.

void setConnectionReference (ImqQueueManager * *manager* = 0);
Sets the connection reference.

virtual ImqBoolean description (ImqString & *description*) = 0 ;
Provides a copy of the description. It returns TRUE if successful.

ImqString description ( );
Returns a copy of the description without any indication of possible errors.

virtual ImqBoolean name (ImqString & *name*);
Provides a copy of the name. It returns TRUE if successful.

ImqString name ( );
Returns a copy of the name without any indication of possible errors.

ImqBoolean setName (const char * *name* = 0);
Sets the name. The name can only be set while the open status is FALSE, and, for an ImqQueueManager, while the connection status is FALSE. It returns TRUE if successful.

ImqObject * nextManagedObject ( ) const ;
Returns the next managed object.

ImqBoolean open ( );
Changes the open status to TRUE by opening the object as necessary, using among other attributes the open options and the name. This method uses the connection reference information and the ImqQueueManager connect method if necessary to ensure that the ImqQueueManager connection status is TRUE. It returns the open status.

ImqBoolean openFor (const MQLONG *required-options* = 0);
Attempts to ensure that the object is open with open options, or with open options that guarantee the behavior implied by the *required-options* parameter value.

If *required-options* is zero, input is required, and any input option suffices. So, if the open options already contain one of:

- MQOO_INPUT_AS_Q_DEF
- MQOO_INPUT_SHARED
- MQOO_INPUT_EXCLUSIVE

the open options are already satisfactory and are not changed; if the open options do not already contain any of these options, MQOO_INPUT_AS_Q_DEF is set in the open options.

If *required-options* is nonzero, the required options are added to the open options ; if *required-options* is any of these options, the others are reset.

If any of the open options are changed and the object is already open, the object is closed temporarily and reopened in order to adjust the open options.

It returns TRUE if successful. Success indicates that the object is open with appropriate options.

MQLONG openOptions ( ) const ;
Returns the open options.

ImqBoolean setOpenOptions (const MQLONG *options*);
Sets the open options. The open options can be set only while the open status is FALSE. It returns TRUE if successful.

ImqBoolean openStatus ( ) const ;
Returns the open status.

ImqObject * previousManagedObject ( ) const ;
Returns the previous managed object.

ImqBoolean queueManagerIdentifier(ImqString & *id*);
Provides a copy of the queue manager identifier. It returns TRUE if successful.

ImqString queueManagerIdentifier( );
Returns the queue manager identifier without any indication of possible errors.

Object methods (protected)

virtual ImqBoolean closeTemporarily ( );
Closes an object safely before reopening. It returns TRUE if successful. This method assumes that the open status is TRUE.

MQHCONN connectionHandle ( ) const ;
Returns the MQHCONN associated with the connection reference. This value is zero if there is no connection reference or if the Manager is not connected.

ImqBoolean inquire (const MQLONG *int-attr*, MQLONG & *value*);
Returns an integer value, the index of which is an MQIA_* value. In case of error, the value is set to MQIAV_UNDEFINED.

ImqBoolean inquire (const MQLONG *char-attr*, char * & *buffer*, const size_t *length*);
Returns a character string, the index of which is an MQCA_* value.

Note: Both of these methods return only a single attribute value. If a *snapshot* is required of more than one value, where the values are consistent with each other for an instant, IBM MQ C++ does not provide this facility and you must use the MQINQ call with appropriate parameters.

virtual void openInformationDisperse ( );
Disperses information from the variable section of the MQOD data structure immediately after an MQOPEN call.

virtual ImqBoolean openInformationPrepare ( );
Prepares information for the variable section of the MQOD data structure immediately before an MQOPEN call, and returns TRUE if successful.

ImqBoolean set (const MQLONG *int-attr*, const MQLONG *value*);
Sets an IBM MQ integer attribute.

ImqBoolean set (const MQLONG *char-attr*, const char * *buffer*, const size_t *required-length*);
Sets an IBM MQ character attribute.

void setNextManagedObject (const ImqObject * *object* = 0);
Sets the next managed object.

Attention: Use this function only if you are sure it will not break the managed object list.

void setPreviousManagedObject (const ImqObject * *object* = 0);
Sets the previous managed object.

Attention: Use this function only if you are sure it will not break the managed object list.

Object data (protected)

MQHOBJ *ohobj*
The IBM MQ object handle (valid only when open status is TRUE).

MQOD *omqod*
The embedded MQOD data structure. The amount of storage allocated for this data structure is that required for an MQOD Version 2. Inspect the version number *(omqod.Version)* and access the other fields as follows:

MQOD_VERSION_1
All other fields in *omqod* can be accessed.

MQOD_VERSION_2
All other fields in *omqod* can be accessed.

MQOD_VERSION_3
omqod.pmqod is a pointer to a dynamically allocated, larger, MQOD. No other fields in *omqod* can be accessed. All fields addressed by *omqod.pmqod* can be accessed.

Note: *omqod.pmqod.Version* can be less than *omqod.Version*, indicating that the IBM MQ MQI client has more functionality than the IBM MQ server.

Reason codes

- MQRC_ATTRIBUTE_LOCKED
- MQRC_INCONSISTENT_OBJECT_STATE
- MQRC_NO_CONNECTION_REFERENCE
- MQRC_STORAGE_NOT_AVAILABLE
- MQRC_REOPEN_SAVED_CONTEXT_ERR
- (reason codes from MQCLOSE)
- (reason codes from MQCONN)
- (reason codes from MQINQ)
- (reason codes from MQOPEN)
- (reason codes from MQSET)

ImqProcess C++ class

This class encapsulates an application process (an IBM MQ object of type MQOT_PROCESS) that can be triggered by a trigger monitor.

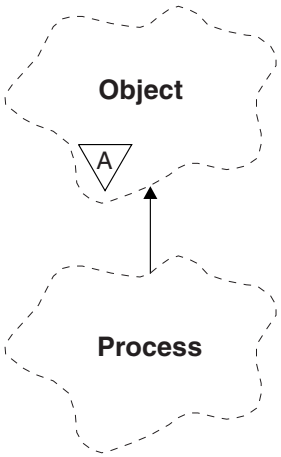


Figure 114. ImqProcess class

- "Object attributes"
- "Constructors"
- "Object methods (public)"

Object attributes

application id
The identity of the application process. This attribute is read-only.

application type
The type of the application process. This attribute is read-only.

environment data
The environment information for the process. This attribute is read-only.

user data
User data for the process. This attribute is read-only.

Constructors

ImqProcess( );
The default constructor.

ImqProcess(const ImqProcess & *process*);
The copy constructor. The ImqObject **open status** is FALSE.

ImqProcess(const char * *name*);
Sets the ImqObject **name**.

Object methods (public)

void operator = (const ImqProcess & *process*);
Performs a close if necessary, and then copies instance data from *process*. The ImqObject **open status** will be FALSE.

ImqBoolean applicationId (ImqString & *id*);
Provides a copy of the **application id**. It returns TRUE if successful.

ImqString applicationId ( );
Returns the **application id** without any indication of possible errors.

ImqBoolean applicationType (MQLONG & *type*);
Provides a copy of the **application type**. It returns TRUE if successful.

MQLONG applicationType ( );
Returns the **application type** without any indication of possible errors.

ImqBoolean environmentData (ImqString & *data*);
Provides a copy of the **environment data**. It returns TRUE if successful.

ImqString environmentData ( );
Returns the **environment data** without any indication of possible errors.

ImqBoolean userData (ImqString & *data*);
Provides a copy of the **user data**. It returns TRUE if successful.

ImqString userData ( );
Returns the **user data** without any indication of possible errors.

ImqPutMessageOptions C++ class

This class encapsulates the MQPMO data structure.

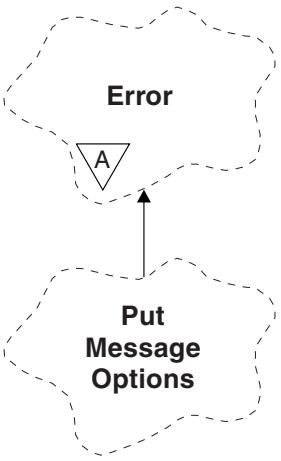


Figure 115. ImqPutMessageOptions class

- "Object attributes"
- "Constructors" on page 3990
- "Object methods (public)" on page 3990
- "Object data (protected)" on page 3991
- "Reason codes" on page 3991

Object attributes

context reference
An ImqQueue that provides a context for messages. Initially there is no reference.

options
The put message options. The initial value is MQPMO_NONE. The following additional values are possible:

- MQPMO_SYNCPOINT
- MQPMO_NO_SYNCPOINT

- MQPMO_NEW_MSG_ID
- MQPMO_NEW_CORREL_ID
- MQPMO_LOGICAL_ORDER
- MQPMO_NO_CONTEXT
- MQPMO_DEFAULT_CONTEXT
- MQPMO_PASS_IDENTITY_CONTEXT
- MQPMO_PASS_ALL_CONTEXT
- MQPMO_SET_IDENTITY_CONTEXT
- MQPMO_SET_ALL_CONTEXT
- MQPMO_ALTERNATE_USER_AUTHORITY
- MQPMO_FAIL_IF_QUIESCING

record fields
The flags that control the inclusion of put message records when a message is put. The initial value is MQPMRF_NONE. The following additional values are possible:

- MQPMRF_MSG_ID
- MQPMRF_CORREL_ID
- MQPMRF_GROUP_ID
- MQPMRF_FEEDBACK
- MQPMRF_ACCOUNTING_TOKEN

ImqMessageTracker attributes are taken from the object for any field that is specified. ImqMessageTracker attributes are taken from the ImqMessage object for any field that is not specified.

resolved queue manager name
Name of a destination queue manager determined during a put. The initial value is null. This attribute is read-only.

resolved queue name
Name of a destination queue determined during a put. The initial value is null. This attribute is read-only.

syncpoint participation
TRUE when messages are put under syncpoint control.

Constructors

ImqPutMessageOptions( );
The default constructor.

ImqPutMessageOptions(const ImqPutMessageOptions & *pmo*);
The copy constructor.

Object methods (public)

void operator = (const ImqPutMessageOptions & *pmo*);
Copies instance data from *pmo*, replacing the existing instance data.

ImqQueue * contextReference ( ) const ;
Returns the context reference.

void setContextReference (const ImqQueue & *queue*);
Sets the context reference.

void setContextReference (const ImqQueue * *queue* = 0);
Sets the context reference.

MQLONG options ( ) const ;
Returns the options.

void setOptions (const MQLONG *options*);
Sets the options, including the syncpoint participation value.

MQLONG recordFields ( ) const ;
Returns the record fields.

void setRecordFields (const MQLONG *fields*);
Sets the record fields.

ImqString resolvedQueueManagerName ( ) const ;
Returns a copy of the resolved queue manager name.

ImqString resolvedQueueName ( ) const ;
Returns a copy of the resolved queue name.

ImqBoolean syncPointParticipation ( ) const ;
Returns the syncpoint participation value, which is TRUE if the options include MQPMO_SYNCPOINT.

void setSyncPointParticipation (const ImqBoolean *sync*);
Sets the syncpoint participation value. If *sync* is TRUE, the options are altered to include MQPMO_SYNCPOINT, and to exclude MQPMO_NO_SYNCPOINT. If *sync* is FALSE, the options are altered to include MQPMO_NO_SYNCPOINT, and to exclude MQPMO_SYNCPOINT.

Object data (protected)

MQPMO *omqpmo*
The MQPMO data structure.

Reason codes

- MQRC_STORAGE_NOT_AVAILABLE

ImqQueue C++ class

This class encapsulates a message queue (an IBM MQ object of type MQOT_Q).

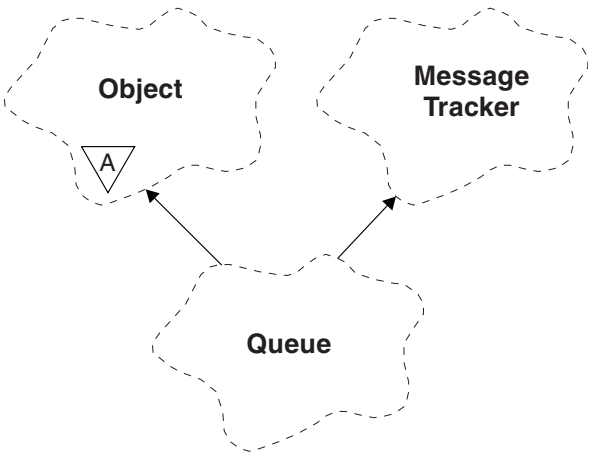


Figure 116. ImqQueue class

This class relates to the MQI calls listed in Table 416 on page 3931.

- "Object attributes" on page 3992
- "Constructors" on page 3995

- "Object methods (public)" on page 3995
- "Object methods (protected)" on page 4002
- "Reason codes" on page 4002

Object attributes

backout requeue name
Excessive backout requeue name. This attribute is read-only.

backout threshold
Backout threshold. This attribute is read-only.

base queue name
Name of the queue that the alias resolves to. This attribute is read-only.

cluster name
Cluster name. This attribute is read-only.

cluster namelist name
Cluster namelist name. This attribute is read-only.

cluster workload rank
Cluster workload rank. This attribute is read-only.

cluster workload priority
Cluster workload priority. This attribute is read-only.

cluster workload use queue
Cluster workload use queue value. This attribute is read-only.

creation date
Queue creation data. This attribute is read-only.

creation time
Queue creation time. This attribute is read-only.

current depth
Number of messages on the queue. This attribute is read-only.

default bind
Default bind. This attribute is read-only.

default input open option
Default open-for-input option. This attribute is read-only.

default persistence
Default message persistence. This attribute is read-only.

default priority
Default message priority. This attribute is read-only.

definition type
Queue definition type. This attribute is read-only.

depth high event
Control attribute for queue depth high events. This attribute is read-only.

depth high limit
High limit for the queue depth. This attribute is read-only.

depth low event
Control attribute for queue depth low events. This attribute is read-only.

depth low limit
Low limit for the queue depth. This attribute is read-only.

depth maximum event
Control attribute for queue depth maximum events. This attribute is read-only.

distribution list reference
Optional reference to an ImqDistributionList that can be used to distribute messages to more than one queue, including this one. The initial value is null.

Note: When an ImqQueue object is opened, any open ImqDistributionList object that it references is automatically closed.

distribution lists
The capability of a transmission queue to support distribution lists. This attribute is read-only.

dynamic queue name
Dynamic queue name. The initial value is AMQ.* for all Windows, UNIX, and Linux platforms.

harden get backout
Whether to harden the backout count. This attribute is read-only.

index type
Index type. This attribute is read-only.

inhibit get
Whether get operations are allowed. The initial value is dependent on the queue definition. This attribute is valid for an alias or local queue only.

inhibit put
Whether put operations are allowed. The initial value is dependent on the queue definition.

initiation queue name
Name of the initiation queue. This attribute is read-only.

maximum depth
Maximum number of messages allowed on the queue. This attribute is read-only.

maximum message length
Maximum length for any message on this queue, which can be less than the maximum for any queue managed by the associated queue manager. This attribute is read-only.

message delivery sequence
Whether message priority is relevant. This attribute is read-only.

next distributed queue
Next object of this class, in no particular order, having the same **distribution list reference** as this object. The initial value is zero.

If an object in a chain is deleted, the previous object and next object are updated so that their distributed queue links no longer point to the deleted object.

non-persistent message class
Level of reliability for non-persistent messages put to this queue. This attribute is read-only.

open input count
Number of ImqQueue objects that are open for input. This attribute is read-only.

open output count
Number of ImqQueue objects that are open for output. This attribute is read-only.

previous distributed queue
Previous object of this class, in no particular order, having the same **distribution list reference** as this object. The initial value is zero.

If an object in a chain is deleted, the previous object and next object are updated so that their distributed queue links no longer point to the deleted object.

process name
Name of the process definition. This attribute is read-only.

queue accounting
Level of accounting information for queues. This attribute is read-only.

queue-manager-name
Name of the queue manager (possibly remote) where the queue resides. Do not confuse the queue manager named here with the ImqObject **connection reference**, which references the (local) queue manager providing a connection. The initial value is null.

queue monitoring
Level of monitoring data collection for the queue. This attribute is read-only.

queue statistics
Level of statistics data for the queue. This attribute is read-only.

queue type
Queue type. This attribute is read-only.

remote queue manager name
Name of the remote queue manager. This attribute is read-only.

remote queue name
Name of the remote queue as known on the remote queue manager. This attribute is read-only.

resolved queue manager name
Resolved queue manager name. This attribute is read-only.

resolved queue name
Resolved queue name. This attribute is read-only.

retention interval
Queue retention interval. This attribute is read-only.

scope Scope of the queue definition. This attribute is read-only.

service interval
Service interval. This attribute is read-only.

service interval event
Control attribute for service interval events. This attribute is read-only.

shareability
Whether the queue can be shared. This attribute is read-only.

storage class
Storage class. This attribute is read-only.

transmission queue name
Name of the transmission queue. This attribute is read-only.

trigger control
Trigger control. The initial value depends on the queue definition. This attribute is valid for a local queue only.

trigger data
Trigger data. The initial value depends on the queue definition. This attribute is valid for a local queue only.

trigger depth
Trigger depth. The initial value depends on the queue definition. This attribute is valid for a local queue only.

trigger message priority
Threshold message priority for triggers. The initial value depends on the queue definition. This attribute is valid for a local queue only.

trigger type
Trigger type. The initial value depends on the queue definition. This attribute is valid for a local queue only.

usage Usage. This attribute is read-only.

Constructors

ImqQueue( );
The default constructor.

ImqQueue(const ImqQueue & *queue*);
The copy constructor. The ImqObject **open status** will be FALSE.

ImqQueue(const char * *name*);
Sets the ImqObject **name**.

Object methods (public)

void operator = (const ImqQueue & *queue*);
Performs a close if necessary, and then copies instance data from *queue*. The ImqObject **open status** will be FALSE.

ImqBoolean backoutRequeueName (ImqString & *name*);
Provides a copy of the **backout requeue name**. It returns TRUE if successful.

ImqString backoutRequeueName ( );
Returns the **backout requeue name** without any indication of possible errors.

ImqBoolean backoutThreshold (MQLONG & *threshold*);
Provides a copy of the **backout threshold**. It returns TRUE if successful.

MQLONG backoutThreshold ( );
Returns the **backout threshold** value without any indication of possible errors.

ImqBoolean baseQueueName (ImqString & *name*);
Provides a copy of the **base queue name**. It returns TRUE if successful.

ImqString baseQueueName ( );
Returns the **base queue name** without any indication of possible errors.

ImqBoolean clusterName(ImqString & *name*);
Provides a copy of the **cluster name**. It returns TRUE if successful.

ImqString clusterName( );
Returns the **cluster name** without any indication of possible errors.

ImqBoolean clusterNamelistName(ImqString & *name*);
Provides a copy of the **cluster namelist name**. It returns TRUE if successful.

ImqString clusterNamelistName( );
Returns the **cluster namelist name** without any indication of errors.

ImqBoolean clusterWorkLoadPriority (MQLONG & priority);
Provides a copy of the cluster workload priority value. It returns TRUE if successful.

MQLONG clusterWorkLoadPriority ( );
Returns the cluster workload priority value without any indication of possible errors.

ImqBoolean clusterWorkLoadRank (MQLONG & rank);
Provides a copy of the cluster workload rank value. It returns TRUE if successful.

MQLONG clusterWorkLoadRank ( );
Returns the cluster workload rank value without any indication of possible errors.

ImqBoolean clusterWorkLoadUseQ (MQLONG & useq);
Provides a copy of the cluster workload use queue value. It returns TRUE if successful.

MQLONG clusterWorkLoadUseQ ( );
Returns the cluster workload use queue value without any indication of possible errors.

ImqBoolean creationDate (ImqString & *date* **);**
Provides a copy of the **creation date**. It returns TRUE if successful.

ImqString creationDate ( );
Returns the **creation date** without any indication of possible errors.

ImqBoolean creationTime (ImqString & *time* **);**
Provides a copy of the **creation time**. It returns TRUE if successful.

ImqString creationTime ( );
Returns the **creation time** without any indication of possible errors.

ImqBoolean currentDepth (MQLONG & *depth* **);**
Provides a copy of the **current depth**. It returns TRUE if successful.

MQLONG currentDepth ( );
Returns the **current depth** without any indication of possible errors.

ImqBoolean defaultInputOpenOption (MQLONG & *option* **);**
Provides a copy of the **default input open option**. It returns TRUE if successful.

MQLONG defaultInputOpenOption ( );
Returns the **default input open option** without any indication of possible errors.

ImqBoolean defaultPersistence (MQLONG & *persistence* **);**
Provides a copy of the **default persistence**. It returns TRUE if successful.

MQLONG defaultPersistence ( );
Returns the **default persistence** without any indication of possible errors.

ImqBoolean defaultPriority (MQLONG & *priority* **);**
Provides a copy of the **default priority**. It returns TRUE if successful.

MQLONG defaultPriority ( );
Returns the **default priority** without any indication of possible errors.

ImqBoolean defaultBind (MQLONG & *bind* **);**
Provides a copy of the **default bind**. It returns TRUE if successful.

MQLONG defaultBind ( );
Returns the **default bind** without any indication of possible errors.

ImqBoolean definitionType (MQLONG & *type* **);**
Provides a copy of the **definition type**. It returns TRUE if successful.

MQLONG definitionType ( );
Returns the **definition type** without any indication of possible errors.

ImqBoolean depthHighEvent (MQLONG & *event* **);**
Provides a copy of the enablement state of the **depth high event**. It returns TRUE if successful.

MQLONG depthHighEvent ( );
Returns the enablement state of the **depth high event** without any indication of possible errors.

ImqBoolean depthHighLimit (MQLONG & *limit* **);**
Provides a copy of the **depth high limit**. It returns TRUE if successful.

MQLONG depthHighLimit ( );
Returns the **depth high limit** value without any indication of possible errors.

ImqBoolean depthLowEvent (MQLONG & *event*);
Provides a copy of the enablement state of the **depth low event**. It returns TRUE if successful.

MQLONG depthLowEvent ( );
Returns the enablement state of the **depth low event** without any indication of possible errors.

ImqBoolean depthLowLimit (MQLONG & *limit*);
Provides a copy of the **depth low limit**. It returns TRUE if successful.

MQLONG depthLowLimit ( );
Returns the **depth low limit** value without any indication of possible errors.

ImqBoolean depthMaximumEvent (MQLONG & *event*);
Provides a copy of the enablement state of the **depth maximum event**. It returns TRUE if successful.

MQLONG depthMaximumEvent ( );
Returns the enablement state of the **depth maximum event** without any indication of possible errors.

ImqDistributionList * distributionListReference ( ) const ;
Returns the **distribution list reference**.

void setDistributionListReference (ImqDistributionList & *list*);
Sets the **distribution list reference**.

void setDistributionListReference (ImqDistributionList * *list* = 0);
Sets the **distribution list reference**.

ImqBoolean distributionLists (MQLONG & *support*);
Provides a copy of the **distribution lists** value. It returns TRUE if successful.

MQLONG distributionLists ( );
Returns the **distribution lists** value without any indication of possible errors.

ImqBoolean setDistributionLists (const MQLONG *support*);
Sets the **distribution lists** value. It returns TRUE if successful.

ImqString dynamicQueueName ( ) const ;
Returns a copy of the **dynamic queue name**.

ImqBoolean setDynamicQueueName (const char * *name*);
Sets the **dynamic queue name**. The **dynamic queue name** can be set only while the ImqObject **open status** is FALSE. It returns TRUE if successful.

ImqBoolean get (ImqMessage & *msg*, ImqGetMessageOptions & *options*);
Retrieves a message from the queue, using the specified *options*. Invokes the ImqObject **openFor** method if necessary to ensure that the ImqObject **open options** include either one of the MQOO_INPUT_* values, or the MQOO_BROWSE value, depending on the *options*. If the *msg* object has an ImqCache **automatic buffer**, the buffer grows to accommodate any message retrieved. The **clearMessage** method is invoked against the *msg* object before retrieval.

This method returns TRUE if successful.

Note: The result of the method invocation is FALSE if the ImqObject **reason code** is MQRC_TRUNCATED_MSG_FAILED, even though this **reason code** is classified as a warning. If a truncated message is accepted, the ImqCache **message length** reflects the truncated length. In either event, the ImqMessage **total message length** indicates the number of bytes that were available.

ImqBoolean get (ImqMessage & *msg*);
As for the previous method, except that default get message options are used.

ImqBoolean get (ImqMessage & *msg*, ImqGetMessageOptions & *options*, const size_t *buffer-size*);
As for the previous two methods, except that an overriding *buffer-size* is indicated. If the *msg* object employs an ImqCache **automatic buffer**, the **resizeBuffer** method is invoked on the *msg* object prior to message retrieval, and the buffer does not grow further to accommodate any larger message.

ImqBoolean get (ImqMessage & *msg*, const size_t *buffer-size*);
As for the previous method, except that default get message options are used.

ImqBoolean hardenGetBackout (MQLONG & *harden*);
Provides a copy of the **harden get backout** value. It returns TRUE if successful.

MQLONG hardenGetBackout ( );
Returns the **harden get backout** value without any indication of possible errors.

ImqBoolean indexType(MQLONG & *type*);
Provides a copy of the **index type**. It returns TRUE if successful.

MQLONG indexType( );
Returns the **index type** without any indication of possible errors.

ImqBoolean inhibitGet (MQLONG & *inhibit*);
Provides a copy of the **inhibit get** value. It returns TRUE if successful.

MQLONG inhibitGet ( );
Returns the **inhibit get** value without any indication of possible errors.

ImqBoolean setInhibitGet (const MQLONG *inhibit*);
Sets the **inhibit get** value. It returns TRUE if successful.

ImqBoolean inhibitPut (MQLONG & *inhibit*);
Provides a copy of the **inhibit put** value. It returns TRUE if successful.

MQLONG inhibitPut ( );
Returns the **inhibit put** value without any indication of possible errors.

ImqBoolean setInhibitPut (const MQLONG *inhibit*);
Sets the **inhibit put** value. It returns TRUE if successful.

ImqBoolean initiationQueueName (ImqString & *name*);
Provides a copy of the **initiation queue name**. It returns TRUE if successful.

ImqString initiationQueueName ( );
Returns the **initiation queue name** without any indication of possible errors.

ImqBoolean maximumDepth (MQLONG & *depth*);
Provides a copy of the **maximum depth**. It returns TRUE if successful.

MQLONG maximumDepth ( );
Returns the **maximum depth** without any indication of possible errors.

ImqBoolean maximumMessageLength (MQLONG & *length*);
Provides a copy of the **maximum message length**. It returns TRUE if successful.

MQLONG maximumMessageLength ( );
Returns the **maximum message length** without any indication of possible errors.

ImqBoolean messageDeliverySequence (MQLONG & *sequence*);
Provides a copy of the **message delivery sequence**. It returns TRUE if successful.

MQLONG messageDeliverySequence ( );
Returns the **message delivery sequence** value without any indication of possible errors.

ImqQueue * nextDistributedQueue ( ) const ;
Returns the **next distributed queue**.

ImqBoolean nonPersistentMessageClass (MQLONG & monq);
Provides a copy of the non persistent message class value. It returns TRUE if successful.

MQLONG nonPersistentMessageClass ( );
Returns the non persistent message class value without any indication of possible errors.

ImqBoolean openInputCount (MQLONG & ***count*** **);**
Provides a copy of the **open input count**. It returns TRUE if successful.

MQLONG openInputCount ( );
Returns the **open input count** without any indication of possible errors.

ImqBoolean openOutputCount (MQLONG & ***count*** **);**
Provides a copy of the **open output count**. It returns TRUE if successful.

MQLONG openOutputCount ( );
Returns the **open output count** without any indication of possible errors.

ImqQueue * previousDistributedQueue ( ) const ;
Returns the **previous distributed queue**.

ImqBoolean processName (ImqString & ***name*** **);**
Provides a copy of the **process name**. It returns TRUE if successful.

ImqString processName ( );
Returns the **process name** without any indication of possible errors.

ImqBoolean put (ImqMessage & ***msg*** **);**
Places a message onto the queue, using default put message options. Uses the ImqObject **openFor** method if necessary to ensure that the ImqObject **open options** include MQOO_OUTPUT.

This method returns TRUE if successful.

ImqBoolean put (ImqMessage & ***msg*****, ImqPutMessageOptions &** ***pmo*** **);**
Places a message onto the queue, using the specified *pmo*. Uses the ImqObject **openFor** method as necessary to ensure that the ImqObject **open options** include MQOO_OUTPUT, and (if the *pmo* **options** include any of MQPMO_PASS_IDENTITY_CONTEXT, MQPMO_PASS_ALL_CONTEXT, MQPMO_SET_IDENTITY_CONTEXT, or MQPMO_SET_ALL_CONTEXT) corresponding MQOO_*_CONTEXT values.

This method returns TRUE if successful.

Note: If the *pmo* includes a **context reference**, the referenced object is opened, if necessary, to provide a context.

ImqBoolean queueAccounting (MQLONG & acctq);
Provides a copy of the queue accounting value. It returns TRUE if successful.

MQLONG queueAccounting ( );
Returns the queue accounting value without any indication of possible errors.

ImqString queueManagerName ( ) const ;
Returns the **queue manager name**.

ImqBoolean setQueueManagerName (const char * ***name*** **);**
Sets the **queue manager name**. The **queue manager name** can be set only while the ImqObject **open status** is FALSE. This method returns TRUE if successful.

ImqBoolean queueMonitoring (MQLONG & monq);
Provides a copy of the queue monitoring value. It returns TRUE if successful.

MQLONG queueMonitoring ( );
Returns the queue monitoring value without any indication of possible errors.

ImqBoolean queueStatistics (MQLONG & statq);
Provides a copy of the queue statistics value. It returns TRUE if successful.

MQLONG queueStatistics ( );
Returns the queue statistics value without any indication of possible errors.

ImqBoolean queueType (MQLONG & *type* **);**
Provides a copy of the **queue type** value. It returns TRUE if successful.

MQLONG queueType ( );
Returns the **queue type** without any indication of possible errors.

ImqBoolean remoteQueueManagerName (ImqString & *name* **);**
Provides a copy of the **remote queue manager name**. It returns TRUE if successful.

ImqString remoteQueueManagerName ( );
Returns the **remote queue manager name** without any indication of possible errors.

ImqBoolean remoteQueueName (ImqString & *name* **);**
Provides a copy of the **remote queue name**. It returns TRUE if successful.

ImqString remoteQueueName ( );
Returns the **remote queue name** without any indication of possible errors.

ImqBoolean resolvedQueueManagerName(ImqString & *name* **);**
Provides a copy of the **resolved queue manager name**. It returns TRUE if successful.

Note: This method fails unless MQOO_RESOLVE_NAMES is among the ImqObject **open options**.

ImqString resolvedQueueManagerName( ) ;
Returns the **resolved queue manager name**, without any indication of possible errors.

ImqBoolean resolvedQueueName(ImqString & *name* **);**
Provides a copy of the **resolved queue name**. It returns TRUE if successful.

Note: This method fails unless MQOO_RESOLVE_NAMES is among the ImqObject **open options**.

ImqString resolvedQueueName( ) ;
Returns the **resolved queue name**, without any indication of possible errors.

ImqBoolean retentionInterval (MQLONG & *interval* **);**
Provides a copy of the **retention interval**. It returns TRUE if successful.

MQLONG retentionInterval ( );
Returns the **retention interval** without any indication of possible errors.

ImqBoolean scope (MQLONG & *scope* **);**
Provides a copy of the **scope**. It returns TRUE if successful.

MQLONG scope ( );
Returns the **scope** without any indication of possible errors.

ImqBoolean serviceInterval (MQLONG & *interval* **);**
Provides a copy of the **service interval**. It returns TRUE if successful.

MQLONG serviceInterval ( );
Returns the **service interval** without any indication of possible errors.

ImqBoolean serviceIntervalEvent (MQLONG & *event*);
Provides a copy of the enablement state of the **service interval event**. It returns TRUE if successful.

MQLONG serviceIntervalEvent ( );
Returns the enablement state of the **service interval event** without any indication of possible errors.

ImqBoolean shareability (MQLONG & *shareability*);
Provides a copy of the **shareability** value. It returns TRUE if successful.

MQLONG shareability ( );
Returns the **shareability** value without any indication of possible errors.

ImqBoolean storageClass(ImqString & *class*);
Provides a copy of the **storage class**. It returns TRUE if successful.

ImqString storageClass( );
Returns the **storage class** without any indication of possible errors.

ImqBoolean transmissionQueueName (ImqString & *name*);
Provides a copy of the **transmission queue name**. It returns TRUE if successful.

ImqString transmissionQueueName ( );
Returns the **transmission queue name** without any indication of possible errors.

ImqBoolean triggerControl (MQLONG & *control*);
Provides a copy of the **trigger control** value. It returns TRUE if successful.

MQLONG triggerControl ( );
Returns the **trigger control** value without any indication of possible errors.

ImqBoolean setTriggerControl (const MQLONG *control*);
Sets the **trigger control** value. It returns TRUE if successful.

ImqBoolean triggerData (ImqString & *data*);
Provides a copy of the **trigger data**. It returns TRUE if successful.

ImqString triggerData ( );
Returns a copy of the **trigger data** without any indication of possible errors.

ImqBoolean setTriggerData (const char * *data*);
Sets the **trigger data**. It returns TRUE if successful.

ImqBoolean triggerDepth (MQLONG & *depth*);
Provides a copy of the **trigger depth**. It returns TRUE if successful.

MQLONG triggerDepth ( );
Returns the **trigger depth** without any indication of possible errors.

ImqBoolean setTriggerDepth (const MQLONG *depth*);
Sets the **trigger depth**. It returns TRUE if successful.

ImqBoolean triggerMessagePriority (MQLONG & *priority*);
Provides a copy of the **trigger message priority**. It returns TRUE if successful.

MQLONG triggerMessagePriority ( );
Returns the **trigger message priority** without any indication of possible errors.

ImqBoolean setTriggerMessagePriority (const MQLONG *priority*);
Sets the **trigger message priority**. It returns TRUE if successful.

ImqBoolean triggerType (MQLONG & *type*);
Provides a copy of the **trigger type**. It returns TRUE if successful.

MQLONG triggerType ( );
Returns the **trigger type** without any indication of possible errors.

ImqBoolean setTriggerType (const MQLONG *type*);
Sets the **trigger type**. It returns TRUE if successful.

ImqBoolean usage (MQLONG & *usage*);
Provides a copy of the **usage** value. It returns TRUE if successful.

MQLONG usage ( );
Returns the **usage** value without any indication of possible errors.

Object methods (protected)

void setNextDistributedQueue (ImqQueue * *queue* = 0);
Sets the **next distributed queue**.

Attention: Use this function only if you are sure it will not break the distributed queue list.

void setPreviousDistributedQueue (ImqQueue * *queue* = 0);
Sets the **previous distributed queue**.

Attention: Use this function only if you are sure it will not break the distributed queue list.

Reason codes

- MQRC_ATTRIBUTE_LOCKED
- MQRC_CONTEXT_OBJECT_NOT_VALID
- MQRC_CONTEXT_OPEN_ERROR
- MQRC_CURSOR_NOT_VALID
- MQRC_NO_BUFFER
- MQRC_REOPEN_EXCL_INPUT_ERROR
- MQRC_REOPEN_INQUIRE_ERROR
- MQRC_REOPEN_TEMPORARY_Q_ERROR
- (reason codes from MQGET)
- (reason codes from MQPUT)

ImqQueueManager C++ class

This class encapsulates a queue manager (an IBM MQ object of type MQOT_Q_MGR).

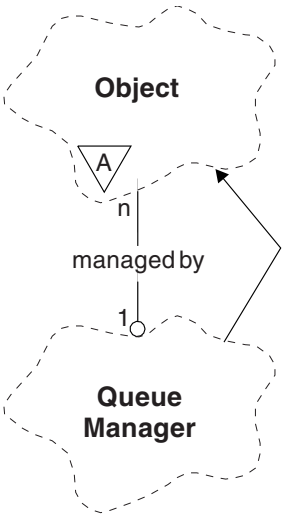


Figure 117. ImqQueueManager class

This class relates to the MQI calls listed in "ImqQueueManager cross-reference" on page 3933. Not all the listed methods are applicable to all platforms; see ALTER QMGR for more details.

Class attributes

behavior

Controls the behavior of implicit connection and disconnection.

IMQ_EXPL_DISC_BACKOUT (0L)
An explicit call to the disconnect method implies backout. This attribute is mutually exclusive with IMQ_EXPL_DISC_COMMIT.

IMQ_EXPL_DISC_COMMIT (1L)
An explicit call to the disconnect method implies commit (the default). This attribute is mutually exclusive with IMQ_EXPL_DISC_BACKOUT.

IMQ_IMPL_CONN (2L)
Implicit connection is allowed (the default).

IMQ_IMPL_DISC_BACKOUT (0L)
An implicit call to the disconnect method, which can occur during object destruction, implies backout. This attribute is mutually exclusive with the IMQ_IMPL_DISC_COMMIT.

IMQ_IMPL_DISC_COMMIT (4L)
An implicit call to the disconnect method, which can occur during object destruction, implies commit (the default). This attribute is mutually exclusive with IMQ_IMPL_DISC_BACKOUT.

At IBM MQ V7.0 and above, C++ applications that make use of an implicit connection, need to specify IMQ_IMPL_CONN along with any other options provided in the `setBehavior()` method on an object of class `ImqQueueManager`. If your application does not use the `setBehavior()` method to explicitly set the behavior options, for example,

```
ImqQueueManager_object.setBehavior(IMQ_IMPL_DISC_COMMIT)
```

this change does not affect you since MQ_IMPL_CONN is enabled by default.

If your application explicitly sets the behavior options, for example,

```
ImqQueueManager_object.setBehavior(IMQ_IMPL_DISC_COMMIT)
```

you need to include IMQ_IMPL_CONN in the `setBehavior()` method as follows, to allow your application to complete an implicit connection:

```
ImqQueueManager_object.setBehavior(IMQ_IMPL_CONN | IMQ_IMPL_DISC_COMMIT)
```

Object attributes

accounting connections override
Allows applications to override the setting of the MQI accounting and queue accounting values.This attribute is read-only.

accounting interval
How long before intermediate accounting records are written (in seconds). This attribute is read-only.

activity recording
Controls the generation of activity reports. This attribute is read-only.

adopt new mca check
The elements checked to determine if an MCA should be adopted when a new inbound channel is detected that has the same name as an MCA that is already active. This attribute is read-only.

adopt new mca type
Whether an orphaned instance of an MCA of a particular channel type should be restarted automatically when a new inbound channel request matching the adopt new mca check parameters is detected. This attribute is read-only.

authentication type
Indicates the type of authentication which is being performed.

authority event
Controls authority events. This attribute is read-only.

begin options
Options that apply to the begin method. The initial value is MQBO_NONE.

bridge event
Whether IMS bridge events are generated. This attribute is read-only.

channel auto definition
Channel auto definition value. This attribute is read-only.

channel auto definition event
Channel auto definition event value. This attribute is read-only.

channel auto definition exit
Channel auto definition exit name. This attribute is read-only.

channel event
Whether channel events are generated. This attribute is read-only.

channel initiator adapters
The number of adapter subtasks to use for processing IBM MQ calls. This attribute is read-only.

channel initiator control
Whether the Channel Initiator should be started automatically when the Queue Manager is started. This attribute is read-only.

channel initiator dispatchers
The number of dispatchers to use for the channel initiator. This attribute is read-only.

channel initiator trace autostart
Whether channel initiator trace should start automatically or not. This attribute is read-only.

channel initiator trace table size
The size of the channel initiator's trace data space (in MB). This attribute is read-only.

channel monitoring
Controls the collection of online monitoring data for channels. This attribute is read-only.

channel reference
A reference to a channel definition for use during client connection. While connected, this attribute can be set to null, but cannot be changed to any other value. The initial value is null.

channel statistics
Controls the collection of statistics data for channels. This attribute is read-only.

character set
Coded character set identifier (CCSID). This attribute is read-only.

cluster sender monitoring
Controls the collection of online monitoring data for automatically-defined cluster sender channels. This attribute is read-only.

cluster sender statistics
Controls the collection of statistics data for automatically defined cluster sender channels. This attribute is read-only.

cluster workload data
Cluster workload exit data. This attribute is read-only.

cluster workload exit
Cluster workload exit name. This attribute is read-only.

cluster workload length
Cluster workload length. This attribute is read-only.

cluster workload mru
Cluster workload most recently used channels value. This attribute is read-only.

cluster workload use queue
Cluster workload use queue value. This attribute is read-only.

command event
Whether command events are generated. This attribute is read-only.

command input queue name
System command input queue name. This attribute is read-only.

command level
Command level supported by the queue manager. This attribute is read-only.

command server control
Whether the Command Server should be started automatically when the Queue Manager is started. This attribute is read-only.

connect options
Options that apply to the connect method. The initial value is MQCNO_NONE. The following additional values may be possible, depending on platform:
- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING
- MQCNO_HANDLE_SHARE_NONE
- MQCNO_HANDLE_SHARE_BLOCK
- MQCNO_HANDLE_SHARE_NO_BLOCK
- MQCNO_SERIALIZE_CONN_TAG_Q_MGR
- MQCNO_SERIALIZE_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_Q_MGR
- MQCNO_RESTRICT_CONN_TAG_QSG

connection id
A unique identifier that allows MQ to reliably identify an application.

connection status
TRUE when connected to the queue manager. This attribute is read-only.

connection tag
A tag to be associated with a connection. This attribute can only be set when not connected. The initial value is null.

cryptographic hardware
Configuration details for cryptographic hardware. For MQ MQI client connections.

dead-letter queue name
Name of the dead-letter queue. This attribute is read-only.

default transmission queue name
Default transmission queue name. This attribute is read-only.

distribution lists
Capability of the queue manager to support distribution lists.

dns group
The name of the group that the TCP listener that handles inbound transmissions for the queue-sharing group should join when using Workload Manager Dynamic Domain Name Services support. This attribute is read-only.

dns wlm
Whether the TCP listener that handles inbound transmissions for the queue-sharing group should register with Workload Manager for Dynamic Domain Name Services. This attribute is read-only.

first authentication record
The first of one or more objects of class ImqAuthenticationRecord, in no particular order, in which the ImqAuthenticationRecord connection reference addresses this object. For MQ MQI client connections.

first managed object
The first of one or more objects of class ImqObject, in no particular order, in which the ImqObject connection reference addresses this object. The initial value is zero.

inhibit event
Controls inhibit events. This attribute is read-only.

ip address version
Which IP protocol (IPv4 or IPv6) to use for a channel connection. This attribute is read-only.

key repository
Location of the key database file in which keys and certificates are stored. For IBM MQ MQI client connections.

key reset count
The number of unencrypted bytes sent and received within a TLS conversation before the secret key is renegotiated. This attribute applies only to client connections using MQCONNX. See also ssl key reset count.

listener timer
The time interval (in seconds) between attempts by IBM MQ to restart the listener if there has been an APPC or TCP/IP failure. This attribute is read-only.

local event
Controls local events. This attribute is read-only.

logger event
Controls whether recovery log events are generated. This attribute is read-only.

lu group name
The generic LU name that the LU 6.2 listener that handles inbound transmissions for the queue-sharing group should use. This attribute is read-only.

lu name
The name of the LU to use for outbound LU 6.2 transmissions. This attribute is read-only.

lu62 arm suffix
The suffix of the SYS1.PARMLIB member APPCPMxx, that nominates the LUADD for this channel initiator. This attribute is read-only.

lu62 channels
The maximum number of channels that can be current or clients that can be connected, that use the LU 6.2 transmission protocol. This attribute is read-only.

maximum active channels
The maximum number of channels that can be active at any time. This attribute is read-only.

maximum channels
The maximum number of channels that can be current (including server-connection channels with connected clients). This attribute is read-only.

maximum handles
Maximum number of handles. This attribute is read-only.

maximum message length
Maximum possible length for any message on any queue managed by this queue manager. This attribute is read-only.

maximum priority
Maximum message priority. This attribute is read-only.

maximum uncommitted messages
Maximum number of uncommitted messages within a unit or work. This attribute is read-only.

mqi accounting
Controls the collection of accounting information for MQI data. This attribute is read-only.

mqi statistics
Controls the collection of statistics monitoring information for the queue manager. This attribute is read-only.

outbound port maximum
The higher end of the range of port numbers to be used when binding outgoing channels. This attribute is read-only.

outbound port minimum
The lower end of the range of port numbers to be used when binding outgoing channels. This attribute is read-only.

password
password associated with user ID

performance event
Controls performance events. This attribute is read-only.

platform
Platform on which the queue manager resides. This attribute is read-only.

queue accounting
Controls the collection of accounting information for queues. This attribute is read-only.

queue monitoring
Controls the collection of online monitoring data for queues. This attribute is read-only.

queue statistics
Controls the collection of statistics data for queues. This attribute is read-only.

receive timeout
Approximately how long a TCP/IP message channel will wait to receive data, including heartbeats, from its partner, before returning to the inactive state. This attribute is read-only.

receive timeout minimum
The minimum time that a TCP/IP channel will wait to receive data, including heartbeats, from its partner, before returning to the inactive state. This attribute is read-only.

receive timeout type
A qualifier applied to receive timeout. This attribute is read-only.

remote event
Controls remote events. This attribute is read-only.

repository name
Repository name. This attribute is read-only.

repository namelist
Repository namelist name. This attribute is read-only.

shared queue manager name
Whether MQOPENs of a shared queue where the ObjectQMgrName is another queue manager in the queue-sharing group should resolve to an open of the shared queue on the local queue manager. This attribute is read-only.

ssl event
Whether SSL events are generated. This attribute is read-only.

ssl FIPS required
Whether only FIPS-certified algorithms should be used if the cryptography is executed in IBM MQ software. This attribute is read-only.

ssl key reset count
The number of unencrypted bytes sent and received within an SSL conversation before the secret key is renegotiated. This attribute is read-only.

start-stop event
Controls start-stop events. This attribute is read-only.

statistics interval
How often statistics monitoring data is written to the monitoring queue. This attribute is read-only.

syncpoint availability
Availability of syncpoint participation. This attribute is read-only.

Note: Queue manager-coordinated global units of work are not supported on the IBM i platform.

You can program a unit of work, externally coordinated by IBM i, using the _Rcommit and _Rback native system calls. Start this type of unit of work by starting the IBM MQ application under job-level commitment control using the STRCMTCTL command. See Interfaces to the IBM i external syncpoint manager for further details. Backout and commit are supported on the IBM i platform for local units of work coordinated by a queue manager.

tcp channels
The maximum number of channels that can be current or clients that can be connected, that use the TCP/IP transmission protocol. This attribute is read-only.

tcp keepalive
Whether the TCP KEEPALIVE facility is to be used to check that the other end of the connection is still available. This attribute is read-only.

tcp name
The name of either the sole or default TCP/IP system to be used, depending on the value of tcp stack type. This attribute is read-only.

tcp stack type
Whether the channel initiator is permitted to only use the TCP/IP address space specified in tcp name or can bind to any selected TCP/IP address. This attribute is read-only.

trace route recording
Controls the recording of route tracing information. This attribute is read-only.

trigger interval
Trigger interval. This attribute is read-only.

user id
On UNIX and Linux platforms, the application's real user ID. On Windows platforms, the application's user ID.

Constructors

ImqQueueManager( );
The default constructor.

ImqQueueManager(const ImqQueueManager & *manager*);
The copy constructor. The connection status will be FALSE.

ImqQueueManager(const char * *name*);
Sets the ImqObject name to *name*.

Destructors

When an ImqQueueManager object is destroyed, it is automatically disconnected.

Class methods (public)

static MQLONG behavior( );
Returns the behavior.

void setBehavior(const MQLONG *behavior* = 0);
Sets the behavior.

Object methods (public)

void operator = (const ImqQueueManager & *mgr*);
Disconnects if necessary, and copies instance data from *mgr*. The connection status is be FALSE.

ImqBoolean accountingConnOverride (MQLONG & statint);
Provides a copy of the accounting connections override value. It returns TRUE if successful.

MQLONG accountingConnOverride ( );
Returns the accounting connections override value without any indication of possible errors.

ImqBoolean accountingInterval (MQLONG & statint);
Provides a copy of the accounting interval value. It returns TRUE if successful.

MQLONG accountingInterval ( );
Returns the accounting interval value without any indication of possible errors.

ImqBoolean activityRecording (MQLONG & rec);
Provides a copy of the activity recording value. It returns TRUE if successful.

MQLONG activityRecording ( );
Returns the activity recording value without any indication of possible errors.

ImqBoolean adoptNewMCACheck (MQLONG & check);
Provides a copy of the adopt new MCA check value. It returns TRUE if successful.

MQLONG adoptNewMCACheck ( );
Returns the adopt new MCA check value without any indication of possible errors.

ImqBoolean adoptNewMCAType (MQLONG & type);
Provides a copy of the adopt new MCA type. It returns TRUE if successful.

MQLONG adoptNewMCAType ( );
Returns the adopt new MCA type without any indication of possible errors.

QLONG authenticationType ( ) const;
Returns the authentication type.

void setAuthenticationType (const MQLONG type = MQCSP_AUTH_NONE);
Sets the authentication type.

ImqBoolean authorityEvent(MQLONG & ***event*** **);**
Provides a copy of the enablement state of the authority event. It returns TRUE if successful.

MQLONG authorityEvent( );
Returns the enablement state of the authority event without any indication of possible errors.

ImqBoolean backout( );
Backs out uncommitted changes. It returns TRUE if successful.

ImqBoolean begin( );
Begins a unit of work. The begin options affect the behavior of this method. It returns TRUE if successful, but it also returns TRUE even if the underlying MQBEGIN call returns MQRC_NO_EXTERNAL_PARTICIPANTS or MQRC_PARTICIPANT_NOT_AVAILABLE (which are both associated with MQCC_WARNING).

MQLONG beginOptions( ) const ;
Returns the begin options.

void setBeginOptions(const MQLONG ***options*** **= MQBO_NONE);**
Sets the begin options.

ImqBoolean bridgeEvent (MQLONG & event);
Provides a copy of the bridge event value. It returns TRUE if successful.

MQLONG bridgeEvent ( );
Returns the bridge event value without any indication of possible errors.

ImqBoolean channelAutoDefinition(MQLONG & ***value*** **);**
Provides a copy of the channel auto definition value. It returns TRUE if successful.

MQLONG channelAutoDefinition( );
Returns the channel auto definition value without any indication of possible errors.

ImqBoolean channelAutoDefinitionEvent(MQLONG & *value*);
Provides a copy of the channel auto definition event value. It returns TRUE if successful.

MQLONG channelAutoDefinitionEvent( );
Returns the channel auto definition event value without any indication of possible errors.

ImqBoolean channelAutoDefinitionExit(ImqString & *name*);
Provides a copy of the channel auto definition exit name. It returns TRUE if successful.

ImqString channelAutoDefinitionExit( );
Returns the channel auto definition exit name without any indication of possible errors.

ImqBoolean channelEvent (MQLONG & event);
Provides a copy of the channel event value. It returns TRUE if successful.

MQLONG channelEvent( );
Returns the channel event value without any indication of possible errors.

MQLONG channelInitiatorAdapters ( );
Returns the channel initiator adapters value without any indication of possible errors.

ImqBoolean channelInitiatorAdapters (MQLONG & adapters);
Provides a copy of the channel initiator adapters value. It returns TRUE if successful.

MQLONG channelInitiatorControl ( );
Returns the channel initiator startup value without any indication of possible errors.

ImqBoolean channelInitiatorControl (MQLONG & init);
Provides a copy of the channel initiator control startup value. It returns TRUE if successful.

MQLONG channelInitiatorDispatchers ( );
Returns the channel initiator dispatchers value without any indication of possible errors.

ImqBoolean channelInitiatorDispatchers (MQLONG & dispatchers);
Provides a copy of the channel initiator dispatchers value. It returns TRUE if successful.

MQLONG channelInitiatorTraceAutoStart ( );
Returns the channel initiator trace auto start value without any indication of possible errors.

ImqBoolean channelInitiatorTraceAutoStart (MQLONG & auto);
Provides a copy of the channel initiator trace auto start value. It returns TRUE if successful.

MQLONG channelInitiatorTraceTableSize ( );
Returns the channel initiator trace table size value without any indication of possible errors.

ImqBoolean channelInitiatorTraceTableSize (MQLONG & size);
Provides a copy of the channel initiator trace table size value. It returns TRUE if successful.

ImqBoolean channelMonitoring (MQLONG & monchl);
Provides a copy of the channel monitoring value. It returns TRUE if successful.

MQLONG channelMonitoring ( );
Returns the channel monitoring value without any indication of possible errors.

ImqBoolean channelReference(ImqChannel * & *pchannel*);
Provides a copy of the channel reference. If the channel reference is invalid, sets *pchannel* to null. This method returns TRUE if successful.

ImqChannel * channelReference( );
Returns the channel reference without any indication of possible errors.

ImqBoolean setChannelReference(ImqChannel & *channel*);
Sets the channel reference. This method returns TRUE if successful.

ImqBoolean setChannelReference(ImqChannel * *channel* = 0);
Sets or resets the channel reference. This method returns TRUE if successful.

ImqBoolean channelStatistics (MQLONG & statchl);
Provides a copy of the channel statistics value. It returns TRUE if successful.

MQLONG channelStatistics ( );
Returns the channel statistics value without any indication of possible errors.

ImqBoolean characterSet(MQLONG & ***ccsid*** **);**
Provides a copy of the character set. It returns TRUE if successful.

MQLONG characterSet( );
Returns a copy of the character set, without any indication of possible errors.

MQLONG clientSslKeyResetCount ( ) const;
Returns the SSL key reset count value used on client connections.

void setClientSslKeyResetCount(const MQLONG count);
Sets the SSL key reset count used on client connections.

ImqBoolean clusterSenderMonitoring (MQLONG & monacls);
Provides a copy of the cluster sender monitoring default value. It returns TRUE if successful.

MQLONG clusterSenderMonitoring ( );
Returns the cluster sender monitoring default value without any indication of possible errors.

ImqBoolean clusterSenderStatistics (MQLONG & statacls);
Provides a copy of the cluster sender statistics value. It returns TRUE if successful.

MQLONG clusterSenderStatistics ( );
Returns the cluster sender statistics value without any indication of possible errors.

ImqBoolean clusterWorkloadData(ImqString & ***data*** **);**
Provides a copy of the cluster workload exit data. It returns TRUE if successful.

ImqString clusterWorkloadData( );
Returns the cluster workload exit data without any indication of possible errors.

ImqBoolean clusterWorkloadExit(ImqString & ***name*** **);**
Provides a copy of the cluster workload exit name. It returns TRUE if successful.

ImqString clusterWorkloadExit( );
Returns the cluster workload exit name without any indication of possible errors.

ImqBoolean clusterWorkloadLength(MQLONG & ***length*** **);**
Provides a copy of the cluster workload length. It returns TRUE if successful.

MQLONG clusterWorkloadLength( );
Returns the cluster workload length without any indication of possible errors.

ImqBoolean clusterWorkLoadMRU (MQLONG & mru);
Provides a copy of the cluster workload most recently used channels value. It returns TRUE if successful.

MQLONG clusterWorkLoadMRU ( );
Returns the cluster workload most recently used channels value without any indication of possible errors.

ImqBoolean clusterWorkLoadUseQ (MQLONG & useq);
Provides a copy of the cluster workload use queue value. It returns TRUE if successful.

MQLONG clusterWorkLoadUseQ ( );
Returns the cluster workload use queue value without any indication of possible errors.

ImqBoolean commandEvent (MQLONG & event);
Provides a copy of the command event value. It returns TRUE if successful.

MQLONG commandEvent ( );
Returns the command event value without any indication of possible errors.

ImqBoolean commandInputQueueName(ImqString & *name*);
Provides a copy of the command input queue name. It returns TRUE if successful.

ImqString commandInputQueueName( );
Returns the command input queue name without any indication of possible errors.

ImqBoolean commandLevel(MQLONG & *level*);
Provides a copy of the command level. It returns TRUE if successful.

MQLONG commandLevel( );
Returns the command level without any indication of possible errors.

MQLONG commandServerControl ( );
Returns the command server startup value without any indication of possible errors.

ImqBoolean commandServerControl (MQLONG & server);
Provides a copy of the command server control startup value. It returns TRUE if successful.

ImqBoolean commit( );
Commits uncommitted changes. It returns TRUE if successful.

ImqBoolean connect( );
Connects to the queue manager with the given ImqObject name, the default being the local queue manager. If you want to connect to a specific queue manager, use the ImqObject setName method before connection. If there is a channel reference, it is used to pass information about the channel definition to MQCONNX in an MQCD. The ChannelType in the MQCD is set to MQCHT_CLNTCONN. channel reference information, which is only meaningful for client connections, is ignored for server connections. The connect options affect the behavior of this method. This method sets the connection status to TRUE if successful. It returns the new connection status.

If there is a first authentication record, the chain of authentication records is used to authenticate digital certificates for secure client channels.

You can connect more than one ImqQueueManager object to the same queue manager. All use the same MQHCONN connection handle and share UOW functionality for the connection associated with the thread. The first ImqQueueManager to connect obtains the MQHCONN handle. The last ImqQueueManager to disconnect performs the MQDISC.

For a multithreaded program, it is recommended that a separate ImqQueueManager object is used for each thread.

ImqBinary connectionId ( ) const ;
Returns the connection ID.

ImqBinary connectionTag ( ) const ;
Returns the connection tag.

ImqBoolean setConnectionTag (const MQBYTE128 *tag* = 0);
Sets the connection tag. If *tag* is zero, clears the connection tag. This method returns TRUE if successful.

ImqBoolean setConnectionTag (const ImqBinary & *tag*);
Sets the connection tag. The data length of *tag* must be either zero (to clear the connection tag) or MQ_CONN_TAG_LENGTH. This method returns TRUE if successful.

MQLONG connectOptions( ) const ;
Returns the connect options.

void setConnectOptions(const MQLONG *options* = MQCNO_NONE);
Sets the connect options.

ImqBoolean connectionStatus( ) const ;
Returns the connection status.

ImqString cryptographicHardware ( );
Returns the cryptographic hardware.

ImqBoolean setCryptographicHardware (const char * *hardware* = 0);
Sets the cryptographic hardware. This method returns TRUE if successful.

ImqBoolean deadLetterQueueName(ImqString & *name*);
Provides a copy of the dead-letter queue name. It returns TRUE if successful.

ImqString deadLetterQueueName( );
Returns a copy of the dead-letter queue name, without any indication of possible errors.

ImqBoolean defaultTransmissionQueueName(ImqString & *name*);
Provides a copy of the default transmission queue name. It returns TRUE if successful.

ImqString defaultTransmissionQueueName( );
Returns the default transmission queue name without any indication of possible errors.

ImqBoolean disconnect( );
Disconnects from the queue manager and sets the connection status to FALSE. Closes all ImqProcess and ImqQueue objects associated with this object, and severs their connection reference before disconnection. If more than one ImqQueueManager object is connected to the same queue manager, only the last to disconnect performs a physical disconnection; others perform a logical disconnection. Uncommitted changes are committed on physical disconnection only.

This method returns TRUE if successful. If it is called when there is no existing connection, the return code is also true.

ImqBoolean distributionLists(MQLONG & *support*);
Provides a copy of the distribution lists value. It returns TRUE if successful.

MQLONG distributionLists( );
Returns the distribution lists value without any indication of possible errors.

ImqBoolean dnsGroup (ImqString & group);
Provides a copy of the DNS group name. It returns TRUE if successful.

ImqString dnsGroup ( );
Returns the DNS group name without any indication of possible errors.

ImqBoolean dnsWlm (MQLONG & wlm);
Provides a copy of the DNS WLM value. It returns TRUE if successful.

MQLONG dnsWlm ( );
Returns the DNS WLM value without any indication of possible errors.

ImqAuthenticationRecord * firstAuthenticationRecord ( ) const ;
Returns the first authentication record.

void setFirstAuthenticationRecord (const ImqAuthenticationRecord * *air* = 0);
Sets the first authentication record.

ImqObject * firstManagedObject( ) const ;
Returns the first managed object.

ImqBoolean inhibitEvent(MQLONG & *event*);
Provides a copy of the enablement state of the inhibit event. It returns TRUE if successful.

MQLONG inhibitEvent( );
Returns the enablement state of the inhibit event without any indication of possible errors.

ImqBoolean ipAddressVersion (MQLONG & version);
Provides a copy of the IP address version value. It returns TRUE if successful.

MQLONG ipAddressVersion ( );
Returns the IP address version value without any indication of possible errors.

ImqBoolean keepAlive (MQLONG & keepalive);
Provides a copy of the keep alive value. It returns TRUE if successful.

MQLONG keepAlive ( );
Returns the keep alive value without any indication of possible errors.

ImqString keyRepository ( );
Returns the key repository.

ImqBoolean setKeyRepository (const char * *repository* = 0);
Sets the key repository. It returns TRUE if successful.

ImqBoolean listenerTimer (MQLONG & timer);
Provides a copy of the listener timer value. It returns TRUE if successful.

MQLONG listenerTimer ( );
Returns the listener timer value without any indication of possible errors.

ImqBoolean localEvent(MQLONG & ***event*** **);**
Provides a copy of the enablement state of the local event. It returns TRUE if successful.

MQLONG localEvent( );
Returns the enablement state of the local event without any indication of possible errors.

ImqBoolean loggerEvent (MQLONG & count);
Provides a copy of the logger event value. It returns TRUE if successful.

MQLONG loggerEvent ( );
Returns the logger event value without any indication of possible errors.

ImqBoolean luGroupName (ImqString & name);.
Provides a copy of the LU group name. It returns TRUE if successful

ImqString luGroupName ( );
Returns the LU group name without any indication of possible errors.

ImqBoolean lu62ARMSuffix (ImqString & suffix);
Provides a copy of the LU62 ARM suffix. It returns TRUE if successful.

ImqString lu62ARMSuffix ( );
Returns the LU62 ARM suffix without any indication of possible errors

ImqBoolean luName (ImqString & name);
Provides a copy of the LU name. It returns TRUE if successful.

ImqString luName ( );
Returns the LU name without any indication of possible errors.

ImqBoolean maximumActiveChannels (MQLONG & channels);
Provides a copy of the maximum active channels value. It returns TRUE if successful.

MQLONG maximumActiveChannels ( );
Returns the maximum active channels value without any indication of possible errors.

ImqBoolean maximumCurrentChannels (MQLONG & channels);
Provides a copy of the maximum current channels value. It returns TRUE if successful.

MQLONG maximumCurrentChannels ( );
Returns the maximum current channels value without any indication of possible errors.

ImqBoolean maximumHandles(MQLONG & *number*);
Provides a copy of the maximum handles. It returns TRUE if successful.

MQLONG maximumHandles( );
Returns the maximum handles without any indication of possible errors.

ImqBoolean maximumLu62Channels (MQLONG & channels);
Provides a copy of the maximum LU62 channels value. It returns TRUE if successful.

MQLONG maximumLu62Channels ( );.
Returns the maximum LU62 channels value without any indication of possible errors

ImqBoolean maximumMessageLength(MQLONG & *length*);
Provides a copy of the maximum message length. It returns TRUE if successful.

MQLONG maximumMessageLength( );
Returns the maximum message length without any indication of possible errors.

ImqBoolean maximumPriority(MQLONG & *priority*);
Provides a copy of the maximum priority. It returns TRUE if successful.

MQLONG maximumPriority( );
Returns a copy of the maximum priority, without any indication of possible errors.

ImqBoolean maximumTcpChannels (MQLONG & channels);
Provides a copy of the maximum TCP channels value. It returns TRUE if successful.

MQLONG maximumTcpChannels ( );
Returns the maximum TCP channels value without any indication of possible errors.

ImqBoolean maximumUncommittedMessages(MQLONG & *number*);
Provides a copy of the maximum uncommitted messages. It returns TRUE if successful.

MQLONG maximumUncommittedMessages( );
Returns the maximum uncommitted messages without any indication of possible errors.

ImqBoolean mqiAccounting (MQLONG & statint);
Provides a copy of the MQI accounting value. It returns TRUE if successful.

MQLONG mqiAccounting ( );
Returns the MQI accounting value without any indication of possible errors.

ImqBoolean mqiStatistics (MQLONG & statmqi);
Provides a copy of the MQI statistics value. It returns TRUE if successful.

MQLONG mqiStatistics ( );
Returns the MQI statistics value without any indication of possible errors.

ImqBoolean outboundPortMax (MQLONG & max);
Provides a copy of the maximum outbound port value. It returns TRUE if successful.

MQLONG outboundPortMax ( );
Returns the maximum outbound port value without any indication of possible errors.

ImqBoolean outboundPortMin (MQLONG & min);
Provides a copy of the minimum outbound port value. It returns TRUE if successful.

MQLONG outboundPortMin ( );
Returns the minimum outbound port value without any indication of possible errors.

ImqBinary password ( ) const;
Returns the password used on client connections.

ImqBoolean setPassword (const ImqString & password);
Sets the password used on client connections.

ImqBoolean setPassword (const char * = 0 password);
Sets the password used on client connections.

ImqBoolean setPassword (const ImqBinary & password);
Sets the password used on client connections.

ImqBoolean performanceEvent(MQLONG & *event* **);**
Provides a copy of the enablement state of the performance event. It returns TRUE if successful.

MQLONG performanceEvent( );
Returns the enablement state of the performance event without any indication of possible errors.

ImqBoolean platform(MQLONG & *platform* **);**
Provides a copy of the platform. It returns TRUE if successful.

MQLONG platform( );
Returns the platform without any indication of possible errors.

ImqBoolean queueAccounting (MQLONG & acctq);
Provides a copy of the queue accounting value. It returns TRUE if successful.

MQLONG queueAccounting ( );
Returns the queue accounting value without any indication of possible errors.

ImqBoolean queueMonitoring (MQLONG & monq);
Provides a copy of the queue monitoring value. It returns TRUE if successful.

MQLONG queueMonitoring ( );
Returns the queue monitoring value without any indication of possible errors.

ImqBoolean queueStatistics (MQLONG & statq);
Provides a copy of the queue statistics value. It returns TRUE if successful.

MQLONG queueStatistics ( );
Returns the queue statistics value without any indication of possible errors.

ImqBoolean receiveTimeout (MQLONG & timeout);
Provides a copy of the receive timeout value. It returns TRUE if successful.

MQLONG receiveTimeout ( );
Returns the receive timeout value without any indication of possible errors.

ImqBoolean receiveTimeoutMin (MQLONG & min);
Provides a copy of the minimum receive timeout value. It returns TRUE if successful.

MQLONG receiveTimeoutMin ( );
Returns the minimum receive timeout value without any indication of possible errors.

ImqBoolean receiveTimeoutType (MQLONG & type);
Provides a copy of the receive timeout type. It returns TRUE if successful.

MQLONG receiveTimeoutType ( );
Returns the receive timeout type without any indication of possible errors.

ImqBoolean remoteEvent(MQLONG & *event* **);**
Provides a copy of the enablement state of the remote event. It returns TRUE if successful.

MQLONG remoteEvent( );
Returns the enablement state of the remote event without any indication of possible errors.

ImqBoolean repositoryName(ImqString & *name* **);**
Provides a copy of the repository name. It returns TRUE if successful.

ImqString repositoryName( );
Returns the repository name without any indication of possible errors.

ImqBoolean repositoryNamelistName(ImqString & *name*);
Provides a copy of the repository namelist name. It returns TRUE if successful.

ImqString repositoryNamelistName( );
Returns a copy of the repository namelist name without any indication of possible errors.

ImqBoolean sharedQueueQueueManagerName (MQLONG & name);
Provides a copy of the shared queue queue manager name value. It returns TRUE if successful.

MQLONG sharedQueueQueueManagerName ( );
Returns the shared queue queue manager name value without any indication of possible errors.

ImqBoolean sslEvent (MQLONG & event);
Provides a copy of the SSL event value. It returns TRUE if successful.

MQLONG sslEvent ( );
Returns the SSL event value without any indication of possible errors.

ImqBoolean sslFips (MQLONG & sslfips);
Provides a copy of the SSL FIPS value. It returns TRUE if successful.

MQLONG sslFips ( );
Returns the SSL FIPS value without any indication of possible errors.

ImqBoolean sslKeyResetCount (MQLONG & count);
Provides a copy of the SSL key reset count value. It returns TRUE if successful.

MQLONG sslKeyResetCount ( );
Returns the SSL key reset count value without any indication of possible errors.

ImqBoolean startStopEvent(MQLONG & *event*);
Provides a copy of the enablement state of the start-stop event. It returns TRUE if successful.

MQLONG startStopEvent( );
Returns the enablement state of the start-stop event without any indication of possible errors.

ImqBoolean statisticsInterval (MQLONG & statint);
Provides a copy of the statistics interval value. It returns TRUE if successful.

MQLONG statisticsInterval ( );
Returns the statistics interval value without any indication of possible errors.

ImqBoolean syncPointAvailability(MQLONG & *sync*);
Provides a copy of the syncpoint availability value. It returns TRUE if successful.

MQLONG syncPointAvailability( );
Returns a copy of the syncpoint availability value, without any indication of possible errors.

ImqBoolean tcpName (ImqString & name);
Provides a copy of the TCP system name. It returns TRUE if successful.

ImqString tcpName ( );
Returns the TCP system name without any indication of possible errors.

ImqBoolean tcpStackType (MQLONG & type);
Provides a copy of the TCP stack type. It returns TRUE if successful.

MQLONG tcpStackType ( );
Returns the TCP stack type without any indication of possible errors.

ImqBoolean traceRouteRecording (MQLONG & routerec);
Provides a copy of the trace route recording value. It returns TRUE if successful.

MQLONG traceRouteRecording ( );
Returns the trace route recording value without any indication of possible errors.

ImqBoolean triggerInterval(MQLONG & *interval*);
Provides a copy of the trigger interval. It returns TRUE if successful.

MQLONG triggerInterval( );
Returns the trigger interval without any indication of possible errors.

ImqBinary userId ( ) const;
Returns the user ID used on client connections.

ImqBoolean setUserId (const ImqString & id);
Sets the user ID used on client connections.

ImqBoolean setUserId (const char * = 0 id);
Sets the user ID used on client connections.

ImqBoolean setUserId (const ImqBinary & id);
Sets the user ID used on client connections.

Object methods (protected)

void setFirstManagedObject (const ImqObject * *object* = 0);
Sets the first managed object.

Object data (protected)

MQHCONN *ohconn*
The IBM MQ connection handle (meaningful only while the connection status is TRUE).

Reason codes

- MQRC_ATTRIBUTE_LOCKED
- MQRC_ENVIRONMENT_ERROR
- MQRC_FUNCTION_NOT_SUPPORTED
- MQRC_REFERENCE_ERROR
- (reason codes for MQBACK)
- (reason codes for MQBEGIN)
- (reason codes for MQCMIT)
- (reason codes for MQCONNX)
- (reason codes for MQDISC)
- (reason codes for MQCONN)

ImqReferenceHeader C++ class

This class encapsulates features of the MQRMH data structure.

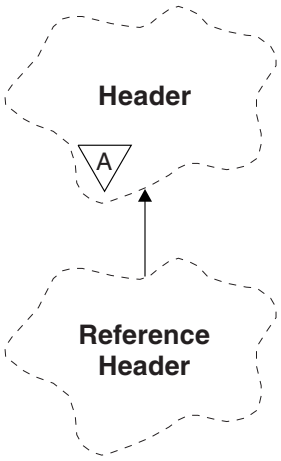


Figure 118. ImqReferenceHeader class

This class relates to the MQI calls listed in "ImqReferenceHeader cross-reference" on page 3936.

Object attributes

destination environment
Environment for the destination. The initial value is a null string.

destination name
Name of the data destination. The initial value is a null string.

instance id
Instance identifier. A binary value (MQBYTE24) of length MQ_OBJECT_INSTANCE_ID_LENGTH. The initial value is MQOII_NONE.

logical length
Logical, or intended, length of message data that follows this header. The initial value is zero.

logical offset
Logical offset for the message data that follows, to be interpreted in the context of the data as a whole, at the ultimate destination. The initial value is zero.

logical offset 2
High-order extension to the logical offset. The initial value is zero.

reference type
Reference type. The initial value is a null string.

source environment
Environment for the source. The initial value is a null string.

source name
Name of the data source. The initial value is a null string.

Constructors

ImqReferenceHeader( );
The default constructor.

ImqReferenceHeader(const ImqReferenceHeader & *header*);
The copy constructor.

Overloaded ImqItem methods

virtual ImqBoolean copyOut (ImqMessage & *msg*);
Inserts an MQRMH data structure into the message buffer at the beginning, moving existing message data further along, and sets the *msg* format to MQFMT_REF_MSG_HEADER.

See the ImqHeader class method description on "ImqHeader C++ class" on page 3965 for further details.

virtual ImqBoolean pasteIn (ImqMessage & *msg*);
Reads an MQRMH data structure from the message buffer.

To be successful, the ImqMessage format must be MQFMT_REF_MSG_HEADER.

See the ImqHeader class method description on "ImqHeader C++ class" on page 3965 for further details.

Object methods (public)

void operator = (const ImqReferenceHeader & *header*);
Copies instance data from *header*, replacing the existing instance data.

ImqString destinationEnvironment ( ) const ;
Returns a copy of the destination environment.

void setDestinationEnvironment (const char * *environment* = 0);
Sets the destination environment.

ImqString destinationName ( ) const ;
Returns a copy of the destination name.

void setDestinationName (const char * *name* = 0);
Sets the destination name.

ImqBinary instanceId ( ) const ;
Returns a copy of the instance id.

ImqBoolean setInstanceId (const ImqBinary & *id*);
Sets the instance id. The data length of *token* must be either 0 or MQ_OBJECT_INSTANCE_ID_LENGTH. This method returns TRUE if successful.

void setInstanceId (const MQBYTE24 *id* = 0);
Sets the instance id. *id* can be zero, which is the same as specifying MQOII_NONE. If *id* is nonzero, it must address MQ_OBJECT_INSTANCE_ID_LENGTH bytes of binary data. When using pre-defined values such as MQOII_NONE, you might need to make a cast to ensure a signature match, for example (MQBYTE *)MQOII_NONE.

MQLONG logicalLength ( ) const ;
Returns the logical length.

void setLogicalLength (const MQLONG *length*);
Sets the logical length.

MQLONG logicalOffset ( ) const ;
Returns the logical offset.

void setLogicalOffset (const MQLONG *offset*);
Sets the logical offset.

MQLONG logicalOffset2 ( ) const ;
Returns the logical offset 2.

void setLogicalOffset2 (const MQLONG *offset*);
Sets the logical offset 2.

ImqString referenceType ( ) const ;
Returns a copy of the reference type.

void setReferenceType (const char * *name* = 0);
Sets the reference type.

ImqString sourceEnvironment ( ) const ;
Returns a copy of the source environment.

void setSourceEnvironment (const char * *environment* = 0);
Sets the source environment.

ImqString sourceName ( ) const ;
Returns a copy of the source name.

void setSourceName (const char * *name* = 0);
Sets the source name.

Object data (protected)

MQRMH *omqrmh*
The MQRMH data structure.

Reason codes

- MQRC_BINARY_DATA_LENGTH_ERROR
- MQRC_STRUC_LENGTH_ERROR
- MQRC_STRUC_ID_ERROR
- MQRC_INSUFFICIENT_DATA
- MQRC_INCONSISTENT_FORMAT
- MQRC_ENCODING_ERROR

ImqString C++ class

This class provides character string storage and manipulation for null-terminated strings.

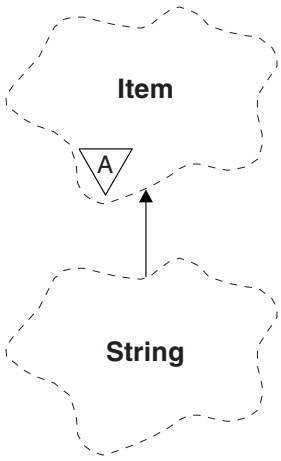


Figure 119. ImqString class

Use an ImqString in place of a **char *** in most situations where a parameter calls for a **char ***.

Object attributes

characters
Characters in the **storage** that precede a trailing null.

length Number of bytes in the **characters**. If there is no **storage**, the **length** is zero. The initial value is zero.

storage
A volatile array of bytes of arbitrary size. A trailing null must always be present in the **storage** after the **characters**, so that the end of the **characters** can be detected. Methods ensure that this situation is maintained, but ensure, when setting bytes in the array directly, that a trailing null exists after modification. Initially, there is no **storage** attribute.

Constructors

ImqString( );
The default constructor.

ImqString(const ImqString & *string*);
The copy constructor.

ImqString(const char *c*);
The **characters** comprise *c*.

ImqString(const char * *text*);
The **characters** are copied from *text*.

ImqString(const void * *buffer*, const size_t *length*);
Copies *length* bytes starting from *buffer* and assigns them to the **characters**. Substitution is made for any null characters copied. The substitution character is a period (.). No special consideration is given to any other non-printable or non-displayable characters copied.

Class methods (public)

static ImqBoolean copy(char * *destination-buffer*, const size_t *length*, const char * *source-buffer*, const char *pad* = 0);

Copies up to *length* bytes from *source-buffer* to *destination-buffer*. If the number of characters in *source-buffer* is insufficient, fills the remaining space in *destination-buffer* with *pad* characters. *source-buffer* can be zero. *destination-buffer* can be zero if *length* is also zero. Any error codes are lost. This method returns TRUE if successful.

static ImqBoolean copy (char * *destination-buffer*, const size_t *length*, const char * *source-buffer*, ImqError & *error-object*, const char *pad* = 0);

Copies up to *length* bytes from *source-buffer* to *destination-buffer*. If the number of characters in *source-buffer* is insufficient, fills the remaining space in *destination-buffer* with *pad* characters. *source-buffer* can be zero. *destination-buffer* can be zero if *length* is also zero. Any error codes are set in *error-object*. This method returns TRUE if successful.

Overloaded ImqItem methods

virtual ImqBoolean copyOut (ImqMessage & *msg*);

Copies the **characters** to the message buffer, replacing any existing content. Sets the *msg* **format** to MQFMT_STRING.

See the parent class method description for further details.

virtual ImqBoolean pasteIn (ImqMessage & *msg*);

Sets the **characters** by transferring the remaining data from the message buffer, replacing the existing **characters**.

To be successful, the **encoding** of the *msg* object must be MQENC_NATIVE. Retrieve messages with MQGMO_CONVERT to MQENC_NATIVE.

To be successful, the ImqMessage **format** must be MQFMT_STRING.

See the parent class method description for further details.

Object methods (public)

char & operator [ ] (const size_t *offset*) const ;

References the character at offset *offset* in the **storage**. Ensure that the relevant byte exists and is addressable.

ImqString operator ( ) (const size_t *offset*, const size_t *length* = 1) const ;

Returns a substring by copying bytes from the **characters** starting at *offset*. If *length* is zero, returns the rest of the **characters**. If the combination of *offset* and *length* does not produce a reference within the **characters**, returns an empty ImqString.

void operator = (const ImqString & *string*);

Copies instance data from *string*, replacing the existing instance data.

ImqString operator + (const char *c*) const ;

Returns the result of appending *c* to the **characters**.

ImqString operator + (const char * *text*) const ;

Returns the result of appending *text* to the **characters**. This can also be inverted. For example:

```
strOne + "string two" ;
"string one" + strTwo ;
```

Note: Although most compilers accept **strOne + "string two";** Microsoft Visual C++ requires **strOne + (char *)"string two" ;**

ImqString operator + (const ImqString & *string1*) const ;

Returns the result of appending *string1* to the **characters**.

ImqString operator + (const double *number*) const ;
Returns the result of appending *number* to the **characters** after conversion to text.

ImqString operator + (const long *number*) const ;
Returns the result of appending *number* to the **characters** after conversion to text.

void operator += (const char *c*);
Appends *c* to the **characters**.

void operator += (const char * *text*);
Appends *text* to the **characters**.

void operator += (const ImqString & *string*);
Appends *string* to the **characters**.

void operator += (const double *number*);
Appends *number* to the **characters** after conversion to text.

void operator += (const long *number*);
Appends *number* to the **characters** after conversion to text.

operator char * ( ) const ;
Returns the address of the first byte in the **storage**. This value can be zero, and is volatile. Use this method only for read-only purposes.

ImqBoolean operator < (const ImqString & *string*) const ;
Compares the **characters** with those of *string* using the **compare** method. The result is TRUE if less than and FALSE if greater than or equal to.

ImqBoolean operator > (const ImqString & *string*) const ;
Compares the **characters** with those of *string* using the **compare** method. The result is TRUE if greater than and FALSE if less than or equal to.

ImqBoolean operator <= (const ImqString & *string*) const ;
Compares the **characters** with those of *string* using the **compare** method. The result is TRUE if less than or equal to and FALSE if greater than.

ImqBoolean operator >= (const ImqString & *string*) const ;
Compares the **characters** with those of *string* using the **compare** method. The result is TRUE if greater than or equal to and FALSE if less than.

ImqBoolean operator == (const ImqString & *string*) const ;
Compares the **characters** with those of *string* using the **compare** method. It returns either TRUE or FALSE.

ImqBoolean operator != (const ImqString & *string*) const ;
Compares the **characters** with those of *string* using the **compare** method. It returns either TRUE or FALSE.

short compare(const ImqString & *string*) const ;
Compares the **characters** with those of *string*. The result is zero if the **characters** are equal, negative if less than and positive if greater than. Comparison is case sensitive. A null ImqString is regarded as less than a nonnull ImqString.

ImqBoolean copyOut(char * *buffer*, const size_t *length*, const char *pad* = 0);
Copies up to *length* bytes from the **characters** to the *buffer*. If the number of **characters** is insufficient, fills the remaining space in *buffer* with *pad* characters. *buffer* can be zero if *length* is also zero. It returns TRUE if successful.

size_t copyOut(long & *number*) const ;
Sets *number* from the **characters** after conversion from text, and returns the number of characters involved in the conversion. If this is zero, no conversion has been performed and *number* is not set. A convertible character sequence must begin with the following values:

```
<blank(s)>
<+|->
digit(s)
```

size_t copyOut(ImqString & *token*, const char *c* = ' ') const ;

If the **characters** contain one or more characters that are different from *c*, identifies a token as the first contiguous sequence of such characters. In this case *token* is set to that sequence, and the value returned is the sum of the number of leading characters *c* and the number of bytes in the sequence. Otherwise, returns zero and does not set *token*.

size_t cutOut(long & *number*);

Sets *number* as for the **copy** method, but also removes from **characters** the number of bytes indicated by the return value. For example, the string shown in the following example can be cut into three numbers by using **cutOut** (*number*) three times:

```
strNumbers = "-1 0   +55 "

while ( strNumbers.cutOut( number ) );
number becomes -1, then 0, then 55
leaving strNumbers == " "
```

size_t cutOut(ImqString & *token*, const char *c* = ' ' '

Sets *token* as for the **copyOut** method, and removes from **characters** the *strToken* characters and also any characters *c* that precede the *token* characters. If *c* is not a blank, removes characters *c* that directly succeed the *token* characters. Returns the number of characters removed. For example, the string shown in the following example can be cut into three tokens by using **cutOut** (*token*) three times:

```
strText = " Program Version 1.1 "

while ( strText.cutOut( token ) );

// token becomes "Program", then "Version",
// then "1.1" leaving strText == " "
```

The following example shows how to parse a DOS path name:

```
strPath = "C:\OS2\BITMAP\OS2LOGO.BMP"

strPath.cutOut( strDrive, ':' );
strPath.stripLeading( ':' );
while ( strPath.cutOut( strFile, '\' ) );

// strDrive becomes "C".
// strFile becomes "OS2", then "BITMAP",
// then "OS2LOGO.BMP" leaving strPath empty.
```

ImqBoolean find(const ImqString & *string*);

Searches for an exact match for *string* anywhere within the **characters**. If no match is found, it returns FALSE. Otherwise, it returns TRUE. If *string* is null, it returns TRUE.

ImqBoolean find(const ImqString & *string*, size_t & *offset*);

Searches for an exact match for *string* somewhere within the **characters** from offset *offset* onwards. If *string* is null, it returns TRUE without updating *offset*. If no match is found, it returns FALSE (the value of *offset* might have been increased). If a match is found, it returns TRUE and updates *offset* to the offset of *string* within the **characters**.

size_t length( ) const ;

Returns the **length**.

ImqBoolean pasteIn(const double *number*, const char * *format* = "%f");

Appends *number* to the **characters** after conversion to text. It returns TRUE if successful.

The specification *format* is used to format the floating point conversion. If specified, it must be one suitable for use with **printf** and floating point numbers, for example **%.3f**.

ImqBoolean pasteIn(const long *number*);
Appends *number* to the **characters** after conversion to text. It returns TRUE if successful.

ImqBoolean pasteIn(const void * *buffer*, const size_t *length*);
Appends *length* bytes from *buffer* to the **characters**, and adds a final trailing null. Substitutes any null characters copied. The substitution character is a period (.). No special consideration is given to any other nonprintable or nondisplayable characters copied. This method returns TRUE if successful.

ImqBoolean set(const char * *buffer*, const size_t *length*);
Sets the **characters** from a fixed-length character field, which might contain a null. Appends a null to the characters from the fixed-length field if necessary. This method returns TRUE if successful.

ImqBoolean setStorage(const size_t *length*);
Allocates (or reallocates) the **storage**. Preserves any original **characters**, including any trailing null, if there is still room for them, but does not initialize any additional storage.

This method returns TRUE if successful.

size_t storage( ) const ;
Returns the number of bytes in the **storage**.

size_t stripLeading(const char *c* = ' ');
Strips leading characters *c* from the **characters** and returns the number removed.

size_t stripTrailing(const char *c* = ' ');
Strips trailing characters *c* from the **characters** and returns the number removed.

ImqString upperCase( ) const ;
Returns an uppercase copy of the **characters**.

Object methods (protected)

ImqBoolean assign (const ImqString & *string*);
Equivalent to the equivalent **operator =** method, but non-virtual. It returns TRUE if successful.

Reason codes

- MQRC_DATA_TRUNCATED
- MQRC_NULL_POINTER
- MQRC_STORAGE_NOT_AVAILABLE
- MQRC_BUFFER_ERROR
- MQRC_INCONSISTENT_FORMAT

ImqTrigger C++ class

This class encapsulates the MQTM (trigger message) data structure.

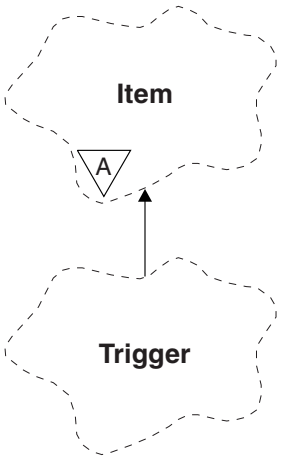


Figure 120. ImqTrigger class

Objects of this class are typically used by a trigger monitor program. The task of a trigger monitor program is to wait for these particular messages and act on them to ensure that other IBM MQ applications are started when messages are waiting for them.

See the IMQSTRG sample program for a usage example.

- "Object attributes"
- "Constructors" on page 4029
- "Overloaded ImqItem methods" on page 4029
- "Object methods (public)" on page 4029
- "Object data (protected)" on page 4030
- "Reason codes" on page 4030

Object attributes

application id
Identity of the application that sent the message. The initial value is a null string.

application type
Type of application that sent the message. The initial value is zero. The following additional values are possible:

- MQAT_AIX
- MQAT_CICS
- MQAT_DOS
- MQAT_IMS
- MQAT_MVS
- MQAT_NOTES_AGENT
- MQAT_OS2
- MQAT_OS390
- MQAT_OS400
- MQAT_UNIX
- MQAT_WINDOWS

- MQAT_WINDOWS_NT
- MQAT_USER_FIRST
- MQAT_USER_LAST

environment data
Environment data for the process. The initial value is a null string.

process name
Process name. The initial value is a null string.

queue name
Name of the queue to be started. The initial value is a null string.

trigger data
Trigger data for the process. The initial value is a null string.

user data
User data for the process. The initial value is a null string.

Constructors

ImqTrigger( );
The default constructor.

ImqTrigger(const ImqTrigger & *trigger*);
The copy constructor.

Overloaded ImqItem methods

virtual ImqBoolean copyOut (ImqMessage & *msg*);
Writes an MQTM data structure to the message buffer, replacing any existing content. Sets the *msg* format to MQFMT_TRIGGER.

See the ImqItem class method description at "ImqItem C++ class" on page 3970 for further details.

virtual ImqBoolean pasteIn (ImqMessage & *msg*);
Reads an MQTM data structure from the message buffer.

To be successful, the ImqMessage format must be MQFMT_TRIGGER.

See the ImqItem class method description at "ImqItem C++ class" on page 3970 for further details.

Object methods (public)

void operator = (const ImqTrigger & *trigger*);
Copies instance data from *trigger*, replacing the existing instance data.

ImqString applicationId ( ) const ;
Returns a copy of the application id.

void setApplicationId (const char * *id*);
Sets the application id.

MQLONG applicationType ( ) const ;
Returns the application type.

void setApplicationType (const MQLONG *type*);
Sets the application type.

ImqBoolean copyOut (MQTMC2 * *ptmc2*);
Encapsulates the MQTM data structure, which is the one received on initiation queues. Fills in an equivalent MQTMC2 data structure provided by the caller, and sets the QMgrName field (which

is not present in the MQTM data structure) to all blanks. The MQTMC2 data structure is traditionally used as a parameter to applications started by a trigger monitor. This method returns TRUE if successful.

ImqString environmentData ( ) const ;
Returns a copy of the environment data.

void setEnvironmentData (const char * *data*);
Sets the environment data.

ImqString processName ( ) const ;
Returns a copy of the process name.

void setProcessName (const char * *name*);
Sets the process name, padded with blanks to 48 characters.

ImqString queueName ( ) const ;
Returns a copy of the queue name.

void setQueueName (const char * *name*);
Sets the queue name, padding with blanks to 48 characters.

ImqString triggerData ( ) const ;
Returns a copy of the trigger data.

void setTriggerData (const char * *data*);
Sets the trigger data.

ImqString userData ( ) const ;
Returns a copy of the user data.

void setUserData (const char * *data*);
Sets the user data.

Object data (protected)

MQTM *omqtm*
The MQTM data structure.

Reason codes

- MQRC_NULL_POINTER
- MQRC_INCONSISTENT_FORMAT
- MQRC_ENCODING_ERROR
- MQRC_STRUC_ID_ERROR

ImqWorkHeader C++ class

This class encapsulates specific features of the MQWIH data structure.

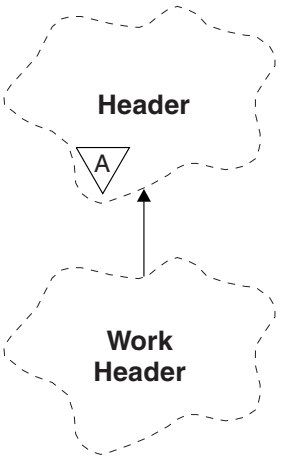


Figure 121. ImqWorkHeader class

Objects of this class are used by applications putting messages to the queue managed by the z/OS Workload Manager.

Object attributes

message token
Message token for the z/OS Workload Manager, of length MQ_MSG_TOKEN_LENGTH. The initial value is MQMTOK_NONE.

service name
The 32-character name of a process. The name is initially blanks.

service step
The 8-character name of a step within the process. The name is initially blanks.

Constructors

ImqWorkHeader( );
The default constructor.

ImqWorkHeader(const ImqWorkHeader & *header*);
The copy constructor.

Overloaded ImqItem methods

virtual ImqBoolean copyOut(ImqMessage & *msg*);
Inserts an MQWIH data structure into the beginning of the message buffer, moving the existing message data further along, and sets the *msg* **format** to MQFMT_WORK_INFO_HEADER.

See the parent class method description for more details.

virtual ImqBoolean pasteIn(ImqMessage & *msg*);
Reads an MQWIH data structure from the message buffer.

To be successful, the encoding of the *msg* object must be MQENC_NATIVE. Retrieve messages with MQGMO_CONVERT to MQENC_NATIVE.

The ImqMessage format must be MQFMT_WORK_INFO_HEADER.

See the parent class method description for more details.

Object methods (public)

void operator = (const ImqWorkHeader & *header*);
Copies instance data from *header*, replacing the existing instance data.

ImqBinary messageToken ( ) const;
Returns the **message token**.

ImqBoolean setMessageToken(const ImqBinary & *token*);
Sets the **message token**. The data length of *token* must be either zero or MQ_MSG_TOKEN_LENGTH. It returns TRUE if successful.

void setMessageToken(const MQBYTE16 *token* = 0);
Sets the **message token**. *token* can be zero, which is the same as specifying MQMTOK_NONE. If *token* is nonzero, it must address MQ_MSG_TOKEN_LENGTH bytes of binary data.

When using predefined values such as MQMTOK_NONE, you might need make a cast to ensure a signature match; for example, (MQBYTE *)MQMTOK_NONE.

ImqString serviceName ( ) const;
Returns the **service name**, including trailing blanks.

void setServiceName(const char * *name*);
Sets the **service name**.

ImqString serviceStep ( ) const;
Returns the **service step**, including trailing blanks.

void setServiceStep(const char * *step*);
Sets the **service step**.

Object data (protected)

MQWIH omqwih
The MQWIH data structure.

Reason codes

- MQRC_BINARY_DATA_LENGTH_ERROR

Properties of IBM MQ classes for JMS objects

All objects in IBM MQ classes for JMS have properties. Different properties apply to different object types. Different properties have different allowable values, and symbolic property values differ between the administration tool and program code.

IBM MQ classes for JMS provides facilities to set and query the properties of objects using the IBM MQ JMS administration tool, IBM MQ Explorer, or in an application. Many of the properties are relevant only to a specific subset of the object types.

For information on how you use the IBM MQ JMS administration tool, see Configuring JMS objects using the administration tool.

Table 419 gives a brief description of each property and shows for each property which object types it applies to. The object types are identified using keywords; see Configuring JMS objects using the administration tool for an explanation of these objects.

Numbers refer to notes at the end of the table. See also "Dependencies between properties of IBM MQ classes for JMS objects" on page 4036.

A property consists of a name-value pair in the format:

```
PROPERTY_NAME(property_value)
```

The topics in this section list, for each property, the name of the property and a brief description, and shows the valid property values used in the administration tool. and the set method that is used to set the value of the property in an application. The topics also show the valid property values for each property and the mapping between symbolic property values used in the tool and their programmable equivalents.

Property names are not case-sensitive, and are restricted to the set of recognized names shown in these topics.

Table 419. Property names and applicable object types

Property	Short form	Object type							
		CF	QCF	TCF	Q	T	XACF	XAQCF	XATCF
"APPLICATIONNAME" on page 4038	APPNAME	Y	Y	Y			Y	Y	Y
"ASYNCEXCEPTION" on page 4038	AEX	Y	Y	Y			Y	Y	Y
"BROKERCCDURSUBQ" on page 4039 [1]	CCDSUB					Y			
"BROKERCCSUBQ" on page 4040 [1]	CCSUB	Y		Y			Y		Y
"BROKERCONQ" on page 4040 [1]	BCON	Y		Y			Y		Y
"BROKERDURSUBQ" on page 4041 [1]	BDSUB					Y			
"BROKERPUBQ" on page 4041 [1]	BPUB	Y		Y		Y	Y		Y
"BROKERPUBQMGR" on page 4042 [1]	BPQM					Y			
"BROKERQMGR" on page 4042 [1]	BQM	Y		Y			Y		Y
"BROKERSUBQ" on page 4043 [1]	BSUB	Y		Y			Y		Y
"BROKERVER" on page 4043 [1]	BVER	Y [2]		Y [2]		Y	Y		Y
"CCDTURL" on page 4044 [3]	CCDT	Y	Y	Y			Y	Y	Y
"CCSID" on page 4044	CCS	Y	Y	Y	Y	Y	Y	Y	Y

Table 419. Property names and applicable object types (continued)

Property	Short form	Object type							
		CF	QCF	TCF	Q	T	XACF	XAQCF	XATCF
"CHANNEL" on page 4045 [3]	CHAN	Y	Y	Y			Y	Y	Y
"CLEANUP" on page 4045 [1]	CL	Y		Y			Y		Y
"CLEANUPINT" on page 4046 [1]	CLINT	Y		Y			Y		Y
"CONNECTIONNAMELIST" on page 4046	CNLIST	Y	Y	Y					
"CLIENTRECONNECTOPTIONS" on page 4047	CROPT	Y	Y	Y					
"CLIENTRECONNECTTIMEOUT" on page 4048	CRT	Y	Y	Y					
"CLIENTID" on page 4048	CID	Y [2]	Y	Y [2]			Y	Y	Y
"CLONESUPP" on page 4049	CLS	Y		Y			Y		Y
"COMPHDR" on page 4049	HC	Y		Y			Y		Y
"COMPMSG" on page 4050	MC	Y	Y	Y			Y	Y	Y
"CONNOPT" on page 4050	CNOPT	Y	Y	Y			Y	Y	Y
"CONNTAG" on page 4051	CNTAG	Y	Y	Y			Y	Y	Y
"DESCRIPTION" on page 4052	DESC	Y [2]	Y	Y [2]	Y	Y	Y	Y	Y
"DIRECTAUTH" on page 4052	DAUTH	Y [2]		Y [2]					
"ENCODING" on page 4053	ENC				Y	Y			
"EXPIRY" on page 4054	EXP				Y	Y			
"FAILIFQUIESCE" on page 4054	FIQ	Y	Y	Y	Y	Y	Y	Y	Y
"HOSTNAME" on page 4055	HOST	Y [2]	Y	Y [2]			Y	Y	Y
"LOCALADDRESS" on page 4055	LA	Y [2]	Y	Y [2]			Y	Y	Y
"MAPNAMESTYLE" on page 4056	MNST	Y	Y	Y			Y	Y	Y
"MAXBUFFSIZE" on page 4057	MBSZ	Y [2]		Y [2]					
"MDREAD" on page 4057	MDR				Y	Y			
"MDWRITE" on page 4058	MDW				Y	Y			
"MDMSGCTX" on page 4058	MDCTX				Y	Y			
"MSGBATCHSZ" on page 4059 [1]	MBS	Y	Y	Y			Y	Y	Y
"MSGBODY" on page 4059	MBODY				Y	Y			
"MSGRETENTION" on page 4060	MRET	Y	Y				Y	Y	
"MSGSELECTION" on page 4060 [1]	MSEL	Y		Y			Y		Y
"MULTICAST" on page 4061	MCAST	Y [2]		Y [2]		Y			
"OPTIMISTICPUBLICATION" on page 4062 [1]	OPTPUB	Y		Y					
"OUTCOMENOTIFICATION" on page 4062 [1]	NOTIFY	Y		Y					
"PERSISTENCE" on page 4063	PER				Y	Y			
"POLLINGINT" on page 4063 [1]	PINT	Y	Y	Y			Y	Y	Y
"PORT" on page 4064	PORT	Y [2]	Y	Y [2]			Y	Y	Y
"PRIORITY" on page 4064	PRI				Y	Y			

Table 419. Property names and applicable object types (continued)

Property	Short form	Object type							
		CF	QCF	TCF	Q	T	XACF	XAQCF	XATCF
"PROCESSDURATION" on page 4065 [1]	PROCDUR	Y		Y					
"PROVIDERVERSION" on page 4065	PVER	Y	Y	Y			Y	Y	Y
"PROXYHOSTNAME" on page 4068	PHOST	Y [2]		Y [2]					
"PROXYPORT" on page 4068	PPORT	Y [2]		Y [2]					
"PUBACKINT" on page 4069 [1]	PAI	Y		Y			Y		Y
"PUTASYNCALLOWED" on page 4069	PAALD				Y	Y			
"QMANAGER" on page 4070	QMGR	Y	Y	Y	Y		Y	Y	Y
"QUEUE" on page 4070	QU				Y				
"READAHEADALLOWED" on page 4071	RAALD				Y	Y			
"READAHEADCLOSEPOLICY" on page 4072	RACP				Y	Y			
"RECEIVECCSID" on page 4072	RCCS				Y	Y			
"RECEIVECONVERSION" on page 4073	RCNV				Y	Y			
"RECEIVEISOLATION" on page 4073 [1]	RCVISOL	Y		Y					
"RECEXIT" on page 4074	RCX	Y	Y	Y			Y	Y	Y
"RECEXITINIT" on page 4074	RCXI	Y	Y	Y			Y	Y	Y
"REPLYTOSTYLE" on page 4075	RTOST				Y	Y			
"RESCANINT" on page 4075 [1]	RINT	Y	Y				Y	Y	
"SECEXIT" on page 4076	SCX	Y	Y	Y			Y	Y	Y
"SECEXITINIT" on page 4077	SCXI	Y	Y	Y			Y	Y	Y
"SENDCHECKCOUNT" on page 4077	SCC	Y	Y	Y			Y	Y	Y
"SENDEXIT" on page 4078	SDX	Y	Y	Y			Y	Y	Y
"SENDEXITINIT" on page 4078	SDXI	Y	Y	Y			Y	Y	Y
"SHARECONVALLOWED" on page 4079	SCALD	Y	Y	Y			Y	Y	Y
"SPARSESUBS" on page 4079 [1]	SSUBS	Y		Y					
"SSLCIPHERSUITE" on page 4080	SCPHS	Y	Y	Y			Y	Y	Y
"SSLCRL" on page 4080	SCRL	Y	Y	Y			Y	Y	Y
"SSLFIPSREQUIRED" on page 4081	SFIPS	Y	Y	Y			Y	Y	Y
"SSLPEERNAME" on page 4081	SPEER	Y	Y	Y			Y	Y	Y
"SSLRESETCOUNT" on page 4082	SRC	Y	Y	Y			Y	Y	Y
"STATREFRESHINT" on page 4082 [1]	SRI	Y		Y			Y		Y
"SUBSTORE" on page 4083 [1]	SS	Y		Y			Y		Y
"SYNCPOINTALLGETS" on page 4083	SPAG	Y	Y	Y			Y	Y	Y

Table 419. Property names and applicable object types (continued)

Property	Short form	Object type							
		CF	QCF	TCF	Q	T	XACF	XAQCF	XATCF
"TARGCLIENT" on page 4084	TC				Y	Y			
"TARGCLIENTMATCHING" on page 4084	TCM	Y	Y				Y	Y	
"TEMPMODEL" on page 4085	TM	Y	Y				Y	Y	
"TEMPQPREFIX" on page 4085	TQP	Y	Y				Y	Y	
"TEMPTOPICPREFIX" on page 4086	TTP	Y		Y			Y		Y
"TOPIC" on page 4086	TOP					Y			
"TRANSPORT" on page 4087	TRAN	Y [2]	Y	Y [2]			Y	Y	Y
"WILDCARDFORMAT" on page 4088	WCFMT	Y		Y			Y		Y

Note:

1. This property can be used with Version 7.0 of IBM MQ classes for JMS but has no effect for an application connected to a Version 7.0 queue manager unless the PROVIDERVERSION property of the connection factory is set to a version number less than 7.
2. Only the BROKERVER, CLIENTID, DESCRIPTION, DIRECTAUTH, HOSTNAME, LOCALADDRESS, MAXBUFFSIZE, MULTICAST, PORT, PROXYHOSTNAME, PROXYPORT, and TRANSPORT properties are supported for a ConnectionFactory or TopicConnectionFactory object when using a real-time connection to a broker.
3. The CCDTURL and CHANNEL properties of an object must not both be set at the same time.

Dependencies between properties of IBM MQ classes for JMS objects

The validity of some properties is dependent on the particular values of other properties.

This dependency can occur in the following groups of properties:

- Client properties
- Properties for a real-time connection to a broker
- Exit initialization strings

Client properties

For a connection to a queue manager, the following properties are relevant only if TRANSPORT has the value CLIENT:

- HOSTNAME
- PORT
- CHANNEL
- LOCALADDRESS
- CCDTURL
- CCSID
- COMPHDR
- COMPMSG
- RECEXIT
- RECEXITINIT
- SECEXIT
- SECEXITINIT
- SENDEXIT
- SENDEXITINIT

- SHARECONVALLOWED
- SSLCIPHERSUITE
- SSLCRL
- SSLFIPSREQUIRED
- SSLPEERNAME
- SSLRESETCOUNT
- APPLICATIONNAME

You cannot set values for these properties by using the administration tool if TRANSPORT has the value BIND.

If TRANSPORT has the value CLIENT, the default value of the BROKERVER property is V1 and the default value of the PORT property is 1414. If you set the value of BROKERVER or PORT explicitly, a later change to the value of TRANSPORT does not override your choices.

Properties for a real-time connection to a broker

Only the following properties are relevant if TRANSPORT has the value DIRECT or DIRECTHTTP:

- BROKERVER
- CLIENTID
- DESCRIPTION
- DIRECTAUTH
- HOSTNAME
- LOCALADDRESS
- MAXBUFFSIZE
- MULTICAST (supported only for DIRECT)
- PORT
- PROXYHOSTNAME (supported only for DIRECT)
- PROXYPORT (supported only for DIRECT)

If TRANSPORT has the value DIRECT or DIRECTHTTP, the default value of the BROKERVER property is V2, and the default value of the PORT property is 1506. If you set the value of BROKERVER or PORT explicitly, a later change to the value of TRANSPORT does not override your choices.

Exit initialization strings

Do not set any of the exit initialization strings without supplying the corresponding exit name. The exit initialization properties are:

- RECEXITINIT
- SECEXITINIT
- SENDEXITINIT

For example, specifying `RECEXITINIT(myString)` without specifying `RECEXIT(some.exit.classname)` causes an error.

Related reference:
"TRANSPORT" on page 4087
The nature of a connection to a queue manager or broker.

APPLICATIONNAME

An application can set a name that identifies its connection to the queue manager. This application name is shown by the **DISPLAY CONN MQSC/PCF** command (where the field is called **APPLTAG**) or in the IBM MQ Explorer Application Connections display (where the field is called **App name**).

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: APPLICATIONNAME

JMS administration tool short name: APPNAME

Programmatic access

Setters/getters

- MQConnectionFactory.setAppName()
- MQConnectionFactory.getAppName()

Values

Any valid string that is no longer than 28 characters. Longer names are adjusted to fit by removing leading package names, if necessary. For example, if the invoking class is `com.example.MainApp`, the full name is used, but if the invoking class is `com.example.dictionaryAndThesaurus.multilingual.mainApp`, the name `multilingual.mainApp` is used, because it is the longest combination of class name and rightmost package name that fits into the available length.

If the class name itself is more than 28 characters long, it is truncated to fit. For example, `com.example.mainApplicationForSecondTestCase` becomes `mainApplicationForSecondTest`.

ASYNCEXCEPTION

This property determines whether IBM MQ classes for JMS informs an ExceptionListener only when a connection is broken, or when any exception occurs asynchronously to a JMS API call. This applies to all Connections created from this ConnectionFactory that have an ExceptionListener registered.

Applicable objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: ASYNCEXCEPTION

JMS administration tool short name: AEX

Programmatic access

Setters/Getters

- MQConnectionFactory.setAsyncExceptions()
- MQConnectionFactory.getAsyncExceptions()

Values

ASYNC_EXCEPTIONS_ALL

Any exception detected asynchronously, outside the scope of a synchronous API call, and all connection broken exceptions are sent to the ExceptionListener.

Environment	Value
JMS Administration Tool	ASYNC_EXCEPTIONS_ALL
Programmatic	WMQCONSTANTS.ASYNC_EXCEPTIONS_ALL = -1
IBM MQ Explorer	All

ASYNC_EXCEPTIONS_CONNECTIONBROKEN

Only exceptions indicating a broken connection are sent to the ExceptionListener. Any other exceptions occurring during asynchronous processing are not reported to the ExceptionListener, and hence the application is not informed of these exceptions. This is the default value from IBM MQ Version 8.0.0, Fix Pack 2 (see JMS: Exception listener changes in Version 8.0).

Environment	Value
JMS Administration Tool	ASYNC_EXCEPTIONS_CONNECTIONBROKEN
Programmatic	WMQCONSTANTS.ASYNC_EXCEPTIONS_CONNECTIONBROKEN = 1
IBM MQ Explorer	Connection Broken

The following additional constant is defined:

- From Version 8.0.0, Fix Pack 2: WMQCONSTANTS.ASYNC_EXCEPTIONS_DEFAULT = ASYNC_EXCEPTIONS_CONNECTIONBROKEN
- Before Version 8.0.0, Fix Pack 2: WMQCONSTANTS.ASYNC_EXCEPTIONS_DEFAULT = ASYNC_EXCEPTIONS_ALL

Related information:

Exceptions in IBM MQ classes for JMS

BROKERCCDURSUBQ

The name of the queue from which durable subscription messages are retrieved for a ConnectionConsumer.

Applicable objects

Topic

JMS administration tool long name: BROKERCCDURSUBQ

JMS administration tool short name: CCDSUB

Programmatic access

Setters/getters

- MQTopic.setBrokerCCDurSubQueue()
- MQTopic.getBrokerCCDurSubQueue()

Values

SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE

This is the default value.

Any valid string

BROKERCCSUBQ

The name of the queue from which non-durable subscription messages are retrieved for a ConnectionConsumer.

Applicable objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: BROKERCCSUBQ

JMS administration tool short name: CCSUB

Programmatic access

Setters/getters

- MQConnectionFactory.setBrokerCCSubQueue()
- MQConnectionFactory.getBrokerCCSubQueue()

Values

SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
This is the default value.

Any valid string

BROKERCONQ

The control queue name of the broker.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: BROKERCONQ

JMS administration tool short name: BCON

Programmatic access

Setters/getters

- MQConnectionFactory.setBrokerControlQueue()
- MQConnectionFactory.getBrokerControlQueue()

Values

SYSTEM.BROKER.CONTROL.QUEUE
This is the default value.

Any valid string

BROKERDURSUBQ

When the IBM MQ classes for JMS are being used in IBM MQ messaging provider migration mode, this property specifies the name of the queue from which durable subscription messages are retrieved.

Applicable objects

Topic

JMS administration tool long name: BROKERDURSUBQ

JMS administration tool short name: BDSUB

Programmatic access

Setters/getters

- MQTopic.setBrokerDurSubQueue()
- MQTopic.getBrokerDurSubQueue()

Values

SYSTEM.JMS.D.SUBSCRIBER.QUEUE
This is the default value.

Any valid string
Starting with SYSTEM.JMS.D

Related information:

Configuring the JMS **PROVIDERVERSION** property

BROKERPUBQ

The name of the queue where published messages are sent (the stream queue).

Applicable Objects

ConnectionFactory, TopicConnectionFactory, Topic, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: BROKERPUBQ

JMS administration tool short name: BPUB

Programmatic access

Setters/getters

- MQConnectionFactory.setBrokerPubQueue
- MQConnectionFactory.getBrokerPubQueue

Values

SYSTEM.BROKER.DEFAULT.STREAM
This is the default value.

Any valid string

BROKERPUBQMGR

The name of the queue manager that owns the queue where messages published on the topic are sent.

Applicable Objects

Topic

JMS administration tool long name: BROKERPUBQMGR

JMS administration tool short name: BPQM

Programmatic access

Setters/getters

- MQTopic.setBrokerPubQueueManager()
- MQTopic.getBrokerPubQueueManager()

Values

null This is the default value.

Any valid string

BROKERQMGR

The name of the queue manager on which the broker is running.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: BROKERQMGR

JMS administration tool short name: BQM

Programmatic access

Setters/getters

- MQConnectionFactory.setBrokerQueueManager()
- MQConnectionFactory.getBrokerQueueManager()

Values

null This is the default value.

Any valid string

BROKERSUBQ

When the IBM MQ classes for JMS are being used in IBM MQ messaging provider migration mode, this property specifies the name of the queue from which non-durable subscription messages are retrieved.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: BROKERSUBQ

JMS administration tool short name: BSUB

Programmatic access

Setters/getters

- MQConnectionFactory.setBrokerSubQueue()
- MQConnectionFactory.getBrokerSubQueue()

Values

SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
This is the default value.

Any valid string
Starting with SYSTEM.JMS.ND

Related information:

Configuring the JMS **PROVIDERVERSION** property

BROKERVER

The version of the broker being used.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, Topic, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: BROKERVER

JMS administration tool short name: BVER

Programmatic access

Setters/getters

- MQConnectionFactory.setBrokerVersion()
- MQConnectionFactory.getBrokerVersion()

Values

V1 To use an IBM MQ Publish/Subscribe broker, or to use a broker of IBM MQ Integrator, WebSphere Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker in compatibility mode. This is the default value if TRANSPORT is set to BIND or CLIENT.

V2 To use a broker of IBM MQ Integrator, WebSphere Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker in native mode. This is the default value if TRANSPORT is set to DIRECT or DIRECTHTTP.

unspecified
After the broker has migrated from V6 to V7, set this property so that RFH2 headers are no longer used. After migration this property is no longer relevant.

CCDTURL

A Uniform Resource Locator (URL) that identifies the name and location of the file containing the client channel definition table and specifies how the file can be accessed.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: CCDTURL

JMS administration tool short name: CCDT

Programmatic access

Setters/getters

- MQConnectionFactory.setCCDTURL()
- MQConnectionFactory.getCCDTURL()

Values

null This is the default value.

A Uniform Resource Locator (URL)

CCSID

The coded character set ID to be used for a connection or destination.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, Queue, Topic, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: CCSID

JMS administration tool short name: CCS

Programmatic access

Setters/getters

- MQConnectionFactory.setCCSID()
- MQConnectionFactory.getCCSID()

Values

819 This is the default value for a connection factory.

1208 This is the default value for a destination.

Any positive integer

CHANNEL

The name of the client connection channel being used.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: CHANNEL

JMS administration tool short name: CHAN

Programmatic access

Setters/getters

- MQConnectionFactory.setChannel()
- MQConnectionFactory.getChannel()

Values

SYSTEM.DEF.SVRCONN
This is the default value.

Any valid string

CLEANUP

Cleanup Level for BROKER or MIGRATE Subscription Stores.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: CLEANUP

JMS administration tool short name: CL

Programmatic access

Setters/getters

- MQConnectionFactory.setCleanupLevel()
- MQConnectionFactory.getCleanupLevel()

Values

SAFE Use safe cleanup. This is the default value.

ASPROP
Use safe, strong, or no cleanup according to a property set on the Java command line.

NONE
Use no cleanup.

STRONG
Use strong cleanup.

CLEANUPINT

The interval, in milliseconds, between background executions of the publish/subscribe cleanup utility.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: CLEANUPINT

JMS administration tool short name: CLINT

Programmatic access

Setters/getters

- MQConnectionFactory.setCleanupInterval()
- MQConnectionFactory.getCleanupInterval()

Values

3600000
This is the default value.

Any positive integer

CONNECTIONNAMELIST

List of TCP/IP connection names. The list is tried in order, once per each reconnection retry attempt.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory

JMS administration tool long name: CONNECTIONNAMELIST

JMS administration tool short name: CNLIST

Programmatic access

Setters/getters

- MQConnectionFactory.setconnectionNameList()
- MQConnectionFactory.getconnectionNameList()

Values

Comma separated list of HOSTNAME(PORT). HOSTNAME(PORT) can be either a DNS name or IP address.

PORT defaults to 1414.

CLIENTRECONNECTOPTIONS

Options governing reconnection.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory

JMS administration tool long name: CLIENTRECONNECTOPTIONS

JMS administration tool short name: CROPT

Programmatic access

Setters/getters

- MQConnectionFactory.setClientReconnectOptions()
- MQConnectionFactory.getClientReconnectOptions()

Values

QMGR

The application can reconnect, but only to the queue manager to which it originally connected.

Use this value if an application can be reconnected, but there is an affinity between the IBM MQ classes for JMS application, and the queue manager to which it first established a connection.

Choose this value if you want an application to automatically reconnect to the standby instance of a highly available queue manager.

To use this value programmatically, use the constant WMQConstants.WMQ_CLIENT_RECONNECT_Q_MGR.

ANY

The application can reconnect to any queue manager.

Use the reconnect option only if there is no affinity between the IBM MQ classes for JMS application and the queue manager with which it initially established a connection.

To use this value from a program, use the constant WMQConstants.WMQ_CLIENT_RECONNECT.

DISABLED

The application will not be reconnected.

To use this value programmatically, use the constant WMQConstants.WMQ_CLIENT_RECONNECT_DISABLED.

ASDEF

Whether the application will reconnect automatically depends on the value of the IBM MQ channel attribute DefReconnect.

To use this value from a program, use the constant WMQConstants.WMQ_CLIENT_RECONNECT_AS_DEF.

CLIENTRECONNECTTIMEOUT

Time before reconnection retries cease.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory

JMS administration tool long name: CLIENTRECONNECTTIMEOUT

JMS administration tool short name: CRT

Programmatic access

Setters/getters

- MQConnectionFactory.setClientReconnectTimeout()
- MQConnectionFactory.setClientReconnectTimeout()

Values

Interval in seconds. Default 1800 (30 minutes).

CLIENTID

The client identifier is used to uniquely identify the application connection for durable subscriptions.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: CLIENTID

JMS administration tool short name: CID

Programmatic access

Setters/getters

- MQConnectionFactory.setClientId()
- MQConnectionFactory.getClientId()

Values

null This is the default value.

Any valid string

CLONESUPP

Whether two or more instances of the same durable topic subscriber can run simultaneously.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: CLONESUPP

JMS administration tool short name: CLS

Programmatic access

Setters/getters

- MQConnectionFactory.setCloneSupport()
- MQConnectionFactory.getCloneSupport()

Values

DISABLED
: Only one instance of a durable topic subscriber can run at a time. This is the default value.

ENABLED
: Two or more instances of the same durable topic subscriber can run simultaneously, but each instance must run in a separate Java virtual machine (JVM).

COMPHDR

A list of the techniques that can be used for compressing header data on a connection.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: COMPHDR

JMS administration tool short name: HC

Programmatic access

Setters/getters

- MQConnectionFactory.setHdrCompList()
- MQConnectionFactory.getHdrCompList()

Values

NONE
: This is the default value.

SYSTEM
: RLE message header compression is performed.

COMPMSG

A list of the techniques that can be used for compressing message data on a connection.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: COMPMSG

JMS administration tool short name: MC

Programmatic access

Setters/getters

- MQConnectionFactory.setMsgCompList()
- MQConnectionFactory.getMsgCompList()

Values

NONE
This is the default value.

A list of one or more of the following values separated by blank characters:
RLE ZLIBFAST ZLIBHIGH

CONNOPT

Controls how IBM MQ classes for JMS applications that use the bindings transport connect to the queue manager.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory.

JMS administration tool long name: CONNOPT

JMS administration tool short name: CNOPT

Programmatic access

Setters/getters

- MQConnectionFactory.setMQConnectionOptions()
- MQConnectionFactory.getMQConnectionOptions()

Values

STANDARD
The nature of the binding between the application and the queue manager depends on the value of the *DefaultBindType* attribute of the queue manager. The STANDARD value maps to the IBM MQ *ConnectOption* MQCNO_STANDARD_BINDING.

SHARED
The application and the local queue manager agent run in separate units of execution but share some resources. This value maps to the IBM MQ *ConnectOption* MQCNO_SHARED_BINDING.

ISOLATED
The application and the local queue manager agent run in separate units of execution and share no resources. The ISOLATED value maps to the IBM MQ *ConnectOption* MQCNO_ISOLATED_BINDING.

FASTPATH
The application and the local queue manager agent run in the same unit of execution. This value maps to the IBM MQ *ConnectOption* MQCNO_FASTPATH_BINDING.

SERIALQM
The application requests exclusive use of the connection tag within the scope of the queue manager. This value maps to the IBM MQ *ConnectOption* MQCNO_SERIALIZE_CONN_TAG_Q_MGR.

SERIALQSG
The application requests exclusive use of the connection tag within the scope of the queue-sharing group to which the queue manager belongs. The SERIALQSG value maps to the IBM MQ *ConnectOption* MQCNO_SERIALIZE_CONN_TAG_QSG.

RESTRICTQM
The application requests shared use of the connection tag, but there are restrictions on the shared use of the connection tag within the scope of the queue manager. This value maps to the IBM MQ *ConnectOption* MQCNO_RESTRICT_CONN_TAG_Q_MGR.

RESTRICTQSG
The application requests shared use of the connection tag, but there are restrictions on the shared use of the connection tag within the scope of the queue-sharing group to which the queue manager belongs. This value maps to the IBM MQ *ConnectOption* MQCNO_RESTRICT_CONN_TAG_QSG.

For further information on IBM MQ connection options, see Connecting to a queue manager using the MQCONNX call.

CONNTAG

A tag that the queue manager associates with the resources updated by the application within a unit of work while the application is connected to the queue manager.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: CONNTAG

JMS administration tool short name: CNTAG

Programmatic access

Setters/getters

- MQConnectionFactory.setConnTag()
- MQConnectionFactory.getConnTag()

Values

A byte array of 128 elements, where each element is 0
This is the default value.

Any string
The value is truncated if it is longer than 128 bytes.

DESCRIPTION

A description of the stored object.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, Queue, Topic, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: DESCRIPTION

JMS administration tool short name: DESC

Programmatic access

Setters/getters

- MQConnectionFactory.setDescription()
- MQConnectionFactory.getDescription()

Values

null This is the default value.

Any valid string
`

DIRECTAUTH

Whether TLS authentication is used on a real-time connection to a broker.

Applicable Objects

ConnectionFactory, TopicConnectionFactory

JMS administration tool long name: DIRECTAUTH

JMS administration tool short name: DAUTH

Programmatic access

Setters/getters

- MQConnectionFactory.setDirectAuth()
- MQConnectionFactory.getDirectAuth()

Values

BASIC
No authentication, username authentication, or password authentication. This is the default value.

CERTIFICATE
Public key certificate authentication.

ENCODING

How numeric data in the body of a message is represented when the message is sent to this destination. The property specifies the representation of binary integers, packed decimal integers, and floating point numbers.

Applicable Objects

Queue, Topic

JMS administration tool long name: ENCODING

JMS administration tool short name: ENC

Programmatic access

Setters/getters

- MQDestination.setEncoding()
- MQDestination.getEncoding()

Values

ENCODING property

The valid values that the `ENCODING` property can take are constructed from the three sub-properties:

integer encoding
Either normal or reversed

decimal encoding
Either normal or reversed

floating-point encoding
IEEE normal, IEEE reversed, or z/OS

The `ENCODING` property is expressed as a three-character string with the following syntax:

```
{N|R}{N|R}{N|R|3}
```

In this string:

- `N` denotes normal
- `R` denotes reversed
- `3` denotes z/OS
- The first character represents *integer encoding*
- The second character represents *decimal encoding*
- The third character represents *floating-point encoding*

This provides a set of twelve possible values for the `ENCODING` property.

There is an additional value, the string `NATIVE`, which sets appropriate encoding values for the Java platform.

The following examples show valid combinations for `ENCODING`:

```
ENCODING(NNR)
ENCODING(NATIVE)
ENCODING(RR3)
```

EXPIRY

The time after which messages at a destination expire.

Applicable Objects

Queue, Topic

JMS administration tool long name: EXPIRY

JMS administration tool short name: EXP

Programmatic access

Setters/getters

- MQDestination.setExpiry()
- MQDestination.getExpiry()

Values

APP Expiry can be defined by the JMS application. This is the default value.

UNLIM
No expiry occurs.

0 No expiry occurs.

Any positive integer representing expiry in milliseconds.

FAILIFQUIESCE

This property determines whether calls to certain methods fail if the queue manager is in a quiescing state.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, Queue, Topic, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: FAILIFQUIESCE

JMS administration tool short name: FIQ

Programmatic access

Setters/getters

- MQConnectionFactory.setFailIfQuiesce()
- MQConnectionFactory.getFailIfQuiesce()

Values

YES Calls to certain methods fail if the queue manager is in a quiescing state. If an application detects that the queue manager is quiescing, the application can complete its immediate task and close the connection, allowing the queue manager to stop. This is the default value.

NO No method call fails because the queue manager is in a quiescing state. If you specify this value, an application cannot detect that the queue manager is quiescing. The application might continue to perform operations against the queue manager, and therefore prevent the queue manager from stopping.

HOSTNAME

For a connection to a queue manager, the host name or IP address of the system on which the queue manager is running or, for a real-time connection to a broker, the host name or IP address of the system on which the broker is running.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: HOSTNAME

JMS administration tool short name: HOST

Programmatic access

Setters/getters

- MQConnectionFactory.setHostName()
- MQConnectionFactory.getHostName()

Values

localhost
This is the default value.

Any valid string

LOCALADDRESS

For a connection to a queue manager, this property specifies either the local network interface to be used, or the local port, or range of local ports, to be used.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: LOCALADDRESS

JMS administration tool short name: LA

Programmatic access

Setters/getters

- MQConnectionFactory.setLocalAddress()
- MQConnectionFactory.getLocalAddress()

Values

"" (empty string)
This is the default value.

A string in the format [ip-addr][(low-port[,high-port])]
Here are some examples:

```
192.0.2.0
```

The channel binds to address 192.0.2.0 locally.

`192.0.2.0(1000)`

The channel binds to address 192.0.2.0 locally and uses port 1000.

`192.0.2.0(1000,2000)`

The channel binds to address 192.0.2.0 locally and uses a port in the range 1000 to 2000.

`(1000)`

The channel binds to port 1000 locally.

`(1000,2000)`

The channel binds to a port in the range 1000 to 2000 locally.

You can specify a host name instead of an IP address. For a real-time connection to a broker, this property is relevant only when multicast is used, and the value of the property must not contain a port number, or a range of port numbers. The only valid values of the property in this case are null, an IP address, or a host name.

MAPNAMESTYLE

Allows compatibility style to be used for MapMessage element names.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: MAPNAMESTYLE

JMS administration tool short name: MNST

Programmatic access

Setters/getters

- MQConnectionFactory.setMapNameStyle()
- MQConnectionFactory.getMapNameStyle()

Values

STANDARD

The standard com.ibm.jms.JMSMapMessage element naming format is to be used. This is the default value and allows non-legal Java identifiers to be used as the element name.

COMPATIBLE

The older com.ibm.jms.JMSMapMessage element naming format is to be used. Only legal Java identifiers can be used as the element name. This is needed only if map messages are being sent to an application that is using a version of IBM MQ classes for JMS earlier than Version 5.3.

MAXBUFFSIZE

The maximum number of received messages that can be stored in an internal message buffer while waiting to be processed by the application. This property applies only when TRANSPORT has the value DIRECT or DIRECTHTTP.

Applicable Objects

ConnectionFactory, TopicConnectionFactory

JMS administration tool long name: MAXBUFFSIZE

JMS administration tool short name: MBSZ

Programmatic access

Setters/getters

- MQConnectionFactory.setMaxBufferSize()
- MQConnectionFactory.getMaxBufferSize()

Values

1000 This is the default value.

Any positive integer

MDREAD

This property determines whether a JMS application can extract the values of MQMD fields.

Applicable Objects

JMS administration tool long name: MDREAD

JMS administration tool short name: MDR

Programmatic access

Setters/getters

- MQDestination.setMQMDReadEnabled()
- MQDestination.getMQMDReadEnabled()

Values

NO When sending messages, the JMS_IBM_MQMD* properties on a sent message are not updated to reflect the updated field values in the MQMD. When receiving messages, none of the JMS_IBM_MQMD* properties are available on a received message, even if the sender had set some or all of them. This is the default value for administrative tools.

For programs, use False.

Yes When sending messages, all of the JMS_IBM_MQMD* properties on a sent message are updated to reflect the updated field values in the MQMD, including the properties that the sender did not set explicitly. When receiving messages, all of the JMS_IBM_MQMD* properties are available on a received message, including the properties that the sender did not set explicitly.

For programs, use True.

MDWRITE

This property determines whether a JMS application can set the values of MQMD fields.

Applicable Objects

Queue, Topic

JMS administration tool long name: MDWRITE

JMS administration tool short name: MDR

Programmatic access

Setters/getters

- MQDestination.setMQMDWriteEnabled()
- MQDestination.getMQMDWriteEnabled()

Values

NO All JMS_IBM_MQMD* properties are ignored and their values are not copied into the underlying MQMD structure. This is the default value for administrative tools.

For programs, use False.

YES JMS_IBM_MQMD* properties are processed. Their values are copied into the underlying MQMD structure.

For programs, use True.

MDMSGCTX

What level of message context is to be set by the JMS application. The application must be running with appropriate context authority for this property to take effect.

Applicable Objects

JMS administration tool long name: MDMSGCTX

JMS administration tool short name: MDCTX

Programmatic access

Setters/getters

- MQDestination.setMQMDMessageContext()
- MQDestination.getMQMDMessageContext()

Values

DEFAULT

The MQOPEN API call and the MQPMO structure specify no explicit message context options. This is the default value for administrative tools.

For programs, use WMQ_MDCTX_DEFAULT.

SET_IDENTITY_CONTEXT

The MQOPEN API call specifies the message context option MQOO_SET_IDENTITY_CONTEXT and the MQPMO structure specifies MQPMO_SET_IDENTITY_CONTEXT.

For programs, use WMQ_MDCTX_SET_IDENTITY_CONTEXT.

SET_ALL_CONTEXT
The MQOPEN API call specifies the message context option MQOO_SET_ALL_CONTEXT and the MQPMO structure specifies MQPMO_SET_ALL_CONTEXT.

For programs, use WMQ_MDCTX_SET_ALL_CONTEXT.

MSGBATCHSZ

The maximum number of messages to be taken from a queue in one packet when using asynchronous message delivery.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: MAXBUFFSIZE

JMS administration tool short name: MBSZ

Programmatic access

Setters/getters

- MQConnectionFactory.setMsgBatchSize()
- MQConnectionFactory.getMsgBatchSize()

Values

10 This is the default value.

Any positive integer

MSGBODY

Determines whether a JMS application accesses the MQRFH2 of an IBM MQ message as part of the message payload.

Applicable Objects

Queue, Topic

JMS administration tool long name: WMQ_MESSAGE_BODY

JMS administration tool short name: MBODY

Programmatic access

Setters/getters

- MQConnectionFactory.setMessageBodyStyle()
- MQConnectionFactory.getMessageBodyStyle()

Values

UNSPECIFIED
When sending, IBM MQ classes for JMS does or does not generate and include an MQRFH2 header, depending on the value of WMQ_TARGET_CLIENT. When receiving, acts as value `JMS`.

JMS When sending, IBM MQ classes for JMS automatically generates an MQRFH2 header and includes it in the IBM MQ message.

When receiving, IBM MQ classes for JMS set the JMS message properties according to values in the MQRFH2 (if present); it does not present the MQRFH2 as part of the JMS message body.

MQ When sending, IBM MQ classes for JMS does not generate an MQRFH2.

When receiving, IBM MQ classes for JMS presents the MQRFH2 as part of the JMS message body.

MSGRETENTION

Whether the connection consumer keeps undelivered messages on the input queue.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory,

JMS administration tool long name: MSGRETENTION

JMS administration tool short name: MRET

Programmatic access

Setters/getters

- MQConnectionFactory.setMessageRetention()
- MQConnectionFactory.getMessageRetention()

Values

Yes Undelivered messages remain on the input queue. This is the default value.

No Undelivered messages are dealt with according to their disposition options.

MSGSELECTION

Determines whether message selection is done by the IBM MQ classes for JMS or by the broker. If TRANSPORT has the value DIRECT, message selection is always done by the broker and the value of MSGSELECTION is ignored. Message selection by the broker is not supported when BROKERVER has the value V1.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: MSGSELECTION

JMS administration tool short name: MSEL

Programmatic access

Setters/getters

- MQConnectionFactory.setMessageSelection()
- MQConnectionFactory.getMessageSelection()

Values

CLIENT
Message selection is done by IBM MQ classes for JMS. This is the default value.

BROKER
Message selection is done by the broker.

MULTICAST

To enable multicast on a real-time connection to a broker and, if enabled, to specify the precise way in which multicast is used to deliver messages from the broker to a message consumer. The property has no effect on how a message producer sends messages to a broker.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, Topic

JMS administration tool long name: MULTICAST

JMS administration tool short name: MCAST

Programmatic access

Setters/getters

- MQConnectionFactory.setMulticast()
- MQConnectionFactory.getMulticast()

Values

DISABLED
Messages are not delivered to a message consumer using multicast transport. This is the default value for ConnectionFactory and TopicConnectionFactory objects.

ASCF Messages are delivered to a message consumer according to the multicast setting for the connection factory associated with the message consumer. The multicast setting for the connection factory is noted at the time that the message consumer is created. This value is valid only for Topic objects, and is the default value for Topic objects.

ENABLED
If the topic is configured for multicast in the broker, messages are delivered to a message consumer using multicast transport. A reliable quality of service is used if the topic is configured for reliable multicast.

RELIABLE
If the topic is configured for reliable multicast in the broker, messages are delivered to the message consumer using multicast transport with a reliable quality of service. If the topic is not configured for reliable multicast, you cannot create a message consumer for the topic.

NOTR
If the topic is configured for multicast in the broker, messages are delivered to the message consumer using multicast transport. A reliable quality of service is not used even if the topic is configured for reliable multicast.

OPTIMISTICPUBLICATION

This property determines whether IBM MQ classes for JMS returns control immediately to a publisher that has published a message, or whether it returns control only after it has completed all the processing associated with the call and can report the outcome to the publisher.

Applicable Objects

ConnectionFactory, TopicConnectionFactory

JMS administration tool long name: OPTIMISTICPUBLICATION

JMS administration tool short name: OPTPUB

Programmatic access

Setters/getters

- MQConnectionFactory.setOptimisticPublication()
- MQConnectionFactory.getOptimisticPublication()

Values

NO When a publisher publishes a message, IBM MQ classes for JMS do not return control to the publisher until it has completed all the processing associated with the call and can report the outcome to the publisher. This is the default value.

YES When a publisher publishes a message, IBM MQ classes for JMS returns control to the publisher immediately, before it has completed all the processing associated with the call and can report the outcome to the publisher. IBM MQ classes for JMS reports the outcome only when the publisher commits the message.

OUTCOMENOTIFICATION

This property determines whether IBM MQ classes for JMS return control immediately to a subscriber that has just acknowledged or committed a message, or whether it returns control only after it has completed all the processing associated with the call and can report the outcome to the subscriber.

Applicable Objects

ConnectionFactory, TopicConnectionFactory

JMS administration tool long name: OUTCOMENOTIFICATION

JMS administration tool short name: NOTIFY

Programmatic access

Setters/getters

- MQConnectionFactory.setOutcomeNotification()
- MQConnectionFactory.getOutcomeNotification()

Values

YES When a subscriber acknowledges or commits a message, IBM MQ classes for JMS do not return control to the subscriber until it has completed all the processing associated with the call and can report the outcome to the subscriber. This is the default value.

NO When a subscriber acknowledges or commits a message, IBM MQ classes for JMS returns control to the subscriber immediately, before it has completed all the processing associated with the call and can report the outcome to the subscriber.

PERSISTENCE

The persistence of messages sent to a destination.

Applicable Objects

Queue, Topic

JMS administration tool long name: PERSISTENCE

JMS administration tool short name: PER

Programmatic access

Setters/getters

- MQDestination.setPersistence()
- MQDestination.getPersistence()

Values

APP Persistence is defined by the JMS application. This is the default value.

QDEF Persistence takes the value of the queue default.

PERS Messages are persistent.

NON Messages are nonpersistent.

HIGH See JMS persistent messages for further information on the use of this value.

POLLINGINT

If each message listener within a session has no suitable message on its queue, this is the maximum interval, in milliseconds, that elapses before each message listener tries again to get a message from its queue. If it frequently happens that no suitable message is available for any of the message listeners in a session, consider increasing the value of this property. This property is relevant only if TRANSPORT has the value BIND or CLIENT.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: POLLINGINT

JMS administration tool short name: PINT

Programmatic access

Setters/getters

- MQConnectionFactory.setPollingInterval()
- MQConnectionFactory.getPollingInterval()

Values

5000 This is the default value.

Any positive integer

PORT

For a connection to a queue manager, the number of the port on which the queue manager is listening or, for a real-time connection to a broker, the number of the port on which the broker is listening for real-time connections.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: PORT

JMS administration tool short name: PORT

Programmatic access

Setters/getters

- MQConnectionFactory.setPort()
- MQConnectionFactory.getPort()

Values

1414 This is the default value if TRANSPORT is set to CLIENT.

1506 This is the default value if TRANSPORT is set to DIRECT or DIRECTHTTP.

Any positive integer

PRIORITY

The priority for messages sent to a destination.

Applicable Objects

Queue, Topic

JMS administration tool long name: PRIORITY

JMS administration tool short name: PRI

Programmatic access

Setters/getters

- MQDestination.setPriority()
- MQDestination.getPriority()

Values

APP Priority is defined by the JMS application. This is the default value.

QDEF Priority takes the value of the queue default.

Any integer in the range 0-9
Lowest to highest.

PROCESSDURATION

This property determines whether a subscriber guarantees to process quickly any message it receives before returning control to IBM MQ classes for JMS.

Applicable Objects

ConnectionFactory, TopicConnectionFactory

JMS administration tool long name: PROCESSDURATION

JMS administration tool short name: PROCDUR

Programmatic access

Setters/getters

- MQConnectionFactory.setProcessDuration()
- MQConnectionFactory.getProcessDuration()

Values

UNKNOWN
A subscriber can give no guarantee about how quickly it can process any message it receives. This is the default value.

SHORT
A subscriber guarantees to process quickly any message it receives before returning control to IBM MQ classes for JMS.

PROVIDERVERSION

This property differentiates between the three IBM MQ messaging modes of operation: IBM MQ messaging provider normal mode, IBM MQ messaging provider normal mode with restrictions, and IBM MQ messaging provider migration mode.

The IBM MQ messaging provider normal mode uses all the features of an IBM MQ queue manager to implement JMS. This mode is optimized to use the JMS 2.0 API and functionality. The IBM MQ messaging provider normal mode with restrictions uses the JMS 2.0 API, but not the new features such as shared subscriptions, delayed delivery, or asynchronous send.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnection Factory, XATopicConnectionFactory

JMS administration tool long name: PROVIDERVERSION

JMS administration tool short name: PVER

Programmatic access

Setters/getters

- MQConnectionFactory.setProviderVersion()
- MQConnectionFactory.getProviderVersion()

Values

You can set the **PROVIDERVERSION** property to any of the values 8 (normal mode), 7 (normal mode with restrictions), 6 (migration mode), or unspecified (the default value). The value that you specify for the **PROVIDERVERSION** property must be a string. If you are specifying an option of 8, 7 or 6, you can do this in any of the following formats:

- V.R.M.F
- V.R.M
- V.R
- V

where V, R, M and F are integer values greater than or equal to zero. The extra R, M and F values are optional and are available for you to use in case fine grained control is needed. For example, if you wanted to use a **PROVIDERVERSION** level of 7, you could set **PROVIDERVERSION**=7, 7.0, 7.0.0 or 7.0.0.0.

8 - Normal mode

The JMS application uses the IBM MQ messaging provider normal mode. Normal mode uses all the features of a IBM MQ queue manager to implement JMS. This mode is optimized to use the JMS 2.0 API and functionality.

If you are connecting to a queue manager with a command level of 800, then all of the JMS 2.0 API and features, such as asynchronous send, delayed delivery, or shared subscription, can be used.

If the queue manager specified in the connection factory settings is not a Version 8.0.0 queue manager, the createConnection method fails with an exception JMSFMQ0003.

The IBM MQ messaging provider normal mode uses the sharing conversations feature and the number of conversations that can be shared is controlled by the **SHARECNV()** property on the server connection channel. If this property is set to 0, you cannot use IBM MQ messaging provider normal mode and the createConnection method fails with an exception JMSCC5007.

7 - Normal mode with restrictions

The JMS application uses the IBM MQ messaging provider normal mode with restrictions. This mode uses the JMS 2.0 API, but not the new features such as shared subscriptions, delayed delivery, or asynchronous send.

If you set **PROVIDERVERSION** to 7 only the IBM MQ messaging provider normal with restrictions mode of operation is available. If the queue manager specified in the connection factory settings is not a Version 7.0.1, or later, queue manager, the createConnection method fails with exception JMSFCC5008.

If you are connecting using normal mode with restrictions, to a queue manager with a command level between 700 and 800 then you can use the JMS 2.0 API, but not the asynchronous send, delayed delivery, or shared subscription features.

The IBM MQ messaging provider normal mode with restrictions uses the sharing conversations feature and the number of conversations that can be shared is controlled by the **SHARECNV()** property on the server connection channel. If this property is set to 0, you cannot use IBM MQ messaging provider normal mode with restrictions and the createConnection method fails with an exception JMSCC5007.

6 - Migration mode

The JMS application uses the IBM MQ messaging provider migration mode.

The IBM MQ classes for JMS use the features and algorithms supplied with IBM WebSphere MQ Version 6.0. If you want to connect to WebSphere Message Broker Version 6.0 or 6.1 using IBM WebSphere MQ Enterprise Transport Version 6.0, you must use this mode. You can connect to a IBM MQ Version 8.0 queue manager using this mode, but none of the new features of a IBM MQ classes for JMS queue manager are used, for example, read ahead or streaming.

If you have a IBM MQ Version 8.0 or later client connecting to a IBM MQ Version 8.0 or later queue manager, then the message selection is done by the queue manager rather than on the client system.

If IBM MQ messaging provider migration mode is specified and you attempt to use any of the JMS 2.0 API, the API method call fails with the exception `JMSCC5007`.

`unspecified (default)`

The **`PROVIDERVERSION`** property is set to *unspecified* by default.

A connection factory that was created with a previous version of IBM MQ classes for JMS in JNDI takes this value when the connection factory is used with the new version of IBM MQ classes for JMS. The following algorithm is used to determine which mode of operation is used. This algorithm is used when the createConnection method is called and uses other aspects of the connection factory to determine if IBM MQ messaging provider normal mode, normal mode with restrictions, or IBM MQ messaging provider migration mode is required.

1. First, an attempt to use IBM MQ messaging provider normal mode is made.
2. If the queue manager connected is not IBM MQ Version 8.0 or later, an attempt to use IBM MQ messaging provider normal mode with restrictions is made.
3. If the queue manager connected is not IBM WebSphere MQ Version 7.0.1, or later, the connection is closed and IBM MQ messaging provider migration mode is used instead.
4. If the **`SHARECNV`** property on the server connection channel is set to 0, the connection is closed and IBM MQ messaging provider migration mode is used instead.
5. If **`BROKERVER`** is set to `V1` or the default *unspecified* value, IBM MQ messaging provider normal mode continues to be used, and therefore any publish/subscribe operations use the new IBM WebSphere MQ Version 7.0.1, or later, features.

   See ALTER QMGR for information about the PSMODE parameter of the ALTER QMGR command for further information on compatibility.
6. If **`BROKERVER`** is set to `V2` the action taken depends on the value of **`BROKERQMGR`** :
   - If the **`BROKERQMGR`** is `blank`:

     If the queue specified by the **`BROKERCONQ`** property can be opened for output (that is, MQOPEN for output succeeds) and **`PSMODE`** on the queue manager is set to `COMPAT` or `DISABLED`, then IBM MQ messaging provider migration mode is used.
   - If the queue specified by the **`BROKERCONQ`** property cannot be opened for output, or the **`PSMODE`** attribute is set to `ENABLED`**:**

     IBM MQ messaging provider normal mode is used.
   - If **`BROKERQMGR`** is `non-blank` :

     IBM MQ messaging provider migration mode is used.

If you cannot change the connection factory that you are using, you can use the `com.ibm.msg.client.wmq.overrideProviderVersion` property to override any setting on the connection factory. This override applies to all connection factories in the JVM but the actual connection factory objects are not modified.

Related information:
Configuring the JMS **PROVIDERVERSION** property

PROXYHOSTNAME

The host name or IP address of the system on which the proxy server is running when using a real-time connection to a broker through a proxy server.

Applicable Objects

ConnectionFactory, TopicConnectionFactory

JMS administration tool long name: PROXYHOSTNAME

JMS administration tool short name: PHOST

Programmatic access

Setters/getters

- MQConnectionFactory.setProxyHostName()
- MQConnectionFactory.getProxyHostName()

Values

null The host name of the proxy server. This is the default value.

PROXYPORT

The number of the port on which the proxy server is listening when using a real-time connection to a broker through a proxy server.

Applicable Objects

ConnectionFactory, TopicConnectionFactory

JMS administration tool long name: PROXYPORT

JMS administration tool short name: PPORT

Programmatic access

Setters/getters

MQConnectionFactory.setProxyPort()

MQConnectionFactory.getProxyPort()

Values

443 The port number of the proxy server. This is the default value.

PUBACKINT

The number of messages published by a publisher before IBM MQ classes for JMS requests an acknowledgment from the broker.

When you lower the value of this property, IBM MQ classes for JMS requests acknowledgments more often, therefore the performance of the publisher decreases. When you raise the value, IBM MQ classes for JMS take a longer time to throw an exception if the broker fails. This property is relevant only if TRANSPORT has the value BIND or CLIENT.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: PROXYPORT

JMS administration tool short name: PPORT

Programmatic access

Setters/getters

MQConnectionFactory.setPubAckInterval()

MQConnectionFactory.getPubAckInterval()

Values

25 Any positive integer may be the default value.

PUTASYNCALLOWED

This property determines whether message producers are allowed to use asynchronous puts to send messages to this destination.

Applicable Objects

Queue, Topic

JMS administration tool long name: PUTASYNCALLOWED

JMS administration tool short name: PAALD

Programmatic access

Setters/getters

MQDestination.setPutAsyncAllowed()

MQDestination.getPutAsyncAllowed()

Values

AS_DEST
Determine whether asynchronous puts are allowed by referring to the queue or topic definition. This is the default value.

AS_Q_DEF
Determine whether asynchronous puts are allowed by referring to the queue definition.

AS_TOPIC_DEF
Determine whether asynchronous puts are allowed by referring to the topic definition.

NO Asynchronous puts are not allowed.

YES Asynchronous puts are allowed.

QMANAGER

The name of the queue manager to connect to.

However, if your application uses a client channel definition table to connect to a queue manager, see Using a client channel definition table with IBM MQ classes for JMS.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, Queue, XAConnectionFactory, XAQueueConnection Factory, XATopicConnectionFactory

JMS administration tool long name: QMANAGER

JMS administration tool short name: QMGR

Programmatic access

Setters/getters

- MQConnectionFactory.setQueueManager()
- MQConnectionFactory.getQueueManager()

Values

" " (empty string)
Any string can be the default value.

QUEUE

The name of the JMS queue destination. This matches the name of the queue used by the queue manager.

Applicable Objects

Queue

JMS administration tool long name: QUEUE

JMS administration tool short name: QU

Values

Any string
Any valid IBM MQ queue name.

Related information:
Rules for naming IBM MQ objects>

READAHEADALLOWED

This property determines whether message consumers and queue browsers are allowed to use read ahead to get nonpersistent messages from this destination into an internal buffer before receiving them.

Applicable Objects

Queue, Topic

JMS administration tool long name: READAHEADALLOWED

JMS administration tool short name: RAALD

Programmatic access

Setters/getters

- MQDestination.setReadAheadAllowed()
- MQDestination.getReadAheadAllowed()

Values

AS_DEST
Determine whether read ahead is allowed by referring to the queue or topic definition. This is the default value in administrative tools.

Use WMQConstants.WMQ_READ_AHEAD_ALLOWED_AS_DEST in programs.

AS_Q_DEF
Determine whether read ahead is allowed by referring to the queue definition.

Use WMQConstants.WMQ_READ_AHEAD_ALLOWED_AS_Q_DEF in programs.

AS_TOPIC_DEF
Determine whether read ahead is allowed by referring to the topic definition.

Use WMQConstants.WMQ_READ_AHEAD_ALLOWED_AS_TOPIC_DEF in programs.

NO Read ahead is not allowed.

Use WMQConstants.WMQ_READ_AHEAD_ALLOWED_DISABLED in programs.

YES Read ahead is allowed.

Use WMQConstants.WMQ_READ_AHEAD_ALLOWED_ENABLED in programs.

READAHEADCLOSEPOLICY

For messages being delivered to an asynchronous message listener, what happens to messages in the internal read ahead buffer when the message consumer is closed.

Applicable Objects

Queue, Topic

JMS administration tool long name: READAHEADCLOSEPOLICY

JMS administration tool short name: RACP

Programmatic access

Setters/getters

- MQDestination.setReadAheadClosePolicy()
- MQDestination.getReadAheadClosePolicy()

Values

DELIVER_ALL
: All messages in the internal read ahead buffer are delivered to the message listener of the application before returning. This is the default value in administrative tools.

  Use WMQConstants.WMQ_READ_AHEAD_DELIVERALL in programs.

DELIVER_CURRENT
: Only the current message listener invocation completes before returning, potentially leaving messages in the internal read ahead buffer, which are then discarded.

  Use WMQConstants.WMQ_READ_AHEAD_DELIVERCURRENT in programs.

RECEIVECCSID

Destination property that sets the target CCSID for queue manager message conversion. The value is ignored unless RECEIVECONVERSION is set to `WMQ_RECEIVE_CONVERSION_QMGR`

Applicable Objects

Queue, Topic

JMS administration tool long name: RECEIVECCSID

JMS administration tool short name: RCCS

Programmatic access

Setters/Getters

- MQDestination.setReceiveCCSID
- MQDestination.getReceiveCCSID

Values

`WMQConstants.WMQ_RECEIVE_CCSID_JVM_DEFAULT`
: **0** - Use JVM Charset.defaultCharset

1208 `UTF-8`

CCSID
: Supported coded character set identifier.

RECEIVECONVERSION

Destination property that determines if data conversion is going to be performed by the queue manager.

Applicable Objects

Queue, Topic

JMS administration tool long name: RECEIVECONVERSION

JMS administration tool short name: RCNV

Programmatic access

Setters/Getters

- MQDestination.setReceiveConversion
- MQDestination.getReceiveConversion

Values

`WMQConstants.WMQ_RECEIVE_CONVERSION_CLIENT_MSG`
1 - Only perform data conversion on the JMS client. The default value from up to V7.0, and from, and including, 7.0.1.5.

`WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR`
2 - Perform data conversion on the queue manager before sending a message to the client. The default (and only) value from V7.0 to V7.0.1.4 inclusive, except if APAR IC72897 is applied.

RECEIVEISOLATION

This property determines whether a subscriber might receive messages that have not been committed on the subscriber queue.

Applicable Objects

ConnectionFactory, TopicConnectionFactory

JMS administration tool long name: RECEIVEISOLATION

JMS administration tool short name: RCVISOL

Values

COMMITTED
A subscriber receives only those messages on the subscriber queue that have been committed. This is the default value in administrative tools.

Use WMQConstants.WMQ_RCVISOL_COMMITTED in programs.

UNCOMMITTED
A subscriber can receive messages that have not been committed on the subscriber queue.

Use WMQConstants.WMQ_RCVISOL_UNCOMMITTED in programs.

RECEXIT

Identifies a channel receive exit, or a sequence of receive exits, to be run in succession.

Additional configuration might be required in order for the IBM MQ classes for JMS to locate receive exits. For more information, see Configuring the IBM MQ classes for JMS to use channel exits.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: RECEXIT

JMS administration tool short name: RCX

Programmatic access

Setters/getters

- MQConnectionFactory.setReceiveExit()
- MQConnectionFactory.getReceiveExit()

Values

- `null`. This is the default value.
- A string comprising one or more items separated by commas, where each item is either:
  - The name of a class that implements the WMQReceiveExit interface (for a channel receive exit written in Java).
  - A string in the format *libraryName(entryPointName)* (for a channel receive exit not written in Java).

RECEXITINIT

The user data that is passed to channel receive exits when they are called.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: RECEXITINIT

JMS administration tool short name: RCXI

Programmatic access

Setters/getters

- MQConnectionFactory.setReceiveExitInit()
- MQConnectionFactory.getReceiveExitInit()

Values

null A string comprising one or more items of user data separated by commas. This is the default value.

REPLYTOSTYLE

Determines how the JMSReplyTo field in a received message is constructed.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: REPLYTOSTYLE

JMS administration tool short name: RTOST

Programmatic access

Setters/getters

- MQConnectionFactory.setReplyToStyle()
- MQConnectionFactory.getReplyToStyle()

Values

DEFAULT
Equivalent to `MQMD`.

RFH2 Use the value supplied in the RFH2 header. If a JMSReplyTo value has been set in the sending application, use that value.

MQMD
Use the MQMD supplied value. This behavior is equivalent to the default behavior of IBM WebSphere MQ Version 6.0.2.4 and 6.0.2.5.

If the JMSReplyTo value set by the sending application does not contain a queue manager name, the receiving queue manager inserts its own name in the MQMD. If you set this parameter to `MQMD`, the reply-to queue you use is on the receiving queue manager. If you set this parameter to `RFH2`, the reply-to queue you use is on the queue manager specified in the RFH2 of the sent message as originally set by the sending application.

If the JMSReplyTo value set by the sending application contains a queue manager name, the value of this parameter is unimportant because both the MQMD and RFH2 contain the same value.

RESCANINT

When a message consumer in the point-to-point domain uses a message selector to select which messages it wants to receive, IBM MQ classes for JMS search the IBM MQ queue for suitable messages in the sequence determined by the `MsgDeliverySequence` attribute of the queue.

After IBM MQ classes for JMS find a suitable message and deliver it to the consumer, IBM MQ classes for JMS resume the search for the next suitable message from its current position in the queue. IBM MQ classes for JMS continue to search the queue in this way until it reaches the end of the queue, or until the interval of time in milliseconds, as determined by the value of this property, has expired. In each case, IBM MQ classes for JMS return to the beginning of the queue to continue the search, and a new time interval commences.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory

JMS administration tool long name: RESCANINT

JMS administration tool short name: RINT

Programmatic access

Setters/getters

- MQConnectionFactory.setRescanInterval()
- MQConnectionFactory.getRescanInterval()

Values

5000 Any positive integer can be the default value.

SECEXIT

Identifies a channel security exit.

Additional configuration might be required in order for the IBM MQ classes for JMS to locate security exits. For more information, see Configuring the IBM MQ classes for JMS to use channel exits.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SECEXIT

JMS administration tool short name: SXC

Programmatic access

Setters/getters

- MQConnectionFactory.setSecurityExit()
- MQConnectionFactory.getSecurityExit()

Values

- `null`. This is the default value.
- A string comprising one or more items separated by commas, where each item is either:
  - The name of a class that implements the WMQSecurityExit interface (for a channel security exit written in Java).
  - A string in the format *libraryName(entryPointName)* (for a channel security exit not written in Java).

SECEXITINIT

The user data that is passed to a channel security exit when it is called.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SECEXITINIT

JMS administration tool short name: SCXI

Programmatic access

Setters/getters

- MQConnectionFactory.setSecurityExitInit()
- MQConnectionFactory.getSecurityExitInit()

Values

null Any string can be the default value.

SENDCHECKCOUNT

The number of send calls to allow between checking for asynchronous put errors, within a single non-transacted JMS session.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SENDCHECKCOUNT

JMS administration tool short name: SCC

Programmatic access

Setters/getters

- MQConnectionFactory.setSendCheckCount()
- MQConnectionFactory.getSendCheckCount()

Values

null Any string can be the default value.

SENDEXIT

Identifies a channel send exit, or a sequence of send exits to be run in succession.

Additional configuration might be required in order for the IBM MQ classes for JMS to locate send exits. For more information, see Configuring the IBM MQ classes for JMS to use channel exits.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SENDEXIT

JMS administration tool short name: SDX

Programmatic access

Setters/getters

- MQConnectionFactory.setSendExit()
- MQConnectionFactory.getSendExit()

Values

- `null`. This is the default value.
- A string comprising one or more items separated by commas, where each item is either:
  - The name of a class that implements the WMQSendExit interface (for a channel send exit written in Java).
  - A string in the format *libraryName(entryPointName)* (for a channel send exit not written in Java).

SENDEXITINIT

The user data that is passed to channel send exits when they are called.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SENDEXITINIT

JMS administration tool short name: SDXI

Programmatic access

Setters/getters

- MQConnectionFactory.setSendExitInit()
- MQConnectionFactory.getSendExitInit()

Values

null Any string comprising one or more items of user data separated by commas can be the default value.

SHARECONVALLOWED

This property determines whether a client connection can share its socket with other top-level JMS connections from the same process to the same queue manager, if the channel definitions match.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SHARECONVALLOWED

JMS administration tool short name: SCALD

Programmatic access

Setters/getters

- MQConnectionFactory.setShareConvAllowed()
- MQConnectionFactory.getShareConvAllowed()

Values

YES This is the default value for administrative tools.

For programs, use WMQConstants.WMQ_SHARE_CONV_ALLOWED_YES.

NO This value is for administrative tools.

For programs, use WMQConstants.WMQ_SHARE_CONV_ALLOWED_NO.

SPARSESUBS

Controls the message retrieval policy of a TopicSubscriber object.

Applicable Objects

ConnectionFactory, TopicConnectionFactory

JMS administration tool long name: SPARSESUBS

JMS administration tool short name: SSUBS

Programmatic access

Setters/getters

- MQConnectionFactory.setSparseSubscriptions()
- MQConnectionFactory.getSparseSubscriptions()

Values

NO Subscriptions receive frequent matching messages. This is the default value for administrative tools.

For programs, use false.

YES Subscriptions receive infrequent matching messages. This value requires that the subscription queue can be opened for browse.

For programs, use true.

SSLCIPHERSUITE

The CipherSuite to use for a TLS connection.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SSLCIPHERSUITE

JMS administration tool short name: SCPHS

Programmatic access

Setters/getters

- MQConnectionFactory.setSSLCipherSuite()
- MQConnectionFactory.getSSLCipherSuite()

Values

null This is the default value. For more information, see TLS properties of JMS objects.

SSLCRL

CRL servers to check for TLS certificate revocation.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SSLCRL

JMS administration tool short name: SCRL

Programmatic access

Setters/getters

- MQConnectionFactory.setSSLCertStores()
- MQConnectionFactory.getSSLCertStores()

Values

null Space-separated list of LDAP URLs. This is the default value. For more information, see TLS properties of JMS objects.

SSLFIPSREQUIRED

This property determines whether a TLS connection must use a CipherSuite that is supported by the IBM Java JSSE FIPS provider (IBMJSSEFIPS).

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SSLFIPSREQUIRED

JMS administration tool short name: SFIPS

Programmatic access

Setters/getters

- MQConnectionFactory.setSSLFipsRequired()
- MQConnectionFactory.getSSLFipsRequired()

Values

NO A TLS connection can use any CipherSuite that is not supported by the IBM Java JSSE FIPS provider (IBMJSSEFIPS).

This is the default value. In programs, use false.

YES A TLS connection must use a CipherSuite that is supported by IBMJSSEFIPS.

In programs, use true.

SSLPEERNAME

For TLS, a *distinguished name* skeleton that must match that provided by the queue manager.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SSLPEERNAME

JMS administration tool short name: SPEER

Programmatic access

Setters/getters

- MQConnectionFactory.setSSLPeerName()
- MQConnectionFactory.getSSLPeerName()

Values

null This is the default value. For more information, see TLS properties of JMS objects.

SSLRESETCOUNT

For TLS, the total number of bytes sent and received by a connection before the secret key that is used for encryption is renegotiated.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SSLRESETCOUNT

JMS administration tool short name: SRC

Programmatic access

Setters/getters

- MQConnectionFactory.setSSLResetCount()
- MQConnectionFactory.getSSLResetCount()

Values

0 Zero, or any positive integer less than or equal to 999, 999, 999. This is the default value. For more information, see TLS properties of JMS objects.

STATREFRESHINT

The interval, in milliseconds, between refreshes of the long running transaction that detects when a subscriber loses its connection to the queue manager.

This property is relevant only if SUBSTORE has the value QUEUE.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: STATREFRESHINT

JMS administration tool short name: SRI

Programmatic access

Setters/getters

- MQConnectionFactory.setStatusRefreshInterval()
- MQConnectionFactory.getStatusRefreshInterval()

Values

60000 Any positive integer can be the default value. For more information, see TLS properties of JMS objects.

SUBSTORE

Where IBM MQ classes for JMS stores persistent data relating to active subscriptions.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SUBSTORE

JMS administration tool short name: SS

Programmatic access

Setters/getters

- MQConnectionFactory.setSubscriptionStore()
- MQConnectionFactory.getSubscriptionStore()

Values

BROKER
: Use the broker-based subscription store to hold details of subscriptions. This is the default value for administrative tools.

  For programs, use WMQConstants.WMQ_SUBSTORE_BROKER.

MIGRATE
: Transfer subscription information from the queue-based subscription store to the broker-based subscription store.

  For programs, use WMQConstants.WMQ_SUBSTORE_MIGRATE.

QUEUE
: Use the queue-based subscription store to hold details of subscriptions.

  For programs, use WMQConstants.WMQ_SUBSTORE_QUEUE.

SYNCPOINTALLGETS

This property determines whether all gets are to be performed under syncpoint.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SYNCPOINTALLGETS

JMS administration tool short name: SPAG

Programmatic access

Setters/getters

- MQConnectionFactory.setSyncpointAllGets()
- MQConnectionFactory.getSyncpointAllGets()

Values

No This is the default value.

Yes

TARGCLIENT

This property determines whether the IBM MQ RFH2 format is used to exchange information with target applications.

Applicable Objects

Queue, Topic

JMS administration tool long name: TARGCLIENT

JMS administration tool short name: TC

Programmatic access

Setters/getters

- MQDestination.setTargetClient()
- MQDestination.getTargetClient()

Values

JMS The target of the message is a JMS application. This is the default value for administrative tools.

For programs, use WMQConstants.WMQ_CLIENT_JMS_COMPLIANT.

MQ The target of the message is a non-JMS IBM MQ application.

For programs, use WMQConstants.WMQ_CLIENT_NONJMS_MQ.

TARGCLIENTMATCHING

This property determines whether a reply message, sent to the queue identified by the JMSReplyTo header field of an incoming message, has an MQRFH2 header only if the incoming message has an MQRFH2 header.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory

JMS administration tool long name: TARGCLIENTMATCHING

JMS administration tool short name: TCM

Programmatic access

Setters/getters

- MQConnectionFactory.setTargetClientMatching()
- MQConnectionFactory.getTargetClientMatching()

Values

YES If an incoming message does not have an MQRFH2 header, the TARGCLIENT property of the Queue object derived from the JMSReplyTo header field of the message is sent to MQ. If the message does have an MQRFH2 header, the TARGCLIENT property is set to JMS instead. This is the default value for administrative tools.

For programs, use true.

NO The TARGCLIENT property of the Queue object derived from the JMSReplyTo header field of an incoming message is always set to JMS.

For programs, use false.

TEMPMODEL

The name of the model queue from which JMS temporary queues are created.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory

JMS administration tool long name: TEMPMODEL

JMS administration tool short name: TM

Programmatic access

Setters/getters

- MQConnectionFactory.setTemporaryModel()
- MQConnectionFactory.getTemporaryModel()

Values

SYSTEM.DEFAULT.MODEL.QUEUE
Any string can be the default value.

TEMPQPREFIX

The prefix that is used to form the name of an IBM MQ dynamic queue.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory

JMS administration tool long name: TEMPQPREFIX

JMS administration tool short name: TQP

Programmatic access

Setters/getters

- MQConnectionFactory.setTempQPrefix()
- MQConnectionFactory.getTempQPrefix()

Values

" " (empty string)
The prefix used is `CSQ.*` on z/OS and `AMQ.*` on all other platforms. These are the default values.

queue prefix
The queue prefix is any string that conforms to the rules for forming contents of the *DynamicQName* field in an IBM MQ object descriptor (structure MQOD), but the last non-blank character must be an asterisk.

TEMPTOPICPREFIX

When creating temporary topics, JMS generates a topic string of the form "TEMP */TEMPTOPICPREFIX/unique_id* ", or if this property is left with the default value, just "TEMP */unique_id* ". Specifying a non-empty TEMPTOPICPREFIX allows specific model queues to be defined for creating the managed queues for subscribers to temporary topics created under this connection.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: TEMPTOPICPREFIX

JMS administration tool short name: TTP

Programmatic access

Setters/getters

- MQConnectionFactory.setTempTopicPrefix()
- MQConnectionFactory.getTempTopicPrefix()

Values

Any non-null string consisting only of valid characters for an IBM MQ topic string. The default value is " " (empty string).

TOPIC

The name of the JMS topic destination, this value is used by the queue manager as the topic string of a publication or subscription.

Applicable Objects

Topic

JMS administration tool long name: TOPIC

JMS administration tool short name: TOP

Values

Any string

A string that forms a valid IBM MQ topic string. When using IBM MQ as a messaging provider with WebSphere Application Server, specify a value that matches the name by which the topic is known for administrative purposes within WebSphere Application Server.

Related information:
Topic strings

TRANSPORT

The nature of a connection to a queue manager or broker.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: TRANSPORT

JMS administration tool short name: TRAN

Programmatic access

Setters/getters

- MQConnectionFactory.setTransportType()
- MQConnectionFactory.getTransportType()

Values

BIND For a connection to a queue manager in bindings mode. This is the default value for administrative tools.

For programs, use WMQConstants.WMQ_CM_BINDINGS.

CLIENT
For a connection to a queue manager in client mode.

For programs, use WMQConstants.WMQ_CM_CLIENT.

DIRECT
For a real-time connection to a broker not using HTTP tunnelling.

For programs, use WMQConstants.WMQ_CM_DIRECT_TCPIP.

DIRECTHTTP
For a real-time connection to a broker using HTTP tunnelling. Only HTTP 1.0 is supported.

For programs, use WMQConstants.WMQ_CM_DIRECT_HTTP.

Related concepts:

"Dependencies between properties of IBM MQ classes for JMS objects" on page 4036
The validity of some properties is dependent on the particular values of other properties.

WILDCARDFORMAT

This property determines which version of wildcard syntax is to be used.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: WILDCARDFORMAT

JMS administration tool short name: WCFMT

Programmatic access

Setters/getters

- MQConnectionFactory.setWildCardFormat()
- MQConnectionFactory.getWildCardFormat()

Values

TOPIC_ONLY
: Recognizes topic level wildcards only, as used in broker version 2. This is the default value for administrative tools.

  For programs, use WMQConstants.WMQ_WILDCARD_TOPIC_ONLY.

CHAR_ONLY
: Recognizes character wildcards only, as used in broker version 1.

  For programs, use WMQConstants.WMQ_WILDCARD_CHAR_ONLY.

The ENCODING property

The ENCODING property comprises three sub-properties, in twelve possible combinations.

The valid values that the `ENCODING` property can take are constructed from the three sub-properties:

integer encoding
: Either normal or reversed

decimal encoding
: Either normal or reversed

floating-point encoding
: IEEE normal, IEEE reversed, or z/OS

The `ENCODING` property is expressed as a three-character string with the following syntax:
`{N|R}{N|R}{N|R|3}`

In this string:

- `N` denotes normal
- `R` denotes reversed
- `3` denotes z/OS
- The first character represents *integer encoding*
- The second character represents *decimal encoding*

- The third character represents *floating-point encoding*

This provides a set of twelve possible values for the ENCODING property.

There is an additional value, the string NATIVE, which sets appropriate encoding values for the Java platform.

The following examples show valid combinations for ENCODING:

```
ENCODING(NNR)
ENCODING(NATIVE)
ENCODING(RR3)
```

TLS properties of JMS objects

Enable Transport Layer Security (TLS) encryption using the SSLCIPHERSUITE property. You can then change the characteristics of the TLS encryption using several other properties.

When you specify TRANSPORT(CLIENT), you can enable TLS encrypted communication using the SSLCIPHERSUITE property. Set this property to a valid CipherSuite provided by your JSSE provider; it must match the CipherSpec named on the SVRCONN channel named by the CHANNEL property.

However, CipherSpecs (as specified on the SVRCONN channel) and CipherSuites (as specified on ConnectionFactory objects) use different naming schemes to represent the same TLS encryption algorithms. If a recognized CipherSpec name is specified on the SSLCIPHERSUITE property, JMSAdmin issues a warning and maps the CipherSpec to its equivalent CipherSuite. See TLS CipherSpecs and CipherSuites in IBM MQ classes for JMS for a list of CipherSpecs recognized by IBM MQ and JMSAdmin.

If you require a connection to use a CipherSuite that is supported by the IBM Java JSSE FIPS provider (IBMJSSEFIPS), set the SSLFIPSREQUIRED property of the connection factory to YES. The default value of this property is NO, which means that a connection can use any supported CipherSuite. The property is ignored if SSLCIPHERSUITE is not set.

The SSLPEERNAME matches the format of the SSLPEER parameter, which can be set on channel definitions. It is a list of attribute name-value pairs separated by commas or semicolons. For example:

```
SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

The set of names and values makes up a *distinguished name*. For more details about distinguished names and their use with IBM MQ, see Securing.

The example given checks the identifying certificate presented by the server at connect-time. For the connection to succeed, the certificate must have a Common Name beginning QMGR., and must have at least two Organizational Unit names, the first of which is IBM and the second WEBSPHERE. Checking is not case-sensitive.

If SSLPEERNAME is not set, no such checking is performed. SSLPEERNAME is ignored if SSLCIPHERSUITE is not set.

The SSLCRL property specifies zero or more CRL (Certificate Revocation List) servers. Use of this property requires a JVM at Java 2 v1.4. This is a space-delimited list of entries of the form:

```
ldap:// hostname:[ port ]
```

optionally followed by a single /. If *port* is omitted, the default LDAP port of 389 is assumed. At connect-time, the TLS certificate presented by the server is checked against the specified CRL servers. See Securing for more about CRL security.

If SSLCRL is not set, no such checking is performed. SSLCRL is ignored if SSLCIPHERSUITE is not set.

The SSLRESETCOUNT property represents the total number of bytes sent and received by a connection before the secret key that is used for encryption is renegotiated. The number of bytes sent is the number before encryption, and the number of bytes received is the number after decryption. The number of bytes also includes control information sent and received by IBM MQ classes for JMS.

For example, to configure a ConnectionFactory object that can be used to create a connection over an TLS enabled MQI channel with a secret key that is renegotiated after 4 MB of data have flowed, issue the following command to JMSAdmin:

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

If the value of SSLRESETCOUNT is zero, which is the default value, the secret key is never renegotiated. The SSLRESETCOUNT property is ignored if SSLCIPHERSUITE is not set.

Messaging REST API reference

V 9.0.4

Reference information about the messaging REST API.

For more information about using the messaging REST API, see Messaging using the REST API.

REST API resources

V 9.0.4

This collection of topics provides reference information for each of the messaging REST API resources.

For more information about using the messaging REST API, see Messaging using the REST API.

/messaging/qmgr/{qmgrName}/queue/{queueName}/message: V 9.0.4

You can use the HTTP POST method with the `/messaging/qmgr/{qmgrName}/queue/{queueName}/message` resource to put messages to the specified queue on the specified queue manager. You can use the HTTP DELETE method with the `/messaging/qmgr/{qmgrName}/queue/{queueName}/message` resource to get messages from the specified queue on the specified queue manager.

POST: V 9.0.4

You can use the HTTP POST method with the `/messaging/qmgr/{`*qmgrName*`}/queue/{`*queueName*`}/message` resource to put messages to the associated queue manager and queue.

Puts an IBM MQ message containing the HTTP request body to the specified queue manager and queue. The method only supports text based HTTP request bodies. Messages are sent as `MQSTR` formatted messages, and are put using the current user context.

Resource URL

`https://host:port/ibmmq/rest/v1/messaging/qmgr/{qmgrName}/queue/{queueName}/message`

qmgrName
Specifies the name of the queue manager to connect to for messaging.

The queue manager name is case-sensitive.

If the queue manager name includes a forward slash, a period, or a percent sign, these characters must be URL encoded:

- A forward slash (/) must be encoded as `%2F`.
- A percent sign (%) must be encoded as `%25`.

queueName
Specifies the name of the queue on which to put the message.

The queue must be defined as being local, remote, or an alias to the specified queue manager - it may also reference a clustered queue.

The queue name is case sensitive.

If the queue name includes a forward slash or a percent sign, these characters must be URL encoded:

- A forward slash, /, must be encoded as `%2F`.
- A percent sign, %, must be encoded as `%25`.

V 9.0.1 You can use HTTP instead of HTTPS if you enable HTTP connections. For more information about enabling HTTP, see Configuring HTTP and HTTPS ports.

Request headers

The following headers must be sent with the request:

Authorization
This header must be sent if you are using basic authentication. For more information, see Using HTTP basic authentication with the REST API.

Content-Type
This header must be sent with one of the following values:

- `text/plain;charset=utf-8`
- `text/html;charset=utf-8`
- `text/xml;charset=utf-8`
- `application/json;charset=utf-8`
- `application/xml;charset=utf-8`

Note: If *charset* is omitted from the `Context-Type` header, `UTF-8` is assumed.

ibm-mq-rest-csrf-token
This header must be sent with a value that is the content of the `csrfToken` cookie. The content of the `csrfToken` cookie is used to confirm that the credentials that are being used to authenticate the request are being used by the owner of the credentials. That is, the token is used to prevent cross-site request forgery attacks.

The `csrfToken` cookie is returned after a request is made with an HTTP GET method. You cannot use a cached version of the content of the cookie because the content of the cookie can change. You must use the latest value of the cookie for each request.

V 9.0.5 The preceding information applies to releases up to and including IBM MQ Version 9.0.4. From IBM MQ Version 9.0.5, this header must be set, but the value can be anything, including

being blank.
The `csrfToken` cookie is no longer sent in responses from the REST API in Version 9.0.5 and later.

The following headers can optionally be sent with the request:

`Accept-Language`
This header specifies the required language for any exceptions or error messages returned in the response message body.

`ibm-mq-md-correlationId`
This header sets the correlation ID of the created message. The header must be specified as a 48 character hexadecimal encoded string, representing 24 bytes.

For example:

```
ibm-mq-md-correlationId: 414d5120514d414445562020202020202067d8bf5923582e02
```

`ibm-mq-md-expiry`
This header sets the expiry duration for the created message. The expiry of a message starts from the time the message arrives on the queue. As a result network latency is ignored. The header must be specified as one of the following values:

- `unlimited` *(default)*
  - The message does not expire.
- *`Integer value`*
  - Milliseconds before message expiry.
  - Limited to the range 0 - 99999999900.

`ibm-mq-md-persistence`
This header sets the persistence for the created message. The header must be specified as one of the following values:

- `nonPersistent` *(default)*
  - The message does not survive system failures or queue manager restarts.
- `persistent`
  - The message survives system failures or queue manager restarts.

`ibm-mq-md-replyTo`
This header sets the reply-to destination for the created message. The format of the header uses the standard notation of supplying the reply-to queue and an optional queue manager: `replyQueue[@replyQmgr]`

For example:

```
ibm-mq-md-replyTo: myReplyQueue@myReplyQMgr
```

Request body format

The request body must be text and use UTF-8 encoding. No specific text structure is required. An `MQSTR` formatted message containing the request body text is created and put to the specified queue.

For more information, see examples.

Security requirements

The caller must be authenticated to the mqweb server. The `MQWebAdmin` and `MQWebAdminRO` roles are not applicable for the messaging REST API. For more information about security for the REST API, see IBM MQ Console and REST API security.

Once authenticated to the mqweb server the user is capable of using both the messaging REST API and the administrative REST API.

The security principal of the caller must be granted the ability to put messages to the specified queue:

- AIX Linux Windows For the queue that is specified by the *{queueName}* portion of the resource URL, `+PUT` authority must be granted to the security principal of the caller.
- z/OS For the queue that is specified by the *{queueName}* portion of the resource URL, `UPDATE` access must be granted to the security principal of the caller.
- The queue that is specified by the *{queueName}* portion of the resource URL, must be `PUT` enabled.

ULW On UNIX, Linux, and Windows, you can grant authority to security principals to use IBM MQ resources by using the **mqsetaut** command. For more information, see mqsetaut.

z/OS On z/OS, see Setting up security on z/OS.

Response status codes

201 Message created and sent successfully.

400 Invalid data provided.

For example, an invalid request header value was specified.

401 Not authenticated.

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. The `ibm-mq-rest-csrf-token` header must also be specified. For more information, see "Security requirements" on page 4092.

403 Not authorized.

The caller is authenticated to the mqweb server and is associated with a valid principal. However, the principal does not have access to all, or a subset of the required IBM MQ resources, or is not in the `MQWebUser` role. For more information about the access that is required, see "Security requirements" on page 4092.

404 Queue does not exist.

405 Queue is PUT inhibited.

415 A message header or body is an unsupported media type.

For example, the `Content-Type` header is set to an unsupported media type.

500 Server issue or error code from IBM MQ.

502 The current security principal cannot send the message as the messaging provider does not support the required function. For example, if the mqweb server class path is invalid.

503 Queue manager not running.

Response headers

The following headers are returned with the response:

`Content-Language`
Specifies the language identifier of the response message in the event of any errors or exceptions. Used in conjunction with `Accept-Language` request header to indicate the required language for any error or exception conditions. The mqweb server default is used if the requested language is unsupported.

`Content-Length`
Specifies the length of the HTTP response body, even when there is no content. Upon success the value is zero.

Content-Type
Specifies the type of response body. Upon success the value is `text/plain;charset=utf-8`. In the event of any errors or exceptions, the value is `application/json;charset=utf-8`.

ibm-mq-md-messageId
Specifies the message ID that is allocated by IBM MQ to this message. Like the `ibm-mq-md-correlationId` request header, it is represented as a 48 character hexadecimal encoded string, representing 24 bytes.

For example:

```
ibm-mq-md-messageId: 414d5120514d4144455620202020202067d8ce5923582f07
```

Response body format

The response body is empty if the message is sent successfully. If an error occurs, the response body contains an error message. For more information, see REST API error handling.

Examples

The following example logs in a user called `mquser` with the password `mquser`. In cURL, the log in request might look like the following Windows example. The LTPA token is stored in the `cookiejar.txt` file by using the `-c` flag:

```
curl -k "https://localhost:9443/ibmmq/rest/v1/login" -X POST
-H "Content-Type: application/json" --data "{\"username\":\"mquser\",\"password\":\"mquser\"}"
-c c:\cookiejar.txt
```

V 9.0.5 After the user is logged in, the LTPA token and `ibm-mq-rest-csrf-token` HTTP header are used to authenticate further requests.

For IBM MQ Version 9.0.4 and earlier, after the user is logged in, the LTPA token and CSRF token are used to authenticate further requests.

V 9.0.5

Note: In the following examples, for releases before IBM MQ Version 9.0.5, `token_value` is the value of the `csrfToken` cookie, and from IBM MQ Version 9.0.5 `token_value` is any value, including blank.

- The following Windows cURL example sends a message to queue `Q1` on queue manager `QM1`, using default options. The message contains the text *"Hello World!"*:

  ```
  curl -k "https://localhost:9443/ibmmq/rest/v1/messaging/qmgr/QM1/queue/Q1/message"
  -X POST -b c:\cookiejar.txt -H "ibm-mq-rest-csrf-token: token_value"
  -H "Content-Type: text/plain;charset=utf-8" --data "Hello World!"
  ```

- The following Windows cURL example sends a persistent message to queue `Q1` on queue manager `QM1`, with an expiry of 2 minutes. The message contains the text *"Hello World!"*:

  ```
  curl -k "https://localhost:9443/ibmmq/rest/v1/messaging/qmgr/QM1/queue/Q1/message"
  -X POST -b c:\cookiejar.txt -H "ibm-mq-rest-csrf-token: token_value"
  -H "Content-Type: text/plain;charset=utf-8" -H "ibm-mq-md-persistence: persistent"
  -H "ibm-mq-md-expiry: 120000" --data "Hello World!"
  ```

- The following Windows cURL example sends a non-persistent message to queue `Q1` on queue manager `QM1`, with no expiry and defined correlation ID. The message contains the text *"Hello World!"*:

  ```
  curl -k "https://localhost:9443/ibmmq/rest/v1/messaging/qmgr/QM1/queue/Q1/message"
  -X POST -b c:\cookiejar.txt -H "ibm-mq-rest-csrf-token: token-value"
  -H "Content-Type: text/plain;charset=utf-8" -H "ibm-mq-md-persistence: nonPersistent"
  -H "ibm-mq-md-expiry: unlimited" -H "ibm-mq-md-correlationId: 414d5120514d4144455620202020202067d8b
  f5923582e02" --data "Hello World!"
  ```

DELETE: V 9.0.4

You can use the HTTP DELETE method with the `/messaging/qmgr/{qmgrName}/queue/{queueName}/message` resource to get messages from the associated queue manager and queue.

Destructively gets the next available message from the specified queue manager and queue, returning the message body in the HTTP response body. The message must have a format of `MQSTR`, and is received using the current user context.

Incompatible messages are left on the queue and an appropriate status code returned to the caller. For example, a message which does not have a `MQSTR` format.

Resource URL

`https://host:port/ibmmq/rest/v1/messaging/qmgr/{qmgrName}/queue/{queueName}/message`

qmgrName
Specifies the name of the queue manager to connect to for messaging.

The queue manager name is case-sensitive.

If the queue manager name includes a forward slash, a period, or a percent sign, these characters must be URL encoded:

- A forward slash (/) must be encoded as `%2F`.
- A percent sign (%) must be encoded as `%25`.

queueName
Specifies the name of the queue from which to get the next message.

The queue must be defined as being local or an alias pointing to a local queue.

The queue name is case sensitive.

If the queue name includes a forward slash or a percent sign, these characters must be URL encoded:

- A forward slash, /, must be encoded as `%2F`.
- A percent sign, %, must be encoded as `%25`.

V 9.0.1 You can use HTTP instead of HTTPS if you enable HTTP connections. For more information about enabling HTTP, see Configuring HTTP and HTTPS ports.

Optional query parameters

correlationId=*hexValue*
Specifies that the HTTP method returns the next message with the corresponding correlation ID.

hexValue
The query parameter must be specified as a 48 character hexadecimal encoded string, representing 24 bytes.

For example:

```
../message?correlationId=414d5120514d4144455620202020202067d8bf5923582e02
```

messageId=*hexValue*
Specifies that the HTTP method returns the next message with the corresponding message ID.

hexValue
The query parameter must be specified as a 48 character hexadecimal encoded string, representing 24 bytes.

For example:

```
../message?messageId=414d5120514d4144455620202020202067d8ce5923582f07
```

wait=*integerValue*
Specifies that the HTTP method will wait *integerValue* milliseconds for the next message to become available.

integerValue
The query parameter must be specified as an integer value representing the millisecond duration. The maximum value is 2147483647.

For example:

```
../message?wait=120000
```

Request headers

The following headers must be sent with the request:

Authorization
This header must be sent if you are using basic authentication. For more information, see Using HTTP basic authentication with the REST API.

ibm-mq-rest-csrf-token
This header must be sent with a value that is the content of the `csrfToken` cookie. The content of the `csrfToken` cookie is used to confirm that the credentials that are being used to authenticate the request are being used by the owner of the credentials. That is, the token is used to prevent cross-site request forgery attacks.

The `csrfToken` cookie is returned after a request is made with an HTTP GET method. You cannot use a cached version of the content of the cookie because the content of the cookie can change. You must use the latest value of the cookie for each request.

V 9.0.5 The preceding information applies to releases up to and including IBM MQ Version 9.0.4. From IBM MQ Version 9.0.5, this header must be set, but the value can be anything, including being blank.
The `csrfToken` cookie is no longer sent in responses from the REST API in Version 9.0.5 and later.

The following headers can optionally be sent with the request:

Accept-Charset
This header can be used to indicate what character set is acceptable for the response. If specified, this header must be set as `UTF-8`.

Accept-Language
This header specifies the required language for any exceptions or error messages returned in the response message body.

Request body format

None.

Security requirements

The caller must be authenticated to the mqweb server. The `MQWebAdmin` and `MQWebAdminRO` roles are not applicable for the messaging REST API. For more information about security for the REST API, see IBM MQ Console and REST API security.

Once authenticated to the mqweb server the user is capable of using both the messaging REST API and the administrative REST API.

The security principal of the caller must be granted the ability to get messages from the specified queue:

- AIX Linux Windows For the queue that is specified by the *{queueName}* portion of the resource URL, `+GET`, `+INQ`, and `+BROWSE` authority must be granted to the security principal of the caller.
- z/OS For the queue that is specified by the *{queueName}* portion of the resource URL, `UPDATE`, access must be granted to the security principal of the caller.
- The queue that is specified by the *{queueName}* portion of the resource URL, must be `GET` enabled.

ULW On UNIX, Linux, and Windows, you can grant authority to security principals to use IBM MQ resources by using the **mqsetaut** command. For more information, see mqsetaut.

z/OS On z/OS, see Setting up security on z/OS.

Response status codes

200 Message received successfully.

204 No message available.

400 Invalid data provided.

For example, an invalid query parameter value was specified.

401 Not authenticated.

The caller must be authenticated to the mqweb server and must be a member of one or more of the `MQWebAdmin`, `MQWebAdminRO`, or `MQWebUser` roles. The `ibm-mq-rest-csrf-token` header must also be specified. For more information, see "Security requirements."

403 Not authorized.

The caller is authenticated to the mqweb server and is associated with a valid principal. However, the principal does not have access to all, or a subset of the required IBM MQ resources, or is not in the `MQWebUser` role. For more information about the access that is required, see "Security requirements."

404 Queue does not exist.

405 Queue is GET inhibited.

500 Server issue or error code from IBM MQ.

501 The HTTP response could not be constructed.

For example, the received message has an incorrect type, or has the correct type but the body could not be processed.

502 The current security principal cannot receive the message as the messaging provider does not support the required function. For example, if the mqweb server class path is invalid.

503 Queue manager not running.

Response headers

The following headers are returned with the response:

Content-Language
Specifies the language identifier of the response message in the event of any errors or exceptions. Used in conjunction with `Accept-Language` request header to indicate the required language for any error or exception conditions. The mqweb server default is used if the requested language is unsupported.

Content-Length
Specifies the length of the HTTP response body, even when there is no content. The value contains the length (bytes) of the message data.

Content-Type
Specifies the type of content returned in the response body of the received message. Upon success the value is `text/plain;charset=utf-8`. In the event of any errors or exceptions, the value is `application/json;charset=utf-8`.

ibm-mq-md-correlationId
Specifies the correlation ID of the received message. The header is returned if the received message contains a valid correlation ID. It is represented as a 48 character hexadecimal encoded string, representing 24 bytes.

For example:

```
ibm-mq-md-correlationId: 414d5120514d414445562020202020067d8bf5923582e02
```

ibm-mq-md-expiry
Specifies the remaining expiry duration of the received message. The header can be one of the following values:

- `unlimited`
  - The message does not expire.
- *`Integer value`*
  - Remaining milliseconds before message expiry.

ibm-mq-md-messageId
Specifies the message ID that is allocated by IBM MQ to this message. Like the `ibm-mq-md-correlationId` header, it is represented as a 48 character hexadecimal encoded string, representing 24 bytes.

For example:

```
ibm-mq-md-messageId: 414d5120514d414445562020202020067d8ce5923582f07
```

ibm-mq-md-persistence
Specifies the persistence of the received message. The header can be one of the following values:

- `nonPersistent`
  - The message does not survive system failures or queue manager restarts.
- `persistent`
  - The message survives system failures or queue manager restarts.

ibm-mq-md-replyTo
Specifies the reply-to destination for the received message. The format of the header uses the standard notation of the reply-to queue and queue manager, `replyQueue@replyQmgr`.

For example:

```
ibm-mq-md-replyTo: myReplyQueue@myReplyQMgr
```

Response body format

Upon success, the response body contains the message body from the received message. If an error occurs, the response body contains a JSON formatted error message. Both responses are `UTF-8` encoded. For more information, see REST API error handling.

Examples

The following example logs in a user called `mquser` with the password `mquser`. In cURL, the log in request might look like the following Windows example. The LTPA token is stored in the `cookiejar.txt` file by using the `-c` flag:

```
curl -k "https://localhost:9443/ibmmq/rest/v1/login" -X POST
-H "Content-Type: application/json" --data "{\"username\":\"mquser\",\"password\":\"mquser\"}"
-c c:\cookiejar.txt
```

After the user is logged in, the LTPA token and `ibm-mq-rest-csrf-token` HTTP header are used to authenticate further requests.

V 9.0.5

Note: In the following examples, for releases before IBM MQ Version 9.0.5, `token-value` is the value of the `csrfToken` cookie, and from IBM MQ Version 9.0.5 `token-value` is any value, including blank.

- The following Windows cURL example sends a message to queue `Q1` on queue manager `QM1`, using default options. The message contains the text *"Hello World!"*:

  ```
  curl -k "https://localhost:9443/ibmmq/rest/v1/messaging/qmgr/QM1/queue/Q1/message"
  -X POST -b c:\cookiejar.txt -H "ibm-mq-rest-csrf-token: token-value"
  -H "Content-Type: text/plain;charset=utf-8" --data "Hello World!"
  ```

- The following Windows cURL example sends a persistent message to queue `Q1` on queue manager `QM1`, with an expiry of 2 minutes. The message contains the text *"Hello World!"*:

  ```
  curl -k "https://localhost:9443/ibmmq/rest/v1/messaging/qmgr/QM1/queue/Q1/message"
  -X POST -b c:\cookiejar.txt -H "ibm-mq-rest-csrf-token: token-value"
  -H "Content-Type: text/plain;charset=utf-8" -H "ibm-mq-md-persistence: persistent"
  -H "ibm-mq-md-expiry: 120000" --data "Hello World!"
  ```

- The following Windows cURL example sends a non-persistent message to queue `Q1` on queue manager `QM1`, with no expiry and defined correlation ID. The message contains the text *"Hello World!"*:

  ```
  curl -k "https://localhost:9443/ibmmq/rest/v1/messaging/qmgr/QM1/queue/Q1/message"
  -X POST -b c:\cookiejar.txt -H "ibm-mq-rest-csrf-token: token-value"
  -H "Content-Type: text/plain;charset=utf-8" -H "ibm-mq-md-persistence: nonPersistent"
  -H "ibm-mq-md-expiry: unlimited" -H "ibm-mq-md-correlationId: 414d5120514d414455620202020202067d8b
  f5923582e02" --data "Hello World!"
  ```

MQ Telemetry Reference

AIX Linux Windows

Information about programming MQTT clients

Related information:

MQTT client libraries programming reference

IBM MQ Telemetry Transport format and protocol

AIX Linux Windows

IBM MQ Telemetry Transport (MQTT) is a lightweight publish/subscribe protocol flowing over TCP/IP to connect large numbers of remote sensors and control devices. MQTT is used by specialized applications on small footprint devices that must tolerate low bandwidth and unreliable communication. You can use MQTT client apps from a business partner, or write your own client apps to use the published protocols. You can get sample client apps and supporting libraries from the Eclipse Paho project.

MQ Telemetry in IBM MQ supports client apps that use the MQTT protocol. There are currently two specifications of this protocol:

- The MQTT Version 3.1.1 Oasis standard
- The MQTT V3.1 Protocol Specification from `mqtt.org`

The Oasis standard is more recent. The functionality is almost identical to the `mqtt.org` specification. The MQTT Version 3.1.1 Oasis standard is supported in IBM MQ Version 8.0.0, Fix Pack 3 and later versions.

If your MQTT client comes from a source other than the IBM Messaging Telemetry Clients SupportPac or the Eclipse Paho project, check the version of the MQTT protocol supported by the client. If your client supports a different level of the MQTT protocol, and does not work correctly with the MQ Telemetry service, a thin conversion layer is required. Check with the source of your client to see if the conversion layer is available as an update to the client you intend to use.

MQXR properties

AIX Linux Windows

MQXR property settings are stored in a platform-specific properties file: `mqxr_win.properties` or `mqxr_unix.properties`. You normally configure these properties by using MQSC admin commands or IBM MQ Explorer.

When you start a queue manager for the first time, the template version of the MQXR properties file for your platform is copied from the *mqinstall*`/mqxr/config` directory to the *mqinstall*`/qmgrs/`*qmgr_name*`/mqxr/config` directory.

You do not normally need to edit the MQXR properties file directly, because all properties except one can be configured through MQSC admin commands or IBM MQ Explorer. If you do decide to edit the file directly, stop the queue manager before you make your changes.

The property that you can only set by editing the file directly is **webcontentpath**. If your telemetry client app is a web app, you also need to serve the web app executable JavaScript to the browser. This requirement is explained in The MQTT messaging client for JavaScript(tm) and web apps. You use the **webcontentpath** property to specify the directory from which the web app executable files are served:

- By default, **webcontentpath** is not present in the MQXR properties file. If **webcontentpath** is not present, the MQ telemetry server serves the web app executable files from the following default location: *mqinstall*/qmgrs/*qmgr_name*/mqxr/WebContent/*your_client_app*
- if **webcontentpath** specifies a path, the MQ telemetry server serves the web app executable files from that location.
- if **webcontentpath** is present and blank, the MQ telemetry server does not serve the web app executable files.

Related reference:

"AuthCallback MQXR class"
AuthCallback is the sole class in package com.ibm.mq.mqxr. It specifies the interface definition that a telemetry server administrator needs when they write an AuthCallback in the MQXR server.

Related information:

Telemetry (MQXR) service

AuthCallback MQXR class

AIX Linux Windows

AuthCallback is the sole class in package com.ibm.mq.mqxr. It specifies the interface definition that a telemetry server administrator needs when they write an AuthCallback in the MQXR server.

Class AuthCallback

```
java.lang.Object
  |
  ≪— com.ibm.mq.mqxr.AuthCallback
```

Implemented interface:

```
javax.security.auth.callback.Callback
```

```
public class AuthCallback
extends java.lang.Object
implements javax.security.auth.callback.Callback
```

Allows a JAAS login module (javax.security.auth.spi.LoginModule) to access WebSphereMQ Server objects.

Methods

getSSLSession

```
public javax.net.ssl.SSLSession getSSLSession()
```

Returns the javax.net.ssl.SSLSession associated with the client connection, or null if the client is connected using a plain text connection.

setSSLSession

```
public void setSSLSession(javax.net.ssl.SSLSession sslSession)
```

The **sslSession** parameter is set by the server to be the sslSession associated with the client connection, or null if the client is connected using a plain text connection.

Constructor

```
public AuthCallback()
```

Related reference:

"MQXR properties" on page 4100
MQXR property settings are stored in a platform-specific properties file: `mqxr_win.properties` or `mqxr_unix.properties`. You normally configure these properties by using MQSC admin commands or IBM MQ Explorer.

Related information:

Telemetry (MQXR) service

Security reference

Use the reference information in this section to help you configure security for IBM MQ.

Related concepts:

"The API exit"
An *API exit* is a program module that monitors or modifies the function of MQI calls. An API exit comprises multiple *API exit functions*, each with its own entry point in the module.

"The API-crossing exit" on page 4104
An *API-crossing exit* is a program that monitors or modifies the function of MQI calls issued by CICS applications on z/OS.

"Certificate validation and trust policy design on UNIX, Linux and Windows systems" on page 4105
IBM MQ validates TLS certificates according to two types of policy, basic, and standard. Standard policy checking conforms to RFC 5280.

"Cryptographic hardware" on page 4119
The way in which IBM MQ provides support for cryptographic hardware depends on which platform you are using.

"IBM MQ rules for SSLPEER values" on page 4120
The SSLPEER attribute is used to check the Distinguished Name (DN) of the certificate from the peer queue manager or client at the other end of an IBM MQ channel. IBM MQ uses certain rules when comparing these values

"Migrating with AltGSKit from Version 7.0.1 to Version 7.1" on page 4122
Perform this task only if you are migrating from IBM WebSphere MQ Version 7.0.1 using the AltGSKit configuration setting to load an alternative GSKit. The alternative GSKit used by IBM WebSphere MQ Version 7.0.1 with the AltGSKit setting is separate from the GSKit used by IBM WebSphere MQ Version 7.1; changes to each GSKit do not affect the other. This is because Version 7.1 uses a private local copy of GSKit in its installation directory and does not support the use of an alternative GSKit.

"CipherSpec mismatches" on page 4124
Both ends of an IBM MQ TLS channel must use the same CipherSpec. Mismatches can be detected during the TLS handshake or during channel startup.

"Authentication failures" on page 4125
There are a number common reasons for authentication failures during the TLS handshake.

Related reference:

"GSKit: Digital certificate signature algorithms compliant with FIPS 140-2" on page 4121
The list of digital certificate signature algorithms in GSKit that are compliant with FIPS 140-2

The API exit

An *API exit* is a program module that monitors or modifies the function of MQI calls. An API exit comprises multiple *API exit functions*, each with its own entry point in the module.

Note: The information in this section does not apply to IBM MQ for z/OS.

There are two categories of exit function:

An exit function that is associated with an MQI call

There are two exit functions in this category for each MQI call and an additional one for an MQGET call with the MQGMO_CONVERT option. The MQCONN and MQCONNX calls share the same exit functions.

For each MQI call, one of the two exit functions is invoked before the queue manager starts to process the call and the other is invoked after the queue manager has completed processing the call. The exit function for an MQGET call with the MQGMO_CONVERT option is invoked during the MQGET call, after the message has been retrieved from the queue by the queue manager but before any data conversion takes place. This allows, for example, a message to be decrypted before data conversion.

An exit function can inspect and modify any of the parameters on an MQI call. On an MQPUT call, for example, an exit function that is invoked before the processing of the call has started can:

- Inspect and modify the contents of the application data in the message being put
- Change the length of the application data in the message
- Modify the contents of the fields in the message descriptor structure, MQMD
- Modify the contents of the fields in the put message options structure, MQPMO

An exit function that is invoked before the processing of an MQI call has started can suppress the call completely. The exit function for an MQGET call with the MQGMO_CONVERT option can suppress data conversion of the message being retrieved.

Initialization and termination exit functions

There are two exit functions in this category, the initialization exit function and the termination exit function.

The initialization exit function is invoked by the queue manager when an application connects to the queue manager. Its primary purpose is to register exit functions and their entry points with the queue manager and perform any initialization processing. You do not have to register all the exit functions, only those that are required for this connection. When the application disconnects from the queue manager, the registrations are removed automatically.

The initialization exit function can also be used to acquire any storage required by the exit and examine the values of any environment variables.

The termination exit function is invoked by the queue manager when an application disconnects from the queue manager. Its purpose is to release any storage used by the exit and perform any required cleanup operations.

An API exit can issue calls to the MQI but, if it does, the API exit is not invoked recursively a second time. The following exit functions, however, are not able to issue MQI calls because the correct environment is not present at the time the exit functions are invoked:

- The initialization exit function
- The exit function for an MQCONN and MQCONNX call that is invoked *before* the queue manager starts to process the call
- The exit function for the MQDISC call that is invoked *after* the queue manager has completed processing the call
- The termination exit function

An API exit can also use other APIs that might be available; for example, it can issue calls to Db2.

An API exit can be used with an IBM MQ client application, but it is important to note that the exit is invoked at the *server* end of an MQI channel. For more information, see Comparing link level security and application level security.

An API exit is written using the C programming language.

To enable an API exit, you must configure it. On IBM i, Windows, UNIX and Linux systems, you do this by editing the IBM MQ configuration file, mqs.ini, and the queue manager configuration file, qm.ini, for each queue manager.

For a client, modify the `ApiExitLocal` stanza in the mqclient.ini file to identify API exit routines for a queue manager.

You configure an API exit by providing the following information:

- The descriptive name of the API exit.
- The name of the module and its location; for example, the full path name.
- The name of the entry point for the initialization exit function.
- The sequence in which the API exit is invoked relative to other API exits. You can configure more than one API exit for a queue manager.
- Optionally, any data to be passed to the API exit.

For more information about how to configure an API exit, see Configuring API exits.

For information about how to write an API exit, see Using and writing API exits.

The API-crossing exit

z/OS

An *API-crossing exit* is a program that monitors or modifies the function of MQI calls issued by CICS applications on z/OS.

Note: The information in this section applies only to CICS applications on z/OS.

The API-crossing exit program is invoked by the CICS adapter and runs in the CICS address space.

The API-crossing exit is invoked for the following MQI calls only:

MQBUFMH
MQCB
MQCB_FUNCTION
MQCLOSE
MQCRTMH
MQCTL
MQDLTMH
MQGET
MQINQ
MQOPEN
MQPUT
MQPUT1
MQSET
MQSTAT
MQSUB
MQSUBRQ

For each MQI call, it is invoked once before the processing of the call has started and once after the processing of the call has been completed.

The exit program can determine the name of an MQI call and can inspect and modify any of the parameters on the call. If it is invoked before an MQI call is processed, it can suppress the call completely.

The exit program can use any of the APIs that a CICS task-related user exit can use; for example, the IMS, Db2, and CICS APIs. It can also use any of the MQI calls except MQCONN, MQCONNX, and MQDISC. However, any MQI calls issued by the exit program do not invoke the exit program a second time.

You can write an API-crossing exit in any programming language supported by IBM MQ for z/OS.

Before an API-crossing exit can be used, the exit program load module must be available when the CICS adapter connects to a queue manager. The load module is a CICS program that must be named CSQCAPX and reside in a library in the DFHRPL concatenation sequence. CSQCAPX must be defined in the CICS system definition file (CSD), and the program must be enabled.

An API-crossing exit can be managed using the CICS adapter control panels, CKQC. When CSQCAPX is loaded, a confirmation message is written to the adapter control panels or to the system console. The adapter control panels can also be used to enable or disable the exit program.

For more information about how to write and implement an API-crossing exit, see “The CICS-IBM MQ Adapter” section in the CICS Transaction Server for z/OS Version 4.1 product documentation. See CICS Transaction Server for z/OS Version 4.1, The CICS-IBM MQ adapter.

Certificate validation and trust policy design on UNIX, Linux and Windows systems

ULW

IBM MQ validates TLS certificates according to two types of policy, basic, and standard. Standard policy checking conforms to RFC 5280.

The information in these topics applies to the following systems:

- IBM MQ for UNIX and Linux systems
- IBM MQ for Windows systems

The following terms are used in this section:

Certificate policy
Determines which fields in a certificate are understood and processed.

OCSP policy
Determines which fields in an OCSP request or response are understood and processed.

CRL policy
Determines which fields in a certificate revocation list are understood and processed.

Path validation policy
Determines how the certificate, OCSP, and CRL policy types interact with each other to determine whether a certificate chain (a trust point "RootCA" to an end-entry "EE") is valid.

The basic and standard path validation policies are described separately because it reflects the implementation within IBM MQ for UNIX, Linux and Windows systems. However, the standard OCSP and CRL policies are the same as the basic policies, and the standard certificate policy is an extended version of the basic policy, so these policies are not described separately.

By default, IBM MQ applies basic policy validation first. If basic policy validation fails, IBM MQ applies standard policy (RFC 5280) validation. If basic policy validation succeeds, standard policy validation is not applied. Thus, a validation failure means that both basic and standard policy validation failed, possibly for different reasons. A validation success means that either basic policy validation succeeded and standard policy validation was therefore not applied, or basic policy validation failed and standard policy validation succeeded.

Enforcing strict RFC 5280 compliance

To enforce strict RFC 5280 compliance, use the certificate validation policy configuration setting. This setting allows you to disable the basic policy, so that only the standard RFC 5280 policy is used. For more information about the certificate validation policy configuration setting, see Certificate validation policies in IBM MQ.

The following examples are digital certificates that are accepted by the basic certificate validation policy, but which are rejected by the RFC 5280 compliant standard policy. In order for a digital certificate chain to be trusted, the entire chain must satisfy the configured validation policy.

To view the full details of a digital certificate, use the **runmqakm** command:

```
runmqakm -cert -details -db key.kdb -pw password -label certificate_label
```

A certificate which has trust status enabled in the **runmqakm** output is not necessarily trusted for use in a TLS handshake. Trust status enabled means that the certificate is eligible to be used as a CA certificate to verify other certificates, if the certificate also satisfies the rules of the certificate validation policy. For more information about the RFC 5280 compliant standard certificate validation policy, see “Standard path validation policy” on page 4115.

Example certificate 1 - incorrect key usage

This example shows a certificate where the key usage field does not comply with the standard certificate validation policy rules for a CA certificate. One of the requirements for a certificate to be valid for use as a CA certificate is that the key usage field must indicate that it is permitted to sign other certificates using the keyCertSign flag. A certificate without this flag cannot be used as a CA certificate.

```
Label : root
Key Size : 1024
Version : X509 V3
Serial : 54cb6f740c7ee410
Issuer : CN=Example Root CA,O=Example,C=GB
Subject : CN=Example Root CA,O=Example,C=GB
Not Before : 9 February 2012 17:19:00 GMT
Not After : 1 October 2019 18:19:00 GMT+01:00
Public Key
    30 81 9F 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01
    05 00 03 81 8D 00 30 81 89 02 81 81 00 CC 44 D9
    25 6D 26 1C 9D B9 FF DE B8 AC 44 AB E3 64 80 44
    AF BE E0 00 93 53 92 33 F8 7E BD D7 71 ED 21 52
    24 75 DF D6 EE 3C 54 97 84 29 EA 93 4C 4A D1 19
    5D C1 A0 82 F5 74 E1 AD D9 87 10 D5 6A 2B 6F 90
    04 0F 7E 6E 85 6D 32 99 33 9C D9 BB 57 86 DE 68
    23 C9 F2 6D 53 E3 F5 FF D1 0B E7 23 19 3A F6 70
    6B C8 C7 EB DB 78 8E 8C 9E 55 58 66 B6 31 DB 40
    5F 6A 97 AB 12 D7 E2 3E 2E 79 EE 78 7B 02 03 01
    00 01
Public Key Type : RSA (1.2.840.113549.1.1.1)
Fingerprint : SHA1 :
    EE 68 D4 4F 73 4F F4 21 DE 1A 01 11 5E DE B1 B8
    DF 40 AA D8
Fingerprint : MD5 :
    50 B5 E9 B2 D7 35 05 6A DC 6D 4B 1E B2 F2 DF A4
Fingerprint : SHA256 :
    B4 D7 6E C4 47 26 24 C7 4F 41 C3 83 03 6F 5C C7
```

```
    07 11 61 E0 0E 36 59 1F 1C E6 69 39 2D 18 05 D2
Extensions
    basicConstraints
        ca = true
        pathLen = 1239876
        critical
    key usage: encipherOnly
Signature Algorithm : SHA256WithRSASignature (1.2.840.113549.1.1.11)
Value
    9D AE 54 A9 9D 68 01 68 15 B5 53 9F 96 C9 5B D1
    52 40 DB CB 33 AF FD B9 26 D5 90 3F 1E 0B FC A6
    D9 8C 04 90 EB AA FD A8 7A 3C AB 60 5F 20 4F 0D
    7B 73 41 27 6A 2B BF 8C 99 91 B6 49 96 82 6A 24
    0A E8 B9 A5 AF 69 3D 2C A3 3C C8 12 39 FB 56 58
    4E 2A FE AC AC 10 89 53 B1 8F 0F C0 50 BF 5E 00
    91 64 B4 A1 4C 9A 4E D5 1F 38 7C AD 32 A9 8A E1
    91 16 2C 6D 1E 4A CA 99 8D CC 22 CD BF 90 49 FC
Trust Status : Enabled
```

In this example, the key usage field contains only the encipherOnly flag. The keyCertSign flag is not set, so this certificate is not permitted to sign other certificates. It therefore cannot be used as a CA certificate.

Example certificate 2 - missing basic constraints extension

This example shows a certificate which lacks the basic constraints extension. The basic constraints extension is used to indicate whether this certificate is permitted for use as a CA. It is also used to indicate the maximum length of any certificate chain which can be signed by the certificate. The standard certificate validation policy requires that the certificate has a basic constraints extension with the isCA flag set in order to be used as a CA.

```
Label : root
Key Size : 1024
Version : X509 V3
Serial : 1c7dfea316570bf6
Issuer : CN=Second Example Root CA,O=Example,C=GB
Subject : CN=Second Example Root CA,O=Example,C=GB
Not Before : 9 February 2012 17:18:22 GMT
Not After : 1 October 2019 18:18:22 GMT+01:00
Public Key
    30 81 9F 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01
    05 00 03 81 8D 00 30 81 89 02 81 81 00 B2 70 49
    7C AE 1B A7 B3 06 49 6C 99 19 BC A8 77 BE 86 33
    21 6B C9 26 CC A6 28 52 9F 7B CF 03 A4 37 A7 4D
    6B 06 AA ED 7D 58 E3 70 F3 F7 C1 06 DA E8 27 C6
    3D 1B AC FA EF AA 59 7A 9A AB C1 14 4E AF 13 14
    4B 71 CA 8D FE C3 F5 2F E8 AC AD EF 21 80 6D 12
    89 4A 2A 84 AA 9D E0 4F C1 93 B1 3E 16 E8 3C 75
    39 2A 74 1E 90 CC B1 C3 2B 1D 55 26 76 D2 65 C1
    06 47 2A BF 79 96 42 76 A9 6E 65 88 5F 02 03 01
    00 01
Public Key Type : RSA (1.2.840.113549.1.1.1)
Fingerprint : SHA1 :
    33 9F A1 81 43 F1 43 95 48 A5 66 B4 CD 98 E8 15
    9C B3 CA 90
Fingerprint : MD5 :
    91 EA D9 C0 2C 05 5B E2 CD 0B F6 DD 8A 11 44 23
Fingerprint : SHA256 :
    62 46 35 0B 0E A1 A7 2A D5 74 70 0F AA 47 9A 9C
    6B 80 1B F1 0B 4C 81 05 85 0E 91 11 A4 21 D2 34
Extensions
    key usage: digitalSignature, keyCertSign
Signature Algorithm : SHA256WithRSASignature (1.2.840.113549.1.1.11)
Value
    79 34 BA 5B 6F DC 06 A3 99 24 4E 8A 2B 27 05 47
    0D 4D BE 6A 77 D1 1D 5F 54 82 9D CC F6 92 D4 9A
    AB 4D B6 DD 6E AD 86 C3 6A A3 32 E3 B3 ED E0 62
```

```
    4A EB 51 08 AC BE 49 9E 9C D7 FE AE C8 9D 17 16
    68 31 6B F4 BA 74 1E 4F 5F 05 48 9F E7 46 BA DC
    17 7A 60 88 F8 5B DB 3C 51 D4 98 97 28 82 CF 36
    47 DA D2 0F 47 FF 70 EA 45 3A 49 66 E6 E2 F9 67
    2C C8 3E 24 A2 3B EC 76 1F D6 31 2B BD A9 B5 08
Trust Status : Enabled
```

In this example, the certificate lacks the basic constraints field entirely. Therefore this certificate cannot be used as a CA certificate.

Example certificate 3 - intermediate CA with old version of X.509

This example shows an intermediate CA certificate which is at X.509 version 1. The standard certificate validation policy requires that all intermediate CA certificates must be at least X.509 version 3. Root CA certificates are exempt from this requirement as there are still some commonly used version 1 root CA certificates in existence. However, this exemption might change in future.

```
Label : intermediate
Key Size : 1024
Version : X509 V1
Serial : 02
Issuer : CN=Test Root CA,O=Example,C=GB
Subject : CN=Test Intermediate CA,O=Example,C=GB
Not Before : 10 February 2012 17:33:45 GMT
Not After : 11 April 2018 18:33:45 GMT+01:00
Public Key
    30 81 9F 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01
    05 00 03 81 8D 00 30 81 89 02 81 81 00 C0 07 C2
    D0 9F 84 DB 7C 20 8F 51 F9 C2 1A 3F CF E2 D7 F2
    F1 56 F2 A4 8F 8F 06 B7 3B 01 31 DE 7C CC 03 63
    AA D3 2F 1C 50 15 E3 56 80 40 7D FF 75 87 D3 F3
    00 89 9A 26 F5 57 05 FA 4F ED 3B DD 93 FA F2 DF
    38 26 D4 3A 92 51 CC F3 70 27 42 7A 9F AD 51 45
    67 B7 AE 11 AD 4F 2D AB D2 CF 73 E6 F0 45 92 F0
    47 16 66 7E 01 C7 76 A3 7B EC D2 76 3F E5 15 EC
    D7 72 2C FE 14 F5 78 83 AA C4 20 AB F7 02 03 01
    00 01
Public Key Type : RSA (1.2.840.113549.1.1.1)
Fingerprint : SHA1 :
    DE BB 75 4B 14 E1 44 B9 B6 44 33 97 49 D0 82 6D
    81 F2 2F DE
Fingerprint : MD5 :
    72 49 44 42 E2 E6 89 F1 CC 37 C9 F6 B5 8F F3 AE
Fingerprint : SHA256 :
    83 A4 52 AF 49 34 F1 DC 49 E6 95 AE 93 67 80 13
    C2 64 D9 26 22 A0 E8 0A 5A A9 71 EC E8 33 E1 D1
Signature Algorithm : SHA256WithRSASignature (1.2.840.113549.1.1.11)
Value
    40 4A 09 94 A0 18 07 5E 96 D7 A6 52 6B 8D 20 50
    E8 91 F7 7E EA 76 B4 08 DF 76 66 1F FA FF 91 79
    2E E0 66 8B 9F 40 FA 14 13 79 81 DB 31 A5 55 1D
    44 67 41 F4 EA 1A F7 83 4F 21 F4 43 78 4E F8 5E
    6F B2 B8 3A F7 6B B4 F5 C6 F8 EB 4C BF 62 6F 3E
    C7 20 EC 53 B3 40 51 36 C1 0A 4E 73 ED 74 D1 93
    02 C5 FB 61 F7 87 64 A5 94 06 7D 25 7C E3 73 DD
    08 D4 07 D0 A4 3F 77 88 12 59 DB A4 DB 68 8F C1
Trust Status : Enabled
```

In this example, the version field is X.509 V1. This certificate is an X.509 version 1 certificate and therefore cannot be used as an intermediate CA.

Basic and standard certificate policies

ULW

The basic and standard certificate policies support the same fields: the standard policy supports additional certificate extensions.

The supported fields for both the basic and standard policies are as follows:

- OuterSigAlgID [11]
- Signature [12]
- Version
- SerialNumber
- InnerSigAlgID [13]
- Issuer
- Validity
- SubjectName
- SubjectPublicKeyInfo
- IssuerUniqueID
- SubjectUniqueID

The supported extensions for the basic policy are as follows. Where an entry is marked as "not supported", IBM MQ does not attempt to process extensions containing a field of that specific type, but does process other types of the same extension.

- AuthorityKeyID
- AuthorityInfoAccess
- SubjectKeyID
- IssuerAltName
- SubjectAltName
- KeyUsage
- BasicConstraints
- PrivateKeyUsage
- CRLDistributionPoints
  - DistributionPoint
    - DistributionPointName (X.500 Name and LDAP Format URI only)
    - NameRelativeToCRLIssuer (not supported)
    - Reasons (ignored)
    - CRLIssuer fields (not supported)

The supported extensions for the standard policy are all those listed for the basic policy and those in the following list. Where an entry is marked as "not supported", IBM MQ does not attempt to process extensions containing a field of that specific type, but does process other types of the same extension.

- NameConstraints
- ExtendedKeyUsage
- CertificatePolicies

---

11. This field is called *signatureAlgorithm* in RFC 5280.
12. This field is called *signatureValue* in RFC 5280.
13. This field is called *signature* in RFC 5280.

  - PolicyInformation
    - PolicyIdentifier
    - PolicyQualifiers (not supported)
- PolicyMappings
- PolicyConstraints

Basic and standard OCSP policies

The basic and standard OCSP policies support the same fields.

The supported fields for a request are as follows. Where an entry is marked as "not supported", IBM MQ does not attempt to process a request containing a field of that specific type, but does process other requests containing the same higher-level field.

- Signature (Optional)
- Version (Version 1 Only)
- RequesterName (Optional)
- RequestList (single request only)
  - CertID [14]
  - singleRequestExtensions (not supported)
- RequestExtensions
  - Nonce (if enabled)

The supported fields for a response are as follows:

- ResponseStatus
- Response
  - responseType (id-pkix-ocsp-basic)
  - BasicOCSPResponse
    - Signature
    - Certs
      - Extensions
      - extendedKeyUsage
        - id-kp-OCSPSigning
      - id-pkix-ocsp-nocheck
    - ResponseData
      - Version (Version 1 Only)
      - ResponderID (by name or by hash)
      - ProducedAt (ignored)
      - Responses (multiple responses supported)
        - SingleResponse
          - certID
          - certStatus
            - RevokedInfo (ignored)
          - thisUpdate (ignored)
          - nextUpdate

14. This field is called reqCert in RFC 2560

- singleExtensions (ignored)
- responseExtensions
  - Nonce (if enabled)

Basic and standard CRL policies

ULW

The basic and standard CRL policies support the same fields and extensions.

The supported fields for these policies are as follows:
- OuterSigAlgID [15]
- Signature [16]
- Version
- InnerSigAlgID [17]
- Issuer
- ThisUpdate
- NextUpdate
- RevokedCertificate
  - UserCertificate
  - RevocationDate

There are no supported CRLEntry extensions.

The supported CRL extensions for these policies are as follows. Where an entry is marked as "not supported", IBM MQ does not attempt to process extensions containing a field of that specific type, but does process other types of the same extension.
- AuthorityKeyID
- IssuerAltName
- CRLNumber
- IssuingDistributionPoint
  - DistributionPoint
  - DistributionPointName
    - FullName (X.500 Name and LDAP Format URI only)
    - NameRelativeToCRLIssuer (not supported)
  - Reasons (ignored)
  - CRLIssuer
  - OnlyContainsUserCerts (not supported)
  - OnlyContainsCACerts (not supported)
  - OnlySomeReasons (not supported)
  - IndirectCRL [18] (rejected)

15. This field is called *signatureAlgorithm* in RFC 5280.
16. This field is called *signatureValue* in RFC 5280.
17. This field is called *signature* in RFC 5280.
18. IndirectCRL extensions will result in CRL validation failing. IndirectCRL extensions must not be used because they cause identified certificates to not be rejected.

Basic path validation policy

ULW

The basic path validation policy determines how the certificate, OCSP, and CRL policy types interact with each other to determine if a certificate chain is valid.

The validation of a chain is performed in the following manner (but not necessarily in the following order):

1. Ensure that the name of the certificate's issuer is equal to the subject name in the previous certificate, and that there is not an empty issuer name in this certificate or the previous certificate subject name. If no previous certificate exists in the path and this is the first certificate in the chain, ensure that the issuer and subject name are identical and that the trust status is set for the certificate [19] .

   Note: IBM MQ for UNIX, Linux and Windows systems will fail path validation in situations where the previous certificate in a path has the same subject name as the current certificate.
2. Ensure that the signature algorithm used to actually sign the certificate matches the signature algorithm indicated within the certificate, by ensuring that the issuer signature algorithm identifier in the certificate matches the algorithm identifier in the signature data.
3. Ensure that the certificate was signed by the issuer, using the subject public key from the previous certificate in the path to verify the signature on the certificate. If no previous certificate exists and this is the first certificate, use the subject public key of the certificate to verify the signature on it. IBM MQ supports DSA and RSA signature algorithms; however it does not support DSA Parameter Inheritance.
4. Ensure that the certificate is a known X509 version, unique IDs are not present for version 1 certificates, and extensions are not present for version 1 and version 2 certificates.
5. Ensure that the certificate has not expired, or not been activated yet, and that its validity period is good [20] .
6. Ensure that there are no unknown critical extensions or any duplicate extensions.
7. Ensure that the certificate has not been revoked. Here, the following operations apply:
   a. If the OCSP connection is enabled and a Responder Address is configured or the Certificate has a valid AuthorityInfoAccess extension specifying a HTTP format GENERALNAME_uniformResourceID check revocation status with OCSP.
   b. If revocation status from 7a above is undetermined the CRLDistributionPoints extension is checked for a list of X.500 distinguished name GENERALNAME_directoryname and URI GENERALNAME_uniformResourceID. Only LDAP, HTTP and FILE format URIs are supported. If the extension is not present, or use of the CRLDistributionPoints extension results in undetermined status and the extension is not Critical, the certificate's issuer's name is used to query revocation status. A CRL database (LDAP) is then queried for CRLs. If the certificate is not the last certificate, or if the last certificate has the basic constraint extension with the "isCA" flag turned on, the database is queried for ARLs and CRLs instead. If CRL checking is enabled, and no CRL database can be queried, the certificate is treated as revoked. Currently, the X500 directory name form and the LDAP/HTTP/FILE URI forms are the only supported name forms used to look up CRLs and ARLs [21] .

19. Trust status is an administrative setting in the key database file. You can access and alter the trust status of a particular signer certificate in iKeyman. Select the required certificate from the signer list and click **View/Edit**. The **Set the certificate as a trusted root** check box on the resulting panel indicates the trust status. You can also set Trust status using **iKeycmd** with the `-trust` flag on the **-cert -modify** command. For further information about this command, see Managing keys and certificates.
20. There are no checks to ensure the subject's validity is within bounds of the issuer's validity. This is not required, and it has been shown that certificates from some CAs do not pass such a check.
21. After they are retrieved from the database, ARLs are evaluated in exactly the same fashion as CRLs. Many CAs do not issue ARLs. However, IBM MQ will look for ARLs and CRLs if checking a CA certificate for revocation status.

Note: RelativeDistinguishedNames are not supported.

c. If revocation status from both 7a on page 4112 and 7b on page 4112 is undetermined, IBM MQ checks the *OCSPAuthentication* configuration setting to decide whether to allow the connection. [22]

8. If the issuerAltName extension is marked critical, ensure that the name forms are recognized. The following general name forms are currently recognized:
   - rfc822
   - DNS
   - directory
   - URI
   - IPAddress(v4/v6)
9. If the subjectAltName extension is marked critical, ensure that the name forms are recognized. The following general name forms are currently recognized:
   - rfc822
   - DNS
   - directory
   - URI
   - IPAddress(v4/v6)
10. If the KeyUsage extension is critical on a non-EE certificate, ensure that the keyCertSign flag is on, and ensure that if the BasicConstraints extension is present, the "isCA" flag is true.
11. If the BasicConstraints extension is present, the following checks are made:
    - If the "isCA" flag is false, ensure the certificate is the last certificate in the chain and that the pathLength field is not present.
    - If the "isCA" flag is true and the certificate is NOT the last certificate in the chain, ensure that the number of certificates until the last certificate in the chain is not greater than the pathLength field.
12. The AuthorityKeyID extension is not used for path validation, but is used when building the certificate chain.
13. The SubjectKeyID extension is not used for path validation, but is used when building the certificate chain.
14. The PrivateKeyUsagePeriod extension is ignored by the validation engine, because it cannot determine when the CA actually signed the certificate. The extension is always non-critical and therefore can be safely ignored.

An OCSP Response is also validated to ensure that the response itself is valid. Validation is performed in the following manner (but not necessarily the following order):

1. Ensure that response status is `Successful` and the response type is PKIX_AD_OCSP_basic.r
2. Ensure that response version data is present and the response is the correct version (Version 1)
3. Ensure that the response is correctly signed. The signature will be rejected if the signer does not meet at least one of the following criteria:
   - The signer matches a local configuration of OCSP signing authority [23] for the certificate.
   - The signer is using the CA key for which the public key is contained in the CA certificate, that is, the CA itself is directly signing the response.
   - The signer is a direct sub-ordinate of the CA that signed the certificate for which revocation information is being checked and is authorized by the CA by including the value of id-ad-ocspSigning in an ExtendedKeyUsage extension.

22. If *OCSPAuthentication* is set to `WARN`, IBM MQ logs the unknown revocation status and allows the connection to continue.

23. This is a Certificate in the KeyStore a user has installed and that has Trust Status set.

Note: Revocation checking of the response signer certificate is not performed if the id-pkix-ocsp-nocheck extension is present.

4. Ensure that response hash algorithm, serialNumber, issuerNameHash, and issuerKeyHash match those of the request.
5. Ensure that the response has not expired, that is, that the nextUpdate time is greater than the current time. [24]
6. Ensure that the certificate has valid revocation status.

The validation of a CRL is also performed to ensure that the CRL itself is valid, and is performed in the following manner (but not necessarily the following order):

1. Ensure that the signature algorithm used to actually sign the CRL matches the signature algorithm indicated within the CRL, by ensuring that the issuer signature algorithm identifier in the CRL matches the algorithm identifier in the signature data.
2. Ensure that the CRL was signed by the issuer of certificate in question, verifying that the CRL has been signed with the key of the certificate issuer.
3. Ensure that the CRL has not expired [25] , or not been activated yet, and that its validity period is good.
4. Ensure that if the version field is present, it is version 2. Otherwise the CRL is version 1 and must not have any extensions. However, IBM MQ for UNIX, Linux and Windows systems only verifies that no critical extensions are present for a version 1 CRL.
5. Ensure that the certificate in question is on the revokedCertificates field list and that the revocation date is not in the future.
6. Ensure that there are no duplicate extensions.
7. If unknown critical extensions, including critical entry extensions, are detected in the CRL, this causes identified certificates to be treated as revoked [26] (provided the CRL passes all other checks).
8. If the authorityKeyID extension in the CRL and the subjectKeyID in the CA certificate are present and if the keyIdentifier field is present within the authorityKeyID of the CRL, match it with the CACertificate's subjectKeyID.
9. If the issuerAltName extension is marked critical, ensure that the name forms are recognized. The following general name forms are currently recognized:
   - rfc822
   - DNS
   - directory
   - URI

---

24. If no current OCSP responses are returned from the responder, IBM MQ will attempt to use out of date responses in determining the revocation status of a Certificate. IBM MQ attempts to use out of date Responses so that security will not be adversely reduced.

25. If no current CRLs are found, IBM MQ for UNIX, Linux and Windows systems will attempt to use out of date CRLs to determine the revocation status of a Certificate. It is not clearly specified in RFC 5280 what action to take in the event of no current CRLs. IBM MQ for UNIX, Linux and Windows systems attempt to use out of date CRLs so that security will not be adversely reduced.

26. ITU X.509 and RFC 5280 are in conflict in this case because the RFC mandates that CRLs with unknown critical extensions must fail validation. However, ITU X.509 requires that identified certificates must still be treated as revoked provided the CRL passes all other checks. IBM MQ for UNIX, Linux and Windows systems adopt the ITU X.509 guidance so that security will not be adversely reduced.

    A potential scenario exists where the CA that issues a CRL might set an unknown critical extension to indicate that even though all other validation checks are successful, a certificate which is identified must not be considered revoked and thus not rejected by the application. In this scenario, following X.509, IBM MQ for UNIX, Linux and Windows systems will function in a fail-secure mode of operation. That is, they might reject certificates that the CA did not intend to be rejected and therefore might deny service to some valid users. A fail-insecure mode ignores a CRL because it has an unknown critical extension and therefore certificates that the CA intended to be revoked are still accepted. The administrator of the system should then query this behavior with the issuing CA.

- IPAddress(v4/v6)

10. If the issuingDistributionPoint extension is present in the CRL, process as follows:
    - If the issuingDistributionPoint specifies an InDirectCRL then fail the CRL validation.
    - If the issuingDistributionPoint indicates that a CRLDistributionPoint is present but no DistributionPointName is found, fail the CRL validation
    - If the issuingDistributionPoint indicates that a CRLDistributionPoint is present and specifies a DistributionPointName ensure that it is a GeneralName or LDAP format URI that matches the name given by the certificate's CRLDistributionPoint or the certificate's issuer's name. If the DistributionPointName is not a GeneralName then the CRL validation will fail.

      Note: RelativeDistinguishedNames are not supported and will fail CRL validation if encountered.

Standard path validation policy

ULW

The standard path validation policy determines how the certificate, OCSP, and CRL policy types interact with each other to determine if a certificate chain is valid. Standard policy checking conforms to RFC 5280.

Path validation uses the following concepts:

- A certification path of length *n*, where the trust point or root certificate is certificate 1, and the EE is *n*.
- A set of initial policy identifiers (each comprising a sequence of policy element identifiers), that identifies one or more certificate policies, any one of which is acceptable for the purposes of certification path processing, or the special value "any-policy". Currently this is always set to "any-policy".

  Note: IBM MQ for UNIX, Linux and Windows systems only supports policy identifiers that are created by IBM MQ for UNIX, Linux and Windows systems.
- Acceptable policy set: a set of certificate policy identifiers comprising the policy or policies recognized by the public key user, together with policies deemed equivalent through policy mapping. The initial value of the acceptable policy set is the special value "any-policy".
- Constrained subtrees: a set of root names defining a set of subtrees within which all subject names in subsequent certificates in the certification path can fall. The initial value is "unbounded".
- Excluded subtrees: a set of root names defining a set of subtrees within which no subject name in subsequent certificates in the certification path can fall. The initial value is "empty".
- Explicit policy: an integer which indicates if an explicit policy identifier is required. The integer indicates the first certificate in the path where this requirement is imposed. When set, this variable can be decreased, but cannot be increased. (That is, if a certificate in the path requires explicit policy identifiers, a later certificate cannot remove this requirement.) The initial value is *n*+1.
- Policy mapping: an integer which indicates if policy mapping is permitted. The integer indicates the last certificate on which policy mapping may be applied. When set, this variable can be decreased, but cannot be increased. (That is, if a certificate in the path specifies policy mapping is not permitted, it cannot be overridden by a later certificate.) The initial value is *n*+1.

The validation of a chain is performed in the following manner (but not necessarily the following order):

1. The information in the following paragraph is consistent with the basic path validation policy described in “Basic path validation policy” on page 4112:

   Ensure that the name of the certificate's issuer is equal to the subject name in the previous certificate, and that there is not an empty issuer name in this certificate or the previous certificate subject name.

If no previous certificate exists in the path and this is the first certificate in the chain, ensure that the issuer and subject name are identical and that the trust status is set for the certificate [27] .

If the certificate does not have a subject name, the subjectAltName extension must be present and critical.

2. The information in the following paragraph is consistent with the basic path validation policy described in “Basic path validation policy” on page 4112:

   Ensure that the signature algorithm used to actually sign the certificate matches the signature algorithm indicated within the certificate, by ensuring that the issuer signature algorithm identifier in the certificate matches the algorithm identifier in the signature data.

   If both the certificate's issuersUniqueID and the issuer's subjectUniqueID are present, ensure they match.

3. The following information is consistent with the basic path validation policy described in “Basic path validation policy” on page 4112:

   Ensure that the certificate was signed by the issuer, using the subject public key from the previous certificate in the path to verify the signature on the certificate. If no previous certificate exists and this is the first certificate, use the subject public key of the certificate to verify the signature on it.

4. The following information is consistent with the basic path validation policy described in “Basic path validation policy” on page 4112:

   Ensure that the certificate is a known X509 version, unique IDs are not present for version 1 certificates and extensions are not present for version 1 and version 2 certificates.

5. The following information is consistent with the basic path validation policy described in “Basic path validation policy” on page 4112:

   Ensure that the certificate has not expired, or not been activated yet, and that its validity period is good [28]

6. The following information is consistent with the basic path validation policy described in “Basic path validation policy” on page 4112:

   Ensure that there are no unknown critical extensions, nor any duplicate extensions.

7. The following information is consistent with the basic path validation policy described in “Basic path validation policy” on page 4112:

   Ensure that the certificate has not been revoked. Here, the following operations apply:

   a. If the OCSP connection is enabled and a Responder Address is configured or the Certificate has a valid AuthorityInfoAccess extension specifying an HTTP format GENERALNAME_uniformResourceID check revocation status with OCSP.

      1) IBM MQ for UNIX and Windows systems allows the OCSP Request to be optionally signed for preconfigured responders but this has otherwise no impact on OCSP Response processing.

   b. If revocation status from 7a is undetermined the CRLDistributionPoints extension is checked for a list of X.500 distinguished name GENERALNAME_directoryname and URI GENERALNAME_uniformResourceID. If the extension is not present, the certificate's issuer's name is used. A CRL database (LDAP) is then queried for CRLs. If the certificate is not the last certificate, or if the last certificate has the basic constraint extension with the "isCA" flag turned on, the database is queried for ARL's and CRL's instead. If CRL checking is enabled, and no CRL database can be queried, the certificate is treated as revoked. Currently, the X500 directory name form and the LDAP/HTTP/FILE URI forms are the only supported name forms used to look up CRLs and ARLs15.

---

27. Trust status is an administrative setting in the key database file. You can access and alter the trust status of a particular signer certificate in iKeyman. Select the required certificate from the signer list and click **View/Edit**. The **Set the certificate as a trusted root** check box on the resulting panel indicates the trust status. You can also set Trust status using **iKeycmd** with the `-trust` flag on the **-cert -modify** command. For further information about this command, see Managing keys and certificates.

28. There are no checks to ensure the subject's validity is within bounds of the issuer's validity. This is not required, and certificates from some CAs have been shown to not pass such a check.

Note: RelativeDistinguishedNames are not supported.

8. The following information is consistent with the basic path validation policy described in "Basic path validation policy" on page 4112:

   If the subjectAltName extension is marked critical, ensure that the name forms are recognized. The following general name forms are currently recognized:

   - rfc822
   - DNS
   - directory
   - URI
   - IPAddress(v4/v6)

9. Ensure that the subject name and subjectAltName extension (critical or noncritical) is consistent with the constrained and excluded subtrees state variables.
10. If the EmailAddress OID is present in the subject name field as an IA5 string, and there is no subjectAltName extension, the EmailAddress must be consistent with the constrained and excluded subtrees state variable.
11. Ensure that policy information is consistent with the initial policy set :
    a. If the explicit policy state variable is less than or equal to the current certificate's numeric sequence value, a policy identifier in the certificate shall be in the initial policy set.
    b. If the policy mapping variable is less than or equal to the current certificate's numeric sequence value, the policy identifier cannot be mapped.
12. Ensure that policy information is consistent with the acceptable policy set:
    a. If the certificate policies extension is marked critical [29] , the intersection of the policies extension and the acceptable policy set is non-null.
    b. The acceptable policy set is assigned the resulting intersection as its new value.
13. Ensure that the intersection of the acceptable policy set and the initial policy set is non-null. If the special Policy of anyPolicy is present then allow it only if it has not been inhibited by the inhibitAnyPolicy extension at this chain position.
14. If an inhibitAnyPolicy extension is present ensure that it is marked Critical and, if so, set the inhibitAnyPolicy state and chain position to the value of the integer value of the extension provided it is not greater than the current value. This is the number of certificates to allow with an anyPolicy Policy before disallowing the anyPolicy Policy.
15. The following steps are performed for all certificates except the last one:
    a. If the issuerAltName extension is marked critical, ensure that the name forms are recognized. The following general name forms are currently recognized:
       - rfc822
       - DNS
       - directory
       - URI
       - IPAddress(v4/v6)
    b.
       1) If the BasicConstraints extension is not present, the certificate is only valid as an EE certificate.
       2) If the BasicConstraints extension is present, ensure that the "isCA" flag is true. Note that "isCA" is always checked to ensure it is true to be as part of the chain building itself, however this specific test is still made. If the pathLength field is present, ensure the number of certificates until the last certificate is not greater than the pathLength field.

29. This is maintained as a legacy requirement from RFC2459 (6.1 (e)(1))

c. If the KeyUsage extension is critical, ensure that the keyCertSign flag is on, and ensure that if the BasicConstraints extension is present, that the "isCA" flag is true [30] .

d. If a policy constraints extension is included in the certificate, modify the explicit policy and policy mapping state variables as follows:
   - i. If requireExplicitPolicy is present and has value *r*, the explicit policy state variable is set to the minimum of its current value and the sum of *r* and *i* (the current certificate in the sequence).
   - ii. If inhibitPolicyMapping is present and has value *q*, the policy mapping state variable is set to the minimum of its current value and the sum of *q* and *i* (the current certificate in the sequence).

e. If the policyMappings extension is present (see 12(b)), ensure that it is not critical, and if policy mapping is allowed, these mappings are used to map between this certificate's policies and its signee's policies.

f. If the nameConstraints extension is present, ensure that it is critical, and that the permitted and excluded subtrees adhere to the following rules before updating the chain's subtree's state in accordance with the algorithm described in RFC 5280 section 6.1.4 part (g):
   1) The minimum field is set to zero.
   2) The maximum field is not present.
   3) The base field name forms are recognized. The following general name forms are currently recognized:
      - rfc822
      - DNS
      - directory
      - URI
      - IPAddress(v4/v6)

16. The ExtendedKeyUsage extension is not checked by IBM MQ.

17. The following information is consistent with the basic path validation policy described in “Basic path validation policy” on page 4112:

    The AuthorityKeyID extension is not used for path validation, but is used when building the certificate chain.

18. The following information is consistent with the basic path validation policy described in “Basic path validation policy” on page 4112:

    The SubjectKeyID extension is not used for path validation, but is used when building the certificate chain.

19. The following information is consistent with the basic path validation policy described in “Basic path validation policy” on page 4112:

    The PrivateKeyUsagePeriod extension is ignored by the validation engine, because it cannot determine when the CA actually signed the certificate. The extension is always non-critical and therefore can be safely ignored.

30. This check is in fact redundant because of step (b), but the check is still made.

Cryptographic hardware

The way in which IBM MQ provides support for cryptographic hardware depends on which platform you are using.

ULW On UNIX, Linux, and Windows systems, IBM MQ provides support for a variety of cryptographic hardware using the PKCS #11 interface.

IBM i z/OS OnIBM i and z/OS, the operating system provides the cryptographic hardware support.

For a list of currently supported cryptography cards, see Cryptography Card List for IBM MQ.

On all platforms, cryptographic hardware is used at the TLS handshaking stage and at secret key reset.

IBM i On IBM i, when you use DCM to create or renew certificates, you can choose to store the key directly in the coprocessor or to use the coprocessor master key to encrypt the private key and store it in a special keystore file.

z/OS On z/OS, when you use RACF to create certificates, you can choose to store the key using ICSF (Integrated Cryptographic Service Facility) to obtain improved performance and more secure key storage. During the TLS handshake, and secret key negotiations, a crypto express card, (if available) is used to do RSA operations. After the handshake completes and data begins to flow, data is decrypted in the CPACF and the crypto express card is not used.

ULW On UNIX, Linux, and Windows systems, IBM MQ support is also provided for TLS cryptographic hardware symmetric cipher operations. When using TLS cryptographic hardware symmetric cipher operations, data sent across a TLS connection is encrypted/decrypted by the cryptographic hardware product.

On the queue manager, this is enabled by setting the SSLCryptoHardware queue manager attribute appropriately (see ALTER QMGR and Change Queue Manager). On the WebSphere MQ MQI client, equivalent variables are provided (see SSL stanza of the client configuration file). The default setting is off.

If this attribute is enabled, IBM MQ attempts to use symmetric cipher operations whether the cryptographic hardware product supports them for the encryption algorithm specified in the current CipherSpec or not. If the cryptographic hardware product does not provide this support, IBM MQ performs the encryption and decryption of data itself, and no error is reported. If the cryptographic hardware product supports symmetric cipher operations for the encryption algorithm specified in the current CipherSpec, this function is activated and the cryptographic hardware product performs the encryption and decryption of the data sent.

In a situation of low processor usage it is often quicker to perform the encryption/decryption in software, rather than copying the data onto the card, encrypting/decrypting it, and copying it back to the TLS protocol software. Hardware symmetric cipher operations become more useful when the processor usage is high.

z/OS On z/OS with cryptographic hardware, support is provided for symmetric cipher operations. This means that the user's data is encrypted and decrypted by the hardware if the hardware has this capability for the CipherSpec chosen, and is configured to support data encryption and decryption.

IBM i On IBM i, cryptographic hardware is not used for encryption and decryption of the user's data, even if the hardware has the capability of performing such encryption for the encryption algorithm

specified in the current CipherSpec.

IBM MQ rules for SSLPEER values

The SSLPEER attribute is used to check the Distinguished Name (DN) of the certificate from the peer queue manager or client at the other end of an IBM MQ channel. IBM MQ uses certain rules when comparing these values

Attention: Only the attributes in the following list can be used in an SSLPEER filter.

When SSLPEER values are compared with DNs, the rules for specifying and matching attribute values are as follows:

1. You can use either a comma or a semicolon as a separator.
2. Spaces before or after the separator are ignored. For example:
   `CN=John Smith, O=IBM ,OU=Test , C=GB`
3. The values of attribute types `SERIALNUMBER`, `MAIL`, `E`, `UID OR USERID`, `CN`, `T`, `OU`, `DC`, `O`, `STREET`, `L`, `ST`, `SP`, `S`, `PC`, `C`, `UNSTRUCTUREDNAME`, `UNSTRUCTUREDADDRESS`, `DNQ` are text strings that typically include only the following:
   - Uppercase and lowercase alphabetic characters `A` through `Z` and `a` through `z`
   - Numeric characters `0` through `9`
   - The space character
   - Characters `, . ; ' " ( ) / -`

   To avoid conversion problems between different platforms, do not use other characters in an attribute value. The attribute types, for example `CN`, must be in uppercase characters.
4. Strings containing the same alphabetic characters match irrespective of case.
5. Spaces are not allowed between the attribute type and the `=` character.
6. Optionally, you can enclose attribute values in double quotation marks, for example `CN="John Smith"`. The quotation marks are discarded when matching values.
7. Spaces at either end of the string are ignored unless the string is enclosed in double quotation marks.
8. The comma and semicolon attribute separator characters are considered to be part of the string when enclosed in double quotation marks.
9. The names of attribute types, for example `CN` or `OU`, are considered to be part of the string when enclosed in double quotation marks.
10. Any of the attribute types `ST`, `SP`, and `S` can be used for the State or Province name.
11. Any attribute value can have an asterisk (`*`) as a pattern-matching character at the beginning, the end, or in both places. The asterisk character substitutes for any number of characters at the beginning or end of the string to be matched. This character enables your SSLPEER value specification to match a range of Distinguished Names. For example, `OU=IBM*` matches every Organizational Unit beginning with IBM, such as IBM Corporation.

    The asterisk character can also be a valid character in a Distinguished Name. To obtain an exact match with an asterisk at the beginning or end of the string, the backslash escape character (`\`) must precede the asterisk: `\*`. Asterisks in the middle of the string are considered to be part of the string and do not require the backslash escape character.
12. The DN can contain multiple OU attributes and multiple DC attributes.
13. When multiple OU attributes are specified, all must exist and be in descending hierarchical order. For an example, see DEFINE CHANNEL.
14. A digital certificate Subject DN can additionally contain multiple attributes of the same type other than OU or DC, but only if the SSLPEER value does not filter on the repeated attribute type. For example, consider a certificate with the following Subject DN:
    `CN=First, CN=Second, O=IBM, C=US`

An SSLPEER value of `O=IBM, C=US` does not filter on CN, so matches this certificate and allows the connection. An SSLPEER value of `CN=First, O=IBM, C=US` fails to match this certificate because the certificate contains multiple CN attributes. You cannot match multiple CN values.

Related information:

Distinguished Names

Channel authentication records

Mapping a TLS Distinguished Name to an MCAUSER user ID

GSKit: Digital certificate signature algorithms compliant with FIPS 140-2

The list of digital certificate signature algorithms in GSKit that are compliant with FIPS 140-2

- RSA with SHA-1
- RSA with SHA-224
- RSA with SHA-256
- RSA with SHA-384
- RSA with SHA-512
- DSA with SHA-1
- ECDSA with SHA-1
- ECDSA with SHA-224
- ECDSA with SHA-256
- ECDSA with SHA-384
- ECDSA with SHA-512
- Curve P-192
- Curve P-224
- Curve P-256
- Curve P-384
- Curve P-521
- Curve K-163
- Curve K-233
- Curve K-283
- Curve K-409
- Curve K-571
- Curve B-163
- Curve B-233
- Curve B-283
- Curve B-409
- Curve B-571

Related information:
Digital certificates and CipherSpec compatibility in IBM MQ

Migrating with AltGSKit from Version 7.0.1 to Version 7.1

Multi

Perform this task only if you are migrating from IBM WebSphere MQ Version 7.0.1 using the AltGSKit configuration setting to load an alternative GSKit. The alternative GSKit used by IBM WebSphere MQ Version 7.0.1 with the AltGSKit setting is separate from the GSKit used by IBM WebSphere MQ Version 7.1; changes to each GSKit do not affect the other. This is because Version 7.1 uses a private local copy of GSKit in its installation directory and does not support the use of an alternative GSKit.

Overview of the main migration steps for AltGSKit

When migrating from IBM WebSphere MQ Version 7.0.1 using AltGSKit to IBM WebSphere MQ Version 7.1, there are a number of tasks to be performed to enable the new GSKit to operate successfully. The main steps to consider when migrating:

1. Ensure that no applications require the use of the currently installed alternative GSKit before initiating removal.
2. Remove the AltGSKit setting from the SSL stanza of each queue manager and client configuration file.
3. Restart each MQI client application which is using the alternative GSKit to ensure that no client applications have the alternative GSKit loaded.
4. Issue the REFRESH SECURITY TYPE(SSL) on each queue manager which is using the alternative GSKit to ensure that no queue managers have the alternative GSKit loaded.
5. Uninstall the alternative GSKit as per the platform specific instructions outlined in this topic.
6. Install the alternative GSKit as per the platform specific instructions referred to in this topic.

Removing the AltGSKit setting

Before the alternative GSKit can be uninstalled, the AltGSKit setting must be removed from the SSL stanza of each queue manager and client configuration file.

To view the contents and for further information about the queue manager configuration files, see Queue manager configuration files, qm.ini

For information about the the SSL stanza of the client configuration file, see SSL stanza of the client configuration file.

Once the configuration file has been altered:

1. Restart each MQI client application which is using the alternative GSKit to ensure that no client applications have the alternative GSKit loaded.
2. Issue the REFRESH SECURITY TYPE(SSL) on each queue manager which is using the alternative GSKit to ensure that no queue managers have the alternative GSKit loaded.

Uninstalling GSKit

For platform specific instructions for uninstalling the alternative GSKit, see the following sections:

- "Uninstalling GSKit Version 8.0 on Windows" on page 4123
- "Uninstalling GSKit Version 8.0 on Linux" on page 4123
- "Uninstalling GSKit Version 8.0 on AIX" on page 4123
- "Uninstalling GSKit Version 8.0 on HP-UX" on page 4123
- "Uninstalling GSKit Version 8.0 on Solaris" on page 4123

Uninstalling GSKit Version 8.0 on Windows

You can uninstall GSKit Version 8.0 interactively using Add or Remove Programs in the Windows Control Panel. You can uninstall GSKit Version 8.0 silently using the Windows Installer **msiexec** utility or the GSKit installation file. If you want to use an accessible interface to uninstall GSKit Version 8.0, use either of the silent uninstallation methods.

Procedure

- To uninstall GSKit Version 8.0 by using **msiexec**:
  1. Issue the command
     ```
     msiexec /x PackageName
     ```
     PackageName is one of the values `GSKit8 SSL 32-bit`, `GSKit8 Crypt 32-bit`, `GSKit8 SSL 64-bit`, or `GSKit8 Crypt 64-bit`.
  2. Repeat for each package to be uninstalled.

Uninstalling GSKit Version 8.0 on Linux

You can uninstall GSKit Version 8.0 using the **rpm** command.

Procedure

Uninstall GSKit Version 8.0 by using the following command:

```
rpm -ev gskssl32-8.0.X.Y gskcrypt32-8.0.X.Y
```

X.Y represents the version number of GSKit installed.

On 64-bit Linux platforms run the following additional command:

```
rpm -ev gskssl64-8.0.X.Y gskcrypt64-8.0.X.Y
```

Uninstalling GSKit Version 8.0 on AIX

You can uninstall GSKit Version 8.0 using the **installp** command.

Procedure

Uninstall GSKit Version 8.0 by using the following command:

```
installp -u -g -V2 gskcrypt32.ppc.rte gskssl32.ppc.rte gskcrypt64.ppc.rte gskssl64.ppc.rte
```

Uninstalling GSKit Version 8.0 on HP-UX

You can uninstall GSKit Version 8.0 using the **swremove** command.

Procedure

Uninstall GSKit Version 8.0 by using the following command:

```
swremove gskcrypt32 gskssl32 gskcrypt64 gskssl64
```

Uninstalling GSKit Version 8.0 on Solaris

You can uninstall GSKit Version 8.0 using the **pkgrm** command.

Procedure

Uninstall GSKit Version 8.0 by using the following command:

```
pkgrm gsk8ssl32 gsk8cry32 gsk8ssl64 gsk8cry64
```

Installing GSKit on IBM WebSphere MQ Version 7.1

On IBM WebSphere MQ Version 7.1 for Windows, GSKit is automatically installed.

To install GSKit on IBM WebSphere MQ Version 7.1 on Linux and UNIX, refer to instructions outlined in the following topics:

- IBM MQ components for Linux systems
- IBM MQ components for HP-UX systems
- IBM MQ components for AIX systems
- IBM MQ components for Solaris systems

CipherSpec mismatches

Both ends of an IBM MQ TLS channel must use the same CipherSpec. Mismatches can be detected during the TLS handshake or during channel startup.

A CipherSpec identifies the combination of the encryption algorithm and hash function. Both ends of an IBM MQ TLS channel must use the same CipherSpec, although they can specify that CipherSpec in a different manner. Mismatches can be detected at two stages:

During the TLS handshake
: The TLS handshake fails when the CipherSpec specified by the TLS client is unacceptable to the TLS support at the TLS server end of the connection. A CipherSpec failure during the TLS handshake arises when the TLS client proposes a CipherSpec that is not supported by the TLS provision on the TLS server. For example, when a TLS client running on AIX proposes the `DES_SHA_EXPORT1024` CipherSpec to a TLS server running on IBM i.

During channel startup
: Channel startup fails when there is a mismatch between the CipherSpec defined for the responding end of the channel and the CipherSpec defined for the calling end of channel. Channel startup also fails when only one end of the channel defines a CipherSpec.

  See Specifying CipherSpecs for more information.

  Note: If Global Server Certificates are used, a mismatch can be detected during channel startup even if the CipherSpecs specified on both channel definitions match.

  Global Server Certificates are a special type of certificate which require that a minimum level of encryption is established on all the communications links with which they are used. If the CipherSpec requested by the IBM MQ channel configuration does not meet this requirement, the CipherSpec is renegotiated during the TLS handshake. This is detected as a failure during IBM MQ channel startup as the CipherSpec no longer matches the one specified on the channel.

  In this case, change the CipherSpec at both sides of the channel to one which meets the requirements of the Global Server Certificate. To establish whether a certificate that has been issued to you is a Global Server Certificate, contact the certificate authority which issued that certificate.

TLS servers do not detect mismatches when an TLS client channel on UNIX, Linux or Windows systems specifies the `DES_SHA_EXPORT1024` CipherSpec, and the corresponding TLS server channel on UNIX, Linux or Windows systems is using the `DES_SHA_EXPORT` CipherSpec. In this case, the channel runs normally.

Authentication failures

There are a number common reasons for authentication failures during the TLS handshake.

These reasons include, but are not limited to, those in the following list:

A certificate has been found in a Certificate Revocation List or Authority Revocation List
You can check certificates against the revocation lists published by the Certificate Authorities.

A Certificate Authority can revoke a certificate that is no longer trusted by publishing it in a Certificate Revocation List (CRL) or Authority Revocation List (ARL). For more information, see Working with revoked certificates.

An OCSP responder has identified a certificate as Revoked or Unknown
You can check certificates using OCSP. An OCSP responder can return a response of Revoked, indicating that a certificate is no longer valid, or Unknown, indicating that it has no revocation data for that certificate. For more information, see Working with revoked certificates.

A certificate has expired or is not yet active
Each digital certificate has a date from which it is valid and a date after which it is no longer valid, so an attempt to authenticate with a certificate that is outside its lifetime fails.

A certificate is corrupted
If the information in a digital certificate is incomplete or damaged, authentication fails.

A certificate is not supported
If the certificate is in a format that is not supported, authentication fails, even if the certificate is still within its lifetime.

The TLS client does not have a certificate
The TLS server always validates the client certificate if one is sent. If the TLS client does not send a certificate, authentication fails if the end of the channel acting as the TLS server is defined:

- With the SSLCAUTH parameter set to REQUIRED or
- With an SSLPEER parameter value

There is no matching CA root certificate or the certificate chain is incomplete
Each digital certificate is issued by a Certificate Authority (CA), which also provides a root certificate that contains the public key for the CA. Root certificates are signed by the issuing CA itself. If the key repository on the computer that is performing the authentication does not contain a valid root certificate for the CA that issued the incoming user certificate, authentication fails.

Authentication often involves a chain of trusted certificates. The digital signature on a user certificate is verified with the public key from the certificate for the issuing CA. If that CA certificate is a root certificate, the verification process is complete. If that CA certificate was issued by an intermediate CA, the digital signature on the intermediate CA certificate must itself be verified. This process continues along a chain of CA certificates until a root certificate is reached. In such cases, all certificates in the chain must be verified correctly. If the key repository on the computer that is performing the authentication does not contain a valid root certificate for the CA that issued the incoming root certificate, authentication fails.

However, certain TLS implementations such as GSKit, DCM, and RACF validate the certificates as long as the trust anchor (ROOT CA) is present, with some of the intermediate CA not present in the trust chain. Therefore, it is important to ensure that the server-side certificate store contains the complete trust chain. Also, the technique of selectively removing signer (CA) certificates must not be used to control connectivity to the queue manager.

For more information, see How certificate chains work.

For more information about the terms used in this topic, see:

- Transport Layer Security (TLS) concepts

- Digital certificates

Monitoring reference

Use the reference information in this section to help you monitor IBM MQ.

- "Structure data types"
- "Object attributes for event data" on page 4151
- "Event message reference" on page 4199

Related information:

Monitoring and performance

Structure data types

Use this topic to understand the structure data types used in the message data that IBM MQ monitoring techniques generate.

The subtopics describe in a language-independent form the structure data types used in monitor message data.

- "MQCFBS - Byte string parameter" on page 4127
- "MQCFGR - Group parameter" on page 4129
- "MQCFH - PCF header" on page 4131
- "MQCFIL - Integer list parameter" on page 4135
- "MQCFIL64 - 64-bit integer list parameter" on page 4137
- "MQCFIN - Integer parameter" on page 4139
- "MQCFIN64 - 64-bit integer parameter" on page 4141
- "MQCFSL - String list parameter" on page 4142
- "MQCFST - String parameter" on page 4145
- "MQEPH - Embedded PCF header" on page 4147

The declarations are shown in the following programming languages:

- C
- COBOL
- PL/I
- IBM i RPG (ILE) (IBM i only)
- z/OS S/390 assembler (z/OS only)
- Windows Visual Basic (Windows only)

MQCFBS - Byte string parameter

Use this page to view the structure of an MQCFBS parameter and the declarations for the following programming languages: C, COBOL, PL/I, RPG/ILE, and S/390 assembler

The MQCFBS structure describes a byte string parameter. Following the links to the declarations is a description of the fields making up the MQCFBS structure:

- C language
- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- S/390 assembler-language (z/OS only)

Type

Description: This indicates that the structure is an MQCFBS structure describing a byte string parameter.
Data type: MQLONG.
Value:

MQCFT_BYTE_STRING
Structure defining a byte string.

StrucLength

Description: This is the length in bytes of the MQCFBS structure, including the variable-length string at the end of the structure (the *String* field).
Data type: MQLONG.

Parameter

Description: This identifies the parameter with a value that is contained in the structure.
Data type: MQLONG.

StringLength

Description: This is the length in bytes of the data in the *String* field, and is zero or greater.
Data type: MQLONG.

String

Description: This is the value of the parameter identified by the *Parameter* field. The string is a byte string, and so is not subject to character-set conversion when sent between different systems.
Note: A null byte in the string is treated as normal data, and does not act as a delimiter for the string.
Data type: MQBYTE x *StringLength*.

C language declaration

```
struct tagMQCFBS {
  MQLONG  Type;          /* Structure type */
  MQLONG  StrucLength;   /* Structure length */
  MQLONG  Parameter;     /* Parameter identifier */
  MQLONG  StringLength;  /* Length of string */
  MQBYTE  String[1];     /* String value -- first character */
 } MQCFBS;
```

COBOL language declaration

```
**   MQCFBS structure
  10 MQCFBS.
**    Structure type
   15 MQCFBS-TYPE         PIC S9(9) BINARY.
**    Structure length
   15 MQCFBS-STRUCLENGTH  PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFBS-PARAMETER    PIC S9(9) BINARY.
**    Length of string
   15 MQCFBS-STRINGLENGTH PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
 1 MQCFBS based,
  3 Type         fixed bin(31), /* Structure type */
  3 StrucLength  fixed bin(31), /* Structure length */
  3 Parameter    fixed bin(31), /* Parameter identifier */
  3 StringLength fixed bin(31); /* Length of string */
```

RPG/ILE language declaration (IBM i only)

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     D* MQCFBS Structure
     D*
     D* Structure type
     D  BSTYP                  1      4I 0 INZ(9)
     D* Structure length
     D  BSLEN                  5      8I 0 INZ(16)
     D* Parameter identifier
     D  BSPRM                  9     12I 0 INZ(0)
     D* Length of string
     D  BSSTL                 13     16I 0 INZ(0)
     D* String value -- first byte
     D  BSSRA                 17     17    INZ
```

S/390 assembler-language declaration (z/OS only)

```
MQCFBS               DSECT
MQCFBS_TYPE          DS   F  Structure type
MQCFBS_STRUCLENGTH   DS   F  Structure length
MQCFBS_PARAMETER     DS   F  Parameter identifier
MQCFBS_STRINGLENGTH  DS   F  Length of string
*
MQCFBS_LENGTH        EQU  *-MQCFBS
                     ORG  MQCFBS
MQCFBS_AREA          DS   CL(MQCFBS_LENGTH)
```

MQCFGR - Group parameter

Use this page to view the structure of an MQCFGR parameter and the declarations for the following programming languages: C, COBOL, PL/I, RPG/ILE, S/390 assembler, and Visual Basic

The MQCFGR structure describes a group parameter. Following the links to the declarations is a description of the fields making up the MQCFGR structure:

- C language
- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- System/390 assembler-language (z/OS only)
- Visual Basic language (Windows only)

The MQCFGR structure is a group parameter in which the subsequent parameter structures are grouped together as a single logical unit. The number of subsequent structures that are included is given by *ParameterCount*. This structure, and the parameter structures it includes, are counted as one structure only in the *ParameterCount* parameter in the PCF header (MQCFH) and the group parameter (MQCFGR).

Type

Description: Indicates that the structure type is MQCFGR describing which parameters are in this group.
Data type: MQLONG.
Value:

MQCFT_GROUP
Structure defining a group of parameters.

StrucLength

Description: Length in bytes of the MQCFGR structure.
Data type: MQLONG.
Value:

MQCFGR_STRUC_LENGTH
Length of the command format group-parameter structure.

Parameter

Description: This identifies the type of group parameter.
Data type: MQLONG.

ParameterCount

Description: The number of parameter structures following the MQCFGR structure that are contained within the group identified by the *Parameter* field. If the group itself contains one or more groups, each group and its parameters count as one structure only.
Data type: MQLONG.

C language declaration

```
typedef struct tagMQCFGR {
  MQLONG  Type;            /* Structure type */
  MQLONG  StrucLength;     /* Structure length */
  MQLONG  Parameter;       /* Parameter identifier */
  MQLONG  ParameterCount;  /* Count of the grouped parameter structures */
 } MQCFGR;
```

COBOL language declaration

```
**   MQCFGR structure
  10 MQCFGR.
**    Structure type
   15 MQCFGR-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFGR-STRUCLENGTH    PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFGR-PARAMETER      PIC S9(9) BINARY.
**    Count of grouped parameter structures
   15 MQCFGR-PARAMETERCOUNT PIC S9(9) BINARY.
```

PL/I language declaration (z/OS and Windows only)

```
dcl
 1 MQCFGR based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Parameter      fixed bin(31), /* Parameter identifier */
  3 ParameterCount fixed bin(31), /* Count of grouped parameter structures */
```

RPG/ILE declaration (IBM i only)

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
  D* MQCFGR Structure
     D*
     D* Structure type
     D  GRTYP                  1      4I INZ(20)
     D* Structure length
     D  GRLEN                  5      8I INZ(16)
     D* Parameter identifier
     D  GRPRM                  9     12I INZ(0)
     D* Count of grouped parameter structures
     D  GRCNT                 13     16I INZ(0)
     D*
```

S/390 assembler-language declaration (z/OS only)

```
MQCFGR                         DSECT
MQCFGR_TYPE                    DS   F        Structure type
MQCFGR_STRUCLENGTH             DS   F        Structure length
MQCFGR_PARAMETER               DS   F        Parameter identifier
MQCFGR_PARAMETERCOUNT          DS   F        Count of grouped parameter structures
MQCFGR_LENGTH                  EQU  *-MQCFGR Length of structure
                               ORG  MQCFGR
MQCFGR_AREA                    DS   CL(MQCFGR_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFGR
  Type As Long            ' Structure type
  StrucLength As Long     ' Structure length
  Parameter As Long       ' Parameter identifier
  ParameterCount As Long  ' Count of grouped parameter structures
  End Type
```

MQCFH - PCF header

Use this page to view the structure of an MQCFH header and the declarations for the following programming languages: C, COBOL, PL/I, RPG/ILE, S/390 assembler, and Visual Basic

The MQCFH structure describes the information that is present at the start of the message data of a monitoring message. Following the links to the declarations is a description of the fields making up the MQCFH structure:

- C language
- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- S/390 assembler language (z/OS only)
- Visual Basic language (Windows only)

Type

Description: Structure type This indicates the content of the message.
Data type: MQLONG.
Values:

MQCFT_ACCOUNTING
Message is an accounting message.

MQCFT_EVENT
Message is reporting an event.

MQCFT_REPORT
Message is an activity report.

MQCFT_RESPONSE
Message is a response to a command.

MQCFT_STATISTICS
Message is a statistics message.

MQCFT_TRACE_ROUTE
Message is a trace-route message.

StrucLength

Description: This is the length in bytes of the MQCFH structure
Data type: MQLONG.
Value:

MQCFH_STRUC_LENGTH
Length of command format header structure.

Version

Description: Structure version number.
Data type: MQLONG.
Value:

MQCFH_VERSION_1
Version number for all events except configuration and command events.

MQCFH_VERSION_2
Version number for configuration events.

MQCFH_VERSION_3
Version number for command events, activity reports, trace-route messages, accounting and statistics messages.

Command

Description: Specifies the category of the message.
Data type: MQLONG.
Value: Refer to the *Command* values in the following structure descriptions:

- "Event message MQCFH (PCF header)" on page 4205.
- Activity report MQCFH (PCF header).
- Trace-route message MQCFH (PCF header).
- Message data in accounting and statistics messages.

MsgSeqNumber

Description: Message sequence number. This is the sequence number of the message within a set of related messages.
Data type: MQLONG.

Control

Description: Control options.
Data type: MQLONG.
Value:

MQCFC_LAST
Last message in the set.

MQCFC_NOT_LAST
Not the last message in the set.

CompCode

Description: Completion code.
Data type: MQLONG.
Value:

MQCC_OK
Events reporting OK condition, activity reports, trace-route messages, accounting messages, or statistics messages.

MQCC_WARNING
Event reporting warning condition.

Reason

Description: Reason code qualifying completion code.
Data type: MQLONG.
Value: For event messages:

MQRC_*
Dependent on the event being reported.
Note: Events with the same reason code are further identified by the **ReasonQualifier** parameter in the event data.

For activity reports, trace-route messages, accounting messages, and statistics messages:

MQRC_NONE

ParameterCount

Description:	Count of parameter structures. This is the number of parameter structures that follow the MQCFH structure.
Data type:	MQLONG.
Value:	0 or greater.

C language declaration

```
typedef struct tagMQCFH {
  MQLONG  Type;            /* Structure type */
  MQLONG  StrucLength;     /* Structure length */
  MQLONG  Version;         /* Structure version number */
  MQLONG  Command;         /* Command identifier */
  MQLONG  MsgSeqNumber;    /* Message sequence number */
  MQLONG  Control;         /* Control options */
  MQLONG  CompCode;        /* Completion code */
  MQLONG  Reason;          /* Reason code qualifying completion code */
  MQLONG  ParameterCount;  /* Count of parameter structures */
 } MQCFH;
```

COBOL language declaration

```
**   MQCFH structure
  10 MQCFH.
**    Structure type
   15 MQCFH-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFH-STRUCLENGTH    PIC S9(9) BINARY.
**    Structure version number
   15 MQCFH-VERSION        PIC S9(9) BINARY.
**    Command identifier
   15 MQCFH-COMMAND        PIC S9(9) BINARY.
**    Message sequence number
   15 MQCFH-MSGSEQNUMBER   PIC S9(9) BINARY.
**    Control options
   15 MQCFH-CONTROL        PIC S9(9) BINARY.
**    Completion code
   15 MQCFH-COMPCODE       PIC S9(9) BINARY.
**    Reason code qualifying completion code
   15 MQCFH-REASON         PIC S9(9) BINARY.
**    Count of parameter structures
   15 MQCFH-PARAMETERCOUNT PIC S9(9) BINARY.
```

PL/I language declaration (z/OS and Windows)

```
dcl
 1 MQCFH based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Version        fixed bin(31), /* Structure version number */
  3 Command        fixed bin(31), /* Command identifier */
  3 MsgSeqNumber   fixed bin(31), /* Message sequence number */
  3 Control        fixed bin(31), /* Control options */
  3 CompCode       fixed bin(31), /* Completion code */
  3 Reason         fixed bin(31), /* Reason code qualifying completion
                                     code */
  3 ParameterCount fixed bin(31); /* Count of parameter structures */
```

RPG language declaration (IBM i only)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQCFH Structure
D*
D* Structure type
D  FHTYP                 1      4I 0 INZ(1)
D* Structure length
D  FHLEN                 5      8I 0 INZ(36)
```

```
D* Structure version number
D  FHVER                  9     12I 0 INZ(1)
D* Command identifier
D  FHCMD                 13     16I 0 INZ(0)
D* Message sequence number
D  FHSEQ                 17     20I 0 INZ(1)
D* Control options
D  FHCTL                 21     24I 0 INZ(1)
D* Completion code
D  FHCMP                 25     28I 0 INZ(0)
D* Reason code qualifying completion code
D  FHREA                 29     32I 0 INZ(0)
D* Count of parameter structures
D  FHCNT                 33     36I 0 INZ(0)
D*
```

S/390 assembler language declaration (z/OS only)

```
MQCFH                           DSECT
MQCFH_TYPE                      DS   F        Structure type
MQCFH_STRUCLENGTH               DS   F        Structure length
MQCFH_VERSION                   DS   F        Structure version number
MQCFH_COMMAND                   DS   F        Command identifier
MQCFH_MSGSEQNUMBER              DS   F        Message sequence number
MQCFH_CONTROL                   DS   F        Control options
MQCFH_COMPCODE                  DS   F        Completion code
MQCFH_REASON                    DS   F        Reason code qualifying
*                                             completion code
MQCFH_PARAMETERCOUNT            DS   F        Count of parameter
*                                             structures
MQCFH_LENGTH                    EQU  *-MQCFH  Length of structure
                                ORG  MQCFH
MQCFH_AREA                      DS   CL(MQCFH_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFH
  Type As Long            'Structure type
  StrucLength As Long     'Structure length
  Version As Long         'Structure version number
  Command As Long         'Command identifier
  MsgSeqNumber As Long    'Message sequence number
  Control As Long         'Control options
  CompCode As Long        'Completion code
  Reason As Long          'Reason code qualifying completion code
  ParameterCount As Long  'Count of parameter structures
End Type
```

MQCFIL - Integer list parameter

Use this page to view the structure of an MQCFIL parameter and the declarations for the following programming languages: C, COBOL, PL/I, RPG/ILE, S/390 assembler, and Visual Basic

The MQCFIL structure describes an integer list parameter. Following the links to the declarations is a description of the fields making up the MQCFIL structure:

- C language
- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- System/390 assembler-language (z/OS only)
- Visual Basic language (Windows only)

Type

Description: Indicates that the structure type is MQCFIL and describes an integer-list parameter.
Data type : MQLONG.
Value:

MQCFT_INTEGER_LIST
Structure defining an integer list.

StrucLength

Description: Length in bytes of the MQCFIL structure, including the array of integers at the end of the structure (the *values* field).
Data type : MQLONG.

Parameter

Description: Identifies the parameter with a value that is contained in the structure.
Data type : MQLONG.

Count

Description: Number of elements in the *Values* array.
Data type : MQLONG.
Values: Zero or greater.

Values

Description: Array of values for the parameter identified by the *Parameter* field.
Data type : MQLONG x *Count*

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure must be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL, PL/I, RPG, and System/390 assembler programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, you must include MQCFIL in a larger structure, and declare additional fields following MQCFIL, to represent the Values field as required.

C language declaration

```
typedef struct tagMQCFIL {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Count;        /* Count of parameter values */
  MQLONG  Values[1];    /* Parameter values - first element */
 } MQCFIL;
```

COBOL language declaration

```
**   MQCFIL structure
  10 MQCFIL.
**    Structure type
   15 MQCFIL-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIL-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIL-PARAMETER   PIC S9(9) BINARY.
**    Count of parameter values
   15 MQCFIL-COUNT       PIC S9(9) BINARY.
```

PL/I language declaration

```
dcl
 1 MQCFIL based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Count       fixed bin(31); /* Count of parameter values */
```

RPG/ILE declaration (IBM i only)

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     D* MQCFIL Structure
     D*
     D* Structure type
     D  ILTYP                  1      4I 0
     D* Structure length
     D  ILLEN                  5      8I 0
     D* Parameter identifier
     D  ILPRM                  9     12I 0
     D* Count of paramter valuee
     D  ILCNT                 13     16I 0
```

S/390 assembler-language declaration

```
MQCFIL                        DSECT
MQCFIL_TYPE                   DS   F        Structure type
MQCFIL_STRUCLENGTH            DS   F        Structure length
MQCFIL_PARAMETER              DS   F        Parameter identifier
MQCFIL_COUNT                  DS   F        Count of parameter values
MQCFIL_LENGTH                 EQU  *-MQCFIL Length of structure
                              ORG  MQCFIL
MQCFIL_AREA                   DS   CL(MQCFIL_LENGTH)
```

Visual Basic language declaration

```
Type MQCFIL
  Type As Long         ' Structure type
  StrucLength As Long  ' Structure length
  Parameter As Long    ' Parameter identifier
  Count As Long        ' Count of parameter value
End Type
```

MQCFIL64 - 64-bit integer list parameter

Use this page to view the structure of an MQCFIL64 parameter and the declarations for the following programming languages: C, COBOL, PL/I, RPG/ILE, and S/390 assembler

The MQCFIL64 structure describes a 64-bit integer list parameter. Following the links to the declarations is a description of the fields making up the MQCFIL64 structure:

- C language
- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- System/390 assembler-language (z/OS only)

Type

Description: Indicates that the structure is a MQCFIL64 structure describing a 64-bit integer list parameter.
Data type: MQLONG.
Value:

MQCFT_INTEGER64_LIST
Structure defining a 64-bit integer list.

StrucLength

Description: Length in bytes of the MQCFIL64 structure, including the array of integers at the end of the structure (the *Values* field).
Data type: MQLONG.

Parameter

Description: Identifies the parameter with a value that is contained in the structure.
Data type: MQLONG.

Count

Description: Number of elements in the *Values* array.
Data type: MQLONG.
Values: 0 or greater.

Values

Description: Array of values for the parameter identified by the *Parameter* field.
Data type: (MQINT64 x *Count*)

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure must be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL, PL/I, RPG, and System/390 assembler programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, you must include MQCFIL64 in a larger structure, and declare additional fields following MQCFIL64, to represent the *Values* field as required.

For COBOL, additional fields should be declared as:

```
PIC  S9(18)
```

For PL/I, additional fields should be declared as FIXED BINARY SIGNED with a precision of 63.

For System/390 assembler, additional fields should be declared D (double word) in the DS declaration.

C language declaration

```
typedef struct tagMQCFIN64 {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Count;        /* Count of parameter values */
  MQINT64 Values[1];        /* Parameter value */
 } MQCFIL64;
```

COBOL language declaration

```
**   MQCFIL64 structure
  10 MQCFIL64.
**    Structure type
   15 MQCFIL64-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIL64-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIL64-PARAMETER   PIC S9(9) BINARY.
**    Count of parameter values
   15 MQCFIL64-COUNT       PIC S9(9) BINARY.
```

PL/I language declaration

```
dcl
 1 MQCFIL64 based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Count       fixed bin(31)  /* Count of parameter values */
```

RPG/ILE language declaration (IBM i only)

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     D* MQCFIL64 Structure
     D*
     D* Structure type
     D  IL64TYP                1      4I 0 INZ(25)
     D* Structure length
     D  IL64LEN                5      8I 0 INZ(16)
     D* Parameter identifier
     D  IL64PRM                9     12I 0 INZ(0)
     D* Count of parameter values
     D  IL64CNT               13     16I 0 INZ(0)
     D* Parameter values -- first element
     D  IL64VAL               17     16    INZ(0)
```

S/390 assembler-language declaration (z/OS only)

```
MQCFIL64                        DSECT
MQCFIL64_TYPE                   DS   F        Structure type
MQCFIL64_STRUCLENGTH            DS   F        Structure length
MQCFIL64_PARAMETER              DS   F        Parameter identifier
MQCFIL64_COUNT                  DS   F        Parameter value high
MQCFIL64_LENGTH                 EQU  *-MQCFIL64 Length of structure
                                ORG  MQCFIL64
MQCFIL64_AREA                   DS   CL(MQCFIL64_LENGTH)
```

MQCFIN - Integer parameter

Use this page to view the structure of an MQCFIN parameter and the declarations for the following programming languages: C, COBOL, PL/I, RPG/ILE, S/390 assembler, and Visual Basic

The MQCFIN structure describes an integer parameter. Following the links to the declarations is a description of the fields making up the MQCFIN structure:

- C language
- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- S/390 assembler-language (z/OS only)
- Visual Basic language (Windows only)

Type

Description: Indicates that the structure type is MQCFIN and describes an integer parameter.
Data type: MQLONG.
Value:

MQCFT_INTEGER
Structure defining an integer.

StrucLength

Description: Length in bytes of the MQCFIN structure.
Data type: MQLONG.
Value:

MQCFIN_STRUC_LENGTH
Length of MQCFIN structure.

Parameter

Description: Identifies the parameter with a value that is contained in the structure.
Data type: MQLONG.

Value

Description: Value of parameter identified by the *Parameter* field.
Data type: MQLONG.

C language declaration

```
typedef struct tagMQCFIN {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Value;        /* Parameter value */
 } MQCFIN;
```

COBOL language declaration

```
**   MQCFIN structure
  10 MQCFIN.
**    Structure type
   15 MQCFIN-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIN-STRUCLENGTH PIC S9(9) BINARY.
```

```
**    Parameter identifier
   15 MQCFIN-PARAMETER   PIC S9(9) BINARY.
**    Parameter value
   15 MQCFIN-VALUE       PIC S9(9) BINARY.
```

PL/I language declaration

```
dcl
 1 MQCFIN based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Value       fixed bin(31); /* Parameter value */
```

RPG/ILE declaration (IBM i only)

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     D* MQCFIN Structure
     D*
     D* Structure type
     D  INTYP                  1      4I 0
     D* Structure length
     D  INLEN                  5      8I 0
     D* Parameter identifier
     D  INPRM                  9     12I 0
     D* Parameter value
     D  INVAL                 13     16I 0
```

S/390 assembler-language declaration

```
MQCFIN                         DSECT
MQCFIN_TYPE                    DS   F        Structure type
MQCFIN_STRUCLENGTH             DS   F        Structure length
MQCFIN_PARAMETER               DS   F        Parameter identifier
MQCFIN_VALUE                   DS   F        Parameter value
MQCFIN_LENGTH                  EQU  *-MQCFIN Length of structure
                               ORG  MQCFIN
MQCFIN_AREA                    DS   CL(MQCFIN_LENGTH)
```

Visual Basic language declaration

```
Type MQCFIN
  Type As Long         ' Structure type
  StrucLength As Long  ' Structure length
  Parameter As Long    ' Parameter identifier
  Value As Long        ' Parameter value
End Type
```

MQCFIN64 - 64-bit integer parameter

Use this page to view the structure of an MQCFIN64 parameter and the declarations for the following programming languages: C, COBOL, PL/I, RPG/ILE, and S/390 assembler

The MQCFIN64 structure describes a 64-bit integer parameter. Following the links to the declarations is a description of the fields making up the MQCFIN64 structure:

- C language
- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- System/390 assembler-language (z/OS only)

Type

Description: Indicates that the structure is a MQCFIN64 structure describing a 64-bit integer parameter.
Data type: MQLONG.
Value:

MQCFT_INTEGER64
Structure defining a 64-bit integer.

StrucLength

Description: Length in bytes of the MQCFIN64 structure.
Data type: MQLONG.
Value:

MQCFIN64_STRUC_LENGTH
Length of 64-bit integer parameter structure.

Parameter

Description: Identifies the parameter with a value that is contained in the structure.
Data type: MQLONG.

Values

Description: This is the value of the parameter identified by the *Parameter* field.
Data type: (MQINT64)

C language declaration

```
typedef struct tagMQCFIN64 {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Reserved;     /* Reserved */
  MQINT64 Value;        /* Parameter value */
 } MQCFIN64;
```

COBOL language declaration

```
**   MQCFIN64 structure
  10 MQCFIN64.
**    Structure type
   15 MQCFIN64-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIN64-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIN64-PARAMETER   PIC S9(9) BINARY.
```

```
**    Reserved
   15 MQCFIN64-RESERVED    PIC S9(9) BINARY.
**    Parameter value
   15 MQCFIN64-VALUE       PIC S9(18) BINARY.
```

PL/I language declaration

```
dcl
 1 MQCFIN64 based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Reserved    fixed bin(31)  /* Reserved */
  3 Value       fixed bin(63); /* Parameter value */
```

RPG/ILE language declaration (IBM i only)

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     D* MQCFIN64 Structure
     D*
     D* Structure type
     D  IN64TYP                1      4I 0 INZ(23)
     D* Structure length
     D  IN64LEN                5      8I 0 INZ(24)
     D* Parameter identifier
     D  IN64PRM                9     12I 0 INZ(0)
     D* Reserved field
     D  IN64RSV               13     16I 0 INZ(0)
     D* Parameter value
     D  IN64VAL               17     16    INZ(0)
```

S/390 assembler-language declaration (z/OS only)

```
MQCFIN64                        DSECT
MQCFIN64_TYPE                   DS   F        Structure type
MQCFIN64_STRUCLENGTH            DS   F        Structure length
MQCFIN64_PARAMETER              DS   F        Parameter identifier
MQCFIN64_RESERVED               DS   F        Reserved
MQCFIN64_VALUE                  DS   D        Parameter value
MQCFIN64_LENGTH                 EQU  *-MQCFIN64 Length of structure
                                ORG  MQCFIN64
MQCFIN64_AREA                   DS   CL(MQCFIN64_LENGTH)
```

MQCFSL - String list parameter

Use this page to view the structure of an MQCFSL parameter and the declarations for the following programming languages: COBOL, PL/I, RPG/ILE, S/390 assembler, and Visual Basic

The MQCFSL structure describes a string list parameter. Following the links to the declarations is a description of the fields making up the MQCFSL structure:

- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- System/390 assembler-language (z/OS only)
- Visual Basic language (Windows only)

Type

Description: This indicates that the structure is an MQCFSL structure describing a string-list parameter.
Data type: MQLONG.
Value:

MQCFT_STRING_LIST
Structure defining a string list.

StrucLength

Description: This is the length in bytes of the MQCFSL structure, including the array of strings at the end of the structure (the *Strings* field).
Data type: MQLONG.

Parameter

Description: This identifies the parameter with values that are contained in the structure.
Data type: MQLONG.

CodedCharSetId

Description: This specifies the coded character set identifier of the data in the *Strings* field.
Data type: MQLONG.

Count

Description: This is the number of strings present in the *Strings* field; zero or greater.
Data type: MQLONG.

StringLength

Description: This is the length in bytes of one parameter value, that is the length of one string in the *Strings* field; all of the strings are this length.
Data type: MQLONG.

String

Description: This is a set of string values for the parameter identified by the *Parameter* field. The number of strings is given by the *Count* field, and the length of each string is given by the *StringLength* field. The strings are concatenated together, with no bytes skipped between adjacent strings. The total length of the strings is the length of one string multiplied by the number of strings present (that is, *StringLength* x *Count*).

In MQFMT_EVENT messages, trailing blanks can be omitted from string parameters (that is, the string may be shorter than the defined length of the parameter). *StringLength* gives the length of the string actually present in the message.
Note: In the MQCFSL structure, a null character in a string is treated as normal data, and does not act as a delimiter for the string. This means that when a receiving application reads a MQFMT_EVENT message, the receiving application receives all of the data specified by the sending application. The data may, of course, have been converted between character sets (for example, by the receiving application specifying the MQGMO_CONVERT option on the MQGET call).
Data type: MQCHAR x *StringLength* x *Count*

COBOL language declaration

```
**   MQCFSL structure
  10 MQCFSL.
**    Structure type
   15 MQCFSL-TYPE          PIC S9(9) BINARY.
```

```
**    Structure length
   15 MQCFSL-STRUCLENGTH    PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFSL-PARAMETER      PIC S9(9) BINARY.
**    Coded character set identifier
   15 MQCFSL-CODEDCHARSETID PIC S9(9) BINARY.
**    Count of parameter values
   15 MQCFSL-COUNT          PIC S9(9) BINARY.
**    Length of one string
   15 MQCFSL-STRINGLENGTH   PIC S9(9) BINARY.
```

PL/I language declaration

```
dcl
 1 MQCFSL based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Parameter      fixed bin(31), /* Parameter identifier */
  3 CodedCharSetId fixed bin(31), /* Coded character set identifier */
  3 Count          fixed bin(31), /* Count of parameter values */
  3 StringLength   fixed bin(31); /* Length of one string */
```

RPG/ILE declaration (IBM i only)

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     D* MQCFSL Structure
     D*
     D* Structure type
     D  SLTYP                  1      4I 0
     D* Structure length
     D  SLLEN                  5      8I 0
     D* Parameter identifier
     D  SLPRM                  9     12I 0
     D* Coded character set identifier
     D  SLCSI                 13     16I 0
     D* Count of parameter values
     D  SLCNT                 17     20I 0
     D* Length of one string
     D  SLSTL                 21     24I 0
```

S/390 assembler-language declaration (z/OS only)

```
MQCFSL                 DSECT
MQCFSL_TYPE            DS   F  Structure type
MQCFSL_STRUCLENGTH     DS   F  Structure length
MQCFSL_PARAMETER       DS   F  Parameter identifier
MQCFSL_CODEDCHARSETID  DS   F  Coded character set identifier
MQCFSL_COUNT           DS   F  Count of parameter values
MQCFSL_STRINGLENGTH    DS   F  Length of one string
*
MQCFSL_LENGTH          EQU  *-MQCFSL
                       ORG  MQCFSL
MQCFSL_AREA            DS   CL(MQCFSL_LENGTH)
```

Visual Basic language declaration (Windows systems only)

```
Type MQCFSL
  Type           As Long 'Structure type'
  StrucLength    As Long 'Structure length'
  Parameter      As Long 'Parameter identifier'
  CodedCharSetId As Long 'Coded character set identifier'
  Count          As Long 'Count of parameter values'
  StringLength   As Long 'Length of one string'
End Type
```

MQCFST - String parameter

Use this page to view the structure of an MQCFST parameter and the declarations for the following programming languages: C, COBOL, PL/I, RPG/ILE, S/390 assembler, and Visual Basic

The MQCFST structure describes a string parameter. Following the links to the declarations is a description of the fields making up the MQCFST structure:

- C language
- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- System/390 assembler-language (z/OS only)
- Visual Basic language (Windows only)

The MQCFST structure ends with a variable-length character string; see the *String* field for further details.

Type

Description: Indicates that the structure type is MQCFST and describes a string parameter.
Data type: MQLONG.
Value:

MQCFT_STRING
Structure defining a string.

StrucLength

Description: Length in bytes of the MQCFST structure, including the string at the end of the structure (the *String* field).
Data type: MQLONG.

Parameter

Description: Identifies the parameter with a value that is contained in the structure.
Data type: MQLONG.
Values: Dependent on the event message.

CodedCharSetId

Description: Coded character set identifier of the data in the *String* field.
Data type: MQLONG.

StringLength

Description: Length in bytes of the data in the *String* field; zero or greater.
Data type: MQLONG.

String

Description: The value of the parameter identified by the *Parameter* field.

In MQFMT_EVENT messages, trailing blanks can be omitted from string parameters (that is, the string may be shorter than the defined length of the parameter). *StringLength* gives the length of the string actually present in the message.

Data type: MQCHAR x *StringLength*

Value: The string can contain any characters that are in the character set defined by *CodedCharSetId*, and that are valid for the parameter identified by *Parameter*.

Language considerations: The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure should be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL, PL/I, System/390 assembler, and Visual Basic programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, the user should include MQCFST in a larger structure, and declare additional fields following MQCFST, to represent the *String* field as required.

A null character in the string is treated as normal data, and does not act as a delimiter for the string. This means that when a receiving application reads an MQFMT_EVENT message, the receiving application receives all of the data specified by the sending application. The data may, of course, have been converted between character sets (for example, by the receiving application specifying the MQGMO_CONVERT option on the MQGET call).

C language declaration

```
typedef struct tagMQCFST {
  MQLONG  Type;            /* Structure type */
  MQLONG  StrucLength;     /* Structure length */
  MQLONG  Parameter;       /* Parameter identifier */
  MQLONG  CodedCharSetId;  /* Coded character set identifier */
  MQLONG  StringLength;    /* Length of string */
  MQCHAR  String[1];       /* String value - first
                              character */
 } MQCFST;
```

COBOL language declaration

```
**   MQCFST structure
  10 MQCFST.
**    Structure type
   15 MQCFST-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFST-STRUCLENGTH    PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFST-PARAMETER      PIC S9(9) BINARY.
**    Coded character set identifier
   15 MQCFST-CODEDCHARSETID PIC S9(9) BINARY.
**    Length of string
   15 MQCFST-STRINGLENGTH   PIC S9(9) BINARY.
```

PL/I language declaration

```
dcl
 1 MQCFST based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Parameter      fixed bin(31), /* Parameter identifier */
  3 CodedCharSetId fixed bin(31), /* Coded character set identifier */
  3 StringLength   fixed bin(31); /* Length of string */
```

RPG/ILE declaration (IBM i only)

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     D* MQCFST Structure
     D*
     D* Structure type
     D  STTYP                  1      4I 0
     D* Structure length
     D  STLEN                  5      8I 0
     D* Parameter identifier
     D  STPRM                  9     12I 0
     D* Coded character set identifier
     D  STCSI                 13     16I 0
     D* Length of string
     D  STSTL                 17     20I 0
```

S/390 assembler-language declaration

```
MQCFST                         DSECT
MQCFST_TYPE                    DS   F        Structure type
MQCFST_STRUCLENGTH             DS   F        Structure length
MQCFST_PARAMETER               DS   F        Parameter identifier
MQCFST_CODEDCHARSETID          DS   F        Coded character set
*                                            identifier
MQCFST_STRINGLENGTH            DS   F        Length of string
MQCFST_LENGTH                  EQU  *-MQCFST Length of structure
                               ORG  MQCFST
MQCFST_AREA                    DS   CL(MQCFST_LENGTH)
```

Visual Basic language declaration

```
Type MQCFST
  Type As Long             ' Structure type
  StrucLength As Long      ' Structure length
  Parameter As Long        ' Parameter identifier
  CodedCharSetId As Long   ' Coded character set identifier
  StringLength As Long     ' Length of string
End Type
```

MQEPH - Embedded PCF header

Use this page to view the structure of an MQEPH embedded PCF header and the declarations for the following programming languages: C, COBOL, PL/I, RPG/ILE, S/390 assembler, and Visual Basic

The MQEPH structure describes the additional data that is present in a message when that message is a programmable command format (PCF) message. Following the links to the declarations is a description of the fields making up the MQEPH structure:

- C language
- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- S/390 assembler-language (z/OS only)
- Visual Basic language (Windows only)

The additional data consists of the MQEPH structure followed by an array of PCF parameter structures. To include the MQEPH structure in a message, the **Format** parameter in the message descriptor is set to MQFMT_EMBEDDED.

StrucId

Description: Structure identifier.
Data type: MQCHAR4.
Value:

MQEPH_STRUC_ID
Identifier for distribution header structure.

Version

Description: Structure version number.
Data type: MQLONG.
Value:

MQEPH_VERSION_1
Version number for embedded PCF header structure.

StrucLength

Description: Structure length. This is the length in bytes of the MQEPH structure and is set to the amount of data preceding the next header structure.
Data type: MQLONG.

Encoding

Description: Numeric encoding. This specifies the numeric encoding of the data that follows the last PCF parameter structure.
Data type: MQLONG.

CodedCharSetId

Description: Coded character set identifier. This specifies the coded character set identifier of the data that follows the last PCF parameter structure.
Data type: MQLONG.

Format

Description: Format. This specifies the format name of the data that follows the last PCF parameter structure.
Data type: MQCHAR8.

Flags

Description: Flags. This is a reserved field.
Data type: MQLONG.
Value:

MQEPH_NONE
No flags have been specified.

MQEPH_CCSID_EMBEDDED
The character set of the parameters containing character data is specified individually within the CodedCharSetId field in each structure. The character set of the StrucId and Format fields is defined by the CodedCharSetId field in the header structure that precedes the MQEPH structure, or by the CodedCharSetId field in the MQMD if the MQEPH is at the start of the message.

PCFHeader

Description: Command format header.
Data type: MQCFH.

C language declaration

```
struct tagMQEPH  {
  MQCHAR4 StrucId;         /* Structure identifier */
  MQLONG  Version;         /* Structure version number */
  MQLONG  StrucLength      /* Structure length */
  MQLONG  Encoding;        /* Numeric encoding */
  MQLONG  CodedCharSetId;  /* Coded character set identifier */
  MQCHAR8 Format;          /* Data format */
  MQLONG  Flags;           /* Flags */
  MQCFH   PCFHeader;       /* PCF header */
 } MQEPH;
```

COBOL language declaration

```
**   MQEPH structure
  10 MQEPH.
**    Structure identifier
   15 MQEPH-STRUCID        PIC X(4).
**    Structure version number
   15 MQEPH-VERSION        PIC S9(9) BINARY.
**    Structure length
   15 MQEPH-STRUCLENGTH    PIC S9(9) BINARY.
**    Numeric encoding
   15 MQEPH-ENCODING       PIC S9(9) BINARY.
**    Coded characeter set identifier
   15 MQEPH-CODEDCHARSETID PIC S9(9) BINARY.
**    Data format
   15 MQEPH-FORMAT         PIC X(8).
**    Flags
   15 MQEPH-FLAGS          PIC S9(9) BINARY.
**    PCF header
   15 MQEPH-PCFHEADER.
**    Structure type
    20 MQEPH-PCFHEADER-TYPE            PIC S9(9) BINARY.
**    Structure length
    20 MQEPH-PCFHEADER-STRUCLENGTH     PIC S9(9) BINARY.
**    Structure version number
    20 MQEPH-PCFHEADER-VERSION         PIC S9(9) BINARY.
**    Command identifier
    20 MQEPH-PCFHEADER-COMMAND         PIC S9(9) BINARY.
**    Message sequence number
    20 MQEPH-PCFHEADER-MSGSEQNUMBER    PIC S9(9) BINARY.
**    Control options
    20 MQEPH-PCFHEADER-CONTROL         PIC S9(9) BINARY.
**    Completion code
    20 MQEPH-PCFHEADER-COMPCODE        PIC S9(9) BINARY.
**    Reason code qualifying completion code
    20 MQEPH-PCFHEADER-REASON          PIC S9(9) BINARY.
**    Count of parameter structures
    20 MQEPH-PCFHEADER-PARAMETERCOUNT  PIC S9(9) BINARY.
```

PL/I language declaration (z/OS and Windows)

```
dcl
 1 MQEPH based,
  3 StrucId         char(4),       /* Structure identifier */
  3 Version         fixed bin(31), /* Structure version number */
  3 StrucLength     fixed bin(31), /* Structure length */
  3 Encoding        fixed bin(31), /* Numeric encoding */
  3 CodedCharSetId  fixed bin(31), /* Coded character set identifier */
  3 Format          char(8),       /* Data format */
  3 Flags           fixed bin(31), /* Flags */
```

```
  3 PCFHeader,                      /* PCF header */
   5 Type           fixed bin(31), /* Structure type */
   5 StrucLength    fixed bin(31), /* Structure length */
   5 Version        fixed bin(31), /* Structure version number */
   5 Command        fixed bin(31), /* Command identifier */
   5 MsgSeqNumber   fixed bin(31), /* Message sequence number */
   5 Control        fixed bin(31), /* Control options */
   5 CompCode       fixed bin(31), /* Completion code */
   5 Reason         fixed bin(31), /* Reason code qualifying completion
                                      code */
   5 ParameterCount fixed bin(31); /* Count of parameter structures */
```

RPG language declaration (IBM i only)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQEPH Structure
D*
D* Structure identifier
D  EPSID                  1      4    INZ('EPH ')
D* Structure version number
D  EPVER                  5      8I 0 INZ(1)
D* Structure length
D  EPLEN                  9     12I 0 INZ(68)
D* Numeric encoding
D  EPENC                 13     16I 0 INZ(0)
D* Coded character set identifier
D  EPCSI                 17     20I 0 INZ(0)
D* Format name
D  EPFMT                 21     28I 0 INZ('        ')
D* Flags
D  EPFLG                 29     32I 0 INZ(0)
D* Programmable Command Format Header
D*
D* Structure type
D  EP1TYPE               33     36I 0 INZ(0)
D* Structure length
D  EP1LEN                37     40I 0 INZ(36)
D* Structure version number
D  EP1VER                41     44I 0 INZ(3)
D* Command identifier
D  EP1CMD                45     48I 0 INZ(0)
D* Message sequence number
D  EP1SEQ                49     52I 0 INZ(1)
D* Control options
D  EP1CTL                53     56I 0 INZ(1)
D* Completion code
D  EP1CMP                57     60I 0 INZ(0)
D* Reason code qualifying completion code
D  EP1REA                61     64I 0 INZ(0)
D* Count of parameter structures
D  EP1CNT                65     68I 0 INZ(0)
```

S/390 assembler-language declaration (z/OS only)

```
MQEPH                           DSECT
MQEPH_STRUCID                   DS   CL4      Structure identifier
MQEPH_VERSION                   DS   F        Structure version number
MQEPH_STRUCLENGTH               DS   F        Structure length
MQEPH_ENCODING                  DS   F        Numeric encoding
MQEPH_CODEDCHARSETID            DS   F        Coded character set identifier
MQEPH_FORMAT                    DS   CL8      Data format
MQEPH_FLAGS                     DS   F        Flags
MQEPH_PCFHEADER                 DS   0F       Force fullword alignment
MQEPH_PCFHEADER_TYPE            DS   F        Structure type
MQEPH_PCFHEADER_STRUCLENGTH     DS   F        Structure length
MQEPH_PCFHEADER_VERSION         DS   F        Structure version number
MQEPH_PCFHEADER_COMMAND         DS   F        Command identifier
MQEPH_PCFHEADER_MSGSEQNUMBER    DS   F        Message sequence number
```

```
MQEPH_PCFHEADER_CONTROL        DS   F        Control options
MQEPH_PCFHEADER_COMPCODE       DS   F        Completion code
MQEPH_PCFHEADER_REASON         DS   F        Reason code qualifying completion code
MQEPH_PCFHEADER_PARAMETERCOUNT DS   F        Count of parameter structures
MQEPH_PCFHEADER_LENGTH         EQU  *-MQEPH_PCFHEADER
                               ORG  MQEPH_PCFHEADER
MQEPH_PCFHEADER_AREA           DS   CL(MQEPH_PCFHEADER_LENGTH)
*
MQEPH_LENGTH                   EQU  *-MQEPH
                               ORG  MQEPH
MQEPH_AREA                     DS   CL(MQEPH_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQEPH
  StrucId As String*4     'Structure identifier
  Version As Long         'Structure version number
  StrucLength As Long     'Structure length
  Encoding As Long        'Numeric encoding
  CodedCharSetId As Long  'Coded characetr set identifier
  Format As String*8      'Format name
  Flags As Long           'Flags
  Reason As Long          'Reason code qualifying completion code
  PCFHeader As MQCFH      'PCF header
End Type
```

Object attributes for event data

Information about the object attributes that IBM MQ monitoring techniques can include in the configuration event data recorded in event messages. The amount of event data depends on the type of object to which the configuration event relates.

Authentication configuration attributes

Event messages relating to objects can include authentication configuration attributes

AuthorityRecordType (MQCFIN)
Object type (parameter identifier: **MQIACF_AUTH_REC_TYPE**).

Describes the object type whose profile is being updated, for example MQOT_Q.

AuthorizationList (MQCFIL)
Authorization list (parameter identifier: **MQIACF_AUTHORIZATION_LIST**).

Displays the MQAUTH_* values; see Inquire Authority Records (Response).

EntityName (MQCFST)
Entity name (parameter identifier: **MQCACF_ENTITY_NAME**).

The entity name can be either a principal name or a group name.

The maximum length of the string is MQ_ENTITY_NAME_LENGTH.

EntityType (MQCFIN)
Entity type (parameter identifier: **MQIACF_ENTITY_TYPE**).

Displays the MQZAET_* values; see Inquire Authority Records (Response).

Authentication information attributes

Event messages relating to objects can include authentication information attributes

AlterationDate (MQCFST)
Alteration date (parameter identifier: **MQCA_ALTERATION_DATE**).

The date when the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: **MQCA_ALTERATION_TIME**).

The time when the information was last altered.

AuthInfoConnName (MQCFST)
Authentication information connection name (parameter identifier: **MQCA_AUTH_INFO_CONN_NAME**).

The maximum length of the string is 48.

AuthInfoDesc (MQCFST)
Authentication information description (parameter identifier: **MQCA_AUTH_INFO_DESC**).

The maximum length of the string is MQ_AUTH_INFO_DESC_LENGTH.

AuthInfoType (MQCFIN)
Authentication information type (parameter identifier: **MQIA_AUTH_INFO_TYPE**).

The value is MQAIT_CRL_LDAP.

LDAPPassword (MQCFST)
LDAP password (parameter identifier: **MQCA_LDAP_PASSWORD**).

The maximum length of the string is MQ_LDAP_PASSWORD_LENGTH.

LDAPUserName (MQCFST)
LDAP user name (parameter identifier: **MQCA_LDAP_USER_NAME**).

The maximum length of the string is 256.

CF structure attributes

Event messages relating to objects can include CF structure attributes

AlterationDate (MQCFST)
Alteration date (parameter identifier: **MQCA_ALTERATION_DATE**).

The date when the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: **MQCA_ALTERATION_TIME**).

The time when the information was last altered.

CFLevel (MQCFIN)
CF level (parameter identifier: **MQIA_CF_LEVEL**).

CFStrucDesc (MQCFST)
CF Structure description (parameter identifier: **MQCA_CF_STRUC_DESC**).

The maximum length of the string is MQCA_CF_STRUC_DESC_LENGTH.

Recovery (MQCFIN)
Recovery (parameter identifier: **MQIA_CF_RECOVER**).

Communication information attributes

AlterationDate (MQCFST)
Alteration date (parameter identifier: **MQCA_ALTERATION_DATE**).

The date when the information was last altered, in the form *yyyy-mm-dd*.

AlterationTime (MQCFST)
Alteration time (parameter identifier: **MQCA_ALTERATION_TIME**).

The time when the information was last altered, in the form *hh.mm.ss*.

Bridge (MQCFIN)
Bridge (parameter identifier: **MQIA_MCAST_BRIDGE**).

Specifies whether publications from applications not using Multicast are bridged to applications using multicast.

The value can be any of the following values:

MQMCB_DISABLED
Bridging is disabled.

MQMCB_ENABLED
Bridging is enabled.

CCSID (MQCFIN)
Coded character set identifier (parameter identifier: **MQIA_CODED_CHAR_SET_ID**).

The CCSID that messages are transmitted on.

CommEvent (MQCFIN)
Communication event (parameter identifier: **MQIA_COMM_EVENT**).

Controls whether event messages are generated for multicast handles that are created using this COMMINFO object.

The value can be any of the following values:

MQEVR_DISABLED
Event messages are not generated.

MQEVR_ENABLED
Event messages are generated.

MQEVR_EXCEPTION
Event messages are generated if the message reliability is below the reliability threshold.

CominfoName (MQCFST)
Communication information name (parameter identifier: **MQCA_COMM_INFO_NAME**).

The name of the administrative communication information definition about which information is to be returned.

Description (MQCFST)
Description (parameter identifier: **MQCA_COMM_INFO_DESC**).

Plain-text comment that provides descriptive information about the communication information object.

Encoding (MQCFIN)
Encoding (parameter identifier: **MQIACF_ENCODING**).

The encoding that the messages are transmitted in.

The value can be any of the following values:

MQENC_AS_PUBLISHED

MQENC_NORMAL

MQENC_REVERSED

MQENC_S390

MQENC_TNS

GrpAddress (MQCFST)
Group address (parameter identifier: **MQCACH_GROUP_ADDRESS**).

The group IP address or DNS name.

MonitorInterval (MQCFIN)
Frequency of monitoring (parameter identifier: **MQIA_MONITOR_INTERVAL**).

How frequently, in seconds, monitoring information is updated and event messages are generated.

MulticastHeartbeat (MQCFIN)
Multicast heartbeat (parameter identifier: **MQIACH_MC_HB_INTERVAL**).

Heartbeat interval measured in milliseconds.

MulticastPropControl (MQCFIN)
Multicast properties control (parameter identifier: **MQIACH_MULTICAST_PROPERTIES**).

Controls how many of the MQMD properties and user properties flow with the message.

The value can be any of the following values:

MQMCP_ALL
All properties are transmitted.

MQMCP_REPLY
Only user properties and MQMD fields that deal with replying to the messages are transmitted.

MQMCP_USER
Only user properties are transmitted.

MQMCP_NONE
No properties are transmitted.

MQMCP_COMPAT
Properties are transmitted in a format compatible with previous IBM MQ multicast clients.

MsgHistory (MQCFIN)
Message history (parameter identifier: **MQIACH_MSG_HISTORY**).

The amount of message history in kilobytes that is kept by the system to handle retransmissions in the case of NACKs.

NewSubHistory (MQCFIN)
New Subscriber History (parameter identifier: **MQIACH_NEW_SUBSCRIBER_HISTORY**).

Controls how much historical data a new subscriber receives. The value can be any of the following values:

MQNSH_NONE
Only publications from the time of the subscription are sent.

MQNSH_ALL
As much history as is known is retransmitted.

PortNumber (MQCFIN)
Port Number (parameter identifier: **MQIACH_PORT**).

The port number to transmit on.

Type (MQCFIN)
Type (parameter identifier: **MQIA_COMM_INFO_TYPE**).

The type of the communications information object.

Channel attributes

Event messages relating to objects can include channel attributes

Only those attributes that apply to the type of channel in question are included in the event data.

AlterationDate (MQCFST)
Alteration date (parameter identifier: **MQCA_ALTERATION_DATE**).

The date when the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: **MQCA_ALTERATION_TIME**).

The time when the information was last altered.

BatchHeartbeat (MQCFIN)
The value being used for the batch heartbeating (parameter identifier: **MQIACH_BATCH_HB**).

The value can be in the range 0 through 999999. A value of 0 indicates heartbeating is not in use.

BatchInterval (MQCFIN)
Batch interval (parameter identifier: **MQIACH_BATCH_INTERVAL**).

BatchSize (MQCFIN)
Batch size (parameter identifier: **MQIACH_BATCH_SIZE**).

ChannelDesc (MQCFST)
Channel description (parameter identifier: **MQCACH_DESC**).

The maximum length of the string is MQ_CHANNEL_DESC_LENGTH.

ChannelMonitoring (MQCFIN)
Level of monitoring data collection for the channel (parameter identifier: **MQIA_MONITORING_CHANNEL**).

The value can be any of the following values:

MQMON_OFF
Monitoring data collection is turned off.

MQMON_LOW
Monitoring data collection is turned on with a low ratio of data collection.

MQMON_MEDIUM
Monitoring data collection is turned on with a medium ratio of data collection.

MQMON_HIGH
Monitoring data collection is turned on with a high ratio of data collection.

MQMON_Q_MGR
The level of monitoring data collected is based on the queue manager attribute **ChannelMonitoring**.

ChannelName (MQCFST)
Channel name (parameter identifier: **MQCACH_CHANNEL_NAME**).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType (MQCFIN)
Channel type (parameter identifier: **MQIACH_CHANNEL_TYPE**).

The value can be:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLNTCONN
Client connection.

MQCHT_CLUSRCVR
Cluster-receiver.

MQCHT_CLUSSDR
Cluster-sender.

CipherSpec (MQCFST)
SSL cipher specification (parameter identifier: **MQCACH_SSL_CIPHER_SPEC**).

The maximum length of the string is MQ_SSL_CIPHER_SPEC_LENGTH.

ClusterName (MQCFST)
Cluster name (parameter identifier: **MQCA_CLUSTER_NAME**).

ClusterNamelist (MQCFST)
Cluster namelist (parameter identifier: **MQCA_CLUSTER_NAMELIST**).

CLWLChannelPriority (MQCFIN)
Cluster workload channel priority (parameter identifier: **MQIACH_CLWL_CHANNEL_PRIORITY**).

CLWLChannelRank (MQCFIN)
Cluster workload channel rank (parameter identifier: **MQIACH_CLWL_CHANNEL_RANK**).

CLWLChannelWeight (MQCFIN)
Cluster workload channel weight (parameter identifier: **MQIACH_CLWL_CHANNEL_WEIGHT**).

ConnectionName (MQCFST)
Connection name (parameter identifier: **MQCACH_CONNECTION_NAME**).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

DataConversion (MQCFIN)
Whether sender should convert application data (parameter identifier: **MQIACH_DATA_CONVERSION**).

The value can be any of the following values:

MQCDC_NO_SENDER_CONVERSION
No conversion by sender.

MQCDC_SENDER_CONVERSION
Conversion by sender.

DiscInterval (MQCFIN)
Disconnection interval (parameter identifier: **MQIACH_DISC_INTERVAL**).

HeaderCompression (MQCFIL)
Header data compression techniques supported by the channel (parameter identifier: **MQIACH_HDR_COMPRESSION**).

For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference.

The value can be one, or more, of the following:

MQCOMPRESS_NONE
No header data compression is performed.

MQCOMPRESS_SYSTEM
Header data compression is performed.

HeartbeatInterval (MQCFIN)
Heartbeat interval (parameter identifier: **MQIACH_HB_INTERVAL**).

KeepAliveInterval (MQCFIN)
Keep alive interval (parameter identifier: **MQIACH_KEEP_ALIVE_INTERVAL**).

LocalAddress (MQCFST)
Local communications address for the channel (parameter identifier: **MQCACH_LOCAL_ADDRESS**).

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

LongRetryCount (MQCFIN)
Long retry count (parameter identifier: **MQIACH_LONG_RETRY**).

LongRetryInterval (MQCFIN)
Long timer (parameter identifier: **MQIACH_LONG_TIMER**).

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: **MQIACH_MAX_MSG_LENGTH**).

MCAName (MQCFST)
Message channel agent name (parameter identifier: **MQCACH_MCA_NAME**).

The maximum length of the string is MQ_MCA_NAME_LENGTH.

MCAType (MQCFIN)
Message channel agent type (parameter identifier: **MQIACH_MCA_TYPE**).

The value can be any of the following values:

MQMCAT_PROCESS
Process

MQMCAT_THREAD
Thread

MCAUserIdentifier (MQCFST)
Message channel agent user identifier (parameter identifier: **MQCACH_MCA_USER_ID**).

The maximum length of the MCA user identifier is MQ_MCA_USER_ID_LENGTH.

MessageCompression (MQCFIL)
Message data compression techniques supported by the channel (parameter identifier: **MQIACH_MSG_COMPRESSION**).

For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference.

The value can be one, or more, of:

MQCOMPRESS_NONE
No message data compression is performed. This is the default value.

MQCOMPRESS_RLE
Message data compression is performed using run-length encoding.

MQCOMPRESS_ZLIBFAST
Message data compression is performed using ZLIB encoding with speed prioritized.

MQCOMPRESS_ZLIBHIGH
Message data compression is performed using ZLIB encoding with compression prioritized.

MQCOMPRESS_ANY
Any compression technique supported by the queue manager can be used. This is only valid for receiver, requester, and server-connection channels.

ModeName (MQCFST)
Mode name (parameter identifier: **MQCACH_MODE_NAME**).

The maximum length of the string is MQ_MODE_NAME_LENGTH.

MsgExit (MQCFSL)
Message exit name (parameter identifier: **MQCACH_MSG_EXIT_NAME**).

The number of names in the list is given by the **Count** field in the MQCFSL structure. It is the same as the count for **MsgUserData**. It may exceed the number of exit names specified for the channel, in which case the excess names are blank; the minimum is 1. The length of each name is given by the **StringLength** field in that structure.

The maximum length of the exit name is MQ_EXIT_NAME_LENGTH.

MsgRetryCount (MQCFIN)
Message retry count (parameter identifier: **MQIACH_MR_COUNT**).

Specifies the number of times that a failing message should be retried.

This parameter is only valid for receiver, cluster-receiver, and requester channels.

MsgRetryExit (MQCFST)
Message retry exit name (parameter identifier: **MQCACH_MR_EXIT_NAME**).

This parameter is only valid for receiver, cluster-receiver, and requester channels.

The maximum length of the string is MQ_MAX_EXIT_NAME_LENGTH.

MsgRetryInterval (MQCFIN)
Message retry interval (parameter identifier: **MQIACH_MR_INTERVAL**).

Specifies the minimum time interval in milliseconds between retries of failing messages.

This parameter is only valid for receiver, cluster-receiver, and requester channels.

MsgRetryUserData (MQCFST)
Message retry exit user data (parameter identifier: **MQCACH_MR_EXIT_USER_DATA**).

Specifies user data that is passed to the message retry exit.

This parameter is only valid for receiver, cluster-receiver, and requester channels.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

MsgUserData (MQCFSL)
Message exit user data (parameter identifier: **MQCACH_MSG_EXIT_USER_DATA**).

The number of names in the list is given by the **Count** field in the MQCFSL structure. It is the same as the count for **MsgExit**. The length of each name is given by the **StringLength** field in that structure.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

NetworkPriority (MQCFIN)
Network priority (parameter identifier: **MQIACH_NETWORK_PRIORITY**).

NonPersistentMsgSpeed (MQCFIN)
Speed at which nonpersistent messages are to be sent (parameter identifier: **MQIACH_NPM_SPEED**).

The value can be any of the following values:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

Password (MQCFST)
Password (parameter identifier: **MQCACH_PASSWORD**).

The maximum length of the string is MQ_PASSWORD_LENGTH.

PeerName (MQCFST)
SSL peer name (parameter identifier: **MQCACH_SSL_PEER_NAME**).

The maximum length of the string is 256.

PutAuthority (MQCFIN)
Put authority (parameter identifier: **MQIACH_PUT_AUTHORITY**).

The value can be:

MQPA_DEFAULT
Default user identifier is used.

MQPA_CONTEXT
Context user identifier is used.

MQPA_ALTERNATE_OR_MCA
Alternate or MCA user identifier is used.

MQPA_ONLY_MCA
Only MCA user identifier is used.

QMgrName (MQCFST)
Queue manager name (parameter identifier: **MQCA_Q_MGR_NAME**).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ReceiveExit (MQCFSL)
Receive exit name (parameter identifier: **MQCACH_RCV_EXIT_NAME**).

The number of names in the list is given by the **Count** field in the MQCFSL structure. It is the same as the count for **ReceiveUserData**. It may exceed the number of exit names specified for the channel, in which case the excess names are blank; the minimum is 1. The length of each name is given by the **StringLength** field in that structure.

For a client-connection channel the maximum length of the exit name is MQ_MAX_EXIT_NAME_LENGTH. For all other channels, the maximum length of the exit name is MQ_EXIT_NAME_LENGTH.

ReceiveUserData (MQCFSL)
Receive exit user data (parameter identifier: **MQCACH_RCV_EXIT_USER_DATA**).

The number of names in the list is given by the **Count** field in the MQCFSL structure. It is the same as the count for **ReceiveExit**. The length of each name is given by the **StringLength** field in that structure.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

SecurityExit (MQCFST)
Security exit name (parameter identifier: **MQCACH_SEC_EXIT_NAME**).

For a client-connection channel the maximum length of the exit name is MQ_MAX_EXIT_NAME_LENGTH. For all other channels, the maximum length of the exit name is MQ_EXIT_NAME_LENGTH.

SecurityUserData (MQCFST)
Security exit user data (parameter identifier: **MQCACH_SEC_EXIT_USER_DATA**).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

SendExit (MQCFSL)
Send exit name (parameter identifier: **MQCACH_SEND_EXIT_NAME**).

The number of names in the list is given by the **Count** field in the MQCFSL structure. It is the same as the count for **SendUserData**. It may exceed the number of exit names specified for the channel, in which case the excess names are blank; the minimum is 1. The length of each name is given by the *StringLength* field in that structure.

For a client-connection channel the maximum length of the exit name is MQ_MAX_EXIT_NAME_LENGTH. For all other channels, the maximum length of the exit name is MQ_EXIT_NAME_LENGTH.

SendUserData (MQCFSL)
Send exit user data (parameter identifier: **MQCACH_SEND_EXIT_USER_DATA**).

The number of names in the list is given by the **Count** field in the MQCFSL structure. It is the same as the count for **SendExit**. The length of each name is given by the **StringLength** field in that structure.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

SeqNumberWrap (MQCFIN)
Sequence wrap number (parameter identifier: **MQIACH_SEQUENCE_NUMBER_WRAP**).

ShortRetryCount (MQCFIN)
Short retry count (parameter identifier: **MQIACH_SHORT_RETRY**).

ShortRetryInterval (MQCFIN)
Short timer (parameter identifier: **MQIACH_SHORT_TIMER**).

SSLClientAuthentication (MQCFIN)
SSL client authentication (parameter identifier: **MQIACH_SSL_CLIENT_AUTH**).

The value can be:

MQSCA_REQUIRED
Certificate required.

MQSCA_OPTIONAL
Certificate optional.

TpName (MQCFST)
Transaction program name (parameter identifier: **MQCACH_TP_NAME**).

The maximum length of the string is MQ_TP_NAME_LENGTH.

TransportType (MQCFIN)
Transmission protocol type (parameter identifier: **MQIACH_XMIT_PROTOCOL_TYPE**).

The value may be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

MQXPT_NETBIOS
NetBIOS.

MQXPT_SPX
SPX.

UserIdentifier (MQCFST)
Task user identifier (parameter identifier: **MQCACH_USER_ID**).

The maximum length of the string is MQ_USER_ID_LENGTH.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: **MQCACH_XMIT_Q_NAME**).

The maximum length of the string is MQ_Q_NAME_LENGTH.

Channel authentication attributes

Event messages relating to objects can include channel authentication attributes

Only those attributes that apply to the type of channel in question are included in the event data.

ChannelProfile (MQCFST)
Channel Profile (parameter identifier: **MQCACH_CHANNEL_NAME**).

Maximum length is MQ_CHANNEL_NAME_LENGTH.

Returned: Always.

ChannelAuthType (MQCFIN)
Channel Authentication Type (parameter identifier: **MQIACF_CHLAUTH_TYPE**).

Returned: Always.

Warning (MQCFIN)
Warning (parameter identifier: **MQIACH_WARNING**).

Returned: Always.

connectionNameList (MQCFSL)
Connection Name List (parameter identifier: **MQCACH_CONNECTION_NAME_LIST**).

Element length: MQ_CONN_NAME_LENGTH.

Returned: Only when **ChannelAuthType** is MQAUT_BLOCKADDR.

MCAUserIdList (MQCFSL)
MCA User Id List (parameter identifier: **MQCACH_MCA_USER_ID_LIST**).

Element length: MQ_MCA_USER_ID_LENGTH.

Returned: Only when **ChannelAuthType** is MQAUT_BLOCKUSER.

MCAUser (MQCFST)
MCA User (parameter identifier: **MQCACH_MCA_USER_ID**).

Maximum length: MQ_MCA_USER_ID_LENGTH.

Returned: Only when **ChannelAuthType** is of a mapping type (MQCAUT_SSLPEERMAP, MQCAUT_ADDRESSMAP, MQCAUT_USERMAP or MQCAUT_QMGRMAP).

ConnectionName (MQCFST)
Connection Name (parameter identifier: **MQCACH_CONNECTION_NAME**).

Maximum length: MQ_CONN_NAME_LENGTH.

Returned: Only when **ChannelAuthType** is of a mapping type (MQCAUT_SSLPEERMAP, MQCAUT_ADDRESSMAP, MQCAUT_USERMAP or MQCAUT_QMGRMAP).

UserSource (MQCFIN)
User Source (parameter identifier: **MQIACH_USER_SOURCE**).

Returned: Only when **ChannelAuthType** is of a mapping type (MQCAUT_SSLPEERMAP, MQCAUT_ADDRESSMAP, MQCAUT_USERMAP or MQCAUT_QMGRMAP).

SSLPeerName (MQCFST)
SSL Peer Name (parameter identifier: **MQCACH_SSL_PEER_NAME**).

Maximum length: MQ_SSL_PEER_NAME_LENGTH.

Returned: Only when **ChannelAuthType** is MQCAUT_SSLPEERMAP.

ClientUserId (MQCFST)
Client User Id (parameter identifier: **MQCACH_CLIENT_USER_ID**).

Maximum length: MQ_MCA_USER_ID_LENGTH.

Returned: Only when **ChannelAuthType** is MQCAUT_USERMAP.

RemoteQueueManagerName (MQCFST)
Remote Queue Manager Name (parameter identifier: **MQCA_REMOTE_Q_MGR_NAME**).

Maximum length: MQ_Q_MGR_NAME_LENGTH.

Returned: Only when **ChannelAuthType** is MQCAUT_QMGRMAP.

Listener attributes

AlterationDate (MQCFST)
Alteration date (parameter identifier: **MQCA_ALTERATION_DATE**).

The date, in the form *yyyy-mm-dd*, on which the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: **MQCA_ALTERATION_TIME**).

The time, in the form *hh.mm.ss*, at which the information was last altered.

Windows **Adapter (MQCIN)**
Adapter number (parameter identifier: **MQIACH_ADAPTER**).

The adapter number on which NetBIOS listens. This parameter is valid only on Windows.

Backlog (MQCIN)
Backlog (parameter identifier: **MQIACH_BACKLOG**).

The number of concurrent connection requests that the listener supports.

Windows **Commands (MQCIN)**
Adapter number (parameter identifier: **MQIACH_COMMAND_COUNT**).

The number of commands that the listener can use. This parameter is valid only on Windows.

IPAddress (MQCFST)
IP address (parameter identifier: **MQCACH_IP_ADDRESS**).

IP address for the listener specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric host name form.

ListenerDesc (MQCFST)
Description of listener definition (parameter identifier: **MQCACH_LISTENER_DESC**).

ListenerName (MQCFST)
Name of listener definition (parameter identifier: **MQCACH_LISTENER_NAME**).

Windows **LocalName (MQCFST)**
NetBIOS local name (parameter identifier: **MQCACH_LOCAL_NAME**).

The NetBIOS local name that the listener uses. This parameter is valid only on Windows.

Windows **NetbiosNames (MQCFIN)**
NetBIOS names (parameter identifier: **MQIACH_NAME_COUNT**).

The number of names that the listener supports. This parameter is valid only on Windows.

Port (MQCFIN)
Port number (parameter identifier: **MQIACH_PORT**).

The port number for TCP/IP. This parameter is valid only if the value of **TransportType** is MQXPT_TCP.

Windows **Sessions (MQCFIN)**
NetBIOS sessions (parameter identifier: MQIACH_SESSION_COUNT).

The number of sessions that the listener can use. This parameter is valid only on Windows.

Socket (MQCFIN)
SPX socket number (parameter identifier: **MQIACH_SOCKET**).

The SPX socket on which to listen. This parameter is valid only if the value of **TransportType** is MQXPT_SPX.

StartMode (MQCFIN)
Service mode (parameter identifier: **MQIACH_LISTENER_CONTROL**).

Specifies how the listener is to be started and stopped. The value can be:

MQSVC_CONTROL_MANUAL
The listener is started and stopped manually, by user command.

MQSVC_CONTROL_Q_MGR
The listener is started and stopped when the queue manager starts and stops.

MQSVC_CONTROL_Q_MGR_START
The listener is started when the queue manager starts, but does not stop when the queue manager stops.

Windows **TPName (MQCFST)**
Transaction program name (parameter identifier: **MQCACH_TP_NAME**).

The LU 6.2 transaction program name. This parameter is valid only on Windows.

TransportType (MQCFIN)
Transmission protocol (parameter identifier: **MQIACH_XMIT_PROTOCOL_TYPE**).

The value can be any of the following values:

MQXPT_TCP
TCP

MQXPT_LU62
LU 6.2

MQXPT_NETBIOS
NetBIOS

MQXPT_SPX
SPX

Namelist attributes

Event messages relating to objects can include namelist attributes

AlterationDate (MQCFST)
Alteration date (parameter identifier: **MQCA_ALTERATION_DATE**).

The date when the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: **MQCA_ALTERATION_TIME**).

The time when the information was last altered.

NameCount (MQCFIN)
Number of names in the namelist (parameter identifier: **MQIA_NAME_COUNT**).

The number of names contained in the namelist.

NamelistDesc (MQCFST)
Description of namelist definition (parameter identifier: **MQCA_NAMELIST_DESC**).

The maximum length of the string is MQ_NAMELIST_DESC_LENGTH.

NamelistName (MQCFST)
The name of the namelist definition (parameter identifier: **MQCA_NAMELIST_NAME**).

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

NamelistType (MQCFIN)
Namelist type (parameter identifier: **MQIA_NAMELIST_TYPE**).

Names (MQCFSL)
The names contained in the namelist (parameter identifier: **MQCA_NAMES**).

The number of names in the list is given by the **Count** field in the MQCFSL structure. The length of each name is given by the **StringLength** field in that structure. The maximum length of a name is MQ_OBJECT_NAME_LENGTH.

Process attributes

Event messages relating to objects can include process attributes

AlterationDate (MQCFST)
Alteration date (parameter identifier: **MQCA_ALTERATION_DATE**).

The date when the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: **MQCA_ALTERATION_TIME**).

The time when the information was last altered.

ApplId (MQCFST)
Application identifier (parameter identifier: **MQCA_APPL_ID**).

The maximum length of the string is MQ_PROCESS_APPL_ID_LENGTH.

ApplType (MQCFIN)
Application type (parameter identifier: **MQIA_APPL_TYPE**).

EnvData (MQCFST)
Environment data (parameter identifier: **MQCA_ENV_DATA**).

The maximum length of the string is MQ_PROCESS_ENV_DATA_LENGTH.

ProcessDesc (MQCFST)
Description of process definition (parameter identifier: **MQCA_PROCESS_DESC**).

The maximum length of the string is MQ_PROCESS_DESC_LENGTH.

ProcessName (MQCFST)
The name of the process definition (parameter identifier: **MQCA_PROCESS_NAME**).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

UserData (MQCFST)
User data (parameter identifier: **MQCA_USER_DATA**).

The maximum length of the string is MQ_PROCESS_USER_DATA_LENGTH.

Queue attributes

Event messages relating to objects can include queue attributes

Only those attributes that apply to the type of queue in question are included in the event data.

AlterationDate (MQCFST)
Alteration date (parameter identifier: **MQCA_ALTERATION_DATE**).

The date when the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: **MQCA_ALTERATION_TIME**).

The time when the information was last altered.

BackoutRequeueName (MQCFST)
Excessive backout requeue name (parameter identifier: **MQCA_BACKOUT_REQ_Q_NAME**).

The maximum length of the string is MQ_Q_NAME_LENGTH.

BackoutThreshold (MQCFIN)
Backout threshold (parameter identifier: **MQIA_BACKOUT_THRESHOLD**).

BaseQName (MQCFST)
Queue name to which the alias resolves (parameter identifier: **MQCA_BASE_Q_NAME**).

This is the name of a queue that is defined to the local queue manager.

The maximum length of the string is MQ_Q_NAME_LENGTH.

CFstructure (MQCFST)
CF structure name (parameter identifier: **MQCA_CF_STRUC_NAME**).

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

ClusterName (MQCFST)
Cluster name (parameter identifier: **MQCA_CLUSTER_NAME**).

ClusterNamelist (MQCFST)
Cluster namelist (parameter identifier: **MQCA_CLUSTER_NAMELIST**).

CLWLQueuePriority (MQCFIN)
Queue priority (parameter identifier: **MQIA_CLWL_Q_PRIORITY**).

CLWLQueueRank (MQCFIN)
Queue rank (parameter identifier: **MQIA_CLWL_Q_RANK**).

CLWLUseQ (MQCFIN)
This defines the behavior of an MQPUT when the target queue has both a local instance and at least one remote cluster instance (parameter identifier: **MQIA_CLWL_USEQ**).

The value can be any of the following values:

MQCLWL_USEQ_ANY
Use remote and local queues.

MQCLWL_USEQ_LOCAL
Do not use remote queues.

MQCLWL_USEQ_AS_Q_MGR
Inherit definition from the queue manager attribute **CLWLUseQ**.

CreationDate (MQCFST)
Queue creation date (parameter identifier: **MQCA_CREATION_DATE**).

The maximum length of the string is MQ_CREATION_DATE_LENGTH.

CreationTime (MQCFST)
Creation time (parameter identifier: **MQCA_CREATION_TIME**).

The maximum length of the string is MQ_CREATION_TIME_LENGTH.

DefBind (MQCFIN)
Default binding (parameter identifier: **MQIA_DEF_BIND**).

The value can be:

MQBND_BIND_ON_OPEN
Binding fixed by MQOPEN call.

MQBND_BIND_NOT_FIXED
Binding not fixed.

MQBND_BIND_ON_GROUP
Allows an application to request that a group of messages are all allocated to the same destination instance.

DefinitionType (MQCFIN)
Queue definition type (parameter identifier: **MQIA_DEFINITION_TYPE**).

The value can be any of the following values:

MQQDT_PREDEFINED
Predefined permanent queue.

MQQDT_PERMANENT_DYNAMIC
Dynamically defined permanent queue.

MQQDT_SHARED_DYNAMIC
Dynamically defined permanent queue that is shared.

DefInputOpenOption (MQCFIN)
Default input open option for defining whether queues can be shared (parameter identifier: **MQIA_DEF_INPUT_OPEN_OPTION**).

The value can be:

MQOO_INPUT_EXCLUSIVE
Open queue to get messages with exclusive access.

MQOO_INPUT_SHARED
Open queue to get messages with shared access.

DefPersistence (MQCFIN)
Default persistence (parameter identifier: **MQIA_DEF_PERSISTENCE**).

The value can be any of the following values:

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

DefPriority (MQCFIN)
Default priority (parameter identifier: **MQIA_DEF_PRIORITY**).

HardenGetBackout (MQCFIN)
Whether to harden backout (parameter identifier: **MQIA_HARDEN_GET_BACKOUT**).

The value can be any of the following values:

MQQA_BACKOUT_HARDENED
Backout count remembered.

MQQA_BACKOUT_NOT_HARDENED
Backout count may not be remembered.

IndexType (MQCFIN)
Index type (parameter identifier: **MQIA_INDEX_TYPE**).

InhibitGet (MQCFIN)
Whether get operations are allowed (parameter identifier: **MQIA_INHIBIT_GET**).

The value can be any of the following values:

MQQA_GET_ALLOWED
Get operations are allowed.

MQQA_GET_INHIBITED
Get operations are inhibited.

InhibitPut (MQCFIN)
Whether put operations are allowed (parameter identifier: **MQIA_INHIBIT_PUT**).

The value can be any of the following values:

MQQA_PUT_ALLOWED
Put operations are allowed.

MQQA_PUT_INHIBITED
Put operations are inhibited.

InitiationQName (MQCFST)
Initiation queue name (parameter identifier: **MQCA_INITIATION_Q_NAME**).

The maximum length of the string is MQ_Q_NAME_LENGTH.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: **MQIA_MAX_MSG_LENGTH**).

MaxQDepth (MQCFIN)
Maximum queue depth (parameter identifier: **MQIA_MAX_Q_DEPTH**).

MsgDeliverySequence (MQCFIN)
Whether priority is relevant (parameter identifier: **MQIA_MSG_DELIVERY_SEQUENCE**).

The value can be any of the following values:

MQMDS_PRIORITY
Messages are returned in priority order.

MQMDS_FIFO
Messages are returned in FIFO order (first in, first out).

ProcessName (MQCFST)
Name of process definition for queue (parameter identifier: **MQCA_PROCESS_NAME**).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

QDepthHiEvent (MQCFIN)
Controls whether Queue Depth High events are generated. (parameter identifier: **MQIA_Q_DEPTH_HIGH_EVENT**).

The value can be any of the following values:

MQEVR_ENABLED
Queue depth high events are enabled.

MQEVR_DISABLED
Queue depth high events are disabled.

QDepthHighLimit (MQCFIN)
High limit for queue depth (parameter identifier: **MQIA_Q_DEPTH_HIGH_LIMIT**).

The threshold against which the queue depth is compared to generate a Queue Depth High event.

QDepthLoEvent (MQCFIN)
Controls whether Queue Depth Low events are generated. (parameter identifier: **MQIA_Q_DEPTH_LOW_EVENT**).

The value can be any of the following values:

MQEVR_ENABLED
Queue depth lowevents are enabled.

MQEVR_DISABLED
Queue depth low events are disabled.

QDepthLowLimit (MQCFIN)
Low limit for queue depth (parameter identifier: **MQIA_Q_DEPTH_LOW_LIMIT**).

The threshold against which the queue depth is compared to generate a Queue Depth Low event.

QDepthMaxEvent (MQCFIN)
Controls whether Queue Full events are generated. (parameter identifier: **MQIA_Q_DEPTH_MAX_EVENT**).

The value can be any of the following values:

MQEVR_ENABLED
Queue depth full events are enabled.

MQEVR_DISABLED
Queue depth full events are disabled.

QDesc (MQCFST)
Queue description (parameter identifier: **MQCA_Q_DESC**).

The maximum length of the string is MQ_Q_DESC_LENGTH.

QName (MQCFST)
Queue name (parameter identifier: **MQCA_Q_NAME**).

The maximum length of the string is MQ_Q_NAME_LENGTH.

QServiceInterval (MQCFIN)
Target for queue service interval (parameter identifier: **MQIA_Q_SERVICE_INTERVAL**).

The service interval used for comparison to generate Queue Service Interval High and Queue Service Interval OK events.

QType (MQCFIN)
Queue type (parameter identifier: **MQIA_Q_TYPE**).

The value can be:

MQQT_ALIAS
Alias queue definition.

MQQT_LOCAL
Local queue.

MQQT_REMOTE
Local definition of a remote queue.

MQQT_MODEL
Model queue definition.

QueueAccounting (MQCFIN)
Specifies whether accounting information is collected (parameter identifier: **MQIA_ACCOUNTING_Q**).

The value can be any of the following values:

MQMON_ON
Accounting information is collected for the queue.

MQMON_OFF
Accounting information is not collected for the queue.

MQMON_Q_MGR
The collection of accounting information for this queue is based is based on the queue manager attribute **QueueAccounting**.

QueueMonitoring (MQCFIN)
Level of monitoring data collection for the queue (parameter identifier: **MQIA_MONITORING_Q**).

The value can be any of the following values:

MQMON_OFF
Monitoring data collection is turned off.

MQMON_LOW
Monitoring data collection is turned on with a low ratio of data collection.

MQMON_MEDIUM
Monitoring data collection is turned on with a moderate ratio of data collection.

MQMON_HIGH
Monitoring data collection is turned on with a high ratio of data collection.

MQMON_Q_MGR
The level of monitoring data collected is based on the queue manager attribute **QueueMonitoring**.

RemoteQMgrName (MQCFST)
Name of remote queue manager (parameter identifier: **MQCA_REMOTE_Q_MGR_NAME**).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

RemoteQName (MQCFST)
Name of remote queue as known locally on the remote queue manager (parameter identifier: **MQCA_REMOTE_Q_NAME**).

The maximum length of the string is MQ_Q_NAME_LENGTH.

RetentionInterval (MQCFIN)
Retention interval (parameter identifier: **MQIA_RETENTION_INTERVAL**).

ServiceIntervalEvent (MQCFIN)
Controls whether Service Interval High or Service Interval OK events are generated. .

The value can be any of the following values:

MQQSIE_NONE
No service interval events are generated.

MQQSIE_OK
Service interval OK events are generated.

MQQSIE_HIGH
Service interval high events are generated.

Shareability (MQCFIN)
Whether queue can be shared (parameter identifier: **MQIA_SHAREABILITY**).

The value can be any of the following values:

MQQA_SHAREABLE
Queue is shareable.

MQQA_NOT_SHAREABLE
Queue is not shareable.

StorageClass (MQCFST)
Storage class name (parameter identifier: **MQCA_STORAGE_CLASS**).

The maximum length of the string is MQ_STORAGE_CLASS_LENGTH.

TriggerControl (MQCFIN)
Trigger control (parameter identifier: **MQIA_TRIGGER_CONTROL**).

The value can be any of the following values:

MQTC_OFF
Trigger messages not required.

MQTC_ON
Trigger messages required.

TriggerData (MQCFST)
Trigger data (parameter identifier: **MQCA_TRIGGER_DATA**).

The maximum length of the string is MQ_TRIGGER_DATA_LENGTH.

TriggerDepth (MQCFIN)
Trigger depth (parameter identifier: **MQIA_TRIGGER_DEPTH**).

TriggerMsgPriority (MQCFIN)
Threshold message priority for triggers (parameter identifier: **MQIA_TRIGGER_MSG_PRIORITY**).

TriggerType (MQCFIN)
Trigger type (parameter identifier: **MQIA_TRIGGER_TYPE**).

The value can be:

MQTT_NONE
No trigger messages.

MQTT_FIRST
Trigger message when queue depth goes from 0 to 1.

MQTT_EVERY
Trigger message for every message.

MQTT_DEPTH
Trigger message when depth threshold exceeded.

Usage (MQCFIN)
Usage (parameter identifier: **MQIA_USAGE**).

The value can be any of the following values:

MQUS_NORMAL
Normal usage.

MQUS_TRANSMISSION
Transmission queue.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: **MQCA_XMIT_Q_NAME**).

The maximum length of the string is MQ_Q_NAME_LENGTH.

Queue manager attributes

Event messages relating to objects can include queue manager attributes.

Multi **AccountingConnOverride (MQCFIN)**

Specifies whether applications can override the settings of the **QueueAccounting** and **MQIAccounting** queue manager parameters (parameter identifier: **MQIA_ACCOUNTING_CONN_OVERRIDE**).

The value can be any of the following values:

MQMON_DISABLED
Applications cannot override the settings of the **QueueAccounting** and **MQIAccounting** parameters.

This value is the initial default value for the queue manager.

MQMON_ENABLED
Applications can override the settings of the **QueueAccounting** and **MQIAccounting** parameters by using the options field of the MQCNO structure of the MQCONNX API call.

This parameter is valid only on IBM i, UNIX, Linux, and Windows.

Multi **AccountingInterval (MQCFIN)**

The time interval, in seconds, at which intermediate accounting records are written (parameter identifier: **MQIA_ACCOUNTING_INTERVAL**).

Specify a value in the range 1 - 604,000.

This parameter is valid only on IBM i, UNIX, Linux, and Windows.

ActivityRecording (MQCFIN)

Specifies whether activity recording is enabled or disabled (parameter identifier: **MQIA_ACTIVITY_RECORDING**).

The value can be any of the following values:

MQRECORDING_MSG
Activity recording is enabled. Activity reports are delivered to the reply-to queue specified in the message descriptor of the message.

MQRECORDING_Q
Activity recording is enabled. Activity reports are delivered to a fixed name queue.

MQRECORDING_DISABLED
Activity recording is disabled.

AdoptNewMCACheck (MQCFIN)

Procedure to determine if an existing receiver MCA is to be adopted when an inbound channel is detected of the same name (parameter identifier: **MQIA_ADOPTNEWMCA_CHECK**).

The value can be any of the following values:

MQADOPT_CHECK_Q_MGR_NAME
Compare the receiver MCA and the inbound channel. If the queue manager names match, the existing receiver MCA is adopted providing it is active. If they don't match, the existing receiver MCA is canceled, and a new MCA is created.

MQADOPT_CHECK_NET_ADDR
Compare the receiver MCA and the inbound channel. If the network addresses match, the existing receiver MCA is adopted providing it is active. If they don't match, the existing receiver MCA is canceled, and a new MCA is created.

MQADOPT_CHECK_ALL
Compare the receiver MCA and the inbound channel. If both the queue manager names, and

the network addresses match, the existing receiver MCA is adopted providing it is active. If they don't match, the existing receiver MCA is canceled, and a new MCA is created.

MQADOPT_CHECK_NONE
If the existing receiver MCA is active it is adopted with no checks.

AdoptNewMCAType (MQCFIN)
Specifies whether orphaned receiver MCAs are to be restarted when an inbound channel matching the **AdoptNewMCACheck** procedure is detected (parameter identifier: **MQIA_ADOPTNEWMCA_TYPE**).

The value can be:

MQADOPT_TYPE_NO
Do not restart and adopt orphaned receiver MCAs.

MQADOPT_TYPE_ALL
Restart and adopt orphaned receiver MCAs.

AlterationDate (MQCFST)
Alteration date (parameter identifier: **MQCA_ALTERATION_DATE**).

The date when the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: **MQCA_ALTERATION_TIME**).

The time when the information was last altered.

AuthorityEvent (MQCFIN)
Controls whether authorization (Not Authorized) events are generated (parameter identifier: **MQIA_AUTHORITY_EVENT**).

The value can be any of the following values:

MQEVR_ENABLED
Authorization event reporting enabled.

MQEVR_DISABLED
Authorization event reporting disabled.

BridgeEvent (MQCFIN)
Determines whether IMS bridge events are generated (parameter identifier: **MQIA_BRIDGE_EVENT**).

The value can be any of the following values:

MQEVR_ENABLED
All IMS bridge events are enabled.

MQEVR_DISABLED
All IMS bridge events are disabled.

CertificateLabel (MQCFST)
Specifies the certificate label for this queue manager to use. The label identifies which personal certificate in the key repository has been selected (parameter identifier: **MQCA_CERT_LABEL**).

ULW **CertificateValPolicy (MQCFIN)**
Specifies which TLS certificate validation policy is used to validate digital certificates received from remote partner systems (parameter identifier: **MQIA_CERT_VAL_POLICY**).

This attribute can be used to control how strictly the certificate chain validation conforms to industry security standards. For more information, see Certificate validation policies in IBM MQ.

The value can be any of the following values:

MQ_CERT_VAL_POLICY_ANY
Apply each of the certificate validation policies supported by the secure sockets library and accept the certificate chain if any of the policies considers the certificate chain valid. This

setting can be used for maximum backwards compatibility with older digital certificates which do not comply with the modern certificate standards.

MQ_CERT_VAL_POLICY_RFC5280
Apply only the RFC 5280 compliant certificate validation policy. This setting provides stricter validation than the ANY setting, but rejects some older digital certificates.

This parameter is only valid on UNIX, Linux, and Windows and can be used only on a queue manager with a command level of 711, or higher.

Changes to **CertificateValPolicy** become effective in the following cases:

- When a new channel process is started.
- For channels that run as threads of the channel initiator, when the channel initiator is restarted.
- For channels that run as threads of the listener, when the listener is restarted.
- For channels that run as threads of a process pooling process, when the process pooling process is started or restarted and first runs a TLS channel. If the process pooling process has already run a TLS channel, and you want the change to become effective immediately, run the MQSC command **REFRESH SECURITY TYPE(SSL)**. The process pooling process is **amqrmppa** on UNIX, Linux, and Windows.
- When a **REFRESH SECURITY TYPE(SSL)** command is issued.

z/OS **CFConlos (MQCFIN)**
Specifies the action to be taken when the queue manager loses connectivity to the administration structure, or any CF structure with **CFConlos** set to ASQMGR (parameter identifier: **MQIA_QMGR_CFCONLOS**).

The value can be:

MQCFCONLOS_TERMINATE
The queue manager terminates when connectivity to CF structures is lost.

MQCFCONLOS_TOLERATE
The queue manager tolerates loss of connectivity to CF structures without terminating.

This parameter applies to z/OS only.

You can select MQCFCONLOS_TOLERATE only if all the queue managers in the queue-sharing group are at command level 710 or greater and have **OPMODE** set to NEWFUNC.

Multi **ChannelAutoDef (MQCFIN)**
Controls whether receiver and server-connection channels can be auto-defined (parameter identifier: **MQIA_CHANNEL_AUTO_DEF**).

Auto-definition for cluster-sender channels is always enabled.

This parameter is valid only on IBM i, UNIX, Linux, and Windows.

The value can be:

MQCHAD_DISABLED
Channel auto-definition disabled.

MQCHAD_ENABLED
Channel auto-definition enabled.

Multi **ChannelAutoDefEvent (MQCFIN)**
Controls whether channel auto-definition events are generated (parameter identifier: **MQIA_CHANNEL_AUTO_DEF_EVENT**), when a receiver, server-connection, or cluster-sender channel is auto-defined.

This parameter is valid only on IBM i, UNIX, Linux, and Windows.

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

ChannelAuthenticationRecords (MQCFIN)
Controls whether channel authentication records are used (parameter identifier: **MQIA_CHLAUTH_RECORDS**).

Channel authentication records can be set and displayed regardless of the value of this attribute.

The value can be any of the following values:

MQCHLA_DISABLED
Channel authentication records are not checked.

MQCHLA_ENABLED
Channel authentication records are checked.

ChannelAutoDefEvent (MQCFIN)
Determines whether channel auto-definition events are generated (parameter identifier: **MQIA_CHANNEL_AUTO_DEF_EVENT**).

The value can be any of the following values:

MQEVR_ENABLED
All channel auto-definition events are enabled.

MQEVR_DISABLED
All channel auto-definition events are disabled.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

ChannelAutoDefExit (MQCFST)
Channel auto-definition exit name (parameter identifier: **MQCA_CHANNEL_AUTO_DEF_EXIT**).

The maximum length of the exit name is **MQ_EXIT_NAME_LENGTH**.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

ChannelAuthenticationRecords (MQCFIN)
Controls whether channel authentication records are used. Channel authentication records can still be set and displayed regardless of the value of this attribute. (parameter identifier: **MQIA_CHLAUTH_RECORDS**).

The value can be:

MQCHLA_DISABLED
Channel authentication records are not checked.

MQCHLA_ENABLED
Channel authentication records are checked.

ChannelEvent (MQCFIN)
Determines whether channel events are generated (parameter identifier: **MQIA_CHANNEL_EVENT**).

The value can be any of the following values:

MQEVR_ENABLED
All channel events are enabled.

MQEVR_EXCEPTION
Only the following channels events are enabled:

- MQRC_CHANNEL_ACTIVATED

- MQRC_CHANNEL_CONV_ERROR
- MQRC_CHANNEL_NOT_ACTIVATED
- MQRC_CHANNEL_STOPPED

MQEVR_DISABLED
All channel events are disabled.

Multi **ChannelInitiatorControl (MQCFIN)**
Specifies whether the channel initiator is to be started when the queue manager starts (parameter identifier: **MQIA_CHINIT_CONTROL**).

The value can be:

MQSVC_CONTROL_MANUAL
The channel initiator is not to be started automatically.

MQSVC_CONTROL_Q_MGR
The channel initiator is to be started automatically when the queue manager starts.

This parameter is valid only on IBM i, UNIX, Linux, and Windows.

ChannelMonitoring (MQCFIN)
Level of real-time monitoring data collection for channels (parameter identifier: **MQIA_MONITORING_CHANNEL**).

The value can be any of the following values:

MQMON_NONE
Monitoring data collection is disabled, regardless of the setting for the **ChannelMonitoring** channel attribute.

MQMON_OFF
Monitoring data collection is turned off for channels specifying MQMON_Q_MGR in the **ChannelMonitoring** channel attribute.

MQMON_LOW
Monitoring data collection is turned on with a low ratio of data collection for channels specifying MQMON_Q_MGR in the **ChannelMonitoring** channel attribute.

MQMON_MEDIUM
Monitoring data collection is turned on with a moderate ratio of data collection for channels specifying MQMON_Q_MGR in the **ChannelMonitoring** channel attribute.

MQMON_HIGH
Monitoring data collection is turned on with a high ratio of data collection for channels specifying MQMON_Q_MGR in the **ChannelMonitoring** channel attribute.

ChannelStatistics(MQCFIN)
Controls whether statistics data is to be collected for channels (parameter identifier: **MQIA_STATISTICS_CHANNEL**).

The value can be:

MQMON_NONE
Statistics data collection is turned off for channels regardless of the setting of their **ChannelStatistics** parameter. This value is the initial default value of the queue manager.

MQMON_OFF
Statistics data collection is turned off for channels specifying a value of MQMON_Q_MGR in their **ChannelStatistics** parameter.

MQMON_LOW
Statistics data collection is turned on, with a low ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their **ChannelStatistics** parameter.

MQMON_MEDIUM
Statistics data collection is turned on, with a moderate ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their **ChannelStatistics** parameter.

MQMON_HIGH
Statistics data collection is turned on, with a high ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their **ChannelStatistics** parameter.

z/OS On z/OS systems, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying LOW, MEDIUM, or HIGH makes no difference to your results.

ChinitAdapters (MQCFIN)
Number of channel initiator adapter subtasks to use for processing IBM MQ calls (parameter identifier: **MQIA_CHINIT_ADAPTERS**).

This value must be in the range 0 through 9999.

ChinitDispatchers (MQCFIN)
Number of dispatchers to use for the channel initiator (parameter identifier: **MQIA_CHINIT_DISPATCHERS**).

ChinitServiceParm (MQCFST)
This attribute is reserved for use by IBM (parameter identifier: **MQCA_CHINIT_SERVICE_PARM**).

ChinitTraceAutoStart (MQCFIN)
Specifies whether the channel initiator trace should start automatically (parameter identifier: **MQIA_CHINIT_TRACE_AUTO_START**).

The value can be:

MQTRAXSTR_YES
Channel initiator trace starts automatically.

MQTRAXSTR_NO
Channel initiator trace does not starts automatically.

ChinitTraceTableSize (MQCFIN)
Size of the channel initiator's trace data space, in MB (parameter identifier: **MQIA_CHINIT_TRACE_TABLE_SIZE**).

ClusterSenderMonitoring (MQCFIN)
Level of real-time monitoring data collection for auto-defined cluster sender channels (parameter identifier: **MQIA_MONITORING_AUTO_CLUSSDR**).

This parameter can have any of the following values:

MQMON_Q_MGR
The collection of monitoring data is inherited from the setting of the **ChannelMonitoring** attribute in the queue manager object.

MQMON_OFF
Monitoring data collection is disabled.

MQMON_LOW
Monitoring data collection is turned on with a low ratio of data collection.

MQMON_MEDIUM
Monitoring data collection is turned on with a moderate ratio of data collection.

MQMON_HIGH
Monitoring data collection is turned on with a high ratio of data collection.

ClusterSenderStatistics (MQCFIN)
Controls whether statistics data is to be collected for auto-defined cluster-sender channels (parameter identifier: **MQIA_STATISTICS_AUTO_CLUSSDR**).

The value can be:

MQMON_Q_MGR
Collection of statistics data is inherited from the setting of the queue manager's **ChannelStatistics** parameter. This value is the initial default value of the queue manager.

MQMON_OFF
Statistics data collection for the channel is disabled.

MQMON_LOW
Unless **ChannelStatistics** is MQMON_NONE, this value specifies a low rate of data collection with a minimal effect on system performance.

MQMON_MEDIUM
Unless **ChannelStatistics** is MQMON_NONE, this value specifies a moderate rate of data collection.

MQMON_HIGH
Unless **ChannelStatistics** is MQMON_NONE, this value specifies a high rate of data collection.

z/OS On z/OS systems, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying LOW, MEDIUM, or HIGH makes no difference to your results.

ClusterWorkLoadData (MQCFST)
Data passed to the cluster workload exit (parameter identifier: **MQCA_CLUSTER_WORKLOAD_DATA**).

ClusterWorkLoadExit (MQCFST)
Name of the cluster workload exit (parameter identifier: **MQCA_CLUSTER_WORKLOAD_EXIT**).

The maximum length of the exit name is MQ_EXIT_NAME_LENGTH.

ClusterWorkLoadLength (MQCFIN)
Cluster workload length (parameter identifier: **MQIA_CLUSTER_WORKLOAD_LENGTH**).

The maximum length of the message passed to the cluster workload exit.

CLWLMRUChannels (MQCFIN)
Maximum number of most recently used channels for cluster workload balancing (parameter identifier: **MQIA_CLWL_MRU_CHANNELS**).

CLWLUseQ (MQCFIN)
This defines the behavior of an MQPUT when the target queue has both a local instance and at least one remote cluster instance (parameter identifier: **MQIA_CLWL_USEQ**).

This parameter can have any of the following values:

MQCLWL_USEQ_ANY
Use remote and local queues.

MQCLWL_USEQ_LOCAL
Do not use remote queues.

CodedCharSetId (MQCFIN)
Coded character set identifier (parameter identifier: **MQIA_CODED_CHAR_SET_ID**).

CommandEvent (MQCFIN)
Controls whether command events are generated (parameter identifier: **MQIA_COMMAND_EVENT**).

The value can be any of the following values:

MQEVR_DISABLED
Command event generation disabled.

MQEVR_ENABLED
Command event generation enabled.

MQEVR_NO_DISPLAY
Command events are generated for all commands other than **MQSC DISPLAY** commands and PCF **Inquire** commands.

CommandEvent (MQCFIN)
Controls whether command events are generated (parameter identifier: **MQIA_COMMAND_EVENT**).

The value can be any of the following values:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

MQEVR_NO_DISPLAY
Event reporting enabled for all successful commands except Inquire commands.

CommandInputQName (MQCFST)
Command input queue name (parameter identifier: **MQCA_COMMAND_INPUT_Q_NAME**).

The maximum length of the string is MQ_Q_NAME_LENGTH.

CommandLevel (MQCFIN)
Command level supported by queue manager (parameter identifier: **MQIA_COMMAND_LEVEL**).

z/OS **CommandScope (MQCFIN)**
Command scope (parameter identifier: **MQCACF_COMMAND_SCOPE**). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following values:

- Blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- A queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue-sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment. The command server must be enabled.
- An asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Multi **CommandServerControl (MQCFIN)**
Specifies whether the command server is to be started when the queue manager starts (parameter identifier: **MQIA_CMD_SERVER_CONTROL**).

The value can be:

MQSVC_CONTROL_MANUAL
The command server is not to be started automatically.

MQSVC_CONTROL_Q_MGR
The command server is to be started automatically when the queue manager starts.

This parameter is valid only on IBM i, UNIX, Linux, and Windows.

ConfigurationEvent (MQCFIN)
Controls whether configuration events are generated (parameter identifier: **MQIA_CONFIGURATION_EVENT**).

The value can be any of the following values:

MQEVR_DISABLED
Configuration event generation disabled.

MQEVR_ENABLED
Configuration event generation enabled.

ConnAuth (MQCFST)
The name of an authentication information object that is used to provide the location of user ID and password authentication (parameter identifier: **MQCA_CONN_AUTH**).

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH. Only authentication information objects with type IDPWOS or IDPWLDAP can be specified; other types result in an error message when the configuration is read by:

- 

The OAM on UNIX, Linux, and Windows.
- 

The security component on z/OS

Custom (MQCFST)
Custom attribute for new features (parameter identifier: **MQCA_CUSTOM**).

This attribute is reserved for the configuration of new features before separate attributes are introduced. It can contain the values of zero or more attributes as pairs of attribute name and value, separated by at least one space. The attribute name-value pairs have the form NAME(VALUE). Single quotation marks must be escaped with another single quotation mark.

This description is updated when features using this attribute are introduced. There are no possible values for **Custom**.

The maximum length of the string is MQ_CUSTOM_LENGTH.

CPILevel (MQCFIN)
CPI level (parameter identifier: **MQIA_CPI_LEVEL**).

DeadLetterQName (MQCFST)
Dead letter (undelivered message) queue name (parameter identifier: **MQCA_DEAD_LETTER_Q_NAME**).

Specifies the name of the local queue that is to be used for undelivered messages. Messages are put on this queue if they cannot be routed to their correct destination.

The maximum length of the string is MQ_Q_NAME_LENGTH.

DefXmitQName (MQCFST)
Default transmission queue name (parameter identifier: **MQCA_DEF_XMIT_Q_NAME**).

This is the name of the default transmission queue that is used for the transmission of messages to remote queue managers, if there is no other indication of which transmission queue to use.

The maximum length of the string is MQ_Q_NAME_LENGTH.

DNSGroup (MQCFST)
This parameter is no longer used. From IBM MQ for z/OS Version 8.0, WLM/DNS is no longer supported by the z/OS Communications Server, so the queue manager attributes **DNSWLM** and **DNSGROUP** are no longer used. (parameter identifier: **MQCA_DNS_GROUP**).

The maximum length of this name is MQ_DNS_GROUP_NAME_LENGTH.

DNSWLM (MQCFIN)
This parameter is no longer used. From IBM MQ for z/OS Version 8.0, WLM/DNS is no longer supported by the z/OS Communications Server, so the queue manager attributes **DNSWLM** and **DNSGROUP** are no longer used. (parameter identifier: **MQIA_DNS_WLM**).

The value can be any of the following values:

MQDNSWLM_YES
This value may be seen on a queue manager migrated from an earlier release. The value is ignored.

MQDNSWLM_NO
This is the only value supported by the queue manager.

EncryptionPolicySuiteB (MQCFIL)
Specifies whether Suite B-compliant cryptography is used and what level of strength is employed (parameter identifier **MQIA_SUITE_B_STRENGTH**).

The value can be one or more of:

MQ_SUITE_B_NONE
Suite B-compliant cryptography is not used.

MQ_SUITE_B_128_BIT
Suite B 128-bit strength security is used.

MQ_SUITE_B_192_BIT
Suite B 192-bit strength security is used.

If invalid lists are specified, such as MQ_SUITE_B_NONE with MQ_SUITE_B_128_BIT, the error MQRCCF_SUITE_B_ERROR is issued.

ExpiryInterval (MQCFIN)
Expiry interval (parameter identifier: **MQIA_EXPIRY_INTERVAL**).

Force (MQCFIN)
Force changes (parameter identifier: **MQIACF_FORCE**).

Specifies whether the command is forced to complete if both of the following are true:

- **DefXmitQName** is specified, and
- An application has a remote queue open, the resolution for which is affected by this change.

GroupUR (MQCFIN)
Controls whether XA client applications can establish transactions with a GROUP unit of recovery disposition (parameter identifier: **MQIA_GROUP_UR**).

The value can be any of the following values:

MQGUR_DISABLED
XA client applications must connect using a queue manager name.

MQGUR_ENABLED
XA client applications can establish transactions with a group unit of recovery disposition by specifying a queue-sharing group name when they connect.

z/OS **IGQPutAuthority (MQCFIN)**
IGQ put authority (parameter identifier: **MQIA_IGQ_PUT_AUTHORITY**).

z/OS **IGQUserId (MQCFST)**
Intra-group queuing agent user identifier (parameter identifier: **MQCA_IGQ_USER_ID**). This parameter is valid only on z/OS when the queue manager is a member of a queue-sharing group.

Specifies the user identifier that is associated with the local intra-group queuing agent. This identifier is one of the user identifiers that might be checked for authorization when the IGQ agent puts messages on local queues. The actual user identifiers checked depend on the setting of the **IGQPutAuthority** attribute, and on external security options.

The maximum length is MQ_USER_ID_LENGTH.

V 9.0.2 Multi **ImageInterval (MQCFIN)**
The target frequency with which the queue manager automatically writes media images, in minutes since the previous media image for an object (parameter identifier: **MQIA_MEDIA_IMAGE_INTERVAL**). This parameter is not valid on z/OS.

The value can be:

Time interval
The time in minutes from 1 - 999 999 999, at which the queue manager automatically writes media images.

The default value is 60 minutes.

MQMEDIMGINTVL_OFF
Automatic media images are not written on a time interval basis.

V 9.0.2 Multi **ImageLogLength (MQCFIN)**

The target size of the recovery log, written before the queue manager automatically writes media images, in number of megabytes since the previous media image for an object. This limits the amount of log to be read when recovering an object (parameter identifier: **MQIA_MEDIA_IMAGE_LOG_LENGTH**). This parameter is not valid on z/OS.

The value can be:

Target log size
The target size of the recovery log in megabytes from 1 - 999 999 999.

MQMEDIMGLOGLN_OFF
Automatic media images are not written based on the size of log written.

MQMEDIMGLOGLN_OFF is the default value.

V 9.0.2 Multi **ImageRecoverObject (MQCFST)**

Specifies whether authentication information, channel, client connection, listener, namelist, process, alias queue, remote queue, and service objects are recoverable from a media image, if linear logging is being used (parameter identifier: **MQIA_MEDIA_IMAGE_RECOVER_OBJ**). This parameter is not valid on z/OS.

The value can be:

MQIMGRCOV_NO
The rcdmqimg and rcrmqobj commands are not permitted for these objects, and automatic media images, if enabled, are not written for these objects.

MQIMGRCOV_YES
These objects are recoverable.

MQIMGRCOV_YES is the default value.

V 9.0.2 Multi **ImageRecoverObject (MQCFST)**

Specifies whether authentication information, channel, client connection, listener, namelist, process, alias queue, remote queue, and service objects are recoverable from a media image, if linear logging is being used (parameter identifier: **MQIA_MEDIA_IMAGE_RECOVER_OBJ**). This parameter is not valid on z/OS.

The value can be:

MQIMGRCOV_NO
The rcdmqimg and rcrmqobj commands are not permitted for these objects, and automatic media images, if enabled, are not written for these objects.

MQIMGRCOV_YES
These objects are recoverable.

MQIMGRCOV_YES is the default value.

V 9.0.2 Multi **ImageRecoverQueue (MQCFST)**

Specifies the default **ImageRecoverQueue** attribute for local and permanent dynamic queue objects, when used with this parameter (parameter identifier: **MQIA_MEDIA_IMAGE_RECOVER_Q**). This parameter is not valid on z/OS.

The value can be:

MQIMGRCOV_NO
The **ImageRecoverQueue** attribute for local and permanent dynamic queue objects is set to MQIMGRCOV_NO .

MQIMGRCOV_YES
The **ImageRecoverQueue** attribute for local and permanent dynamic queue objects is set to MQIMGRCOV_YES .

MQIMGRCOV_YES is the default value.

V 9.0.2 Multi **ImageSchedule (MQCFST)**
Whether the queue manager automatically writes media images (parameter identifier: **MQIA_MEDIA_IMAGE_SCHEDULING**). This parameter is not valid on z/OS.

The value can be:

MQMEDIMGSCHED_AUTO
The queue manager attempts to automatically write a media image for an object, before **ImageInterval** minutes have elapsed, or **ImageLogLength** megabytes of recovery log have been written, since the previous media image for the object was taken.

The previous media image might have been taken manually or automatically, depending on the settings of **ImageInterval** or **ImageLogLength**.

MQMEDIMGSCHED_MANUAL
Automatic media images are not written.

MQMEDIMGSCHED_MANUAL is the default value.

InhibitEvent (MQCFIN)
Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated (parameter identifier: **MQIA_INHIBIT_EVENT**).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

IntraGroupQueuing (MQCFIN)
Intra group queuing (parameter identifier: **MQIA_INTRA_GROUP_QUEUING**).

IPAddressVersion (MQCFIN)
Specifies the IP version to be used (parameter identifier: **MQIA_IP_ADDRESS_VERSION**).

The value can be any of the following values:

MQIPADDR_IPV4
The IPv4 stack is used.

MQIPADDR_IPV6
The IPv6 stack is used.

ListenerTimer (MQCFIN)
The time interval, in seconds, between attempts to restart a listener following an APPC or TCP/IP failure (parameter identifier: **MQCA_LISTENER_TIMER**).

LocalEvent (MQCFIN)
Controls whether local error events are generated (parameter identifier: **MQIA_LOCAL_EVENT**).

The value can be any of the following values:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

Multi **LoggerEvent (MQCFIN)**
Controls whether recovery log events are generated (parameter identifier: **MQIA_LOGGER_EVENT**).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled. This value is valid only on queue managers that use linear logging.

This parameter is valid only on IBM i, UNIX, Linux, and Windows.

z/OS **LUGroupName (MQCFST)**
Generic LU name for the LU 6.2 listener (parameter identifier: **MQCA_LU_GROUP_NAME**).

The generic LU name to be used by the LU 6.2 listener that handles inbound transmissions for the queue-sharing group.

This parameter applies to z/OS only.

The maximum length of the string is MQ_LU_NAME_LENGTH.

z/OS **LUName (MQCFST)**
LU name to use for outbound LU 6.2 transmissions (parameter identifier: **MQCA_LU_NAME**).

The name of the LU to use for outbound LU 6.2 transmissions. Set this parameter to be the same as the name of the LU to be used by the listener for inbound transmissions.

This parameter applies to z/OS only.

The maximum length of the string is MQ_LU_NAME_LENGTH.

LU62ARMSuffix (MQCFST)
The suffix of the SYS1.PARMLIB member APPCPMxx, that nominates the LUADD for this channel initiator (parameter identifier: MQCA_LU62_ARM_SUFFIX).

The maximum length of this name is MQ_ARM_SUFFIX_LENGTH.

LU62Channels (MQCFIN)
Maximum number of current channels that use the LU 6.2 transmission protocol, including clients connected to server connection channels (parameter identifier: **MQIA_LU62_CHANNELS**).

LUGroupName (MQCFST)
The generic LU name that the LU 6.2 listener that handles inbound transmissions for the queue-sharing group is to use. This name must be the same as **LUName** (parameter identifier: **MQCA_LU_GROUP_NAME**).

The maximum length of this name is MQ_LU_NAME_LENGTH.

LUName (MQCFST)
The LU name that the LU 6.2 listener that handles outbound transmissions is to use. This name must be the same as **LUGroupName** (parameter identifier: **MQCA_LU_NAME**).

The maximum length of this name is MQ_LU_NAME_LENGTH.

MaxActiveChannels (MQCFIN)
Maximum number of channels that can be active at the same time (parameter identifier: **MQIA_ACTIVE_CHANNELS**).

MaxChannels (MQCFIN)
Maximum number of current channels, including clients connected to server connection channels (parameter identifier: **MQIA_MAX_CHANNELS**).

MaxHandles (MQCFIN)
Maximum number of handles (parameter identifier: **MQIA_MAX_HANDLES**).

Specifies the maximum number of handles that any one job can have open at the same time.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: **MQIA_MAX_MSG_LENGTH**).

MaxPriority (MQCFIN)
Maximum priority (parameter identifier: MQIA_MAX_PRIORITY).

MaxUncommittedMsgs (MQCFIN)
Maximum number of uncommitted messages within a unit of work (parameter identifier: **MQIA_MAX_UNCOMMITTED_MSGS**).

That is:

- The number of messages that can be retrieved, plus
- The number of messages that can be put on a queue, plus
- Any trigger messages generated within this unit of work

under any one syncpoint. This limit does not apply to messages that are retrieved or put outside syncpoint.

Multi **MQIAccounting(MQCFIN**
Controls whether accounting information for MQI data is to be collected (parameter identifier: **MQIA_ACCOUNTING_MQI**).

The value can be:

MQMON_OFF
MQI accounting data collection is disabled. This value is the initial default value of the queue manager.

MQMON_ON
MQI accounting data collection is enabled.

This parameter is valid only on IBM i, UNIX, Linux, and Windows.

Multi **MQIStatistics(MQCFIN)**
Controls whether statistics monitoring data is to be collected for the queue manager (parameter identifier: **MQIA_STATISTICS_MQI**).

The value can be:

MQMON_OFF
Data collection for MQI statistics is disabled. This value is the initial default value of the queue manager.

MQMON_ON
Data collection for MQI statistics is enabled.

This parameter is valid only on IBM i, UNIX, Linux, and Windows.

MsgMarkBrowseInterval(MQCFIN)
Mark-browse interval (parameter identifier: **MQIA_MSG_MARK_BROWSE_INTERVAL**).

Specifies the time interval in milliseconds after which the queue manager can automatically unmark messages.

This parameter can either have a value in the range 0 - 999,999,999, or the special value MQMMBI_UNLIMITED.

A value of 0 causes the queue manager to unmark messages immediately.

MQMMBI_UNLIMITED indicates that the queue manager does not automatically unmark messages.

OutboundPortMax (MQCFIN)
Outbound port range maximum (parameter identifier: **MQIA_OUTBOUND_PORT_MAX**).

The upper limit for the range of port numbers used when binding outgoing channels.

OutboundPortMin (MQCFIN)
Outbound port range minimum (parameter identifier: **MQIA_OUTBOUND_PORT_MIN**).

The lower limit for the range of port numbers used when binding outgoing channels.

Parent(MQCFST)
The name of the queue manager to which this queue manager is to connect hierarchically as its child (parameter identifier: **MQCA_PARENT**).

A blank value indicates that this queue manager has no parent queue manager. If there is an existing parent queue manager it is disconnected. This value is the initial default value of the queue manager.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Note:

- The use of IBM MQ hierarchical connections requires that the queue manager attribute **PSMode** is set to MQPSM_ENABLED.
- The value of **Parent** can be set to a blank value if **PSMode** is set to MQPSM_DISABLED.
- Before connecting to a queue manager hierarchically as its child, channels in both directions must exist between the parent queue manager and child queue manager.
- If a parent is defined, the **Change Queue Manager** command disconnects from the original parent and sends a connection flow to the new parent queue manager.
- Successful completion of the command does not mean that the action completed or that it is going to complete successfully. Use the **Inquire Pub/Sub Status** command to track the status of the requested parent relationship.

PerformanceEvent (MQCFIN)
Controls whether performance-related events are generated (parameter identifier: **MQIA_PERFORMANCE_EVENT**).

The value can be any of the following values:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

Platform (MQCFIN)
Platform on which the queue manager resides (parameter identifier: **MQIA_PLATFORM**).

PubSubClus (MQCFIN)
Controls whether the queue manager participates in publish/subscribe clustering (parameter identifier: **MQIA_PUBSUB_CLUSTER**).

The value can be:

MQPSCLUS_ENABLED
The creating or receipt of clustered topic definitions and cluster subscriptions is permitted.

Note: The introduction of a clustered topic into a large IBM MQ cluster can cause a degradation in performance. This degradation occurs because all partial repositories are notified of all the other members of the cluster. Unexpected subscriptions might be created at all other nodes; for example; where **proxysub(FORCE)** is specified. Large numbers of channels might be started from a queue manager; for example, on resync after a queue manager failure.

MQPSCLUS_DISABLED
The creating or receipt of clustered topic definitions and cluster subscriptions is inhibited. The creations or receipts are recorded as warnings in the queue manager error logs.

PubSubMaxMsgRetryCount (MQCFIN)
The number of attempts to reprocess a message when processing a failed command message under sync point (parameter identifier: **MQIA_PUBSUB_MAXMSG_RETRY_COUNT**).

The value of this parameter must be a number in the range 0 to 999 999 999. The initial value is 5.

PubSubMode (MQCFIN)
Specifies whether the publish/subscribe engine and the queued publish/subscribe interface are running. The publish/subscribe engine enables applications to publish or subscribe by using the application programming interface. The publish/subscribe interface monitors the queues used the queued publish/subscribe interface (parameter identifier: **MQIA_PUBSUB_MODE**).

The value can be:

MQPSM_COMPAT
The publish/subscribe engine is running. It is therefore possible to publish or subscribe by using the application programming interface. The queued publish/subscribe interface is not running. Therefore any message that is put to the queues that are monitored by the queued publish/subscribe interface is not acted on. MQPSM_COMPAT is used for compatibility with versions of IBM Integration Bus (formerly known as WebSphere Message Broker) prior to version 7 that use this queue manager.

MQPSM_DISABLED
The publish/subscribe engine and the queued publish/subscribe interface are not running. It is therefore not possible to publish or subscribe using the application programming interface. Any publish/subscribe messages that are put to the queues that are monitored by the queued publish/subscribe interface are not acted on.

MQPSM_ENABLED
The publish/subscribe engine and the queued publish/subscribe interface are running. It is therefore possible to publish or subscribe by using the application programming interface and the queues that are monitored by the queued publish/subscribe interface. This value is the initial default value of the queue manager.

PubSubNPInputMsg (MQCFIN)
Whether to discard (or keep) an undelivered input message (parameter identifier: **MQIA_PUBSUB_NP_MSG**).

The value can be:

MQUNDELIVERED_DISCARD
Non-persistent input messages are discarded if they cannot be processed.

MQUNDELIVERED_KEEP
Non-persistent input messages are not discarded if they cannot be processed. In this situation, the queued publish/subscribe interface continues to try the process again at appropriate intervals and does not continue processing subsequent messages.

PubSubNPResponse (MQCFIN)
Controls the behavior of undelivered response messages (parameter identifier: **MQIA_PUBSUB_NP_RESP**).

The value can be:

MQUNDELIVERED_NORMAL
Non-persistent responses that cannot be placed on the reply queue are put on the dead letter queue. If they cannot be placed on the dead letter queue they are discarded.

MQUNDELIVERED_SAFE
Non-persistent responses that cannot be placed on the reply queue are put on the dead letter queue. If the response cannot be sent and cannot be placed on the dead letter queue the queued publish/subscribe interface rolls back the current operation. The operation is tried again at appropriate intervals and does not continue processing subsequent messages.

MQUNDELIVERED_DISCARD
Non-persistent responses that are not placed on the reply queue are discarded.

MQUNDELIVERED_KEEP
Non-persistent responses are not placed on the dead letter queue or discarded. Instead, the queued publish/subscribe interface backs out the current operation and then try it again at appropriate intervals.

PubSubSyncPoint (MQCFIN)
Whether only persistent (or all) messages must be processed under sync point (parameter identifier: **MQIA_PUBSUB_SYNC_PT**).

The value can be:

MQSYNCPOINT_IFPER
This value makes the queued publish/subscribe interface receive non-persistent messages outside sync point. If the interface receives a publication outside sync point, the interface forwards the publication to subscribers known to it outside sync point.

MQSYNCPOINT_YES
This value makes the queued publish/subscribe interface receive all messages under sync point.

QMgrDesc (MQCFST)
Queue manager description (parameter identifier: **MQCA_Q_MGR_DESC**).

The maximum length of the string is MQ_Q_MGR_DESC_LENGTH.

QMgrIdentifier (MQCFST)
Queue manager identifier (parameter identifier: **MQCA_Q_MGR_IDENTIFIER**).

The unique identifier of the queue manager.

QMgrName (MQCFST)
Name of local queue manager (parameter identifier: **MQCA_Q_MGR_NAME**).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QSGName (MQCFST)
Queue-sharing group name (parameter identifier: **MQCA_QSG_NAME**).

The maximum length of the string is MQ_QSG_NAME_LENGTH.

z/OS **QSGCertificateLabel (MQCFST)**
Specifies the certificate label for the queue-sharing group to use (parameter identifier: **MQCA_QSG_CERT_LABEL**).

QueueAccounting (MQCFIN)
Specifies whether accounting information is collected for queues (parameter identifier: **MQIA_ACCOUNTING_Q**).

The value can be any of the following values:

MQMON_ON
For all queues that have the queue parameter **QueueAccounting** specified as MQMON_Q_MGR, accounting information is collected.

MQMON_OFF
For all queues that have the queue parameter **QueueAccounting** specified as MQMON_Q_MGR, accounting information is not collected.

MQMON_NONE
Accounting information is not collected for queues.

QueueMonitoring (MQCFIN)
Level of real-time monitoring data collection for queues (parameter identifier: **MQIA_MONITORING_Q**).

The value can be any of the following values:

MQMON_NONE
Monitoring data collection is disabled, regardless of the setting for the **QueueMonitoring** queue attribute.

MQMON_OFF
Monitoring data collection is turned off for queues specifying MQMON_Q_MGR in the **QueueMonitoring** queue attribute.

MQMON_LOW
Monitoring data collection is turned on with a low ratio of data collection for queues specifying MQMON_Q_MGR in the **QueueMonitoring** queue attribute.

MQMON_MEDIUM
Monitoring data collection is turned on with a moderate ratio of data collection for queues specifying MQMON_Q_MGR in the **QueueMonitoring** queue attribute.

MQMON_HIGH
Monitoring data collection is turned on with a high ratio of data collection for queues specifying MQMON_Q_MGR in the **QueueMonitoring** queue attribute.

Multi **QueueStatistics (MQCFIN)**
Controls whether statistics data is to be collected for queues (parameter identifier: **MQIA_STATISTICS_Q**).

The value can be:

MQMON_NONE
Statistics data collection is turned off for queues regardless of the setting of their **QueueStatistics** parameter. This value is the initial default value of the queue manager.

MQMON_OFF
Statistics data collection is turned off for queues specifying a value of MQMON_Q_MGR in their **QueueStatistics** parameter.

MQMON_ON
Statistics data collection is turned on for queues specifying a value of MQMON_Q_MGR in their **QueueStatistics** parameter.

This parameter is valid only on IBM i, UNIX, Linux, and Windows.

ReceiveTimeout(MQCFIN)
In conjunction with **ReceiveTimeoutType** specifies how long a TCP/IP channel will wait to receive data, including heartbeats, from its partner before returning to the inactive state (parameter identifier: **MQIA_RECEIVE_TIMEOUT**).

ReceiveTimeoutMin(MQCFIN)
The minimum time, in seconds, that a TCP/IP channel will wait to receive data, including heartbeats, from its partner before returning to the inactive state (parameter identifier: **MQIA_RECEIVE_TIMEOUT_MIN**).

ReceiveTimeoutType (MQCFIN)
In conjunction with **ReceiveTimeout** specifies how long a TCP/IP channel will wait to receive data, including heartbeats, from its partner before returning to the inactive state (parameter identifier: **MQIA_RECEIVE_TIMEOUT_TYPE**).

The value can be any of the following values:

MQRCVTIME_MULTIPLY
The **ReceiveTimeout** value is a multiplier to be applied to the negotiated value of **HeartbeatInterval** to determine how long a channel will wait. This is the queue manager's initial default value.

MQRCVTIME_ADD
ReceiveTimeout is a value, in seconds, to be added to the negotiated value of **HeartbeatInterval** to determine how long a channel will wait.

MQRCVTIME_EQUAL
ReceiveTimeout is a value, in seconds, representing how long a channel will wait.

RemoteEvent (MQCFIN)
Controls whether remote error events are generated (parameter identifier: **MQIA_REMOTE_EVENT**).

The value can be any of the following values:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

RepositoryName (MQCFST)
Repository name (parameter identifier: **MQCA_REPOSITORY_NAME**).

The name of a cluster for which this queue manager is to provide a repository service.

RepositoryNamelist (MQCFST)
Repository name list (parameter identifier: **MQCA_REPOSITORY_NAMELIST**).

The name of a list of clusters for which this queue manager is to provide a repository service.

RevDns (MQCFIN)
Whether reverse lookup of the host name from a Domain Name Server is carried out. (parameter identifier: **MQIA_REVERSE_DNS_LOOKUP**).

This attribute has an effect only on channels using a transport type (TRPTYPE) of TCP.

The value can be:

MQRDNS_DISABLED
DNS host names are not reverse looked-up for the IP addresses of inbound channels. With this setting any CHLAUTH rules using host names are not matched.

MQRDNS_ENABLED
DNS host names are reverse looked-up for the IP addresses of inbound channels when this information is required. This setting is required for matching against CHLAUTH rules that contain host names, and for writing out error messages.

z/OS **SecurityCase (MQCFIN)**
Security case supported (parameter identifier: **MQIA_SECURITY_CASE**).

Specifies whether the queue manager supports security profile names in mixed case, or in uppercase only. The value is activated when a Refresh Security command is run with **SecurityType(MQSECTYPE_CLASSES)** specified. This parameter is valid only on z/OS.

The value can be:

MQSCYC_UPPER
Security profile names must be in uppercase.

MQSCYC_MIXED
Security profile names can be in uppercase or in mixed case.

z/OS **SharedQueueQueueManagerName (MQCFIN)**
Specifies how messages are put on a shared queue that specifies another queue manager from a queue-sharing group as the object queue manager (parameter identifier: **MQIA_SHARED_Q_Q_MGR_NAME**).

The value can be:

MQSQQM_USE
Messages are delivered to the object queue manager before being put on the shared queue.

MQSQQM_IGNORE
Messages are put directly on the shared queue.

SSLCRLNameList (MQCFST)
TLS CRL name list (parameter identifier: **MQCA_SSL_CRL_NAMELIST**).

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

SSLEvent (MQCFIN)
Determines whether IMS bridge events are generated (parameter identifier: **MQIA_SSL_EVENT**).

The value can be any of the following values:

MQEVR_ENABLED
All TLS events are enabled.

MQEVR_DISABLED
All TLS events are disabled.

ULW **SSLCryptoHardware(MQCFST)**
The TLS cryptographic hardware (parameter identifier: **MQCA_SSL_CRYPTO_HARDWARE**).

The length of the string is MQ_SSL_CRYPTO_HARDWARE_LENGTH.

Sets the name of the parameter string required to configure the cryptographic hardware present on the system.

This parameter is valid only on UNIX, Linux, and Windows.

All supported cryptographic hardware supports the PKCS #11 interface. Specify a string of the following format:

GSK_PKCS11=*PKCS_#11_driver_path_and_filename*;*PKCS_#11_token_label*;*PKCS_#11_token_password*;*symmetric_cipher_setting*;

The PKCS #11 driver path is an absolute path to the shared library providing support for the PKCS #11 card. The PKCS #11 driver filename is the name of the shared library. An example of the value required for the PKCS #11 driver path and filename is/usr/lib/pkcs11/PKCS11_API.so.

To access symmetric cipher operations through GSKit, specify the symmetric cipher setting parameter. The value of this parameter is either:

SYMMETRIC_CIPHER_OFF
Do not access symmetric cipher operations.

SYMMETRIC_CIPHER_ON
Access symmetric cipher operations.

If the symmetric cipher setting is not specified, this value has the same effect as specifying SYMMETRIC_CIPHER_OFF.

The maximum length of the string is 256 characters. The default value is blank.

If you specify a string in the wrong format, you get an error.

When the **SSLCryptoHardware (MQCFST)** value is changed, the cryptographic hardware parameters specified become the ones used for new TLS connection environments. The new information becomes effective:

- When a new channel process is started.
- For channels that run as threads of the channel initiator, when the channel initiator is restarted.
- For channels that run as threads of the listener, when the listener is restarted.
- When a Refresh Security command is issued to refresh the contents of the TLS key repository.

SSLEvent (MQCFIN)
Controls whether TLS events are generated (parameter identifier: **MQIA_SSL_EVENT**).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

ULW z/OS **SSLFipsRequired (MQCFIN)**
SSLFIPS specifies whether only FIPS-certified algorithms are to be used if cryptography is carried out in IBM MQ, rather than in cryptographic hardware (parameter identifier: **MQIA_SSL_FIPS_REQUIRED**).

If cryptographic hardware is configured, the cryptographic modules used are those modules provided by the hardware product. These modules might, or might not, be FIPS-certified to a particular level depending on the hardware product in use. This parameter applies to z/OS, UNIX, Linux, and Windows platforms only.

The value can be any of the following values:

MQSSL_FIPS_NO
IBM MQ provides an implementation of TLS cryptography which supplies some FIPS-certified modules on some platforms. If you set **SSLFIPSRequired** to MQSSL_FIPS_NO, any CipherSpec supported on a particular platform can be used. This value is the initial default value of the queue manager.

If the queue manager runs without using cryptographic hardware, refer to the CipherSpecs listed in Specifying CipherSpecs employing FIPS 140-2 certified cryptography:

MQSSL_FIPS_YES
Specifies that only FIPS-certified algorithms are to be used in the CipherSpecs allowed on all TLS connections from and to this queue manager.

For a listing of appropriate FIPS 140-2 certified CipherSpecs; see Specifying CipherSpecs.

Changes to **SSLFIPS** become effective in the following cases:

- ULW On UNIX, Linux, and Windows, when a new channel process is started.
- ULW For channels that run as threads of the channel initiator on UNIX, Linux, and Windows, when the channel initiator is restarted.
- ULW For channels that run as threads of the listener on UNIX, Linux, and Windows, when the listener is restarted.

- ULW For channels that run as threads of a process pooling process, when the process pooling process is started or restarted and first runs a TLS channel. If the process pooling process has already run a TLS channel, and you want the change to become effective immediately, run the MQSC command **REFRESH SECURITY TYPE(SSL)**. The process pooling process is **amqrmppa** on UNIX, Linux, and Windows.
- z/OS On z/OS, when the channel initiator is restarted.
- z/OS When a **REFRESH SECURITY TYPE(SSL)** command is issued, except on z/OS.

SSLKeyRepository (MQCFST)
TLS key repository (parameter identifier: **MQCA_SSL_KEY_REPOSITORY**).

The maximum length of the string is MQ_SSL_KEY_REPOSITORY_LENGTH.

SSLKeyResetCount (MQCFIN)
TLS key reset count (parameter identifier: **MQIA_SSL_RESET_COUNT**).

The maximum length of the string is MQ_SSL_KEY_REPOSITORY_LENGTH.

SSLTasks (MQCFIN)
TLS tasks (parameter identifier: **MQIA_SSL_TASKS**).

StartStopEvent (MQCFIN)
Controls whether start and stop events are generated (parameter identifier: **MQIA_START_STOP_EVENT**).

The value can be any of the following values:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

Multi **StatisticsInterval (MQCFIN)**
The time interval, in seconds, at which statistics monitoring data is written to the monitoring queue (parameter identifier: **MQIA_STATISTICS_INTERVAL**).

Specify a value in the range 1 - 604,000.

This parameter is valid only on IBM i, UNIX, Linux, and Windows.

SyncPoint (MQCFIN)
Syncpoint availability (parameter identifier: **MQIA_SYNCPOINT**).

TCPChannels (MQCFIN)
Maximum number of current channels that use the TCP/IP transmission protocol, including clients connected to server connection channels (parameter identifier: **MQIA_TCP_CHANNELS**).

TCPKeepAlive (MQCFIN)
Specifies whether to use the TCP KEEPALIVE facility to check whether the MCA at the opposite end of a channel is available (parameter identifier: **MQIA_TCP_KEEP_ALIVE**).

The value can be any of the following values:

MQTCPKEEP_YES
Use the TCP KEEPALIVE facility as specified in the TCP profile configuration data set.

MQTCPKEEP_NO
Do not use the TCP KEEPALIVE facility.

TCPName (MQCFST)
TCP name (parameter identifier: **MQIA_TCP_NAME**).

The name of the current TCP/IP system in use.

The maximum length of this value is MQ_TCP_NAME_LENGTH.

TCPStackType (MQCFIN)
TCP stack type (parameter identifier: **MQIA_TCP_STACK_TYPE**).

Specifies whether the channel initiator uses the TCP/IP address space specified in TCPNAME only, or whether it can bind to any selected TCP/IP address.

The value can be:

MQTCPSTACK_SINGLE
The channel initiator uses the TCP/IP address space specified in TCPNAME only.

MQTCPSTACK_MULTIPLE
The initiator can use any TCP/IP address space available to it. If no other address spaces are available, the address space specified in TCPNAME is used.

TraceRouteRecording (MQCFIN)
Specifies whether trace-route messaging is enabled or disabled (parameter identifier: **MQIA_TRACE_ROUTE_RECORDING**).

The value can be:

MQRECORDING_MSG
Trace-route messaging is enabled. Trace-route reply messages are delivered to the reply-to queue specified in the message descriptor of the message.

MQRECORDING_Q
Trace-route messaging is enabled. Trace-route reply messages are delivered to a fixed name queue.

MQRECORDING_DISABLED.
Trace-route messaging is disabled.

TreeLifeTime (MQCFIN)
The lifetime, in seconds, of non-administrative topics (parameter identifier: **MQIA_TREE_LIFE_TIME**).

Non-administrative topics are those topics created when an application publishes to, or subscribes as, a topic string that does not exist as an administrative node. When this non-administrative node no longer has any active subscriptions, this parameter determines how long the queue manager waits before removing that node. Only non-administrative topics that are in use by a durable subscription remain after the queue manager is recycled.

Specify a value in the range 0 - 604,000. A value of 0 means that non-administrative topics are not removed by the queue manager. The initial default value of the queue manager is 1800.

TriggerInterval (MQCFIN)
Trigger interval (parameter identifier: **MQIA_TRIGGER_INTERVAL**).

Specifies the trigger time interval, expressed in milliseconds, for use only with queues where `TriggerType` has a value of `MQTT_FIRST`.

Storage class attributes

Event messages relating to objects can include storage class attributes

AlterationDate (MQCFST)
Alteration date (parameter identifier: **MQCA_ALTERATION_DATE**).

The date when the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: **MQCA_ALTERATION_TIME**).

The time when the information was last altered.

PageSetId (MQCFIN)
Page set identifier (parameter identifier: **MQIA_PAGESET_ID**).

PassTicketApplication (MQCFST)
Name of the application used to authenticate IMS bridge passtickets (parameter identifier: **MQCA_PASS_TICKET_APPL**).

The maximum length of the string is MQ_PASS_TICKET_APPL_LENGTH.

StgClassDesc (MQCFST)
Storage class description (parameter identifier: **MQCA_STORAGE_CLASS_DESC**).

The maximum length of the string is MQ_STORAGE_CLASS_DESC_LENGTH.

XCFGroupName (MQCFST)
XCF group name (parameter identifier: **MQCA_XCF_GROUP_NAME**).

The maximum length of the string is MQ_XCF_GROUP_NAME_LENGTH.

XCFMemberName (MQCFST)
XCF member name (parameter identifier: MQCA_XCF_MEMBER_NAME).

The maximum length of the string is **MQ_XCF_MEMBER_NAME_LENGTH**.

Topic attributes

Event messages relating to objects can include topic attributes

AlterationDate (MQCFST)
Alteration date (parameter identifier: **MQCA_ALTERATION_DATE**).

The date when the information was last altered, in the form *yyyy-mm-dd*.

AlterationTime (MQCFST)
Alteration time (parameter identifier: **MQCA_ALTERATION_TIME**).

The time when the information was last altered, in the form *hh.mm.ss*.

ClusterName (MQCFST)
The name of the cluster to which this topic belongs. (parameter identifier: **MQCA_CLUSTER_NAME**).

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH. Setting this parameter to a cluster that this queue manager is a member of makes all queue managers in the cluster aware of this topic. Any publication to this topic or a topic string below it put to any queue manager in the cluster is propagated to subscriptions on any other queue manager in the cluster. For more details, see Distributed publish/subscribe networks.

The value can be any of the following values:

Blank If no topic object above this topic in the topic tree has set this parameter to a cluster name, then this topic does not belong to a cluster. Publications and subscriptions for this topic are not propagated to publish/subscribe cluster-connected queue managers. If a topic node higher in the topic tree has a cluster name set, publications and subscriptions to this topic are also propagated throughout the cluster.

This value is the default value for this parameter if no value is specified.

String The topic belongs to this cluster. It is not recommended that this is set to a different cluster from a topic object above this topic object in the topic tree. Other queue managers in the cluster will honor this object's definition unless a local definition of the same name exists on those queue managers.

Additionally, if **PublicationScope** or **SubscriptionScope** are set to MQSCOPE_ALL, this value is the cluster to be used for the propagation of publications and subscriptions, for this topic, to publish/subscribe cluster-connected queue managers.

DefPersistence (MQCFIN)
Default persistence (parameter identifier: **MQIA_TOPIC_DEF_PERSISTENCE**).

The value can be:

MQPER_PERSISTENCE_AS_PARENT
The default persistence is based on the setting of the closest parent administrative topic object in the topic tree.

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

DefPriority (MQCFIN)
Default priority (parameter identifier: **MQIA_DEF_PRIORITY**).

DefPutResponse (MQCFIN)
Default put response (parameter identifier: **MQIA_DEF_PUT_RESPONSE_TYPE**).

The value can be:

MQPRT_ASYNC_RESPONSE
The put operation is issued asynchronously, returning a subset of MQMD fields.

MQPRT_RESPONSE_AS_PARENT
The default put response is based on the setting of the closest parent administrative topic object in the topic tree.

MQPRT_SYNC_RESPONSE
The put operation is issued synchronously, returning a response.

DurableModelQName (MQCFST)
Name of the model queue to be used for durable managed subscriptions (parameter identifier: **MQCA_MODEL_DURABLE_Q**).

The maximum length of the string is MQ_Q_NAME_LENGTH.

DurableSubscriptions (MQCFIN)
Whether applications are permitted to make durable subscriptions (parameter identifier: **MQIA_DURABLE_SUB**).

The value can be:

MQSUB_DURABLE_AS_PARENT
Whether durable subscriptions are permitted is based on the setting of the closest parent administrative topic object in the topic tree.

MQSUB_DURABLE
Durable subscriptions are permitted.

MQSUB_NON_DURABLE
Durable subscriptions are not permitted.

InhibitPublications (MQCFIN)
Whether publications are allowed for this topic (parameter identifier: **MQIA_INHIBIT_PUB**).

The value can be:

MQTA_PUB_AS_PARENT
Whether messages can be published to this topic is based on the setting of the closest parent administrative topic object in the topic tree.

MQTA_PUB_INHIBITED
Publications are inhibited for this topic.

MQTA_PUB_ALLOWED
Publications are allowed for this topic.

InhibitSubscriptions (MQCFIN)
Whether subscriptions are allowed for this topic (parameter identifier: **MQIA_INHIBIT_SUB**).

The value can be:

MQTA_SUB_AS_PARENT
Whether applications can subscribe to this topic is based on the setting of the closest parent administrative topic object in the topic tree.

MQTA_SUB_INHIBITED
Subscriptions are inhibited for this topic.

MQTA_SUB_ALLOWED
Subscriptions are allowed for this topic.

NonDurableModelQName (MQCFST)
Name of the model queue to be used for non durable managed subscriptions (parameter identifier: **MQCA_MODEL_NON_DURABLE_Q**).

The maximum length of the string is MQ_Q_NAME_LENGTH.

NonPersistentMsgDelivery (MQCFIN)
The delivery mechanism for non-persistent messages published to this topic (parameter identifier: **MQIA_NPM_DELIVERY**).

The value can be:

MQDLV_AS_PARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

MQDLV_ALL
Non-persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_DUR
Non-persistent messages must be delivered to all durable subscribers. Failure to deliver a non-persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_AVAIL
Non-persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

PersistentMsgDelivery (MQCFIN)
The delivery mechanism for persistent messages published to this topic (parameter identifier: **MQIA_PM_DELIVERY**).

The value can be:

MQDLV_AS_PARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

MQDLV_ALL
Persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_DUR
Persistent messages must be delivered to all durable subscribers. Failure to deliver a persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_AVAIL
Persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

ProxySubscriptions (MQCFIN)
Whether a proxy subscription is to be sent for this topic, even if no local subscriptions exist, to directly connected queue managers (parameter identifier: **MQIA_PROXY_SUB**).

The value can be:

MQTA_PROXY_SUB_FORCE
A proxy subscription is sent to connected queue managers even if no local subscriptions exist.

MQTA_PROXY_SUB_FIRSTUSE
A proxy subscription is sent for this topic only when a local subscription exists.

PublicationScope (MQCFIN)
Whether this queue manager propagates publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: **MQIA_PUB_SCOPE**).

The value can be:

MQSCOPE_ALL
Publications for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

MQSCOPE_AS_PARENT
Whether this queue manager will propagate publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

This is the default value for this parameter if no value is specified.

MQSCOPE_QMGR
Publications for this topic are not propagated to other queue managers.

Note: You can override this behavior on a publication-by-publication basis, using MQPMO_SCOPE_QMGR on the Put Message Options.

QMgrName (MQCFST)
Name of local queue manager (parameter identifier: **MQCA_CLUSTER_Q_MGR_NAME**).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

SubscriptionScope (MQCFIN)
Whether this queue manager propagates subscriptions to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: **MQIA_SUB_SCOPE**).

The value can be:

MQSCOPE_ALL
Subscriptions for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

MQSCOPE_AS_PARENT
Whether this queue manager will propagate subscriptions to queue managers as part of a hierarchy or as part of a publish/subscribe cluster is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

This is the default value for this parameter if no value is specified.

MQSCOPE_QMGR
Subscriptions for this topic are not propagated to other queue managers.

Note: You can override this behavior on a subscription-by-subscription basis, using MQSO_SCOPE_QMGR on the Subscription Descriptor or **SUBSCOPE(QMGR)** on **DEFINE SUB**.

TopicDesc (MQCFST)
Topic description (parameter identifier: **MQCA_TOPIC_DESC**).

The maximum length is MQ_TOPIC_DESC_LENGTH.

TopicName (MQCFST)
Topic object name (parameter identifier: **MQCA_TOPIC_NAME**).

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

TopicString (MQCFST)
The topic string (parameter identifier: **MQCA_TOPIC_STRING**).

The '/' character within this string has special meaning. It delimits the elements in the topic tree. A topic string can start with the '/' character but is not required to. A string starting with the '/' character is not the same as the string which starts without the '/' character. A topic string cannot end with the "/" character.

The maximum length of the string is MQ_TOPIC_STR_LENGTH.

TopicType (MQCFIN)
Whether this object is a local or cluster topic (parameter identifier: **MQIA_TOPIC_TYPE**).

The value can be:

MQTOPT_LOCAL
This object is a local topic.

MQTOPT_CLUSTER
This object is a cluster topic.

WildcardOperation (MQCFIN)
Behavior of subscriptions including wildcards made to this topic (parameter identifier: **MQIA_WILDCARD_OPERATION**).

The value can be any of the following values:

MQTA_PASSTHRU
Subscriptions made using wildcard topic names that are less specific than the topic string at this topic object will receive publications made to this topic and to topic strings more specific than this topic. This is the default supplied with IBM MQ.

MQTA_BLOCK
Subscriptions made using wildcard topic names that are less specific than the topic string at this topic object will not receive publications made to this topic or to topic strings more specific than this topic.

Event message reference

Use this page to obtain an overview of information about the format of event messages.

For each instrumentation event, information is returned in both the message descriptor and message data parts of the events messages.

Related concepts:

"Event message descriptions" on page 4208
The event message data contains information specific to the event that was generated. This data includes the name of the queue manager and, where appropriate, the name of the queue.

Related reference:

"Event message format"
Event messages are standard IBM MQ messages containing a message descriptor and message data.

"Event message MQMD (message descriptor)" on page 4201
The message descriptor for an event message contains information that a system monitoring application can use, such as the message type and format, and the date and time that the message was put on the event queue.

"Event message MQCFH (PCF header)" on page 4205
The message data in event messages is in programmable command format (PCF), as used in PCF command inquiries and responses. The message data consists of two parts: the event header and the event data.

Related information:

Instrumentation events

Event message format

Event messages are standard IBM MQ messages containing a message descriptor and message data.

Table 420 on page 4200 shows the basic structure of event messages and, in the Event data column, the names of the fields in an event message for queue service interval events.

Table 420. Event message structure for queue service interval events

Message descriptor	**Message data**	
MQMD structure	**PCF header** **MQCFH structure**	**Event data** [1]
Structure identifier Structure version Report options Message type Expiration time Feedback code Encoding Coded character set ID Message format Message priority Persistence Message identifier Correlation identifier Backout count Reply-to queue Reply-to queue manager User identifier Accounting token Application identity data Application type Application name Put date Put time Application origin data Group identifier Message sequence number Offset Message flags Original length	Structure type Structure length Structure version Command identifier Message sequence number Control options Completion code Reason code Parameter count	Queue manager name Queue name Time since last reset Maximum number of messages on queue Number of messages put to queue Number of messages retrieved from queue

Note:

1. The parameters shown are those returned for a queue service interval event. The actual event data depends on the specific event.

In general, you need only a subset of this information for any system management programs that you write. For example, your application might need the following data:

- The name of the application causing the event
- The name of the queue manager on which the event occurred
- The queue on which the event was generated
- The event statistics

Event message MQMD (message descriptor)

The message descriptor for an event message contains information that a system monitoring application can use, such as the message type and format, and the date and time that the message was put on the event queue.

The information in the descriptor informs a system management application that the message type is MQMT_DATAGRAM, and the message format is MQFMT_EVENT.

Many of the fields in an event message contain fixed data, which is supplied by the queue manager that generated the message. The MQMD also specifies the name of the queue manager (truncated to 28 characters) that put the message.

For an event message, the MQMD structure contains the following values:

StrucId

Description: Structure identifier.
Data type: MQCHAR4.
Value: MQMD_STRUC_ID

Version

Description: Structure version number.
Data type: MQLONG.
Values:

MQMD_VERSION_1
Version-1 message descriptor structure, supported in all environments.

MQMD_VERSION_2
Version-2 message descriptor structure, supported on AIX, HP-UX, z/OS, IBM i, Solaris, Linux, Windows, and all IBM MQ MQI clients connected to these systems.

Report

Description: Options for report messages.
Data type: MQLONG.
Value:

MQRO_NONE
No reports required.

MsgType

Description: Indicates type of message.
Data type: MQLONG.
Value: MQMT_DATAGRAM.

Expiry

Description: Message lifetime.
Data type: MQLONG.
Value:

MQEI_UNLIMITED
The message does not have an expiry time.

Feedback

Description: Feedback or reason code.
Data type: MQLONG.
Value: MQFB_NONE.

Encoding

Description: Numeric encoding of message data.
Data type: MQLONG.
Value: MQENC_NATIVE.

CodedCharSetId

Description: Character set identifier of event message data.
Data type: MQLONG.
Value: Coded character set ID (CCSID) of the queue manager generating the event.

Format

Description: Format name of message data.
Data type: MQCHAR8.
Value:

MQFMT_EVENT
Event message.

Priority

Description: Message priority.
Data type: MQLONG.
Value:

MQPRI_PRIORITY_AS_Q_DEF
The priority is that of the event queue.

Persistence

Description: Message persistence.
Data type: MQLONG.
Value:

MQPER_PERSISTENCE_AS_Q_DEF
The priority is that of the event queue.

MsgId

Description: Message identifier.
Data type: MQBYTE24.
Value: A unique value generated by the queue manager.

CorrelId

Description: Correlation identifier.
Data type: MQBYTE24.
Value: For performance, queue manager, logger, channel, bridge, and SSL events:

MQCI_NONE
No correlation identifier is specified. This is for private queues only.

For such events on a shared queue, a nonzero correlation identifier is set. This parameter is set so that you can track multiple event messages from different queue managers. The characters are specified in the following way:

1-4 Product identifier ('CSQ ')

5-8 Queue-sharing group name

9 Queue manager identifier

10-17 Time stamp

18-24 Nulls

For configuration and command events:

A unique nonzero correlation identifier
All messages relating to the same event have the same CorrelId.

BackoutCount

Description: Backout counter.
Data type: MQLONG.
Value: 0.

ReplyToQ

Description: Name of reply queue.
Data type: MQCHAR48.
Values: Blank.

ReplyToQMgr

Description: Name of reply queue manager.
Data type: MQCHAR48.
Value: The queue manager name at the originating system.

UserIdentifier

Description: Identifies the application that originated the message.
Data type: MQCHAR12.
Value: Blank.

AccountingToken

Description: Accounting token that allows an application to charge for work done as a result of the message.
Data type: MQBYTE32.
Value: MQACT_NONE.

ApplIdentityData

Description: Application data relating to identity.
Data type: MQCHAR32.
Values: Blank.

PutApplType

Description: Type of application that put the message.
Data type: MQLONG.
Value:

MQAT_QMGR
Queue manager generated message.

PutApplName

Description: Name of application that put the message.
Data type: MQCHAR28.
Value: The queue manager name at the originating system.

PutDate

Description: Date when message was put.
Data type: MQCHAR8.
Value: As generated by the queue manager.

PutTime

Description: Time when message was put.
Data type: MQCHAR8.
Value: As generated by the queue manager.

ApplOriginData

Description: Application data relating to origin.
Data type: MQCHAR4.
Value: Blank.

Note: If *Version* is MQMD_VERSION_2, the following additional fields are present:

GroupId

Description: Identifies to which message group or logical message the physical message belongs.
Data type: MQBYTE24.
Value:

MQGI_NONE
No group identifier specified.

MsgSeqNumber

Description: Sequence number of logical message within group.
Data type: MQLONG.
Value: 1.

Offset

Description: Offset of data in physical message from start of logical message.
Data type: MQLONG.
Value: 0.

MsgFlags

Description: Message flags that specify attributes of the message or control its processing.
Data type: MQLONG.
Value: MQMF_NONE.

OriginalLength

Description: Length of original message.
Data type: MQLONG.
Value: MQOL_UNDEFINED.

Event message MQCFH (PCF header)

The message data in event messages is in programmable command format (PCF), as used in PCF command inquiries and responses. The message data consists of two parts: the event header and the event data.

The MQCFH header specifies the following information:

- The category of event: whether the event is a queue manager, performance, channel, configuration, command, or logger event.
- A reason code specifying the cause of the event. For events caused by MQI calls, this reason code is the same as the reason code for the MQI call.

Reason codes have names that begin with the characters MQRC_. For example, the reason code MQRC_PUT_INHIBITED is generated when an application attempts to put a message on a queue that is not enabled for puts.

For an event, the MQCFH structure contains the following values:

Type

Description: Structure type that identifies the content of the message.
Data type: MQLONG.
Value:

MQCFT_EVENT
Message is reporting an event.

StrucLength

Description: Structure length.
Data type: MQLONG.
Value:

MQCFH_STRUC_LENGTH
Length in bytes of MQCFH structure.

Version

Description: Structure version number.
Data type: MQLONG.
Values:

MQCFH_VERSION_1
Version-1 in all events except configuration and command events.

MQCFH_VERSION_2
Version-2 for configuration events.

MQCFH_VERSION_3
Version-3 for command events.

Command

Description: Command identifier. This identifies the event category.
Data type: MQLONG.
Values:

MQCMD_Q_MGR_EVENT
Queue manager event.

MQCMD_PERFM_EVENT
Performance event.

MQCMD_CHANNEL_EVENT
Channel event.

MQCMD_CONFIG_EVENT
Configuration event.

MQCMD_COMMAND_EVENT
Command event.

MQCMD_LOGGER_EVENT
Logger event.

MsgSeqNumber

Description: Message sequence number. This is the sequence number of the message within a group of related messages.
Data type: MQLONG.
Values:

1 For change object configuration events with attribute values before the changes, and for all other types of events.

2 For change object configuration events with the attribute values after the changes

Control

Description: Control options.
Data type: MQLONG.
Values:

MQCFC_LAST
For change object configuration events with attribute values after the changes, and for all other types of events.

MQCFC_NOT_LAST
For Change Object configurations events only, with the attribute values from before the changes.

CompCode

Description: Completion code.
Data type: MQLONG.
Values:

MQCC_OK
Event reporting OK condition.

MQCC_WARNING
Event reporting warning condition. All events have this completion code, unless otherwise specified.

Reason

Description: Reason code qualifying completion code.
Data type: MQLONG.
Values: MQRC_* Dependent on the event being reported.
Note: Events with the same reason code are further identified by the **ReasonQualifier** parameter in the event data.

ParameterCount

Description: Count of parameter structures. This is the number of parameter structures that follow the MQCFH structure. A group structure (MQCFGR), and its included parameter structures, are counted as one structure only.
Data type: MQLONG.
Values: 0 or greater.

Event message descriptions

The event message data contains information specific to the event that was generated. This data includes the name of the queue manager and, where appropriate, the name of the queue.

The data structures returned depend on which particular event was generated. In addition, for some events, certain parameters of the structures are optional, and are returned only if they contain information that is relevant to the circumstances giving rise to the event. The values in the data structures depend on the circumstances that caused the event to be generated.

Note:

1. The PCF structures in the message data are not returned in a defined order. They must be identified from the parameter identifiers shown in the description.
2. Events are available on all platforms, unless specific limitations are shown at the start of an event description.

Alias Base Queue Type Error:

Event name:	Alias Base Queue Type Error.
Reason code in MQCFH:	MQRC_ALIAS_BASE_Q_TYPE_ERROR (2001, X'7D1'). Alias base queue not a valid type.
Event description:	An MQOPEN or MQPUT1 call was issued specifying an alias queue as the destination, but the *BaseObjectName* in the alias queue definition resolves to a queue that is not a local queue, or local definition of a remote queue.
Event type:	Local.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

BaseObjectName

Description: Object name to which the alias resolves.
Identifier: MQCA_BASE_OBJECT_NAME. For compatibility with existing applications you can still use MQCA_BASE_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

QType

Description: Type of queue to which the alias resolves.
Identifier: MQIA_Q_TYPE.
Data type: MQCFIN.
Values:

MQQT_ALIAS
Alias queue definition.

MQQT_MODEL
Model queue definition.

Returned: Always.

ApplType

Description: Type of the application making the call that caused the event.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the call that caused the event.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Bridge Started:

Event name:	Bridge Started.
Reason code in MQCFH:	MQRC_BRIDGE_STARTED (2125, X'84D'). Bridge started.
Event description:	The IMS bridge has been started.
Event type:	IMS bridge.
Platforms:	IBM MQ for z/OS only.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

BridgeType

Description: Bridge type.
Identifier: MQIACF_BRIDGE_TYPE.
Data type: MQCFIN.
Values:

MQBT_OTMA
OTMA bridge.

Returned: Always.

BridgeName

Description:	Bridge name. For bridges of type MQBT_OTMA, the name is of the form `XCFgroupXCFmember`, where `XCFgroup` is the XCF group name to which both IMS and IBM MQ belong. `XCFmember` is the XCF member name of the IMS system.
Identifier:	MQCACF_BRIDGE_NAME.
Data type:	MQCFST.
Maximum length:	MQ_BRIDGE_NAME_LENGTH.
Returned:	Always.

Bridge Stopped:

Event name:	Bridge Stopped.
Reason code in MQCFH:	MQRC_BRIDGE_STOPPED (2126, X'84E'). Bridge stopped.
Event description:	The IMS bridge has been stopped.
Event type:	IMS bridge.
Platforms:	IBM MQ for z/OS only.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ReasonQualifier

Description:	Identifier that qualifies the reason code in MQCFH.
Identifier:	MQIACF_REASON_QUALIFIER.
Data type:	MQCFIN.
Values:	**MQRQ_BRIDGE_STOPPED_OK** Bridge has been stopped with either a zero return code or a warning return code. For MQBT_OTMA bridges, one side or the other issued a normal IXCLEAVE request. **MQRQ_BRIDGE_STOPPED_ERROR** Bridge has been stopped but there is an error reported.
Returned:	Always.

BridgeType

Description: Bridge type.
Identifier: MQIACF_BRIDGE_TYPE.
Data type: MQCFIN.
Value:

MQBT_OTMA
OTMA bridge.

Returned: Always.

BridgeName

Description: Bridge name. For bridges of type MQBT_OTMA, the name is of the form XCFgroupXCFmember, where XCFgroup is the XCF group name to which both IMS and IBM MQ belong. XCFmember is the XCF member name of the IMS system.
Identifier: MQCACF_BRIDGE_NAME.
Data type: MQCFST.
Maximum length: MQ_BRIDGE_NAME_LENGTH.
Returned: Always.

ErrorIdentifier

Description: When a bridge is stopped because of an error, this code identifies the error. If the event reports a bridge stop failure, the IMS sense code is set.
Identifier: MQIACF_ERROR_IDENTIFIER.
Data type: MQCFIN.
Returned: If *ReasonQualifier* is MQRQ_BRIDGE_STOPPED_ERROR.

Change Authority Record:

Event name:	Change Authority Record
Reason code in MQCFH:	MQRC_CONFIG_CHANGE_OBJECT (2368, X'0940'). Object changed.
Event description:	A Set Authority Record command was issued that successfully changed an existing authority record.
Event type:	Configuration
Platforms:	All except z/OS.
Event queue:	SYSTEM.ADMIN.CONFIG.EVENT.

Note that two event messages are generated for the change authority record event. The first has the authority record attribute values *before* the change; the second has the attribute values *after* the change.

Event data

EventQMgr

Description: The queue manager where the command or call was entered. That is, the queue manager where the command is processed and that generates the event is in the MQMD of the event message.
Identifier: MQCACF_EVENT_Q_MGR
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

EventUserId

Description: The user ID that issued the command or call that generated the event.

This is the same user ID that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (**UserIdentifier**) from the message descriptor of the command message..
Identifier: MQCACF_EVENT_USER_ID
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

EventOrigin

Description: The origin of the action causing the event.
Identifier: MQIACF_EVENT_ORIGIN
Data type: MQCFIN.
Values:

MQEVO_CONSOLE
Console command (runmqsc or setmqaut)

MQEVO_INTERNAL
Directly by queue manager

MQEVO_MSG
Command message on SYSTEM.ADMIN.COMMAND.QUEUE

Returned: Always

EventAccountingToken

Description: For commands received as a message (MQEVO_MSG), the accounting token (**AccountingToken**) from the message descriptor of the command message.
Identifier: MQBACF_EVENT_ACCOUNTING_TOKEN
Data type: MQCFBS
Maximum length: MQ_ACCOUNTING_TOKEN_LENGTH
Returned: Only if **EventOrigin** is MQEVO_MSG.

EventApplIdentity

Description: For commands received as a message (MQEVO_MSG), application identity data (**ApplIdentityData**) from the message descriptor of the command message.
Identifier: MQMQCACF_EVENT_APPL_IDENTITY
Data type: MQCFST
Maximum length: MQ_APPL_IDENTITY_DATA_LENGTH
Returned: Only if **EventOrigin** is MQEVO_MSG.

EventApplType

Description: For commands received as a message (MQEVO_MSG), the type of application (**PutApplType**) from the message descriptor of the command message.
Identifier: MQIACF_EVENT_APPL_TYPE
Data type: MQCFIN
Values:
Returned: Only if **EventOrigin** is MQEVO_MSG.

EventApplName

Description: For commands received as a message (MQEVO_MSG), the name of the application (**PutApplName**) from the message descriptor of the command message.
Identifier: MQCACF_EVENT_APPL_NAME
Data type: MQCFST
Maximum length: MQ_APPL_NAME_LENGTH
Returned: Only if **EventOrigin** is MQEVO_MSG.

EventApplOrigin

Description: For commands received as a message (MQEVO_MSG), the application origin data (**ApplOriginData**) from the message descriptor of the command message.
Identifier: MQCACF_EVENT_APPL_ORIGIN
Data type: MQCFST
Maximum length: MQ_APPL_ORIGIN_DATA_LENGTH
Returned: Only if **EventOrigin** is MQEVO_MSG.

ObjectType

Description: Object type
Identifier: MQIACF_OBJECT_TYPE
Data type: MQCFIN
Values: MQOT_AUTH_REC
Returned: Always

ProfileName

Description: Object or generic profile name
Identifier: MQCACF_AUTH_PROFILE_NAME
Data type: MQCFST
Maximum length: MQ_AUTH_PROFILE_NAME_LENGTH
Returned: Always

Object attributes

A parameter structure is returned for each attribute of the object. The attributes returned depend on the object type. For more information see "Object attributes for event data" on page 4151.

Change object:

Event name:	Change object.
Reason code in MQCFH:	MQRC_CONFIG_CHANGE_OBJECT (2368, X'940'). Existing object changed.
Event description:	An ALTER or DEFINE REPLACE command or an MQSET call was issued that successfully changed an existing object.
Event type:	Configuration.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CONFIG.EVENT.

Note: Two event messages are generated for the change object event. The first has the object attribute values **before** the change, the second has the attribute values **after** the change.

Event data

EventUserId

Description: The user ID that issued the command or call that generated the event. (This is the same user ID that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (UserIdentifier) from the MQMD of the command message).
Identifier: MQCACF_EVENT_USER_ID.
Datatype: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

EventOrigin

Description: The origin of the action causing the event.
Identifier: MQIACF_EVENT_ORIGIN.
Datatype: MQCFIN.
Values:

MQEVO_CONSOLE
Console command.

MQEVO_INIT
Initialization input data set command.

MQEVO_INTERNAL
Directly by queue manager.

MQEVO_MQSET
MQSET call.

MQEVO_MSG
Command message on SYSTEM.COMMAND.INPUT.

MQEVO_OTHER
None of the above.

Returned: Always.

EventQMgr

Description: The queue manager where the command or call was entered. (The queue manager where the command is executed and that generates the event is in the MQMD of the event message).
Identifier: MQCACF_EVENT_Q_MGR.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

EventAccountingToken

Description: For commands received as a message (MQEVO_MSG), the accounting token (AccountingToken) from the MQMD of the command message.
Identifier: MQBACF_EVENT_ACCOUNTING_TOKEN.
Datatype: MQCFBS.
Maximum length: MQ_ACCOUNTING_TOKEN_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplIdentity

Description: For commands received as a message (MQEVO_MSG), application identity data (ApplIdentityData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_IDENTITY.
Datatype: MQCFST.
Maximum length: MQ_APPL_IDENTITY_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplType

Description: For commands received as a message (MQEVO_MSG), the type of application (PutApplType) from the MQMD of the command message.
Identifier: MQIACF_EVENT_APPL_TYPE.
Datatype: MQCFIN.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplName

Description: For commands received as a message (MQEVO_MSG), the name of the application (PutApplName) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_NAME.
Datatype: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplOrigin

Description: For commands received as a message (MQEVO_MSG), the application origin data (ApplOriginData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_ORIGIN.
Datatype: MQCFST.
Maximum length: MQ_APPL_ORIGIN_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

ObjectType

Description: Object type:
Identifier: MQIACF_OBJECT_TYPE.
Datatype: MQCFIN.
Values:

MQOT_CHANNEL
Channel.

MQOT_CHLAUTH
Channel authentication record.

MQOT_NAMELIST
Namelist.

MQOT_NONE
No object.

MQOT_PROCESS
Process.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_STORAGE_CLASS
Storage class.

MQOT_AUTH_INFO
Authentication information.

MQOT_CF_STRUC
CF structure.

MQOT_TOPIC
Topic.

MQOT_COMM_INFO
Communication information.

MQOT_LISTENER
Channel Listener.

Returned: Always.

ObjectName

Description: Object name:
Identifier : Identifier will be according to object type.

- MQCACH_CHANNEL_NAME
- MQCA_NAMELIST_NAME
- MQCA_PROCESS_NAME
- MQCA_Q_NAME
- MQCA_Q_MGR_NAME
- MQCA_STORAGE_CLASS
- MQCA_AUTH_INFO_NAME
- MQCA_CF_STRUC_NAME
- MQCA_TOPIC_NAME
- MQCA_COMM_INFO_NAME
- MQCACH_LISTENER_NAME

Note: MQCACH_CHANNEL_NAME can also be used for channel authentication.

Datatype: MQCFST.
Maximum length: MQ_OBJECT_NAME_LENGTH.

Returned: Always

Disposition

Description: Object disposition:
Identifier: MQIA_QSG_DISP.
Datatype: MQCFIN.
Values:

MQQSGD_Q_MGR
Object resides on page set of queue manager.

MQQSGD_SHARED
Object resides in shared repository and messages are shared in coupling facility.

MQQSGD_GROUP
Object resides in shared repository.

MQQSGD_COPY
Object resides on page set of queue manager and is a local copy of a GROUP object.

Returned: Always, except for queue manager and CF structure objects.

Object attributes

A parameter structure is returned for each attribute of the object. The attributes returned depend on the object type. For more information see "Object attributes for event data" on page 4151.

Channel Activated:

Event name:	Channel Activated.
Reason code in MQCFH:	MQRC_CHANNEL_ACTIVATED (2295, X'8F7'). Channel activated.
Event description:	This condition is detected when a channel that has been waiting to become active, and for which a Channel Not Activated event has been generated, is now able to become active, because an active slot has been released by another channel. This event is not generated for a channel that is able to become active without waiting for an active slot to be released.
Event type:	Channel.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ChannelName

Description:	Channel Name.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH.
Returned:	Always.

XmitQName

Description:	Transmission queue name.
Identifier:	MQCACH_XMIT_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	For sender, server, cluster-sender, and cluster-receiver channels only.

ConnectionName

Description:	If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the *ConnectionName* field in the channel definition.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	Only for commands that do not contain a generic name.

Channel Auto-definition Error:

Event name:	Channel Auto-definition Error.
Reason code in MQCFH:	MQRC_CHANNEL_AUTO_DEF_ERROR (2234, X'8BA'). Automatic channel definition failed.
Event description:	This condition is detected when the automatic definition of a channel fails; this may be because an error occurred during the definition process, or because the channel automatic-definition exit inhibited the definition. Additional information indicating the reason for the failure is returned in the event message.
Event type:	Channel.
Platforms:	All, except IBM MQ for z/OS.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ChannelName

Description: Name of the channel for which the auto-definiton has failed.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: Always.

ChannelType

Description: Channel Type. This specifies the type of channel for which the auto-definition has failed.
Identifier: MQIACH_CHANNEL_TYPE.
Data type: MQCFIN.
Values:

MQCHT_RECEIVER
Receiver.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLUSSDR
Cluster-sender.

Returned: Always.

ErrorIdentifier

Description: Identifier of the cause of the error. This contains either the reason code (MQRC_* or MQRCCF_*) resulting from the channel definition attempt or the value MQRCCF_SUPPRESSED_BY_EXIT if the attempt to create the definition was disallowed by the exit.
Identifier: MQIACF_ERROR_IDENTIFIER.
Data type: MQCFIN.
Returned: Always.

ConnectionName

Description: Name of the partner attempting to establish connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: Always.

AuxErrorDataInt1

Description: Auxiliary error data. This contains the value returned by the exit in the *Feedback* field of the MQCXP to indicate why the auto definition has been disallowed.
Identifier: MQIACF_AUX_ERROR_DATA_INT_1.
Data type: MQCFIN.
Returned: Only if *ErrorIdentifier* contains MQRCCF_SUPPRESSED_BY_EXIT.

Channel Auto-definition OK:

Event name:	Channel Auto-definition OK.
Reason code in MQCFH:	MQRC_CHANNEL_AUTO_DEF_OK (2233, X'8B9'). Automatic channel definition succeeded.
Event description:	This condition is detected when the automatic definition of a channel is successful. The channel is defined by the MCA.
Event type:	Channel.
Platforms:	All, except IBM MQ for z/OS.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ChannelName

Description: Name of the channel being defined.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: Always.

ChannelType

Description: Type of channel being defined.
Identifier: MQIACH_CHANNEL_TYPE.
Data type: MQCFIN.
Values:

MQCHT_RECEIVER
Receiver.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLUSSDR
Cluster-sender.

Returned: Always.

ConnectionName

Description: Name of the partner attempting to establish connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: Always.

Channel Blocked:

Event name:	Channel Blocked.
Reason code in MQCFH:	MQRC_CHANNEL_BLOCKED Channel blocked. MQRC_CHANNEL_BLOCKED_WARNING Channel blocked - warning mode.
Event description:	This event is issued when an attempt to start an inbound channel is blocked. For MQRC_CHANNEL_BLOCKED_WARNING, temporary access has been granted to the channel because the channel authentication record is defined with WARN set to YES.
Event type:	Channel.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

Reason qualifier

Description: Identifier that qualifies the reason code
Identifier: MQIACF_REASON_QUALIFIER
Data type: MQCFIN.
Values:

MQRQ_CHANNEL_BLOCKED_ADDRESS
Channel was blocked due to its IP address being in the list to be refused

MQRQ_CHANNEL_BLOCKED_USERID
Channel was blocked due to its asserted or mapped user ID being in the list to be refused.

MQRQ_CHANNEL_BLOCKED_NOACCESS
Channel was blocked due to its IP address; TLS Peer name; remote queue manager name or client user ID being mapped to have no access.

Returned: Always.

ChannelName

Description: Channel Name.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: If the Reason Qualifier is not MQRQ_CHANNEL_BLOCKED_ADDRESS. In that case the inbound connection is blocked before the channel name is known.

UserIdentifier

Description: User identifier that was blocked.
Identifier: MQCACF_USER_IDENTIFIER
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH
Returned: Only if the Reason Qualifier is MQRQ_CHANNEL_BLOCKED_USERID

ConnectionName

Description: Address of the partner attempting to establish connection
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: Always

RemoteQMgrName

Description: Name of the partner queue manager attempting to establish connection.
Identifier: MQCA_REMOTE_Q_MGR_NAME
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH
Returned: Only for inbound queue manager connections.

SSLPeerName

Description: The Distinguished Name in the certificate sent from the remote system.
Identifier: MQCACH_SSL_PEER_NAME
Data type: MQCFST.
Maximum length: MQ_DISTINGUISHED_NAME_LENGTH
Returned: Whenever the channel is using TLS and the client has not connected anonymously.

ClientUserIdentifier

Description: Client side user identifier of the partner attempting to establish connection.
Identifier: MQCACH_CLIENT_USER_ID
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH
Returned: Only for inbound client connections, if the Reason Qualifier is not MQRQ_CHANNEL_BLOCKED_ADDRESS. In that case the inbound connection is blocked before the client user Id name is known.

ApplType

Description: Type of application that made the API call.
Identifier: MQIA_APPL_TYPE
Data type: MQCFIN.
Returned: Only for inbound client connections. If the Reason Qualifier is not MQRQ_CHANNEL_BLOCKED_ADDRESS. In that case the inbound connection is blocked before the application name is known.

ApplName

Description: Name of the application that made the API call.
Identifier: MQCACF_APPL_NAME
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH
Returned: Only for inbound client connections. If the Reason Qualifier is not MQRQ_CHANNEL_BLOCKED_ADDRESS. In that case the inbound connection is blocked before the application name is known.

Channel Conversion Error:

Event name:	Channel Conversion Error.
Reason code in MQCFH:	MQRC_CHANNEL_CONV_ERROR (2284, X'8EC'). Channel conversion error.
Event description:	This condition is detected when a channel is unable to carry out data conversion and the MQGET call to get a message from the transmission queue resulted in a data conversion error. The reason for the failure is identified by *ConversionReasonCode*.
Event type:	Channel.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ConversionReasonCode

Description: Identifier of the cause of the conversion error.
Identifier: MQIACF_CONV_REASON_CODE.
Data type: MQCFIN.
Values:

MQRC_CONVERTED_MSG_TOO_BIG (2120, X'848')
Converted message too big for application buffer.

MQRC_FORMAT_ERROR (2110, X'83E')
Message format not valid.

MQRC_NOT_CONVERTED (2119, X'847')
Application message data not converted.

MQRC_SOURCE_CCSID_ERROR (2111, X'83F')
Source coded character set identifier not valid.

MQRC_SOURCE_DECIMAL_ENC_ERROR (2113, X'841')
Packed-decimal encoding in message not recognized.

MQRC_SOURCE_FLOAT_ENC_ERROR (2114, X'842')
Floating-point encoding in message not recognized.

MQRC_SOURCE_INTEGER_ENC_ERROR (2112, X'840')
Integer encoding in message not recognized.

MQRC_TARGET_CCSID_ERROR (2115, X'843')
Target coded character set identifier not valid.

MQRC_TARGET_DECIMAL_ENC_ERROR (2117, X'845')
Packed-decimal encoding specified by receiver not recognized.

MQRC_TARGET_FLOAT_ENC_ERROR (2118, X'846')
Floating-point encoding specified by receiver not recognized.

MQRC_TARGET_INTEGER_ENC_ERROR (2116, X'844')
Integer encoding specified by receiver not recognized.

MQRC_TRUNCATED_MSG_ACCEPTED (2079, X'81F')
Truncated message returned (processing completed).

MQRC_TRUNCATED_MSG_FAILED (2080, X'820')
Truncated message returned (processing not completed).

Returned: Always.

ChannelName

Description: Channel name.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: Always.

Format

Description: Format name.
Identifier: MQCACH_FORMAT_NAME.
Data type: MQCFST.
Maximum length: MQ_FORMAT_LENGTH.
Returned: Always.

XmitQName

Description: Transmission queue name.
Identifier: MQCACH_XMIT_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ConnectionName

Description:	If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the *ConnectionName* field in the channel definition.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	Always.

Channel Not Activated:

Event name:	Channel Not Activated.
Reason code in MQCFH:	MQRC_CHANNEL_NOT_ACTIVATED (2296, X'8F8'). Channel cannot be activated.
Event description:	This condition is detected when a channel is required to become active, either because it is starting, or because it is about to make another attempt to establish connection with its partner. However, it is unable to do so because the limit on the number of active channels has been reached. See the following: • MaxActiveChannels parameter in the qm.ini file for AIX, HP-UX, and Solaris • MaxActiveChannels parameter in the Registry for Windows. • ACTCHL parameter on the ALTER QMGR command for z/OS The channel waits until it is able to take over an active slot released when another channel ceases to be active. At that time a Channel Activated event is generated.
Event type:	Channel.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ChannelName

Description:	Channel name.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH.
Returned:	Always.

XmitQName

Description:	Transmission queue name.
Identifier:	MQCACH_XMIT_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	For sender, server, cluster-sender, and cluster-receiver channel types only.

ConnectionName

Description:	If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the *ConnectionName* field in the channel definition.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	Only for commands that do not contain a generic name.

Channel Not Available:

Event name:	Channel Not Available.
Reason code in MQCFH:	MQRC_CHANNEL_NOT_AVAILABLE (2537, X'9E9'). Channel not available.
Event description:	This is issued when an attempt to start an inbound channel is rejected.
Event type:	Channel.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ReasonQualifier

Description:	Identifier that qualifies the reason code.
Identifier:	MQIACF_REASON_QUALIFIER.
Data type:	MQCFIN.

Values:

MQRQ_MAX_ACTIVE_CHANNELS
Channel was unavailable due to maximum active channel instances (MaxActiveChannels qm.ini stanza on Multiplatforms or ACTCHL MQSC keyword on z/OS) limit being reached for the queue manager.

MQRQ_MAX_CHANNELS
Channel was unavailable due to maximum channel instances (MaxChannels qm.ini stanza on Multiplatforms or MAXCHL MQSC keyword on z/OS) limit being reached for the queue manager.

MQRQ_SVRCONN_INST_LIMIT
Channel was unavailable due to maximum active channel instances (MAXINST) limit being reached for the channel.

MQRQ_CLIENT_INST_LIMIT
Channel was unavailable due to maximum active channel instances (MAXINSTC) limit being reached for the client for the channel.

Returned: Always.

ChannelName

Description: Channel name.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: Always.

ConnectionName

Description: Address of the partner attempting to establish connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: Always.

MaximumActiveChannels

Description: Maximum active channels.
Identifier: MQIA_ACTIVE_CHANNELS
Data type: MQCFIN.
Returned: Only where reason qualifier MQRQ_MAX_ACTIVE_CHANNELS.

MaximumChannels

Description: Maximum channels.
Identifier: MQIA_MAX_CHANNELS
Data type: MQCFIN
Returned: Only where reason qualifier MQRQ_MAX_CHANNELS.

MaximumInstances

Description:	Maximum channel instances.
Identifier:	MQIACH_MAX_INSTANCES
Data type:	MQCFIN
Returned:	Only where reason qualifier MQRQ_SVRCONN_INST_LIMIT.

MaximumClientInstances

Description:	Maximum channel instances per client.
Identifier:	MQIACH_MAX_INSTS_PER_CLIENT
Data type:	MQCFIN
Returned:	Only where reason qualifier MQRQ_CLIENT_INST_LIMIT.

Channel SSL Error:

Event name:	Channel SSL Error.
Reason code in MQCFH:	MQRC_CHANNEL_SSL_ERROR (2371, X'943'). Channel SSL Error.
Event description:	This condition is detected when a channel using Transport Layer Security (TLS) fails to establish a connection. *ReasonQualifier* identifies the nature of the error.
Event type:	SSL.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ReasonQualifier

Description:	Identifier that qualifies the reason code.
Identifier:	MQIACF_REASON_QUALIFIER.
Data type:	MQCFIN.

Values:

MQRQ_SSL_HANDSHAKE_ERROR
The key exchange / authentication failure arose during the TLS handshake.

MQRQ_SSL_CIPHER_SPEC_ERROR
This error can mean any one of the following:

- The TLS client CipherSpec does not match that on the TLS server channel definition.
- An invalid CipherSpec has been specified.
- A CipherSpec has only been specified on one end of the TLS channel.

MQRQ_SSL_PEER_NAME_ERROR
The Distinguished Name in the certificate sent by one end of the TLS channel does not match the peer name on the end of the channel definition at the other end of the TLS channel.

MQRQ_SSL_CLIENT_AUTH_ERROR
The TLS server channel definition specified either SSLCAUTH(REQUIRED) or a SSLPEER value that was not blank, but the TLS client did not provide a certificate.

Returned: Always.

ChannelName

Description: Channel Name.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: The *ChannelName* might not be available if the channel has not yet got far enough through its start-up process, in this case the channel name will not be returned. Otherwise always.

XmitQName

Description: Transmission queue name.
Identifier: MQCACH_XMIT_Q_NAME.
Data type: MQCFST.
Returned: For sender, server, cluster-sender and cluster-receiver channels only.

ConnectionName

Description: If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the ConnectionName field in the channel definition.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: The *ConnectionName* might not be available if the channel has not yet got far enough through its start-up process, in this case the connection name will not be returned. Otherwise always.

SSLHandshakeStage

Description: Information about the TLS function call giving the error. For z/OS, details of function names can be found in the *System Secure Sockets Layer Programming Guide and Reference* SC24-5877.
Identifier: MQCACH_SSL_HANDSHAKE_STAGE.
Data type: MQCFST.
Maximum length: MQ_SSL_HANDSHAKE_STAGE_LENGTH.
Returned: This field is only present if *ReasonQualifier* is set to MQRQ_SSL_HANDSHAKE_ERROR.

SSLReturnCode

Description: A numeric return code from a failing TLS call.

Details of TLS Return Codes for specific platforms can be found as follows:

- z/OS For z/OS, see "Transport Layer Security (TLS) return codes for z/OS" on page 5221.
- Multi For Multiplatforms, see Transport Layer Security (TLS) return codes.

Identifier: MQIACH_SSL_RETURN_CODE.
Data type: MQCFIN.
Returned: This field is only present if *ReasonQualifier* is set to MQRQ_SSL_HANDSHAKE_ERROR.

SSLPeerName

Description: The Distinguished Name in the certificate sent from the remote system.
Identifier: MQCACH_SSL_PEER_NAME.
Data type: MQCFST.
Maximum length: MQ_DISTINGUISHED_NAME_LENGTH.
Returned: This field is only present if *ReasonQualifier* is set to MQRQ_SSL_PEER_NAME_ERROR and is not always present for this reason qualifier.

Channel SSL Warning:

Event name:	Channel SSL Warning.
Reason code in MQCFH:	MQRC_CHANNEL_SSL_WARNING (2552, X'9F8'). Channel SSL Warning.
Event description:	This condition is detected when a channel using Transport Layer Security (TLS) experiences a problem that does not cause it to fail to establish a TLS connection. *ReasonQualifier* identifies the nature of the event.
Event type:	SSL.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier that qualifies the reason code.
Identifier: MQIACF_REASON_QUALIFIER.
Data type: MQCFIN.
Values:

MQRQ_SSL_UNKNOWN_REVOCATION
An OCSP responder returned a response of Unknown. IBM MQ is configured to produce warnings but allow the connection to continue.

Returned: Always.

ChannelName

Description: Channel Name.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: The *ChannelName* might not be available if the channel has not yet got far enough through its start-up process, in this case the channel name will not be returned. Otherwise always.

XmitQName

Description: Transmission queue name.
Identifier: MQCACH_XMIT_Q_NAME.
Data type: MQCFST.
Returned: For sender, server, cluster-sender and cluster-receiver channels only.

ConnectionName

Description: If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the ConnectionName field in the channel definition.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: The *ConnectionName* may not be available if the channel has not yet got far enough through its start-up process, in this case the connection name will not be returned. Otherwise always.

Channel Started:

Event name:	Channel Started.
Reason code in MQCFH:	MQRC_CHANNEL_STARTED (2282, X'8EA'). Channel started.
Event description:	Either an operator has issued a Start Channel command, or an instance of a channel has been successfully established. This condition is detected when Initial Data negotiation is complete and resynchronization has been performed where necessary, such that message transfer can proceed.
Event type:	Channel.
Platforms:	All. Client connections do not produce this event.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ChannelName

Description:	Channel name.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH.
Returned:	Always.

XmitQName

Description:	Transmission queue name.
Identifier:	MQCACH_XMIT_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	For sender, server, cluster-sender, and cluster-receiver channels only.

ConnectionName

Description:	If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the *ConnectionName* field in the channel definition.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	Only for commands that do not contain a generic name.

Channel Stopped:

Event name:	Channel Stopped.
Reason code in MQCFH:	MQRC_CHANNEL_STOPPED (2283, X'8EB'). Channel stopped.
Event description:	This is issued when a channel instance stops. It will only be issued if the channel instance previously issued a channel started event.
Event type:	Channel.
Platforms:	All. Client connections do not produce this event.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier that qualifies the reason code.
Identifier: MQIACF_REASON_QUALIFIER.
Data type: MQCFIN.
Values:

MQRQ_CHANNEL_STOPPED_OK
Channel has been closed with either a zero return code or a warning return code.

MQRQ_CHANNEL_STOPPED_ERROR
Channel has been closed but there is an error reported and the channel is not in stopped or retry state.

MQRQ_CHANNEL_STOPPED_RETRY
Channel has been closed and it is in retry state.

MQRQ_CHANNEL_STOPPED_DISABLED
Channel has been closed and it is in a stopped state.

Returned: Always.

ChannelName

Description: Channel name.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: Always.

ErrorIdentifier

Description: Identifier of the cause of the error. If a channel is stopped due to an error, this is the code that identifies the error. If the event message is because of a channel stop failure, the following fields are set:

1. *ReasonQualifier*, containing the value MQRQ_CHANNEL_STOPPED_ERROR
2. *ErrorIdentifier*, containing the code number of an error message that describes the error
3. *AuxErrorDataInt1*, containing error message integer insert 1
4. *AuxErrorDataInt2*, containing error message integer insert 2
5. *AuxErrorDataStr1*, containing error message string insert 1
6. *AuxErrorDataStr2*, containing error message string insert 2
7. *AuxErrorDataStr3*, containing error message string insert 3

The meanings of the error message inserts depend on the code number of the error message. Details of error-message code numbers and the inserts for specific platforms can be found as follows:

- z/OS For z/OS, see Distributed queuing message codes.
- Multi For Multiplatforms, the last four digits of *ErrorIdentifier* when displayed in hexadecimal notation indicate the decimal code number of the error message.

  For example, if *ErrorIdentifier* has the value X'xxxxyyyy', the message code of the error message explaining the error is AMQyyyy. See "IBM MQ messages on Multiplatforms" on page 4328 for a description of these error messages.

Identifier: MQIACF_ERROR_IDENTIFIER.
Data type: MQCFIN.
Returned: Always.

AuxErrorDataInt1

Description: First integer of auxiliary error data for channel errors. If a channel is stopped due to an error, this is the first integer parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.
Identifier: MQIACF_AUX_ERROR_DATA_INT_1.
Data type: MQCFIN.
Returned: Always.

AuxErrorDataInt2

Description: Second integer of auxiliary error data for channel errors. If a channel is stopped due to an error, this is the second integer parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.
Identifier: MQIACF_AUX_ERROR_DATA_INT_2.
Data type: MQCFIN.
Returned: Always.

AuxErrorDataStr1

Description: First string of auxiliary error data for channel errors. If a channel is stopped due to an error, this is the first string parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.
Identifier: MQCACF_AUX_ERROR_DATA_STR_1.
Data type: MQCFST.
Returned: Always.

AuxErrorDataStr2

Description: Second string of auxiliary error data for channel errors. If a channel is stopped due to an error, this is the second string parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.
Identifier: MQCACF_AUX_ERROR_DATA_STR_2.
Data type: MQCFST.
Returned: Always.

AuxErrorDataStr3

Description: Third string of auxiliary error data for channel errors. If a channel is stopped due to an error, this is the third string parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.
Identifier: MQCACF_AUX_ERROR_DATA_STR_3.
Data type: MQCFST.
Returned: Always.

XmitQName

Description: Transmission queue name.
Identifier: MQCACH_XMIT_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: For sender, server, cluster-sender, and cluster-receiver channels only.

ConnectionName

Description: If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the *ConnectionName* field in the channel definition.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST or MQCFSL.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: Only for commands that do not contain a generic name.

Channel Stopped By User:

Event name:	Channel Stopped By User.
Reason code in MQCFH:	MQRC_CHANNEL_STOPPED_BY_USER (2279, X'8E7'). Channel stopped by user.
Event description:	This is issued when a user issues a STOP CHL command. *ReasonQualifier* identifies the reasons for stopping.
Event type:	Channel.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ReasonQualifier

Description:	Identifier that qualifies the reason code.
Identifier:	MQIACF_REASON_QUALIFIER.
Data type:	MQCFIN.
Values:	**MQRQ_CHANNEL_STOPPED_DISABLED** Channel has been closed and it is in a stopped state.
Returned:	Always.

ChannelName

Description:	Channel name.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH.
Returned:	Always.

Command:

Event name:	Command.
Reason code in MQCFH:	MQRC_COMMAND_MQSC (2412, X'96C'). MQSC command successfully issued, or, MQRC_COMMAND_PCF (2413, X'96D'). PCF command successfully issued.
Event description:	Command successfully issued.
Event type:	Command.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.COMMAND.EVENT.

Event data

The event data consists of two groups, *CommandContext* and *CommandData*.

CommandContext

Description:	PCF group containing the elements related to the context of the issued command.
Identifier:	MQGACF_COMMAND_CONTEXT.
Data type:	MQCFGR.
PCF elements in group:	• *EventUserId* • *EventOrigin* • *EventQMgr* • *EventAccountingToken* • *EventIdentityData* • *EventApplType* • *EventApplName* • *EventApplOrigin* • *Command*
Returned:	Always.

EventUserId

Description:	The user ID that issued the command or call that generated the event. (This is the same user ID that is used to check the authority to issue the command; for commands received from a queue, this is also the user identifier (UserIdentifier) from the MQMD of the command message).
Identifier:	MQCACF_EVENT_USER_ID.
Data type:	MQCFST.
Maximum length:	MQ_USER_ID_LENGTH.
Returned:	Always.

EventOrigin

Description: The origin of the action causing the event.
Identifier: MQIACF_EVENT_ORIGIN.
Data type: MQCFIN.
Values:

MQEVO_CONSOLE
Console command.

MQEVO_INIT
Initialization input data set command.

MQEVO_MSG
Command message on SYSTEM.COMMAND.INPUT.

MQEVO_INTERNAL
Directly by queue manager.

MQEVO_OTHER
None of the above.

Returned: Always.

EventQMgr

Description: The queue manager where the command was entered. (The queue manager where the command is executed and that generates the event is in the MQMD of the event message).
Identifier: MQCACF_EVENT_Q_MGR.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

EventAccountingToken

Description: For commands received as a message (MQEVO_MSG), the accounting token (AccountingToken) from the MQMD of the command message.
Identifier: MQBACF_EVENT_ACCOUNTING_TOKEN.
Data type: MQCFBS.
Maximum length: MQ_ACCOUNTING_TOKEN_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventIdentityData

Description: For commands received as a message (MQEVO_MSG), application identity data (ApplIdentityData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_IDENTITY.
Data type: MQCFST.
Maximum length: MQ_APPL_IDENTITY_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplType

Description: For commands received as a message (MQEVO_MSG), the type of application (PutApplType) from the MQMD of the command message.
Identifier: MQIACF_EVENT_APPL_TYPE.
Data type: MQCFIN.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplName

Description: For commands received as a message (MQEVO_MSG), the name of the application (PutApplName) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplOrigin

Description: For commands received as a message (MQEVO_MSG), the application origin data (ApplOriginData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_ORIGIN.
Data type: MQCFST.
Maximum length: MQ_APPL_ORIGIN_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

Command

Description: The command code.
Identifier: MQIACF_COMMAND.
Data type: MQCFIN.
Values:

- If the event relates to a PCF command, then the value is that of the Command parameter in the MQCFH structure in the command message.
- If the event relates to an MQSC command, then the value is as follows:

MQCMD_ARCHIVE_LOG
ARCHIVE LOG

MQCMD_BACKUP_CF_STRUC
BACKUP CFSTRUCT

MQCMD_CHANGE_AUTH_INFO
ALTER AUTHINFO

MQCMD_CHANGE_BUFFER_POOL
ALTER BUFFPOOL

MQCMD_CHANGE_CF_STRUC
ALTER CFSTRUCT

MQCMD_CHANGE_CHANNEL
ALTER CHANNEL

MQCMD_CHANGE_COMM_INFO
ALTER COMMINFO

MQCMD_CHANGE_LISTENER
ALTER LISTENER

MQCMD_CHANGE_NAMELIST
ALTER NAMELIST

MQCMD_CHANGE_PAGE_SET
ALTER PSID

MQCMD_CHANGE_PROCESS
ALTER PROCESS

MQCMD_CHANGE_Q
ALTER QLOCAL/QREMOTE/QALIAS/QMODEL

MQCMD_CHANGE_Q_MGR
ALTER QMGR, DEFINE MAXSMSGS

MQCMD_CHANGE_SECURITY
ALTER SECURITY

MQCMD_CHANGE_SERVICE
ALTER SERVICE

MQCMD_CHANGE_STG_CLASS
ALTER STGCLASS

MQCMD_CHANGE_SUBSCRIPTION
ALTER SUBSCRIPTION

MQCMD_CHANGE_TOPIC
ALTER TOPIC

MQCMD_CHANGE_TRACE
ALTER TRACE

MQCMD_CLEAR_Q
CLEAR QLOCAL

MQCMD_CLEAR_TOPIC_STRING
CLEAR TOPICSTR

MQCMD_CREATE_AUTH_INFO
DEFINE AUTHINFO

MQCMD_CREATE_BUFFER_POOL
DEFINE BUFFPOOL

MQCMD_CREATE_CF_STRUC
DEFINE CFSTRUCT

MQCMD_CREATE_CHANNEL
DEFINE CHANNEL

MQCMD_CREATE_COMM_INFO
DEFINE COMMINFO

MQCMD_CREATE_LISTENER
DEFINE LISTENER

MQCMD_CREATE_NAMELIST
DEFINE NAMELIST

MQCMD_CREATE_PAGE_SET
DEFINE PSID

MQCMD_CREATE_PROCESS
DEFINE PROCESS

MQCMD_CREATE_Q
DEFINE QLOCAL/QREMOTE/QALIAS/QMODEL

MQCMD_CREATE_SERVICE
DEFINE SERVICE

MQCMD_CREATE_STG_CLASS
DEFINE STGCLASS

MQCMD_CREATE_SUBSCRIPTION
DEFINE SUB

MQCMD_CREATE_TOPIC
DEFINE TOPIC

MQCMD_DELETE_AUTH_INFO
DELETE AUTHINFO

MQCMD_DELETE_CF_STRUC
DELETE CFSTRUCT

MQCMD_DELETE_CHANNEL
DELETE CHANNEL

MQCMD_DELETE_COMM_INFO
DELETE COMMINFO

MQCMD_DELETE_LISTENER
DELETE LISTENER

MQCMD_DELETE_NAMELIST
DELETE NAMELIST

MQCMD_DELETE_PAGE_SET
DELETE PSID

MQCMD_DELETE_PROCESS
DELETE PROCESS

MQCMD_DELETE_Q
DELETE QLOCAL/QREMOTE/QALIAS/QMODEL

MQCMD_DELETE_SERVICE
DELETE SERVICE

MQCMD_DELETE_STG_CLASS
DELETE STGCLASS

MQCMD_DELETE_SUBSCRIPTION
DELETE SUBSCRIPTION

MQCMD_DELETE_TOPIC
DELETE TOPIC

MQCMD_INQUIRE_ARCHIVE
DISPLAY ARCHIVE

MQCMD_INQUIRE_AUTH_INFO
DISPLAY AUTHINFO

MQCMD_INQUIRE_CF_STRUC
DISPLAY CFSTRUCT

MQCMD_INQUIRE_CF_STRUC_STATUS
DISPLAY CFSTATUS

MQCMD_INQUIRE_CHANNEL
DISPLAY CHANNEL

MQCMD_INQUIRE_CHANNEL_INIT
DISPLAY CHINIT

MQCMD_INQUIRE_CHANNEL_STATUS
DISPLAY CHSTATUS

MQCMD_INQUIRE_CHLAUTH_RECS
DISPLAY CHLAUTH

MQCMD_INQUIRE_CLUSTER_Q_MGR
DISPLAY CLUSQMGR

MQCMD_INQUIRE_CMD_SERVER
DISPLAY CMDSERV

MQCMD_INQUIRE_COMM_INFO
DISPLAY COMMINFO

MQCMD_INQUIRE_CONNECTION
DISPLAY CONN

MQCMD_INQUIRE_LISTENER
DISPLAY LISTENER

MQCMD_INQUIRE_LOG
DISPLAY LOG

MQCMD_INQUIRE_NAMELIST
DISPLAY NAMELIST

MQCMD_INQUIRE_PROCESS
DISPLAY PROCESS

MQCMD_INQUIRE_PUBSUB_STATUS
DISPLAY PUBSUB

MQCMD_INQUIRE_Q
DISPLAY QUEUE

MQCMD_INQUIRE_Q_MGR
DISPLAY QMGR, DISPLAY MAXSMSGS

MQCMD_INQUIRE_QSG
DISPLAY GROUP

MQCMD_INQUIRE_Q_STATUS
DISPLAY QSTATUS

MQCMD_INQUIRE_SECURITY
DISPLAY SECURITY

MQCMD_INQUIRE_SERVICE
DISPLAY SERVICE

MQCMD_INQUIRE_STG_CLASS
DISPLAY STGCLASS

MQCMD_INQUIRE_SUBSCRIPTION
DISPLAY SUB

MQCMD_INQUIRE_SUB_STATUS
DISPLAY SBSTATUS

MQCMD_INQUIRE_SYSTEM
DISPLAY SYSTEM

MQCMD_INQUIRE_THREAD
DISPLAY THREAD

MQCMD_INQUIRE_TOPIC
DISPLAY TOPIC

MQCMD_INQUIRE_TOPIC_STATUS
DISPLAY TPSTATUS

MQCMD_INQUIRE_TRACE
DISPLAY TRACE

MQCMD_INQUIRE_USAGE
DISPLAY USAGE

MQCMD_MOVE_Q
MOVE QLOCAL

MQCMD_PING_CHANNEL
PING CHANNEL

MQCMD_RECOVER_BSDS
RECOVER BSDS

MQCMD_RECOVER_CF_STRUC
RECOVER CFSTRUCT

MQCMD_REFRESH_CLUSTER
REFRESH CLUSTER

MQCMD_REFRESH_Q_MGR
REFRESH QMGR

MQCMD_REFRESH_SECURITY
REFRESH SECURITY

MQCMD_RESET_CHANNEL
RESET CHANNEL

MQCMD_RESET_CLUSTER
RESET CLUSTER

MQCMD_RESET_Q_MGR
RESET QMGR

MQCMD_RESET_Q_STATS
RESET QSTATS

MQCMD_RESET_TPIPE
RESET TPIPE

MQCMD_RESOLVE_CHANNEL
RESOLVE CHANNEL

MQCMD_RESOLVE_INDOUBT
RESOLVE INDOUBT

MQCMD_RESUME_Q_MGR
RESUME QMGR other than CLUSTER/CLUSNL

MQCMD_RESUME_Q_MGR_CLUSTER
RESUME QMGR CLUSTER/CLUSNL

MQCMD_REVERIFY_SECURITY
REVERIFY SECURITY

MQCMD_SET_ARCHIVE
SET ARCHIVE

MQCMD_SET_CHLAUTH_REC
SET CHLAUTH

MQCMD_SET_LOG
SET LOG

MQCMD_SET_SYSTEM
SET SYSTEM

MQCMD_START_CHANNEL
START CHANNEL

MQCMD_START_CHANNEL_INIT
START CHINIT

MQCMD_START_CHANNEL_LISTENER
START LISTENER

MQCMD_START_CMD_SERVER
START CMDSERV

MQCMD_START_SERVICE
START SERVICE

MQCMD_START_TRACE
START TRACE

MQCMD_STOP_CHANNEL
STOP CHANNEL

MQCMD_STOP_CHANNEL_INIT
STOP CHINIT

MQCMD_STOP_CHANNEL_LISTENER
STOP LISTENER

MQCMD_STOP_CMD_SERVER
STOP CMDSERV

MQCMD_STOP_CONNECTION
STOP CONN

MQCMD_STOP_SERVICE
STOP SERVICE

MQCMD_STOP_TRACE
STOP TRACE

MQCMD_SUSPEND_Q_MGR
SUSPEND QMGR other than CLUSTER/CLUSNL

MQCMD_SUSPEND_Q_MGR_CLUSTER
SUSPEND QMGR CLUSTER/CLUSNL

Returned: Always.

CommandData

Description: PCF group containing the elements related to the command data.
Identifier: MQGACF_COMMAND_DATA.
Data type: MQCFGR.
PCF elements in group:
- If generated for an MQSC command, this group only contains the PCF element *CommandMQSC*.
- If generated for a PCF command, this group contains the PCF elements that make up the PCF command, exactly as in the command message.

Returned: Always.

CommandMQSC

Description: The text of the MQSC command.
Identifier: MQCACF_COMMAND_MQSC.
Data type: MQCFST.
Maximum length: MQ_COMMAND_MQSC_LENGTH.
Returned: Only if Reason in the message descriptor is MQRC_COMMAND_MQSC.

Create object:

Event name:	Create object.
Reason code in MQCFH:	MQRC_CONFIG_CREATE_OBJECT (2367, X'93F'). New object created.
Event description:	A DEFINE or DEFINE REPLACE command was issued which successfully created a new object.
Event type:	Configuration.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CONFIG.EVENT.

Event data

EventUserId

Description: The user ID that issued the command or call that generated the event. (This is the same user ID that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (UserIdentifier) from the MQMD of the command message).
Identifier: MQCACF_EVENT_USER_ID.
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

EventOrigin

Description: The origin of the action causing the event.
Identifier: MQIACF_EVENT_ORIGIN.
Data type: MQCFIN.
Values:

MQEVO_CONSOLE
Console command.

MQEVO_INIT
Initialization input data set command.

MQEVO_INTERNAL
Directly by queue manager.

MQEVO_MQSET
MQSET call.

MQEVO_MSG
Command message on SYSTEM.COMMAND.INPUT.

MQEVO_OTHER
None of the above.

Returned: Always.

EventQMgr

Description: The queue manager where the command or call was entered. (The queue manager where the command is executed and that generates the event is in the MQMD of the event message).
Identifier: MQCACF_EVENT_Q_MGR.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

EventAccountingToken

Description: For commands received as a message (MQEVO_MSG), the accounting token (AccountingToken) from the MQMD of the command message.
Identifier: MQBACF_EVENT_ACCOUNTING_TOKEN.
Data type: MQCFBS.
Maximum length: MQ_ACCOUNTING_TOKEN_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplIdentity

Description: For commands received as a message (MQEVO_MSG), application identity data (ApplIdentityData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_IDENTITY.
Data type: MQCFST.
Maximum length: MQ_APPL_IDENTITY_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplType

Description: For commands received as a message (MQEVO_MSG), the type of application (PutApplType) from the MQMD of the command message.
Identifier: MQIACF_EVENT_APPL_TYPE.
Data type: MQCFIN.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplName

Description: For commands received as a message (MQEVO_MSG), the name of the application (PutApplName) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplOrigin

Description: For commands received as a message (MQEVO_MSG), the application origin data (ApplOriginData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_ORIGIN.
Data type: MQCFST.
Maximum length: MQ_APPL_ORIGIN_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

ObjectType

Description: Object type:
Identifier: MQIACF_OBJECT_TYPE.
Data type: MQCFIN.

Values:

MQOT_CHANNEL
Channel.

MQOT_CHLAUTH
Channel authentication record.

MQOT_NAMELIST
Namelist.

MQOT_NONE
No object.

MQOT_PROCESS
Process.

MQOT_Q
Queue.

MQOT_STORAGE_CLASS
Storage class.

MQOT_AUTH_INFO
Authentication information.

MQOT_CF_STRUC
CF structure.

MQOT_TOPIC
Topic.

MQOT_COMM_INFO
Communication information.

MQOT_LISTENER
Channel Listener.

Returned: Always.

ObjectName

Description: Object name:
Identifier : Identifier will be according to object type.

- MQCACH_CHANNEL_NAME
- MQCA_NAMELIST_NAME
- MQCA_PROCESS_NAME
- MQCA_Q_NAME
- MQCA_STORAGE_CLASS
- MQCA_AUTH_INFO_NAME
- MQCA_CF_STRUC_NAME
- MQCA_TOPIC_NAME
- MQCA_COMM_INFO_NAME
- MQCACH_LISTENER_NAME

Note: MQCACH_CHANNEL_NAME can also be used for channel authentication.

Data type: MQCFST.
Maximum length: MQ_OBJECT_NAME_LENGTH.
Returned: Always

Disposition

Description:	Object disposition:
Identifier:	MQIA_QSG_DISP.
Data type:	MQCFIN.
Values:	**MQQSGD_Q_MGR** Object resides on page set of queue manager. **MQQSGD_SHARED** Object resides in shared repository and messages are shared in coupling facility. **MQQSGD_GROUP** Object resides in shared repository. **MQQSGD_COPY** Object resides on page set of queue manager and is a local copy of a GROUP object.
Returned:	Always, except for CF structure objects.

Object attributes

A parameter structure is returned for each attribute of the object. The attributes returned depend on the object type. For more information see "Object attributes for event data" on page 4151

Default Transmission Queue Type Error:

Event name:	Default Transmission Queue Type Error.
Reason code in MQCFH:	MQRC_DEF_XMIT_Q_TYPE_ERROR (2198, X'896'). Default transmission queue not local.
Event description:	An MQOPEN or MQPUT1 call was issued specifying a remote queue as the destination. Either a local definition of the remote queue was specified, or a queue manager alias was being resolved, but in either case the **XmitQName** attribute in the local definition is blank. No transmission queue is defined with the same name as the destination queue manager, so the local queue manager has attempted to use the default transmission queue. However, although there is a queue defined by the **DefXmitQName** queue manager attribute, it is not a local queue. See Defining a transmission queue for more information about transmission queues.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

XmitQName

Description: Default transmission queue name.
Identifier: MQCA_XMIT_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

QType

Description: Type of default transmission queue.
Identifier: MQIA_Q_TYPE.
Data type: MQCFIN.
Values:

MQQT_ALIAS
Alias queue definition.

MQQT_REMOTE
Local definition of a remote queue.

Returned: Always.

ApplType

Description: Type of application making the MQI call that caused the event.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

ConnName

Description:	Connection name for client connection.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description:	Channel name for client connection.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	If the application making the MQI call that caused the event is a client attached application.

Related information:

Defining a transmission queue

Sample definitions supplied with IBM MQ

DefXmitQName (MQCHAR48)

DefXmitQName (48-byte character string)
Default transmission queue name.

DefaultTransmissionQueueName property

Defining system objects

Default Transmission Queue Usage Error:

Event name:	Default Transmission Queue Usage Error.
Reason code in MQCFH:	MQRC_DEF_XMIT_Q_USAGE_ERROR (2199, X'897'). Default transmission queue usage error.
Event description:	An MQOPEN or MQPUT1 call was issued specifying a remote queue as the destination. Either a local definition of the remote queue was specified, or a queue manager alias was being resolved, but in either case the **XmitQName** attribute in the local definition is blank. No transmission queue is defined with the same name as the destination queue manager, so the local queue manager has attempted to use the default transmission queue. However, the queue defined by the **DefXmitQName** queue manager attribute does not have a **Usage** attribute of MQUS_TRANSMISSION. See the Defining a transmission queue for more information about default transmission queues.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

XmitQName

Description: Default transmission queue name.
Identifier: MQCA_XMIT_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ApplType

Description: Type of application making the MQI call that caused the event.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

ConnName

Description:	Connection name for client connection.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description:	Channel name for client connection.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	If the application making the MQI call that caused the event is a client attached application.

Related information:

Defining a transmission queue

Sample definitions supplied with IBM MQ

DefXmitQName (MQCHAR48)

DefXmitQName (48-byte character string)
Default transmission queue name.

DefaultTransmissionQueueName property

Defining system objects

Delete Authority Record:

Event name:	Delete Authority Record
Reason code in MQCFH:	MQRC_CONFIG_DELETE_OBJECT (2369, X'0941'). Object deleted.
Event description:	A Delete Authority Record command was issued, or an object was deleted, which successfully deleted an authority record.
Event type:	Configuration
Platforms:	All except z/OS.
Event queue:	SYSTEM.ADMIN.CONFIG.EVENT.

Event data

EventQMgr

Description:	The queue manager where the command or call was entered. That is, the queue manager where the command is processed and that generates the event is in the MQMD of the event message.
Identifier:	MQCACF_EVENT_Q_MGR
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

EventUserId

Description: The user ID that issued the command or call that generated the event.

This is the same user ID that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (**UserIdentifier**) from the message descriptor of the command message..

Identifier: MQCACF_EVENT_USER_ID
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

EventOrigin

Description: The origin of the action causing the event.
Identifier: MQIACF_EVENT_ORIGIN
Data type: MQCFIN.
Values:

MQEVO_CONSOLE
Console command (runmqsc or setmqaut)

MQEVO_INTERNAL
Directly by queue manager

MQEVO_MSG
Command message on SYSTEM.ADMIN.COMMAND.QUEUE

Returned: Always

EventAccountingToken

Description: For commands received as a message (MQEVO_MSG), the accounting token (**AccountingToken**) from the message descriptor of the command message.
Identifier: MQBACF_EVENT_ACCOUNTING_TOKEN
Data type: MQCFBS
Maximum length: MQ_ACCOUNTING_TOKEN_LENGTH
Returned: Only if **EventOrigin** is MQEVO_MSG.

EventApplIdentity

Description: For commands received as a message (MQEVO_MSG), application identity data (**ApplIdentityData**) from the message descriptor of the command message.
Identifier: MQMQCACF_EVENT_APPL_IDENTITY
Data type: MQCFST
Maximum length: MQ_APPL_IDENTITY_DATA_LENGTH
Returned: Only if **EventOrigin** is MQEVO_MSG.

EventApplType

Description: For commands received as a message (MQEVO_MSG), the type of application (**PutApplType**) from the message descriptor of the command message.
Identifier: MQIACF_EVENT_APPL_TYPE
Data type: MQCFIN
Values:
Returned: Only if **EventOrigin** is MQEVO_MSG.

EventApplName

Description:	For commands received as a message (MQEVO_MSG), the name of the application (**PutApplName**) from the message descriptor of the command message.
Identifier:	MQCACF_EVENT_APPL_NAME
Data type:	MQCFST
Maximum length:	MQ_APPL_NAME_LENGTH
Returned:	Only if **EventOrigin** is MQEVO_MSG.

EventApplOrigin

Description:	For commands received as a message (MQEVO_MSG), the application origin data (**ApplOriginData**) from the message descriptor of the command message.
Identifier:	MQCACF_EVENT_APPL_ORIGIN
Data type:	MQCFST
Maximum length:	MQ_APPL_ORIGIN_DATA_LENGTH
Returned:	Only if **EventOrigin** is MQEVO_MSG.

ObjectType

Description:	Object type
Identifier:	MQIACF_OBJECT_TYPE
Data type:	MQCFIN
Values:	MQOT_AUTH_REC
Returned:	Always

ProfileName

Description:	Object or generic profile name
Identifier:	MQCACF_AUTH_PROFILE_NAME
Data type:	MQCFST
Maximum length:	MQ_AUTH_PROFILE_NAME_LENGTH
Returned:	Always

Object attributes

A parameter structure is returned for each attribute of the object. The attributes returned depend on the object type. For more information see “Object attributes for event data” on page 4151.

Delete object:

Event name:	Delete object.
Reason code in MQCFH:	MQRC_CONFIG_DELETE_OBJECT (2369, X'941'). Object deleted.
Event description:	A DELETE command or MQCLOSE call was issued that successfully deleted an object.
Event type:	Configuration.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CONFIG.EVENT.

Event data

EventUserId

Description: The user ID that issued the command or call that generated the event. (This is the same user ID that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (UserIdentifier) from the MQMD of the command message).
Identifier: MQCACF_EVENT_USER_ID.
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

EventOrigin

Description: The origin of the action causing the event.
Identifier: MQIACF_EVENT_ORIGIN.
Data type: MQCFIN.
Values:

MQEVO_CONSOLE
Console command.

MQEVO_INIT
Initialization input data set command.

MQEVO_INTERNAL
Directly by queue manager.

MQEVO_MSG
Command message on SYSTEM.COMMAND.INPUT.

MQEVO_OTHER
None of the above.

Returned: Always.

EventQMgr

Description: The queue manager where the command or call was entered. (The queue manager where the command is executed and that generates the event is in the MQMD of the event message).
Identifier: MQCACF_EVENT_Q_MGR.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

EventAccountingToken

Description: For commands received as a message (MQEVO_MSG), the accounting token (AccountingToken) from the MQMD of the command message.
Identifier: MQBACF_EVENT_ACCOUNTING_TOKEN.
Data type: MQCFBS.
Maximum length: MQ_ACCOUNTING_TOKEN_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplIdentity

Description: For commands received as a message (MQEVO_MSG), application identity data (ApplIdentityData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_IDENTITY.
Data type: MQCFST.
Maximum length: MQ_APPL_IDENTITY_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplType

Description: For commands received as a message (MQEVO_MSG), the type of application (PutApplType) from the MQMD of the command message.
Identifier: MQIACF_EVENT_APPL_TYPE.
Data type: MQCFIN.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplName

Description: For commands received as a message (MQEVO_MSG), the name of the application (PutApplName) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplOrigin

Description: For commands received as a message (MQEVO_MSG), the application origin data (ApplOriginData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_ORIGIN.
Data type: MQCFST.
Maximum length: MQ_APPL_ORIGIN_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

ObjectType

Description: Object type:
Identifier: MQIACF_OBJECT_TYPE.
Data type: MQCFIN.

Values:

MQOT_CHANNEL
Channel.

MQOT_CHLAUTH
Channel authentication record.

MQOT_NAMELIST
Namelist.

MQOT_NONE
No object.

MQOT_PROCESS
Process.

MQOT_Q
Queue.

MQOT_STORAGE_CLASS
Storage class.

MQOT_AUTH_INFO
Authentication information.

MQOT_CF_STRUC
CF structure.

MQOT_TOPIC
Topic.

MQOT_COMM_INFO
Communication information.

MQOT_LISTENER
Channel Listener.

Returned: Always.

ObjectName

Description: Object name:
Identifier : Identifier will be according to object type.

- MQCACH_CHANNEL_NAME
- MQCA_NAMELIST_NAME
- MQCA_PROCESS_NAME
- MQCA_Q_NAME
- MQCA_STORAGE_CLASS
- MQCA_AUTH_INFO_NAME
- MQCA_CF_STRUC_NAME
- MQCA_TOPIC_NAME
- MQCA_COMM_INFO_NAME
- MQCACH_LISTENER_NAME

Note: MQCACH_CHANNEL_NAME can also be used for channel authentication.

Data type: MQCFST.
Maximum length: MQ_OBJECT_NAME_LENGTH.
Returned: Always

Disposition

Description: Object disposition:
Identifier: MQIA_QSG_DISP.
Data type: MQCFIN.
Values:

MQQSGD_Q_MGR
Object resides on page set of queue manager.

MQQSGD_SHARED
Object resides in shared repository and messages are shared in coupling facility.

MQQSGD_GROUP
Object resides in shared repository.

MQQSGD_COPY
Object resides on page set of queue manager and is a local copy of a GROUP object.

Returned: Always, except for CF structure objects.

Object attributes

A parameter structure is returned for each attribute of the object. The attributes returned depend on the object type. For more information see "Object attributes for event data" on page 4151.

Get Inhibited:

Event name:	Get Inhibited.
Reason code in MQCFH:	MQRC_GET_INHIBITED (2016, X'7E0'). Gets inhibited for the queue.
Event description:	MQGET calls are currently inhibited for the queue (see InhibitGet (MQLONG) for the **InhibitGet** queue attribute) or for the queue to which this queue resolves.
Event type:	Inhibit.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ApplType

Description: Type of application that issued the get.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application that issued the get.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Related information:
Setting queue attributes
InhibitGet property

IBM i InhibitGet (10-digit signed integer)
Controls whether get operations for this queue are allowed.

Logger:

Event name:	Logger.
Reason code in MQCFH:	MQRC_LOGGER_STATUS (2411, X'96B'). New log extent started.
Event description:	Issued when a queue manager starts writing to a new log extent IBM i or on IBM i a new journal receiver.
Event type:	Logger.
Platforms:	All, except IBM MQ for z/OS.
Event queue:	SYSTEM.ADMIN.LOGGER.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

CurrentLogExtent

Description:	Name of the log extent IBM i , or on IBM i the journal receiver being written, when the event message was generated.
Identifier:	MQCACF_CURRENT_LOG_EXTENT_NAME.
Data type:	MQCFST.
Maximum length:	MQ_LOG_EXTENT_NAME_LENGTH.
Returned:	Always.

RestartRecoveryLogExtent

Description:	Name of the oldest log extent IBM i , or on IBM i the oldest journal receiver, required by the queue manager to perform restart recovery.
Identifier:	MQCACF_RESTART_LOG_EXTENT_NAME.
Data type:	MQCFST.
Maximum length:	MQ_LOG_EXTENT_NAME_LENGTH.
Returned:	Always.

MediaRecoveryLogExtent

Description:	Name of the oldest log extent IBM i , or on IBM i the oldest journal receiver, required by the queue manager to perform media recovery.
Identifier:	MQCACF_MEDIA_LOG_EXTENT_NAME.
Data type:	MQCFST.
Maximum length:	MQ_LOG_EXTENT_NAME_LENGTH.
Returned:	Always.

LogPath

Description: The directory where log files are created by the queue manager.
Identifier: MQCACF_LOG_PATH.
Data type: MQCFST.
Maximum length: MQ_LOG_PATH_LENGTH.
Returned: Always.

Not Authorized (type 1):

Event name:	Not Authorized (type 1).
Reason code in MQCFH:	MQRC_NOT_AUTHORIZED (2035, X'7F3'). Not authorized for access.
Event description:	On an MQCONN or system connection call, the user is not authorized to connect to the queue manager. *ReasonQualifier* identifies the nature of the error.
Event type:	Authority.
Platforms:	All, except IBM MQ for z/OS.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier for type 1 authority events.
Identifier: MQIACF_REASON_QUALIFIER.
Data type: MQCFIN.
Values:

MQRQ_CONN_NOT_AUTHORIZED
Connection not authorized.

MQRQ_SYS_CONN_NOT_AUTHORIZED
Missing system authority.

MQRQ_CSP_NOT_AUTHORIZED
MQCSP user identifier and password not authorized.

Returned: Always.

UserIdentifier

Description: User identifier that caused the authorization check.
Identifier: MQCACF_USER_IDENTIFIER.
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

ApplType

Description: Type of application causing the event.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application causing the event.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

CSPUserIdentifier

Description: The user identifier from the Connection Security Parameters (MQCSP) structure.
Identifier: MQCACF_CSP_USER_IDENTIFIER.
Data type: MQCFST.
Maximum length: MQ_CLIENT_USER_ID_LENGTH
Returned: Only for MQRQ_CSP_NOT_AUTHORIZED.

Not Authorized (type 2):

Event name:	Not Authorized (type 2).
Reason code in MQCFH:	MQRC_NOT_AUTHORIZED (2035, X'7F3'). Not authorized for access.
Event description:	On an MQOPEN or MQPUT1 call, the user is not authorized to open the object for the options specified.
Event type:	Authority.
Platforms:	All, except IBM MQ for z/OS.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ReasonQualifier

Description:	Identifier for type 2 authority events.	
Identifier:	MQIACF_REASON_QUALIFIER.	
Data type:	MQCFIN.	
Values:	MQRQ_OPEN_NOT_AUTHORIZED	Open not authorized.
Returned:	Always.	

Options

Description:	Options specified on the MQOPEN call.
Identifier:	MQIACF_OPEN_OPTIONS.
Data type:	MQCFIN.
Returned:	Always.

UserIdentifier

Description:	User identifier that caused the authorization check.
Identifier:	MQCACF_USER_IDENTIFIER.
Data type:	MQCFST.
Maximum length:	MQ_USER_ID_LENGTH.
Returned:	Always.

ApplType

Description: Type of application that caused the authorization check.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application that caused the authorization check.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Object queue manager name from object descriptor (MQOD).
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectQMgrName* in the object descriptor (MQOD) when the object was opened is not the queue manager currently connected.

QName

Description: Object name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: If the object opened is a queue object.

ProcessName

Description: Name of process object from object descriptor (MQOD).
Identifier: MQCA_PROCESS_NAME.
Data type: MQCFST.
Maximum length: MQ_PROCESS_NAME_LENGTH.
Returned: If the object opened is a process object.

TopicString

Description: Topic string being subscribed to, or opened.
Identifier: MQCA_TOPIC_STRING.
Data type: MQCFST.
Maximum length: MQ_TOPIC_STR_LENGTH.
Returned: If the object opened is a topic object.

AdminTopicNames

Description: List of topic admin objects against which authority is checked.
Identifier: MQCA_ADMIN_TOPIC_NAME.
Data type: MQCFSL.
Maximum length: MQ_TOPIC_NAME_LENGTH.
Returned: If the object opened is a topic object.

ObjectType

Description: Object type from object descriptor (MQOD).
Identifier: MQIACF_OBJECT_TYPE.
Data type: MQCFIN.
Values:

MQOT_NAMELIST	Namelist.
MQOT_PROCESS	Process.
MQOT_Q	Queue.
MQOT_Q_MGR	Queue manager.
MQOT_TOPIC	Topic.

Returned: Always.

NamelistName

Description: Object name from object descriptor (MQOD).
Identifier: MQCA_NAMELIST_NAME.
Data type: MQCFST.
Maximum length: MQ_NAMELIST_NAME_LENGTH.
Returned: If the object opened is a namelist object.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Not Authorized (type 3):

Event name:	Not Authorized (type 3).
Reason code in MQCFH:	MQRC_NOT_AUTHORIZED (2035, X'7F3'). Not authorized for access.
Event description:	When closing a queue using the MQCLOSE call, the user is not authorized to delete the object, which is a permanent dynamic queue, and the **Hobj** parameter specified on the MQCLOSE call is not the handle returned by the MQOPEN call that created the queue. When closing a subscription using an MQCLOSE call, the user has requested that the subscription is removed using the MQCO_REMOVE_SUB option, but the user is not the creator of the subscription or does not have *sub* authority on the topic associated with the subscription.
Event type:	Authority.
Platforms:	All, except IBM MQ for z/OS.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ReasonQualifier

Description:	Identifier for type 3 authority events.
Identifier:	MQIACF_REASON_QUALIFIER.
Data type:	MQCFIN.
Values:	**MQRQ_CLOSE_NOT_AUTHORIZED** Close not authorized.
Returned:	Always.

UserIdentifier

Description:	User identifier that caused the authorization check
Identifier:	MQCACF_USER_IDENTIFIER
Data type:	MQCFST.
Maximum length:	MQ_USER_ID_LENGTH.
Returned:	Always.

ApplType

Description: Type of application causing the authorization check.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application causing the authorization check.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

QName

Description: Object name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: If the handle being closed is a queue

SubName

Description: Name of subscription being removed.
Identifier: MQCACF_SUB_NAME.
Data type: MQCFST.
Maximum length: MQ_SUB_NAME_LENGTH.
Returned: If the handle being closed is a subscription.

TopicString

Description: Topic string of the subscription.
Identifier: MQCA_TOPIC_STRING
Data type: MQCFST.
Maximum length: MQ_TOPIC_STR_LENGTH.
Returned: If the handle being closed is a subscription.

AdminTopicNames

Description: List of topic administration objects against which authority was checked.
Identifier: MQCA_ADMIN_TOPIC_NAME
Data type: MQCFSL.
Maximum length: MQ_TOPIC_NAME_LENGTH.
Returned: If the handle being closed is a subscription.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Not Authorized (type 4):

Event name:	Not Authorized (type 4).
Reason code in MQCFH:	MQRC_NOT_AUTHORIZED (2035, X'7F3'). Not authorized for access.
Event description:	Indicates that a command has been issued from a user ID that is not authorized to access the object specified in the command.
Event type:	Authority.
Platforms:	All, except IBM MQ for z/OS.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier for type 4 authority events.
Identifier: MQIACF_REASON_QUALIFIER.
Data type: MQCFIN.
Values:

MQRQ_CMD_NOT_AUTHORIZED
Command not authorized.

Returned: Always.

Command

Description: Command identifier. See the MQCFH header structure, described in "Event message MQCFH (PCF header)" on page 4205.
Identifier: MQIACF_COMMAND.
Data type: MQCFIN.
Returned: Always.

UserIdentifier

Description: User identifier that caused the authorization check.
Identifier: MQCACF_USER_IDENTIFIER.
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

Not Authorized (type 5):

Event name:	Not Authorized (type 5).
Reason code in MQCFH:	MQRC_NOT_AUTHORIZED (2035, X'7F3'). Not authorized for access.
Event description:	On an MQSUB call, the user is not authorized to subscribe to the specified topic.
Event type:	Authority.
Platforms:	All, except IBM MQ for z/OS.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier for type 5 authority events.
Identifier: MQIACF_REASON_QUALIFIER.
Data type: MQCFIN.
Values:

MQRQ_SUB_NOT_AUTHORIZED
Subscribe not authorized.

Returned: Always.

Options

Description: Options specified on the MQSUB call.
Identifier: MQIACF_SUB_OPTIONS
Data type: MQCFIN.
Returned: Always.

UserIdentifier

Description: User identifier that caused the authorization check.
Identifier: MQCACF_USER_IDENTIFIER.
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

ApplType

Description: Type of application that caused the authorization check.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application that caused the authorization check.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

TopicString

Description: Topic string being opened or subscribed to.
Identifier: MQCA_TOPIC_STRING.
Data type: MQCFST.
Maximum length: MQ_TOPIC_STR_LENGTH.
Returned: Always.

AdminTopicNames

Description: List of topic administration objects against which authority is checked.
Identifier: MQCA_ADMIN_TOPIC_NAME.
Data type: MQCFSL.
Maximum length: MQ_TOPIC_NAME_LENGTH.
Returned: Always.

ConnName

Description:	Connection name for client connection.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description:	Channel name for client connection.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	If the application making the MQI call that caused the event is a client attached application.

Not Authorized (type 6):

Event name:	Not Authorized (type 6).
Reason code in MQCFH:	MQRC_NOT_AUTHORIZED (2035, X'7F3'). Not authorized for access.
Event description:	On an MQSUB call, the user is not authorized to use the destination queue with the required level of access. This event is only returned for subscriptions using non-managed destination queues. When creating, altering, or resuming a subscription, and a handle to the destination queue is supplied on the request, the user does not have PUT authority on the destination queue provided. When resuming or alerting a subscription and the handle to the destination queue is to be returned on the MQSUB call, and the user does not have PUT, GET and BROWSE authority on the destination queue.
Event type:	Authority.
Platforms:	All, except IBM MQ for z/OS.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ReasonQualifier

Description: Identifier for type 6 authority events.
Identifier: MQIACF_REASON_QUALIFIER.
Data type: MQCFIN.
Values:

MQRQ_SUB_DEST_NOT_AUTHORIZED
Subscription destination queue usage not authorized.

Returned: Always.

Options

Description: Options specified on the MQSUB call.
Identifier: MQIACF_SUB_OPTIONS
Data type: MQCFIN.
Returned: Always.

UserIdentifier

Description: User identifier that caused the authorization check.
Identifier: MQCACF_USER_IDENTIFIER.
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

ApplType

Description: Type of application that caused the authorization check.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application that caused the authorization check.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

TopicString

Description: Topic string being subscribed to.
Identifier: MQCA_TOPIC_STRING.
Data type: MQCFST.
Maximum length: MQ_TOPIC_STR_LENGTH.
Returned: Always.

DestQMgrName

Description: Hosting queue manager name of the subscription's destination queue.
Identifier: MQCACF_OBJECT_Q_MGR_NAME
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the queue manager hosting the destination queue is not the queue manager to which the application is currently connected.

DestQName

Description: The name of the destination queue of the subscription..
Identifier: MQCA_Q_NAME
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

DestOpenOptions

Description: The open options requested for the destination queue.
Identifier: MQIACF_OPEN_OPTIONS
Data type: MQCFIN.
Returned: Always.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Put Inhibited:

Event name:	Put Inhibited.
Reason code in MQCFH:	MQRC_PUT_INHIBITED (2051, X'803'). Put calls inhibited for the queue or topic.
Event description:	MQPUT and MQPUT1 calls are currently inhibited for the queue or topic (see the **InhibitPut** queue attribute in InhibitPut (MQLONG) or the **InhibitPublications** topic attribute in "Topic attributes" on page 4194 for the queue to which this queue resolves.
Event type:	Inhibit.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	If the object opened is a queue object

ApplType

Description:	Type of application that issued the put.
Identifier:	MQIA_APPL_TYPE.
Data type:	MQCFIN.
Returned:	Always.

ApplName

Description:	Name of the application that issued the put.
Identifier:	MQCACF_APPL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Always.

ObjectQMgrName

Description:	Name of queue manager from object descriptor (MQOD).
Identifier:	MQCACF_OBJECT_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Only if this parameter has a value different from *QMgrName*. This occurs when the *ObjectQMgrName* field in the object descriptor provided by the application on the MQOPEN or MQPUT1 call is neither blank nor the name of the application's local queue manager. However, it can also occur when *ObjectQMgrName* in the object descriptor is blank, but a name service provides a queue manager name that is not the name of the application's local queue manager.

TopicString

Description:	Topic String being opened
Identifier:	MQCA_TOPIC_STRING
Data type:	MQCFST.
Maximum length:	MQ_TOPIC_STR_LENGTH.
Returned:	If the object opened is a topic.

ConnName

Description:	Connection name for client connection.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description:	Channel name for client connection.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	If the application making the MQI call that caused the event is a client attached application.

Related information:

InhibitPut property

IBM i InhibitPut (10-digit signed integer)
Controls whether put operations for this queue are allowed.

Inquire Queue (Response)
The response to the Inquire Queue command MQCMD_INQUIRE_Q consists of the response header followed by the *QName* structure. On z/OS only, response includes the *QSGDisposition* structure, and the requested combination of attribute parameter structures.

Inquire Topic (Response)
The response to the Inquire Topic (MQCMD_INQUIRE_TOPIC) command consists of the response header followed by the *TopicName* structure (and on z/OS only, the *QSG Disposition* structure), and the requested combination of attribute parameter structures (where applicable).

Inquire Topic Status (Response)
The response of the Inquire topic (MQCMD_INQUIRE_TOPIC_STATUS) command consists of the response header, followed by the *TopicString* structure, and the requested combination of attribute parameter structures (where applicable). The Inquire Topic Status command returns the values requested when the *StatusType* is MQIACF_TOPIC_STATUS. The Inquire Topic Status command returns the values requested when the *StatusType* is MQIACF_TOPIC_STATUS_SUB. The Inquire Topic Status command returns the values requested when the *StatusType* is MQIACF_TOPIC_STATUS_PUB.

Change, Copy, and Create Topic
The Change Topic command changes existing topic definitions. The Copy and Create Topic commands create new topic definitions - the Copy command uses attribute values of an existing topic definition.

Queue Depth High:

Event name:	Queue Depth High.
Reason code in MQCFH:	MQRC_Q_DEPTH_HIGH (2224, X'8B0'). Queue depth high limit reached or exceeded.
Event description:	An MQPUT or MQPUT1 call has caused the queue depth to be incremented to or above the limit specified in the **QDepthHighLimit** attribute.
Event type:	Performance.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.PERFM.EVENT.

Note:

1. IBM MQ for z/OS supports queue depth events on shared queues. You might receive a NULL event message for a shared queue if a queue manager has performed no activity on that shared queue.
2. For shared queues, the correlation identifier, *CorrelId* in the message descriptor (MQMD) is set. See "Event message MQMD (message descriptor)" on page 4201 for more information.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Name of the queue on which the limit has been reached.
Identifier:	MQCA_BASE_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

TimeSinceReset

Description:	Time, in seconds, since the statistics were last reset. The value recorded by this timer is also used as the *interval time* in queue service interval events.
Identifier:	MQIA_TIME_SINCE_RESET.
Data type:	MQCFIN.
Returned:	Always.

HighQDepth

Description:	Maximum number of messages on the queue since the queue statistics were last reset.
Identifier:	MQIA_HIGH_Q_DEPTH.
Data type:	MQCFIN.
Returned:	Always.

MsgEnqCount

Description:	Number of messages enqueued. This is the number of messages put on the queue since the queue statistics were last reset.
Identifier:	MQIA_MSG_ENQ_COUNT.
Data type:	MQCFIN.
Returned:	Always.

MsgDeqCount

Description:	Number of messages removed from the queue since the queue statistics were last reset.
Identifier:	MQIA_MSG_DEQ_COUNT.
Data type:	MQCFIN.
Returned:	Always.

Queue Depth Low:

Event name:	Queue Depth Low.
Reason code in MQCFH:	MQRC_Q_DEPTH_LOW (2225, X'8B1'). Queue depth low limit reached or exceeded.
Event description:	A get operation has caused the queue depth to be decremented to or below the limit specified in the **QDepthLowLimit** attribute.
Event type:	Performance.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.PERFM.EVENT.

Note:

1. IBM MQ for z/OS supports queue depth events on shared queues. You might receive a NULL event message for a shared queue if a queue manager has performed no activity on that shared queue.
2. For shared queues, the correlation identifier, *CorrelId* in the message descriptor (MQMD) is set. See "Event message MQMD (message descriptor)" on page 4201 for more information.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description: Name of the queue on which the limit has been reached.
Identifier: MQCA_BASE_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

TimeSinceReset

Description: Time, in seconds, since the statistics were last reset. The value recorded by this timer is also used as the *interval time* in queue service interval events.
Identifier: MQIA_TIME_SINCE_RESET.
Data type: MQCFIN.
Returned: Always.

HighQDepth

Description: Maximum number of messages on the queue since the queue statistics were last reset.
Identifier: MQIA_HIGH_Q_DEPTH.
Data type: MQCFIN.
Returned: Always.

MsgEnqCount

Description: Number of messages enqueued. This is the number of messages put on the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_ENQ_COUNT.
Data type: MQCFIN.
Returned: Always.

MsgDeqCount

Description: Number of messages removed from the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_DEQ_COUNT.
Data type: MQCFIN.
Returned: Always.

Queue Full:

Event name:	Queue Full.
Reason code in MQCFH:	MQRC_Q_FULL (2053, X'805'). Queue already contains maximum number of messages.
Event description:	On an MQPUT or MQPUT1 call, the call failed because the queue is full. That is, it already contains the maximum number of messages possible (see the *MaxQDepth* local-queue attribute
Event type:	Performance.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.PERFM.EVENT.

Note:

1. IBM MQ for z/OS supports queue depth events on shared queues. You might receive a NULL event message for a shared queue if a queue manager has performed no activity on that shared queue.

2. For shared queues, the correlation identifier, *CorrelId* in the message descriptor (MQMD) is set. See "Event message MQMD (message descriptor)" on page 4201 for more information.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Name of the queue on which the put was rejected.
Identifier: MQCA_BASE_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

TimeSinceReset

Description: Time, in seconds, since the statistics were last reset.
Identifier: MQIA_TIME_SINCE_RESET.
Data type: MQCFIN.
Returned: Always.

HighQDepth

Description: Maximum number of messages on a queue.
Identifier: MQIA_HIGH_Q_DEPTH.
Data type: MQCFIN.
Returned: Always.

MsgEnqCount

Description: Number of messages enqueued. This is the number of messages put on the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_ENQ_COUNT.
Data type: MQCFIN.
Returned: Always.

MsgDeqCount

Description: Number of messages removed from the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_DEQ_COUNT.
Data type: MQCFIN.
Returned: Always.

Queue Manager Active:

Event name:	Queue Manager Active.
Reason code in MQCFH:	MQRC_Q_MGR_ACTIVE (2222, X'8AE'). Queue manager active.
Event description:	This condition is detected when a queue manager becomes active.
Event type:	Start And Stop.
Platforms:	All, except the first start of an IBM MQ for z/OS queue manager. In this case it is produced only on subsequent restarts. The *ReasonQualifier* and *HostName* fields apply only to those platforms that support multi-instance availability; that is not z/OS
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ReasonQualifier

Description:	Identifier of causes for this reason code. This specifies the type of start that is happening.
Identifier:	MQIACF_REASON_QUALIFIER.
Data type:	MQCFIN.
Values:	**MQRQ_FAILOVER_PERMITTED** Queue manager has started normally and allows a standby instance. **MQRQ_FAILOVER_NOT_PERMITTED** Queue manager has started normally but does not allow a standby instance. **MQRQ_STANDBY_ACTIVATED** Queue manager has moved out of standby mode into active mode.
Returned:	Always.

HostName

Description:	The host name of the machine on which the queue manager is running.
Identifier:	MQCACF_HOST_NAME.
Data type:	MQCFST.
Returned:	Always.

Queue Manager Not Active:

Event name:	Queue Manager Not Active.
Reason code in MQCFH:	MQRC_Q_MGR_NOT_ACTIVE (2223, X'8AF'). Queue manager unavailable.
Event description:	This condition is detected when a queue manager is requested to stop or quiesce.
Event type:	Start And Stop.
Platforms:	All, except IBM MQ for z/OS.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier of causes of this reason code. This specifies the type of stop that was requested.
Identifier: MQIACF_REASON_QUALIFIER.
Data type: MQCFIN.
Values:

MQRQ_Q_MGR_STOPPING
Queue manager stopping.

MQRQ_Q_MGR_QUIESCING
Queue manager quiescing.

Returned: Always.

Queue Service Interval High:

Event name:	Queue Service Interval High.
Reason code in MQCFH:	MQRC_Q_SERVICE_INTERVAL_HIGH (2226, X'8B2'). Queue service interval high.
Event description:	No successful get operations or MQPUT calls have been detected within an interval greater than the limit specified in the **QServiceInterval** attribute.
Event type:	Performance.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.PERFM.EVENT.

Note: IBM MQ for z/OS does not support service interval events on shared queues.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Name of the queue specified on the command that caused this queue service interval event to be generated.
Identifier: MQCA_BASE_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

TimeSinceReset

Description: Time, in seconds, since the statistics were last reset. For a service interval high event, this value is greater than the service interval.
Identifier: MQIA_TIME_SINCE_RESET.
Data type: MQCFIN.
Returned: Always.

HighQDepth

Description: Maximum number of messages on the queue since the queue statistics were last reset.
Identifier: MQIA_HIGH_Q_DEPTH.
Data type: MQCFIN.
Returned: Always.

MsgEnqCount

Description: Number of messages enqueued. This is the number of messages put on the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_ENQ_COUNT.
Data type: MQCFIN.
Returned: Always.

MsgDeqCount

Description: Number of messages removed from the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_DEQ_COUNT.
Data type: MQCFIN.
Returned: Always.

Queue Service Interval OK:

Event name:	Queue Service Interval OK.
Reason code in MQCFH:	MQRC_Q_SERVICE_INTERVAL_OK (2227, X'8B3'). Queue service interval OK.
Event description:	A successful get operation has been detected within an interval less than or equal to the limit specified in the **QServiceInterval** attribute.
Event type:	Performance.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.PERFM.EVENT.

Note: IBM MQ for z/OS does not support service interval events on shared queues.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name specified on the command that caused this queue service interval event to be generated.
Identifier:	MQCA_BASE_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

TimeSinceReset

Description:	Time, in seconds, since the statistics were last reset.
Identifier:	MQIA_TIME_SINCE_RESET.
Data type:	MQCFIN.
Returned:	Always.

HighQDepth

Description:	Maximum number of messages on the queue since the queue statistics were last reset.
Identifier:	MQIA_HIGH_Q_DEPTH.
Data type:	MQCFIN.
Returned:	Always.

MsgEnqCount

Description: Number of messages enqueued. This is the number of messages put on the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_ENQ_COUNT.
Data type: MQCFIN.
Returned: Always.

MsgDeqCount

Description: Number of messages removed from the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_DEQ_COUNT.
Data type: MQCFIN.
Returned: Always.

Queue Type Error:

Event name:	Queue Type Error.
Reason code in MQCFH:	MQRC_Q_TYPE_ERROR (2057, X'809'). Queue type not valid.
Event description:	On an MQOPEN call, the *ObjectQMgrName* field in the object descriptor specifies the name of a local definition of a remote queue (in order to specify a queue manager alias). In that local definition the **RemoteQMgrName** attribute is the name of the local queue manager. However, the *ObjectName* field specifies the name of a model queue on the local queue manager, which is not allowed. See the Queue manager events for more information.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ApplType

Description:	Type of application making the MQI call that caused the event.
Identifier:	MQIA_APPL_TYPE.
Data type:	MQCFIN.
Returned:	Always.

ApplName

Description:	Name of the application making the MQI call that caused the event.
Identifier:	MQCACF_APPL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Always.

ObjectQMgrName

Description:	Name of the object queue manager.
Identifier:	MQCACF_OBJECT_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ConnName

Description:	Connection name for client connection.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description:	Channel name for client connection.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	If the application making the MQI call that caused the event is a client attached application.

Refresh Authority Record:

Event name:	Refresh Authority Record
Reason code in MQCFH:	MQRC_CONFIG_REFRESH_OBJECT (2370, X'0942'). Refresh queue manager configuration - authority records.
Event description:	A REFRESH QMGR command specifying **TYPE**(*CONFIGEV*) was issued.
Event type:	Configuration
Platforms:	All except z/OS.
Event queue:	SYSTEM.ADMIN.CONFIG.EVENT.

Note that the REFRESH QMGR command can produce many configuration events; one event is generated for each authority record that is selected by the command.

Event data

EventQMgr

Description: The queue manager where the command or call was entered. That is, the queue manager where the command is processed and that generates the event is in the MQMD of the event message.
Identifier: MQCACF_EVENT_Q_MGR
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

EventUserId

Description: The user ID that issued the command or call that generated the event.

This is the same user ID that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (**UserIdentifier**) from the message descriptor of the command message..
Identifier: MQCACF_EVENT_USER_ID
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

EventOrigin

Description: The origin of the action causing the event.
Identifier: MQIACF_EVENT_ORIGIN
Data type: MQCFIN.
Values:

MQEVO_CONSOLE
Console command (runmqsc or setmqaut)

MQEVO_INTERNAL
Directly by queue manager

MQEVO_MSG
Command message on SYSTEM.ADMIN.COMMAND.QUEUE

Returned: Always

EventAccountingToken

Description: For commands received as a message (MQEVO_MSG), the accounting token (**AccountingToken**) from the message descriptor of the command message.
Identifier: MQBACF_EVENT_ACCOUNTING_TOKEN
Data type: MQCFBS
Maximum length: MQ_ACCOUNTING_TOKEN_LENGTH
Returned: Only if **EventOrigin** is MQEVO_MSG.

EventApplIdentity

Description: For commands received as a message (MQEVO_MSG), application identity data (**ApplIdentityData**) from the message descriptor of the command message.
Identifier: MQMQCACF_EVENT_APPL_IDENTITY
Data type: MQCFST
Maximum length: MQ_APPL_IDENTITY_DATA_LENGTH
Returned: Only if **EventOrigin** is MQEVO_MSG.

EventApplType

Description: For commands received as a message (MQEVO_MSG), the type of application (**PutApplType**) from the message descriptor of the command message.
Identifier: MQIACF_EVENT_APPL_TYPE
Data type: MQCFIN
Values:
Returned: Only if **EventOrigin** is MQEVO_MSG.

EventApplName

Description: For commands received as a message (MQEVO_MSG), the name of the application (**PutApplName**) from the message descriptor of the command message.
Identifier: MQCACF_EVENT_APPL_NAME
Data type: MQCFST
Maximum length: MQ_APPL_NAME_LENGTH
Returned: Only if **EventOrigin** is MQEVO_MSG.

EventApplOrigin

Description: For commands received as a message (MQEVO_MSG), the application origin data (**ApplOriginData**) from the message descriptor of the command message.
Identifier: MQCACF_EVENT_APPL_ORIGIN
Data type: MQCFST
Maximum length: MQ_APPL_ORIGIN_DATA_LENGTH
Returned: Only if **EventOrigin** is MQEVO_MSG.

ObjectType

Description: Object type
Identifier: MQIACF_OBJECT_TYPE
Data type: MQCFIN
Values: MQOT_AUTH_REC
Returned: Always

ProfileName

Description: Object or generic profile name
Identifier: MQCACF_AUTH_PROFILE_NAME
Data type: MQCFST
Maximum length: MQ_AUTH_PROFILE_NAME_LENGTH
Returned: Always

Object attributes

A parameter structure is returned for each attribute of the object. The attributes returned depend on the object type. For more information see "Object attributes for event data" on page 4151.

Refresh object:

Event name:	Refresh object.
Reason code in MQCFH:	MQRC_CONFIG_REFRESH_OBJECT (2370, X'942'). Refresh queue manager configuration.
Event description:	A REFRESH QMGR command specifying TYPE (CONFIGEV) was issued.
Event type:	Configuration.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CONFIG.EVENT.

Note: The REFRESH QMGR command can produce many configuration events; one event is generated for each object that is selected by the command.

Event data

EventUserId

Description: The user ID that issued the command or call that generated the event. (This is the same user ID that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (UserIdentifier) from the MQMD of the command message).
Identifier: MQCACF_EVENT_USER_ID.
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

EventOrigin

Description: The origin of the action causing the event.
Identifier: MQIACF_EVENT_ORIGIN.
Data type: MQCFIN.
Values:

MQEVO_CONSOLE
Console command.

MQEVO_INIT
Initialization input data set command.

MQEVO_INTERNAL
Directly by queue manager.

MQEVO_MSG
Command message on SYSTEM.COMMAND.INPUT.

MQEVO_OTHER
None of the above.

Returned: Always.

EventQMgr

Description: The queue manager where the command or call was entered. (The queue manager where the command is executed and that generates the event is in the MQMD of the event message).
Identifier: MQCACF_EVENT_Q_MGR.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

EventAccountingToken

Description: For commands received as a message (MQEVO_MSG), the accounting token (AccountingToken) from the MQMD of the command message.
Identifier: MQBACF_EVENT_ACCOUNTING_TOKEN.
Data type: MQCFBS.
Maximum length: MQ_ACCOUNTING_TOKEN_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplIdentity

Description: For commands received as a message (MQEVO_MSG), application identity data (ApplIdentityData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_IDENTITY.
Data type: MQCFST.
Maximum length: MQ_APPL_IDENTITY_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplType

Description: For commands received as a message (MQEVO_MSG), the type of application (PutApplType) from the MQMD of the command message.
Identifier: MQIACF_EVENT_APPL_TYPE.
Data type: MQCFIN.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplName

Description: For commands received as a message (MQEVO_MSG), the name of the application (PutApplName) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplOrigin

Description: For commands received as a message (MQEVO_MSG), the application origin data (ApplOriginData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_ORIGIN.
Data type: MQCFST.
Maximum length: MQ_APPL_ORIGIN_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

ObjectType

Description: Object type:
Identifier: MQIACF_OBJECT_TYPE.
Data type: MQCFIN.
Values:

MQOT_CHANNEL
Channel.

MQOT_CHLAUTH
Channel authentication record.

MQOT_NAMELIST
Namelist.

MQOT_NONE
No object.

MQOT_PROCESS
Process.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_STORAGE_CLASS
Storage class.

MQOT_AUTH_INFO
Authentication information.

MQOT_CF_STRUC
CF structure.

MQOT_TOPIC
Topic.

MQOT_COMM_INFO
Communication information.

MQOT_LISTENER
Channel Listener.

Returned: Always.

ObjectName

Description: Object name:
Identifier : Identifier will be according to object type.

- MQCACH_CHANNEL_NAME
- MQCA_NAMELIST_NAME
- MQCA_PROCESS_NAME
- MQCA_Q_NAME
- MQCA_Q_MGR_NAME
- MQCA_STORAGE_CLASS
- MQCA_AUTH_INFO_NAME
- MQCA_CF_STRUC_NAME
- MQCA_TOPIC_NAME
- MQCA_COMM_INFO_NAME
- MQCACH_LISTENER_NAME

Note: MQCACH_CHANNEL_NAME can also be used for channel authentication.

Data type: MQCFST.
Maximum length: MQ_OBJECT_NAME_LENGTH.

Returned: Always

Disposition

Description: Object disposition:
Identifier: MQIA_QSG_DISP.
Data type: MQCFIN.
Values:

MQQSGD_Q_MGR
Object resides on page set of queue manager.

MQQSGD_SHARED
Object resides in shared repository and messages are shared in coupling facility.

MQQSGD_GROUP
Object resides in shared repository.

MQQSGD_COPY
Object resides on page set of queue manager and is a local copy of a GROUP object.

Returned: Always, except for queue manager and CF structure objects.

Object attributes

A parameter structure is returned for each attribute of the object. The attributes returned depend on the object type. For more information see "Object attributes for event data" on page 4151.

Remote Queue Name Error:

Event name:	Remote Queue Name Error.
Reason code in MQCFH:	MQRC_REMOTE_Q_NAME_ERROR (2184, X'888'). Remote queue name not valid.
Event description:	On an MQOPEN or MQPUT1 call one of the following occurs: • A local definition of a remote queue (or an alias to one) was specified, but the **RemoteQName** attribute in the remote queue definition is blank. Note that this error occurs even if the *XmitQName* in the definition is not blank. • The *ObjectQMgrName* field in the object descriptor is not blank and not the name of the local queue manager, but the *ObjectName* field is blank.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ApplType

Description: Type of application making the MQI call that caused the event.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description:	Channel name for client connection.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	If the application making the MQI call that caused the event is a client attached application.

Transmission Queue Type Error:

Event name:	Transmission Queue Type Error.
Reason code in MQCFH:	MQRC_XMIT_Q_TYPE_ERROR (2091, X'82B'). Transmission queue not local.
Event description:	On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager. The *ObjectName* or *ObjectQMgrName* field in the object descriptor specifies the name of a local definition of a remote queue but one of the following applies to the **XmitQName** attribute of the definition. Either: • *XmitQName* is not blank, but specifies a queue that is not a local queue, or • *XmitQName* is blank, but *RemoteQMgrName* specifies a queue that is not a local queue This also occurs if the queue name is resolved through a cell directory, and the remote queue manager name obtained from the cell directory is the name of a queue, but this is not a local queue.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

XmitQName

Description: Transmission queue name.
Identifier: MQCA_XMIT_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

QType

Description: Type of transmission queue.
Identifier: MQIA_Q_TYPE.
Data type: MQCFIN.
Values:

MQQT_ALIAS
Alias queue definition.

MQQT_REMOTE
Local definition of a remote queue.

Returned: Always.

ApplType

Description: Type of application making the MQI call that caused the event.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description:	Channel name for client connection.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	If the application making the MQI call that caused the event is a client attached application.

Transmission Queue Usage Error:

Event name:	Transmission Queue Usage Error.
Reason code in MQCFH:	MQRC_XMIT_Q_USAGE_ERROR (2092, X'82C'). Transmission queue with wrong usage.
Event description:	On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager, but one of the following occurred. Either: • *ObjectQMgrName* specifies the name of a local queue, but it does not have a **Usage** attribute of MQUS_TRANSMISSION. • The *ObjectName* or *ObjectQMgrName* field in the object descriptor specifies the name of a local definition of a remote queue but one of the following applies to the **XmitQName** attribute of the definition: – *XmitQName* is not blank, but specifies a queue that does not have a **Usage** attribute of MQUS_TRANSMISSION – *XmitQName* is blank, but *RemoteQMgrName* specifies a queue that does not have a **Usage** attribute of MQUS_TRANSMISSION • The queue name is resolved through a cell directory, and the remote queue manager name obtained from the cell directory is the name of a local queue, but it does not have a **Usage** attribute of MQUS_TRANSMISSION.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

XmitQName

Description: Transmission queue name.
Identifier: MQCA_XMIT_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ApplType

Description: Type of application making the MQI call that caused the event.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Unknown Alias Base Queue:

Event name:	Unknown Alias Base Queue.
Reason code in MQCFH:	MQRC_UNKNOWN_ALIAS_BASE_Q (2082, X'822'). Unknown alias base queue or topic.
Event description:	An MQOPEN or MQPUT1 call was issued specifying an alias queue as the destination, but the *BaseObjectName* in the alias queue attributes is not recognized as a queue or topic name.
Event type:	Local.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

BaseObjectName

Description:	Object name to which the alias resolves.
Identifier:	MQCA_BASE_OBJECT_NAME. For compatibility with existing applications, you can still use MQCA_BASE_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

ApplType

Description:	Type of application making the MQI call that caused the event.
Identifier:	MQIA_APPL_TYPE.
Data type:	MQCFIN.
Returned:	Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

BaseType

Description: Type of object to which the alias resolves.
Identifier: MQIA_BASE_TYPE.
Data type: MQCFIN.
Values:

MQOT_Q
Base object type is a queue

MQOT_TOPIC
Base object type is a topic

Returned: Always.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Unknown Default Transmission Queue:

Event name:	Unknown Default Transmission Queue.
Reason code in MQCFH:	MQRC_UNKNOWN_DEF_XMIT_Q (2197, X'895'). Unknown default transmission queue.
Event description:	An MQOPEN or MQPUT1 call was issued specifying a remote queue as the destination. If a local definition of the remote queue was specified, or if a queue manager alias is being resolved, the **XmitQName** attribute in the local definition is blank. No queue is defined with the same name as the destination queue manager. The queue manager has therefore attempted to use the default transmission queue. However, the name defined by the **DefXmitQName** queue manager attribute is not the name of a locally-defined queue.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

XmitQName

Description:	Default transmission queue name.
Identifier:	MQCA_XMIT_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

ApplType

Description: Type of application attempting to open the remote queue.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application attempting to open the remote queue.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Unknown Object Name:

Event name:	Unknown Object Name.
Reason code in MQCFH:	MQRC_UNKNOWN_OBJECT_NAME (2085, X'825'). Unknown object name.
Event description:	On an MQOPEN or MQPUT1 call, the *ObjectQMgrName* field in the object descriptor MQOD is set to one of the following options. It is either: • Blank • The name of the local queue manager • The name of a local definition of a remote queue (a queue manager alias) in which the **RemoteQMgrName** attribute is the name of the local queue manager However, the *ObjectName* in the object descriptor is not recognized for the specified object type.
Event type:	Local.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ApplType

Description:	Type of application making the MQI call that caused the event.
Identifier:	MQIA_APPL_TYPE.
Data type:	MQCFIN.
Returned:	Always.

ApplName

Description:	Name of the application making the MQI call that caused the event.
Identifier:	MQCACF_APPL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: If the object opened is a queue object. Either *QName* or *TopicName* is returned.

ProcessName

Description: Process object name from object descriptor (MQOD).
Identifier: MQCA_PROCESS_NAME.
Data type: MQCFST.
Maximum length: MQ_PROCESS_NAME_LENGTH.
Returned: If the object opened is a process object. One of *ProcessName*, *QName*, or *TopicName* is returned.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

TopicName

Description: Topic object name from object descriptor (MQOD).
Identifier: MQCA_TOPIC_NAME.
Data type: MQCFST.
Maximum length: MQ_TOPIC_NAME_LENGTH.
Returned: If the object opened is a topic object. One of *ProcessName*, *QName*, or *TopicName* is returned.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Unknown Remote Queue Manager:

Event name:	Unknown Remote Queue Manager.
Reason code in MQCFH:	MQRC_UNKNOWN_REMOTE_Q_MGR (2087, X'827'). Unknown remote queue manager.
Event description:	On an MQOPEN or MQPUT1 call, an error occurred with queue-name resolution, for one of the following reasons: • *ObjectQMgrName* is either blank or the name of the local queue manager, and *ObjectName* is the name of a local definition of a remote queue that has a blank *XmitQName*. However, there is no (transmission) queue defined with the name of *RemoteQMgrName*, and the **DefXmitQName** queue manager attribute is blank. • *ObjectQMgrName* is the name of a queue manager alias definition (held as the local definition of a remote queue) that has a blank *XmitQName*. However, there is no (transmission) queue defined with the name of *RemoteQMgrName*, and the **DefXmitQName** queue manager attribute is blank. • *ObjectQMgrName* specified is not: – Blank – The name of the local queue manager – The name of a local queue – The name of a queue manager alias definition (that is, a local definition of a remote queue with a blank *RemoteQName*) and the **DefXmitQName** queue manager attribute is blank. • *ObjectQMgrName* is blank or is the name of the local queue manager, and *ObjectName* is the name of a local definition of a remote queue (or an alias to one), for which *RemoteQMgrName* is either blank or is the name of the local queue manager. This error occurs even if the *XmitQName* is not blank. • *ObjectQMgrName* is the name of a local definition of a remote queue. In this case, it should be a queue manager alias definition, but the *RemoteQName* in the definition is not blank. • *ObjectQMgrName* is the name of a model queue. • The queue name is resolved through a cell directory. However, there is no queue defined with the same name as the remote queue manager name obtained from the cell directory. Also, the **DefXmitQName** queue manager attribute is blank. • On z/OS only: a message was put to a queue manager in a queue-sharing group and *SQQMNAME* is set to USE. This routes the message to the specified queue manager in order to be put on the queue. If *SQQMNAME* is set to IGNORE, the message is put to the queue directly.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ApplType

Description: Type of application attempting to open the remote queue.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application attempting to open the remote queue.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description:	Channel name for client connection.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	If the application making the MQI call that caused the event is a client attached application.

Unknown Transmission Queue:

Event name:	Unknown Transmission Queue.
Reason code in MQCFH:	MQRC_UNKNOWN_XMIT_Q (2196, X'894'). Unknown transmission queue.
Event description:	On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager. The *ObjectName* or the *ObjectQMgrName* in the object descriptor specifies the name of a local definition of a remote queue (in the latter case queue manager aliasing is being used). However, the **XmitQName** attribute of the definition is not blank and not the name of a locally-defined queue.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

XmitQName

Description:	Transmission queue name.
Identifier:	MQCA_XMIT_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

ApplType

Description: Type of application making the MQI call that caused the event.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Troubleshooting and support reference

Use the reference information in this section to help you diagnose errors with IBM MQ.

Select the appropriate topic from the following list to diagnose problems and errors in IBM MQ:

- "Example trace data for Windows"
- "Example trace data for UNIX and Linux" on page 4310
- "Examples of trace output" on page 4314
- "Examples of CEDF output" on page 4316
- "Return code 00000461 for TCP/IP" on page 4327

Related information:

Troubleshooting and support

Troubleshooting overview

Using trace

z/OS Using trace on z/OS

Example trace data for Windows

Windows

An extract from an IBM MQ for Windows trace file.

```
Counter  TimeStamp          PID.TID    Ident        Data
=========================================================

00000EF7 16:18:56.381367   2512.1           :       !! - Thread stack
00000EF8 16:18:56.381406   2512.1           :       !! - -> InitProcessInitialisation
00000EF9 16:18:56.381429   2512.1           :       --{  InitProcessInitialisation
00000EFA 16:18:56.381514   2512.1           :       ---{  xcsReleaseThreadMutexSem
00000EFB 16:18:56.381529   2512.1           :       ---}  xcsReleaseThreadMutexSem (rc=OK)
00000EFC 16:18:56.381540   2512.1           :       ---{  xcsGetEnvironmentString
00000EFD 16:18:56.381574   2512.1           :       xcsGetEnvironmentString[AMQ_REUSE_SHARED_THREAD] = NULL
00000EFE 16:18:56.381587   2512.1           :       ---}! xcsGetEnvironmentString (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000EFF 16:18:56.381612   2512.1           :       ---{  xcsGetEnvironmentInteger
00000F00 16:18:56.381622   2512.1           :       ----{  xcsGetEnvironmentString
00000F01 16:18:56.381647   2512.1           :       xcsGetEnvironmentString[AMQ_AFFINITY_MASK] = NULL
00000F02 16:18:56.381660   2512.1           :       ----}! xcsGetEnvironmentString (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000F03 16:18:56.381673   2512.1           :       ---}! xcsGetEnvironmentInteger (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000F04 16:18:56.381684   2512.1           :       ---{  xcsGetEnvironmentString
00000F05 16:18:56.381708   2512.1           :       xcsGetEnvironmentString[AMQ_FFSTINFO] = NULL
00000F06 16:18:56.381747   2512.1           :       ---}! xcsGetEnvironmentString (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000F07 16:18:56.381760   2512.1           :       ---{  xcsIsEnvironment
00000F08 16:18:56.381783   2512.1           :       xcsIsEnvironment[AMQ_DEBUG_MTIME] = FALSE
00000F09 16:18:56.381793   2512.1           :       ---}  xcsIsEnvironment (rc=OK)
00000F0A 16:18:56.381804   2512.1           :       ---{  xcsGetEnvironmentInteger
00000F0B 16:18:56.381811   2512.1           :       ----{  xcsGetEnvironmentString
00000F0C 16:18:56.381835   2512.1           :       xcsGetEnvironmentString[AMQ_CBM_REUSE_FACTOR] = NULL
00000F0D 16:18:56.381848   2512.1           :       ----}! xcsGetEnvironmentString (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000F0E 16:18:56.381861   2512.1           :       ---}! xcsGetEnvironmentInteger (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000F0F 16:18:56.381874   2512.1           :       ---{  xcsGetEnvironmentInteger
00000F10 16:18:56.381885   2512.1           :       ----{  xcsGetEnvironmentString
00000F11 16:18:56.381908   2512.1           :       xcsGetEnvironmentString[AMQ_CBM_MAX_CACHEABLE_SIZE] = NULL
00000F12 16:18:56.381919   2512.1           :       ----}! xcsGetEnvironmentString (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000F13 16:18:56.381929   2512.1           :       ---}! xcsGetEnvironmentInteger (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000F14 16:18:56.381941   2512.1           :       ---{  xcsGetEnvironmentInteger
00000F15 16:18:56.381952   2512.1           :       ----{  xcsGetEnvironmentString
00000F16 16:18:56.381976   2512.1           :       xcsGetEnvironmentString[AMQ_CBM_LEN] = NULL
00000F17 16:18:56.381992   2512.1           :       ----}! xcsGetEnvironmentString (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000F18 16:18:56.382003   2512.1           :       ---}! xcsGetEnvironmentInteger (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000F19 16:18:56.382016   2512.1           :       --}  InitProcessInitialisation (rc=OK)
00000F1A 16:18:56.383045   2512.1           :       --{  DLLMain
00000F1B 16:18:56.383059   2512.1           :       ---{  MCSInitCriticalSection
00000F1C 16:18:56.383068   2512.1           :       ---} MCSInitCriticalSection (rc=OK)
```

Figure 122. Sample IBM MQ for Windows trace

Example trace data for UNIX and Linux

UNIX Linux

Extracts from traces file for UNIX and Linux.

AIX

Example for AIX

Figure 123 on page 4311 shows an extract from an IBM MQ for AIX trace:

```
Timestamp        Process.Thread  Trace Ident  Trace Data
=========================================================
 12:06:32.904335   622742.1           :           Header.v02:7.0:AIX 5.3:64:-1:1:GMT
 12:06:32.904427   622742.1           :           Version : 7.0.0.0    Level : p000-L090514
 12:06:32.904540   622742.1           :           UTC   Date : 05/15/09  Time : 11:06:32.904302
 12:06:32.904594   622742.1           :           Local Date : 05/15/09  Time : 12:06:32.904302  GMT
 12:06:32.904697   622742.1           :           PID : 622742 Process : dltmqm_nd (64-bit)
 12:06:32.904728   622742.1           :           Host : dynamo
 12:06:32.904755   622742.1           :           Operating System : AIX 5.3
 12:06:32.904781   622742.1           :           Product Long Name : WebSphere MQ for AIX
 12:06:32.904806   622742.1           :           -------------------------------
 12:06:32.904832   622742.1           :           xtrNullFd: 3, xihTraceFileNum: 5
 12:06:32.904916   622742.1           :           Data: 0x00000000
 12:06:32.904952   622742.1           :           Thread stack
 12:06:32.904982   622742.1           :           -> InitProcessInitialisation
 12:06:32.905007   622742.1           :      {  InitProcessInitialisation
 12:06:32.905033   622742.1           :      -{  xcsIsEnvironment
 12:06:32.905062   622742.1           :           xcsIsEnvironment[AMQ_NO_CS_RELOAD] = FALSE
 12:06:32.905088   622742.1           :      -}  xcsIsEnvironment rc=OK
 12:06:32.905117   622742.1           :      -{  xcsLoadFunction
 12:06:32.905145   622742.1           :           LibName(libmqmcs_r.a(shr.o)) LoadType(2097200)
 12:06:32.905178   622742.1           :           General, comms, CS, OAM, or WAS
 12:06:32.905204   622742.1           :      --{  xcsQueryValueForSubpool
 12:06:32.905282   622742.1           :      --}  xcsQueryValueForSubpool rc=OK
 12:06:32.905504   622742.1           :           FullPathLibName(/usr/mqm/lib64/libmqmcs_r.a(shr.o)) loaded with load
 12:06:32.905540   622742.1           :      --{  xcsGetMem
 12:06:32.905575   622742.1           :           component:24 function:176 length:2088 options:0 cbmindex:-1 *pointer:
 12:06:32.905601   622742.1           :      --}  xcsGetMem rc=OK
 12:06:32.905638   622742.1           :           Handle(0) Function(0) FullPathLibName(/usr/mqm/lib64/libmqmcs_r.a(shr
 12:06:32.905665   622742.1           :      -}  xcsLoadFunction rc=OK
```

Figure 123. Sample IBM MQ for AIX trace

HP-UX

Example for HP-UX

```
Timestamp        Process.Thread  Trace Ident  Trace Data
=========================================================
 10:36:38.973286    11352.1           :           Header.v02:7.0:HP-UX B.11.31:64:0:1:GMT
 10:36:38.973328    11352.1           :           Version : 7.0.1.3    Level : p701-103-100814
 10:36:38.973347    11352.1           :           UTC   Date : 02/28/12  Time : 10:36:38.973271
 10:36:38.973356    11352.1           :           Local Date : 02/28/12  Time : 10:36:38.973271  GMT
 10:36:38.973378    11352.1           :           PID : 11352 Process : dltmqm_nd (64-bit)
 10:36:38.973384    11352.1           :           Host : myhost
 10:36:38.973389    11352.1           :           Operating System : HP-UX B.11.31
 10:36:38.973394    11352.1           :           Product Long Name : WebSphere MQ for HP-UX (Itanium platform)
 10:36:38.973399    11352.1           :           --------------------------------
 10:36:38.973405    11352.1           :           xtrNullFd: 4, xihTraceFileNum: 5
 10:36:38.973434    11352.1           :           Thread stack
 10:36:38.974303    11352.1           :           -> InitProcessInitialisation
 10:36:38.974309    11352.1           :      {  InitProcessInitialisation
 10:36:38.974314    11352.1           :      -{  xcsIsEnvironment
 10:36:38.974338    11352.1           :           xcsIsEnvironment[AMQ_NO_CS_RELOAD] = FALSE
 10:36:38.974343    11352.1           :      -}  xcsIsEnvironment rc=OK
 10:36:38.974356    11352.1           :      -{  xcsLoadFunction
 10:36:38.974362    11352.1           :           LibName(libmqmcs_r.so) LoadType(2097200)
 10:36:38.974368    11352.1           :           General, comms, CS, OAM, or WAS
 10:36:38.974388    11352.1           :      --{  xcsQueryValueForSubpool
 10:36:38.974401    11352.1           :      --}  xcsQueryValueForSubpool rc=OK
 10:36:38.974451    11352.1           :           FullPathLibName(/opt/mqm/lib64/libmqmcs_r.so) loaded with dlopen
 10:36:38.974456    11352.1           :      --{  xcsGetMemFn
 10:36:38.974463    11352.1           :           component:24 function:176 length:2088 options:0 cbmindex:-1 *pointer:600
 10:36:38.974468    11352.1           :      --}  xcsGetMemFn rc=OK
 10:36:38.974475    11352.1           :           Handle(0000000000000000) Function(0000000000000000) FullPathLibName(/opt
 10:36:38.974480    11352.1           :      -}  xcsLoadFunction rc=OK
 10:36:38.974486    11352.1           :           SystemPageSize is 4096.
 10:36:38.974493    11352.1           :           getrlimit for RLIMIT_NOFILE returned rlim_cur=2048 rlim_max=4096
```

Figure 124. Sample HP-UX trace

Linux

Example for Linux

Figure 125 on page 4313 shows an extract from an IBM MQ for Linux trace:

```
Timestamp        Process.Thread  Trace Ident  Trace Data
=========================================================
 11:02:23.643879     1239.1          :           Header.v02:7.0:Linux 2.6.5-7.276-smp:32:-1:1:GMT
 11:02:23.643970     1239.1          :           Version : 7.0.0.0    Level : p000-L090514
 11:02:23.644025     1239.1          :           UTC   Date : 05/15/09  Time : 10:02:23.643841
 11:02:23.644054     1239.1          :           Local Date : 05/15/09  Time : 11:02:23.643841  GMT
 11:02:23.644308     1239.1          :           PID : 1239 Process : dltmqm (32-bit)
 11:02:23.644324     1239.1          :           Host : hal1
 11:02:23.644334     1239.1          :           Operating System : Linux 2.6.5-7.276-smp
 11:02:23.644344     1239.1          :           Product Long Name : WebSphere MQ for Linux (x86 platform)
 11:02:23.644353     1239.1          :           --------------------------------
 11:02:23.644363     1239.1          :           xtrNullFd: 3, xihTraceFileNum: 4
 11:02:23.644394     1239.1          :           Thread stack
 11:02:23.644412     1239.1          :           -> InitProcessInitialisation
 11:02:23.644427     1239.1          :      {  InitProcessInitialisation
 11:02:23.644439     1239.1          :      -{  xcsIsEnvironment
 11:02:23.644469     1239.1          :           xcsIsEnvironment[AMQ_NO_CS_RELOAD] = FALSE
 11:02:23.644485     1239.1          :      -}  xcsIsEnvironment rc=OK
 11:02:23.644504     1239.1          :      -{  xcsLoadFunction
 11:02:23.644519     1239.1          :           LibName(libmqmcs_r.so) LoadType(2097200)
 11:02:23.644537     1239.1          :           General, comms, CS, OAM, or WAS
 11:02:23.644558     1239.1          :      --{  xcsQueryValueForSubpool
 11:02:23.644579     1239.1          :      --}  xcsQueryValueForSubpool rc=OK
 11:02:23.644641     1239.1          :           FullPathLibName(/opt/mqm/lib/libmqmcs_r.so) loaded with dlopen
 11:02:23.644652     1239.1          :      --{  xcsGetMem
 11:02:23.644675     1239.1          :           component:24 function:176 length:8212 options:0 cbmindex:-1 *pointer:
 11:02:23.644685     1239.1          :      --}  xcsGetMem rc=OK
 11:02:23.644722     1239.1          :           Handle((nil)) Function((nil)) FullPathLibName(/opt/mqm/lib/libmqmcs_r
 11:02:23.644732     1239.1          :      -}  xcsLoadFunction rc=OK
 11:02:23.644753     1239.1          :           SystemPageSize is 4096.
```

Figure 125. Sample IBM MQ for Linux trace

Solaris

Example for Solaris

Figure 126 on page 4314 shows an extract from an IBM MQ for Solaris trace:

```
Timestamp        Process.Thread  Trace Ident  Trace Data
=========================================================
 11:48:57.905466     7078.1           :           Header.v02:7.0:SunOS 5.9:64:-1:1:GMT
 11:48:57.905625     7078.1           :           Version : 7.0.0.0    Level : p000-L090514
 11:48:57.905770     7078.1           :           UTC   Date : 05/15/09  Time : 10:48:57.905364
 11:48:57.905816     7078.1           :           Local Date : 05/15/09  Time : 11:48:57.905364  GMT
 11:48:57.906104     7078.1           :           PID : 7078 Process : dltmqm_nd (64-bit)
 11:48:57.906129     7078.1           :           Host : computer.v6.hursley.ibm.com
 11:48:57.906148     7078.1           :           Operating System : SunOS 5.9
 11:48:57.906167     7078.1           :           Product Long Name : WebSphere MQ for Solaris (SPARC platform)
 11:48:57.906184     7078.1           :           --------------------------------
 11:48:57.906203     7078.1           :           xtrNullFd: 4, xihTraceFileNum: 5
 11:48:57.906276     7078.1           :           Thread stack
 11:48:57.906353     7078.1           :      {  xcsInitialize
 11:48:57.906385     7078.1           :      -{  InitPrivateServices
 11:48:57.906439     7078.1           :      --{  xcsGetEnvironmentString
 11:48:57.906566     7078.1           :           xcsGetEnvironmentString[MQS_ACTION_ON_EXCEPTION] = NULL
 11:48:57.906608     7078.1           :      --}! xcsGetEnvironmentString rc=xecE_E_ENV_VAR_NOT_FOUND
 11:48:57.906709     7078.1           :      --{  xcsIsEnvironment
 11:48:57.906738     7078.1           :           xcsIsEnvironment[AMQ_SIGCHLD_SIGACTION] = FALSE
 11:48:57.906755     7078.1           :      --}  xcsIsEnvironment rc=OK
 11:48:57.906771     7078.1           :           AMQ_SIGCHLD_SIGACTION is not set
 11:48:57.906835     7078.1           :      --{  xcsIsEnvironment
 11:48:57.906862     7078.1           :           xcsIsEnvironment[MQS_NO_SYNC_SIGNAL_HANDLING] = FALSE
 11:48:57.906878     7078.1           :      --}  xcsIsEnvironment rc=OK
 11:48:57.907000     7078.1           :           FPE Handler installed, New=7e0b0f38, Old=0
 11:48:57.907035     7078.1           :           SEGV Handler installed, New=7e0b0f38, Old=0
 11:48:57.907063     7078.1           :           BUS Handler installed, New=7e0b0f38, Old=0
 11:48:57.907091     7078.1           :           ILL Handler installed, New=7e0b0f38, Old=0
 11:48:57.907109     7078.1           :           Synchronous Signal Handling Activated
```

Figure 126. Sample IBM MQ for Solaris trace

Examples of trace output

Use this topic as an example of how to interpret trace output.

Figure 127 on page 4315 shows an example of a trace taken on entry to an MQPUT1 call. The following items have been produced:

- Queue request parameter list
- Object descriptor (MQOD)
- Message descriptor (MQMD)
- Put message options (MQPMO)
- The first 256 bytes of message data

Compare this to Figure 128 on page 4316, which illustrates the same control blocks on exit from IBM MQ.

```
USRD9 5E9 ASCB 00F87E80           JOBN ECIC330
CSQW072I ENTRY: MQ user parameter trace
PUTONE
    Thread... 004C2B10  Userid... CICSUSER  pObjDesc. 106B2010
    pMsgDesc. 106B20B8  pPMO..... 106B2200
    BufferL.. 00000064  pBuffer.. 106A0578  RSV1..... 00000000
    RSV2..... 00000000  RSV3..... 116BC830
    C9E8C1E8  C5C3C9C3  AA8E8583  76270484  | IYAYECIC..ec...d |
    D4D8E3E3  0000048C  00000000  00000000  | MQTT............ |
    00000000  1910C7C2  C9C2D4C9  E8C14BC9  | ......GBIBMIYA.I |
    C7C3E2F2  F0F48E85  83762979  00010000  | GCS204.ec..`.... |

              GMT-01/30/05 14:42:08.412320    LOC-01/30/05 14:42:08.412320

USRD9 5E9 ASCB 00F87E80           JOBN ECIC330
CSQW072I ENTRY: MQ user parameter trace
 +0000  D6C44040  00000001  00000000  C2404040  | OD  ........B    |
 +0010  40404040  40404040  40404040  40404040  |                  |
 ...
 +00A0  00000000  00000000                     | ........        |

              GMT-01/30/05 14:42:08.412345    LOC-01/30/05 14:42:08.412345

USRD9 5E9 ASCB 00F87E80           JOBN ECIC330
CSQW072I ENTRY: MQ user parameter trace
 +0000  D4C44040  00000001  00000000  00000008  | MD  ............ |
 ...
 +0130  40404040  40404040  40404040  40404040  |                  |
 +0140  40404040                               |                  |

              GMT-01/30/05 14:42:08.412370    LOC-01/30/05 14:42:08.412370

USRD9 5E9 ASCB 00F87E80           JOBN ECIC330
CSQW072I ENTRY: MQ user parameter trace
 +0000  D7D4D640  00000001  00000000  FFFFFFFF  | PMO ............ |
 ...
 +0070  40404040  40404040  40404040  40404040  |                  |

              GMT-01/30/05 14:42:08.412393    LOC-01/30/05 14:42:08.412393

USRD9 5E9 ASCB 00F87E80           JOBN ECIC330
CSQW072I ENTRY: MQ user parameter trace
 +0000  C1C1C1C1  C1C1C1C1  C1404040  40404040  | AAAAAAAAA        |
 ...
 +0060  40404040                               |                  |

              GMT-01/30/05 14:42:08.412625    LOC-01/30/05 14:42:08.412625
```

Figure 127. Example trace data from an entry trace of an MQPUT1 request

```
USRD9 5EA ASCB 00F87E80           JOBN ECIC330
CSQW073I EXIT:  MQ user parameter trace
PUTONE
    Thread... 004C2B10  Userid... CICSUSER  pObjDesc. 106B2010
    pMsgDesc. 106B20B8  pPMO..... 106B2200
    BufferL.. 00000064  pBuffer.. 106A0578  RSV1..... 00000000
    RSV2..... 00000000  RSV3..... 116BC830
    CompCode. 00000002  Reason... 000007FB
    C9E8C1E8  C5C3C9C3  AA8E8583  76270484  | IYAYECIC..ec...d |
    D4D8E3E3  0000048C  00000000  00000000  | MQTT............ |
    00000000  1910C7C2  C9C2D4C9  E8C14BC9  | ......GBIBMIYA.I |
    C7C3E2F2  F0F48E85  83762979  00010000  | GCS204.ec..`.... |
MQRC_OBJECT_TYPE_ERROR

               GMT-01/30/05 14:42:08.412678    LOC-01/30/05 14:42:08.412678

USRD9 5EA ASCB 00F87E80           JOBN ECIC330
CSQW073I EXIT:  MQ user parameter trace
 +0000  D6C44040  00000001  00000000  C2404040  | OD  ........B    |
 ...
 +00A0  00000000  00000000                     | ........         |

               GMT-01/30/05 14:42:08.412789    LOC-01/30/05 14:42:08.412789

USRD9 5EA ASCB 00F87E80           JOBN ECIC330
CSQW073I EXIT:  MQ user parameter trace
 +0000  D4C44040  00000001  00000000  00000008  | MD  ............ |
 ...
 +0140  40404040                               |                  |

               GMT-01/30/05 14:42:08.412814    LOC-01/30/05 14:42:08.412814

USRD9 5EA ASCB 00F87E80           JOBN ECIC330
CSQW073I EXIT:  MQ user parameter trace
 +0000  D7D4D640  00000001  00000000  FFFFFFFF  | PMO ............ |
 ...
 +0070  40404040  40404040  40404040  40404040  |                  |

               GMT-01/30/05 14:42:08.412836    LOC-01/30/05 14:42:08.412836

USRD9 5EA ASCB 00F87E80           JOBN ECIC330
CSQW073I EXIT:  MQ user parameter trace
 +0000  C1C1C1C1  C1C1C1C1  C1404040  40404040  | AAAAAAAAA        |
 ...
 +0060  40404040                               |                  |

               GMT-01/30/05 14:42:08.412858    LOC-01/30/05 14:42:08.412858
```

Figure 128. Example trace data from an exit trace of an MQPUT1 request

Examples of CEDF output

Use this topic as a reference for example CEDF output from MQI calls.

This topic gives examples of the output produced by the CICS execution diagnostic facility (CEDF) when using IBM MQ. The examples show the data produced on entry to and exit from the following MQI calls, in both hexadecimal and character format. Other MQI calls produce similar data.

Example CEDF output for the MQOPEN call

The parameters for this call are:

Parameter	Description
ARG 000	Connection handle
ARG 001	Object descriptor
ARG 002	Options
ARG 003	Object handle
ARG 004	Completion code
ARG 005	Reason code

```
  STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000000200004044')            AT X'05ECAFD8'
 001: ARG 001 (X'D6C4404000000010000000 1C3C5C4C6')            AT X'00144910'
 001: ARG 002 (X'00000072000000000000000000000000')            AT X'001445E8'
 001: ARG 003 (X'00000000000000720000000000000000')            AT X'001445E4'
 001: ARG 004 (X'00000000000000000000000000000000')            AT X'001445EC'
 001: ARG 005 (X'00000000000000000000000000000000')            AT X'001445F0'
```

Figure 129. Example CEDF output on entry to an MQOPEN call (hexadecimal)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000000200004044')            AT X'05ECAFD8'
 001: ARG 001 (X'D6C4404000000010000000 1C3C5C4C6')            AT X'00144910'
 001: ARG 002 (X'00000072000000000000000000000000')            AT X'001445E8'
 001: ARG 003 (X'00000001000000720000000000000000')            AT X'001445E4'
 001: ARG 004 (X'00000000000000000000000000000000')            AT X'001445EC'
 001: ARG 005 (X'00000000000000000000000000000000')            AT X'001445F0'
```

Figure 130. Example CEDF output on exit from an MQOPEN call (hexadecimal)

```
  STATUS:  ABOUT TO EXECUTE COMMAND
  CALL TO RESOURCE MANAGER MQM
   001: ARG 000 ('.............. .')
   001: ARG 001 ('OD  ........CEDF')
   001: ARG 002 ('................')
   001: ARG 003 ('................')
   001: ARG 004 ('................')
   001: ARG 005 ('................')
```

Figure 131. Example CEDF output on entry to an MQOPEN call (character)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('.............. .')
 001: ARG 001 ('OD  ........CEDF')
 001: ARG 002 ('................')
 001: ARG 003 ('................')
 001: ARG 004 ('................')
 001: ARG 005 ('................')
```

Figure 132. Example CEDF output on exit from an MQOPEN call (character)

Example CEDF output for the MQCLOSE call

The parameters for this call are:

Parameter	Description
ARG 000	Connection handle
ARG 001	Object handle
ARG 002	Options
ARG 003	Completion code
ARG 004	Reason code

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000007200000000')            AT X'001445E0'
 001: ARG 001 (X'0000000100000072000000000000000')             AT X'001445E4'
 001: ARG 002 (X'00000000000000010000000200004044')            AT X'05ECAFD8'
 001: ARG 003 (X'00000000000000000000000800000008')            AT X'001445EC'
 001: ARG 004 (X'00000000000000080000000800000060')            AT X'001445F0'
```

Figure 133. Example CEDF output on entry to an MQCLOSE call (hexadecimal)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000000000007200000000')            AT X'001445E0'
 001: ARG 001 (X'00000000000000720000000000000000')            AT X'001445E4'
 001: ARG 002 (X'00000000000000010000000200004044')            AT X'05ECAFD8'
 001: ARG 003 (X'00000000000000000000000800000008')            AT X'001445EC'
 001: ARG 004 (X'00000000000000080000000800000060')            AT X'001445F0'
```

Figure 134. Example CEDF output on exit from an MQCLOSE call (hexadecimal)

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('................')
 001: ARG 001 ('................')
 001: ARG 002 ('.............. .')
 001: ARG 003 ('................')
 001: ARG 004 ('................-')
```

Figure 135. Example CEDF output on entry to an MQCLOSE call (character)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('................')
 001: ARG 001 ('................')
 001: ARG 002 ('.............. .')
 001: ARG 003 ('................')
 001: ARG 004 ('................-')
```

Figure 136. Example CEDF output on exit from an MQCLOSE call (character)

Example CEDF output for the MQPUT call

The parameters for this call are:

Parameter	Description
ARG 000	Connection handle
ARG 001	Object handle
ARG 002	Message descriptor
ARG 003	Put message options
ARG 004	Buffer length
ARG 005	Message data
ARG 006	Completion code
ARG 007	Reason code

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000007200000000')            AT X'001445E0'
 001: ARG 001 (X'00000001000000720000000000000000')            AT X'001445E4'
 001: ARG 002 (X'D4C4404000000001000000000000008')             AT X'001449B8'
 001: ARG 003 (X'D7D4D64000000001000002400000000')             AT X'00144B48'
 001: ARG 004 (X'00000080000000000000000000040000')            AT X'001445F4'
 001: ARG 005 (X'5C5CC8C5D3D3D640E6D6D9D3C45C5C5C')            AT X'00144BF8'
 001: ARG 006 (X'00000000000000000000000800000000')            AT X'001445EC'
 001: ARG 007 (X'00000000000000080000000000000000')            AT X'001445F0'
```

Figure 137. Example CEDF output on entry to an MQPUT call (hexadecimal)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000007200000000')            AT X'001445E0'
 001: ARG 001 (X'00000001000000720000000000000000')            AT X'001445E4'
 001: ARG 002 (X'D4C4404000000001000000000000008')             AT X'001449B8'
 001: ARG 003 (X'D7D4D64000000001000002400000000')             AT X'00144B48'
 001: ARG 004 (X'00000008000000000000000000040000')            AT X'001445F4'
 001: ARG 005 (X'5C5CC8C5D3D3D640E6D6D9D3C45C5C5C')            AT X'00144BF8'
 001: ARG 006 (X'00000000000000000000000800000000')            AT X'001445EC'
 001: ARG 007 (X'00000000000000080000000000000000')            AT X'001445F0'
```

Figure 138. Example CEDF output on exit from an MQPUT call (hexadecimal)

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('................')
 001: ARG 001 ('................')
 001: ARG 002 ('MD  ............')
 001: ARG 003 ('PMO ............')
 001: ARG 004 ('................')
 001: ARG 005 ('**HELLO WORLD***')
 001: ARG 006 ('................')
 001: ARG 007 ('................')
```

Figure 139. Example CEDF output on entry to an MQPUT call (character)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('................')
 001: ARG 001 ('................')
 001: ARG 002 ('MD  ............')
 001: ARG 003 ('PMO ............')
 001: ARG 004 ('................')
 001: ARG 005 ('**HELLO WORLD***')
 001: ARG 006 ('................')
 001: ARG 007 ('................')
```

Figure 140. Example CEDF output on exit from an MQPUT call (character)

Example CEDF output for the MQPUT1 call

The parameters for this call are:

Parameter	Description
ARG 000	Connection handle
ARG 001	Object descriptor
ARG 002	Message descriptor
ARG 003	Put message options
ARG 004	Buffer length
ARG 005	Message data
ARG 006	Completion code
ARG 007	Reason code

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000000000007200000000')            AT X'001445E0'
 001: ARG 001 (X'D6C4404000000010000000001C3C5C4C6')            AT X'00144910'
 001: ARG 002 (X'D4C4404000000010000000000000008')              AT X'001449B8'
 001: ARG 003 (X'D7D4D64000000010000002400000000')              AT X'00144B48'
 001: ARG 004 (X'00000008000000080000006000040000')             AT X'001445F4'
 001: ARG 005 (X'5C5CC8C5D3D3D640E6D6D9D3C45C5C5C')             AT X'00144BF8'
 001: ARG 006 (X'00000000000000000000000800000008')             AT X'001445EC'
 001: ARG 007 (X'00000000000000080000000800000060')             AT X'001445F0'
```

Figure 141. Example CEDF output on entry to an MQPUT1 call (hexadecimal)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000000000007200000000')            AT X'001445E0'
 001: ARG 001 (X'D6C4404000000010000000001C3C5C4C6')            AT X'00144910'
 001: ARG 002 (X'D4C4404000000010000000000000008')              AT X'001449B8'
 001: ARG 003 (X'D7D4D64000000010000002400000000')              AT X'00144B48'
 001: ARG 004 (X'00000008000000080000006000040000')             AT X'001445F4'
 001: ARG 005 (X'5C5CC8C5D3D3D640E6D6D9D3C45C5C5C')             AT X'00144BF8'
 001: ARG 006 (X'00000000000000000000000800000008')             AT X'001445EC'
 001: ARG 007 (X'00000000000000080000000800000060')             AT X'001445F0'
```

Figure 142. Example CEDF output on exit from an MQPUT1 call (hexadecimal)

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('................')
 001: ARG 001 ('OD  ........CEDF')
 001: ARG 002 ('MD  ............')
 001: ARG 003 ('PMO ............')
 001: ARG 004 ('...........-....')
 001: ARG 005 ('**HELLO WORLD***')
 001: ARG 006 ('................')
 001: ARG 007 ('...............-')
```

Figure 143. Example CEDF output on entry to an MQPUT1 call (character)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('................')
 001: ARG 001 ('OD  ........CEDF')
 001: ARG 002 ('MD  ............')
 001: ARG 003 ('PMO ............')
 001: ARG 004 ('...........-....')
 001: ARG 005 ('**HELLO WORLD***')
 001: ARG 006 ('................')
 001: ARG 007 ('...............-')
```

Figure 144. Example CEDF output on exit from an MQPUT1 call (character)

Example CEDF output for the MQGET call

The parameters for this call are:

Parameter	Description
ARG 000	Connection handle
ARG 001	Object handle
ARG 002	Message descriptor
ARG 003	Get message options
ARG 004	Buffer length
ARG 005	Message buffer
ARG 006	Message length
ARG 007	Completion code
ARG 008	Reason code

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000007200000000')            AT X'001445E0'
 001: ARG 001 (X'0000000100000072000000000000000')             AT X'001445E4'
 001: ARG 002 (X'D4C44040000000010000000000000000')            AT X'001449B8'
 001: ARG 003 (X'C7D4D64000000000100004044FFFFFFFF')           AT X'00144B00'
 001: ARG 004 (X'00000008000000000000000000040000')            AT X'001445F4'
 001: ARG 005 (X'00000000000000000000000000000000')            AT X'00144C00'
 001: ARG 006 (X'00000000000000000004000000000000')            AT X'001445F8'
 001: ARG 007 (X'00000000000000000000000800000000')            AT X'001445EC'
 001: ARG 008 (X'00000000000000080000000000000000')            AT X'001445F0'
```

Figure 145. Example CEDF output on entry to an MQGET call (hexadecimal)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000007200000000')            AT X'001445E0'
 001: ARG 001 (X'0000000100000072000000000000000')             AT X'001445E4'
 001: ARG 002 (X'D4C4404000000001000000000000008')             AT X'001449B8'
 001: ARG 003 (X'C7D4D64000000000100004044FFFFFFFF')           AT X'00144B00'
 001: ARG 004 (X'00000008000000080000000000040000')            AT X'001445F4'
 001: ARG 005 (X'5C5CC8C5D3D3D640E6D6D9D3C45C5C5C')            AT X'00144C00'
 001: ARG 006 (X'00000008000000000000040000000000')            AT X'001445F8'
 001: ARG 007 (X'00000000000000000000000800000008')            AT X'001445EC'
 001: ARG 008 (X'00000000000000080000000800000000')            AT X'001445F0'
```

Figure 146. Example CEDF output on exit from an MQGET call (hexadecimal)

```
 STATUS:  ABOUT TO EXECUTE COMMAND
 CALL TO RESOURCE MANAGER MQM
  001: ARG 000 ('................')
  001: ARG 001 ('................')
  001: ARG 002 ('MD  ............')
  001: ARG 003 ('GMO ...... .....')
  001: ARG 004 ('................')
  001: ARG 005 ('................')
  001: ARG 006 ('................')
  001: ARG 007 ('................')
  001: ARG 008 ('................')
```

Figure 147. Example CEDF output on entry to an MQGET call (character)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('................')
 001: ARG 001 ('................')
 001: ARG 002 ('MD  ............')
 001: ARG 003 ('GMO ...... .....')
 001: ARG 004 ('................')
 001: ARG 005 ('**HELLO WORLD***')
 001: ARG 006 ('................')
 001: ARG 007 ('................')
 001: ARG 008 ('................')
```

Figure 148. Example CEDF output on exit from an MQGET call (character)

Example CEDF output for the MQINQ call

The parameters for this call are:

Parameter	Description
ARG 000	Connection handle
ARG 001	Object handle
ARG 002	Count of selectors
ARG 003	Array of attribute selectors
ARG 004	Count of integer attributes
ARG 005	Integer attributes
ARG 006	Length of character attributes buffer
ARG 007	Character attributes
ARG 008	Completion code
ARG 009	Reason code

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000000200004044')            AT X'05ECAFCC'
 001: ARG 001 (X'0000000100000072000000000000000')             AT X'001445E4'
 001: ARG 002 (X'0000000200000404485ECA00885ECA220')           AT X'05ECAFD4'
 001: ARG 003 (X'0000000D0000000C0000000000000000')            AT X'00144C08'
 001: ARG 004 (X'0000000200000404485ECA00885ECA220')           AT X'05ECAFD4'
 001: ARG 005 (X'00000000000000000000000000000000')            AT X'00144C10'
 001: ARG 006 (X'00000000000000010000000200004044')            AT X'05ECAFCC'
 001: ARG 007 (X'00000000000000000000000000000000')            AT X'00144C18'
 001: ARG 008 (X'00000000000000000000000800000008')            AT X'001445EC'
 001: ARG 009 (X'00000000000000080000000800040000')            AT X'001445F0'
```

Figure 149. Example CEDF output on entry to an MQINQ call (hexadecimal)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000000200004044')            AT X'05ECAFCC'
 001: ARG 001 (X'0000000100000072000000000000000')             AT X'001445E4'
 001: ARG 002 (X'0000000200000404485ECA00885ECA220')           AT X'05ECAFD4'
 001: ARG 003 (X'0000000D0000000C0040000000000000')            AT X'00144C08'
 001: ARG 004 (X'0000000200000404485ECA00885ECA220')           AT X'05ECAFD4'
 001: ARG 005 (X'00400000000000000000000000000000')            AT X'00144C10'
 001: ARG 006 (X'00000000000000010000000200004044')            AT X'05ECAFCC'
 001: ARG 007 (X'00000000000000000000000000000000')            AT X'00144C18'
 001: ARG 008 (X'00000000000000000000000800000008')            AT X'001445EC'
 001: ARG 009 (X'00000000000000080000000800040000')            AT X'001445F0'
```

Figure 150. Example CEDF output on exit from an MQINQ call (hexadecimal)

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('.............. .')
 001: ARG 001 ('................')
 001: ARG 002 ('...... .e...e.s.')
 001: ARG 003 ('................')
 001: ARG 004 ('...... .e...e.s.')
 001: ARG 005 ('................')
 001: ARG 006 ('.............. .')
 001: ARG 007 ('................')
 001: ARG 008 ('................')
 001: ARG 009 ('................')
```

Figure 151. Example CEDF output on entry to an MQINQ call (character)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('.............. .')
 001: ARG 001 ('................')
 001: ARG 002 ('...... .e...e.s.')
 001: ARG 003 ('......... ......')
 001: ARG 004 ('...... .e...e.s.')
 001: ARG 005 ('. ..............')
 001: ARG 006 ('.............. .')
 001: ARG 007 ('................')
 001: ARG 008 ('................')
 001: ARG 009 ('................')
```

Figure 152. Example CEDF output on exit from an MQINQ call (character)

Example CEDF output for the MQSET call

The parameters for this call are:

Parameter	Description
ARG 000	Connection handle
ARG 001	Object handle
ARG 002	Count of selectors
ARG 003	Array of attribute selectors
ARG 004	Count of integer attributes
ARG 005	Integer attributes
ARG 006	Length of character attributes buffer
ARG 007	Character attributes
ARG 008	Completion code
ARG 009	Reason code

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000007200000000')            AT X'001445E0'
 001: ARG 001 (X'0000000100000072000000000000000000')          AT X'001445E4'
 001: ARG 002 (X'0000000100000002000040448 5ECA008')           AT X'05ECAFDC'
 001: ARG 003 (X'00000018000007DF0000000000000000')            AT X'00144C08'
 001: ARG 004 (X'0000000100000002000040448 5ECA008')           AT X'05ECAFDC'
 001: ARG 005 (X'00000000000000000000000000000000')            AT X'00144C10'
 001: ARG 006 (X'00000000000000010000000200004044')            AT X'05ECAFD8'
 001: ARG 007 (X'00000000000000000000000000000000')            AT X'00144C18'
 001: ARG 008 (X'00000000000000000000000800000008')            AT X'001445EC'
 001: ARG 009 (X'00000000000000080000000800000060')            AT X'001445F0'
```

Figure 153. Example CEDF output on entry to an MQSET call (hexadecimal)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000007200000000')            AT X'001445E0'
 001: ARG 001 (X'0000000100000072000000000000000000')          AT X'001445E4'
 001: ARG 002 (X'0000000100000002000040448 5ECA008')           AT X'05ECAFDC'
 001: ARG 003 (X'00000018000007DF0000000000000000')            AT X'00144C08'
 001: ARG 004 (X'0000000100000002000040448 5ECA008')           AT X'05ECAFDC'
 001: ARG 005 (X'00000000000000000000000000000000')            AT X'00144C10'
 001: ARG 006 (X'00000000000000010000000200004044')            AT X'05ECAFD8'
 001: ARG 007 (X'00000000000000000000000000000000')            AT X'00144C18'
 001: ARG 008 (X'00000000000000000000000800000008')            AT X'001445EC'
 001: ARG 009 (X'00000000000000080000000800000060')            AT X'001445F0'
```

Figure 154. Example CEDF output on exit from an MQSET call (hexadecimal)

```
 STATUS:  ABOUT TO EXECUTE COMMAND
 CALL TO RESOURCE MANAGER MQM
  001: ARG 000 ('................')
  001: ARG 001 ('................')
  001: ARG 002 ('.......... .e...')
  001: ARG 003 ('................')
  001: ARG 004 ('.......... .e...')
  001: ARG 005 ('................')
  001: ARG 006 ('............... .')
  001: ARG 007 ('................')
  001: ARG 008 ('................')
  001: ARG 009 ('...............-')
```

Figure 155. Example CEDF output on entry to an MQSET call (character)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('................')
 001: ARG 001 ('................')
 001: ARG 002 ('.......... .e...')
 001: ARG 003 ('................')
 001: ARG 004 ('.......... .e...')
 001: ARG 005 ('................')
 001: ARG 006 ('............... .')
 001: ARG 007 ('................')
 001: ARG 008 ('................')
 001: ARG 009 ('................-')
```

Figure 156. Example CEDF output on exit from an MQSET call (character)

Return code 00000461 for TCP/IP

z/OS

There is a channel failure, and you receive the following: `CSQX208E TRPTYPE=TCP RC=00000461`, or `CSQX208E TRPTYPE=TCP RC=00000461 reason=76650446`.

Cause

Return code 10054 or RC 461 means the connection is reset by peer (ECONNRESET). This return code is often the results of a problem in the TCP/IP network. There are various reasons for TCP/IP sending a reset:

- An unorderly connection termination, such as a rebooting the client box, can cause a reset.
- An application requests a connect to a port and IP address on which no server is listening.
- An application closes a socket with data still in the application receive buffer. The connection is reset to allow the remote partner to know that the data was not delivered.
- Any data that arrives for a connection that has been closed can cause a reset.
- An application closes a socket and sets the linger socket option to zero. This notifies TCP/IP that the connection should not linger.

  Note: IBM MQ does not code the linger time = `0`, therefore IBM MQ itself does not cause a reset.
- A TCP segment that is not valid arrives for a connection. For example, a bad acknowledge or sequence number can cause a reset.
- The connect request times out. TCP stops trying to connect to a particular port and IP address and reset the connection.
- A firewall can reset connections if the packet does not adhere to the firewall rules and policies. For example, a source or destination port, or IP address does not match the firewall rule or policy.
- The retransmit timer expires. TCP stops trying to retransmit a packet and reset the connection.
- A bad hardware device can cause resets.

You need to be aware that the effect of your configuration at higher levels, for example, the channel initiator dispatching priority being too low, could exhibit itself as a reset. Therefore, you should also consider the effect of your configuration when trying to determine why a reset is happening.

Diagnosing the problem

Use TCP/IP packet traces to determine why the reset occurred.

See z/OS UNIX reason codes for the last two bytes of the reason code found in the CSQX208E error message.

Messages

You can use the following messages to help you solve problems with your IBM MQ components or applications.

IBM MQ messages on Multiplatforms

Multi

IBM MQ diagnostic messages are listed in this section in numerical order, grouped according to the part of IBM MQ from which they originate.

Note: If a message is specific to a single platform, this is indicated after the message identifier. Although some messages are listed several times, each instance relates to a different platform. If present, the version common to a number of platforms is listed first, followed by versions for individual platforms. Ensure that you read the appropriate version.

For details of these messages, see IBM Knowledge Center:

- AMQ3xxx: Utilities and tools
- AMQ4xxx: User interface messages (Windows and Linux systems)
- AMQ5xxx: Installable services
- AMQ6xxx: Common services
- AMQ7xxx: IBM MQ product
- AMQ8xxx: Administration
- AMQ9xxx: Remote

Reading a message

The following information is provided for each message:

Message identifier
The message identifier is in two parts:

1. The characters "AMQ", which identify the message as being from IBM MQ
2. A four-digit decimal code

Message text
A summary of the message

Severity
0: Information
10: Warning
20: Error
30: Severe error
40: Stop Error
50: System Error

Explanation
An explanation of the message giving further information.

Response
The response required from the user. In some cases, particularly for information messages, this might be "No action is required".

Message variables

Some messages display text or numbers that vary according to the circumstances giving rise to the message; these are known as *message variables*. The message variables are indicated as <insert_1>, <insert_2>, and so on.

In some cases a message might have variables in the Explanation or Response. Find the values of the message variables by looking in the error log. The complete message, including the Explanation and the Response, is recorded there.

Related concepts:

z/OS "IBM MQ for z/OS messages, completion, and reason codes" on page 4353
Use this topic to interpret and understand the messages and codes issued by IBM MQ for z/OS.

Related information:

API completion and reason codes

PCF reason codes

Transport Layer Security (TLS) return codes

WCF custom channel exceptions

Telemetry messages

AIX Linux Windows

Reference information to help you identify and interpret the messages for MQ Telemetry.

AMQCO1001E
MQXR service unexpectedly caught communications exception={0}(Exception).

Explanation
An exception was caught by the Communications Manager and it was not able to take a reasonable action in response to the exception.

User action
Investigate and resolve the cause of the underlying exception.

AMQCO1002E
A selection key={0} was found in an unexpected state.

Explanation
A selection key was found in a state that was not expected.

User action
Contact your IBM support center.

AMQCO1003E
Connection={0}(Connection) has insufficient data available to satisfy a get request.

Explanation
The application tried to read more data than is immediately available. After the application has processed the information available to it, it should release control and wait to be called again when more data is available.

User action
Change the application to handle this exception, or use Connection.available() before the get() method is called in order to determine if the get() will succeed.

AMQCO1004E
Connection Close error: {0}.

Explanation
An error occurred when a connection was closed. The session might not have completed normally.

User action
Check that the session completed normally.

AMQCO1005E
SSL key repository file invalid or not found for channel "{1}". The following exception was thrown: {0}

Explanation
The SSL key repository file specified for the channel is not valid.

User action
Check the validity of the specified SSL key repository file.

AMQCO1006I
Channel "{0}" has stopped.

Explanation
The channel has stopped. No further communication with clients will occur on this channel.

User action
No action is required.

AMQCO1007E
Connection "{0}" did not send or receive data for "{1}" milliseconds and has been closed.

Explanation
The application set the idle timer on the connection to {1} milliseconds, but did not send or receive any data within this time, so the connection was closed.

User action
Determine why the connection did not send or receive data and if appropriate set the idleTimer to a longer value.

AMQCO1008E
An SSL Handshake error occurred when a client at "{1}" attempted to connect to channel "{0}": {2}.

Explanation
An error occurred when performing an SSL handshake with a client application. This is often because the client is presenting certificates that the MQXR service does not trust.

User action
Use the information in the exception to diagnose and fix the problem.

AMQCO1009E
An invalid keystore name="{1}" was specified.

Explanation
The keystore name or the pass phrase specified is not valid.

User action
Specify a valid keystore file name and password.

AMQCO1010E
An SSL Exception occurred when a client at "{1}" attempted to connect to channel "{0}": {2}.

Explanation
An error occurred when performing an SSL operation with a client application.

User action
Use the information in the exception to diagnose and fix the problem.

AMQCO2001E
An error (probe: {0}) occurred and a Failure Data Capture (FDC) file has been written.

Explanation
A problem was detected and a FDC file was written to aid diagnostics.

User action
Look at the FDC file and attempt to resolve the problem. If the problem cannot be resolved, contact your IBM support center.

AMQCO2002I
Trace is disabled.

Explanation
Tracing the MQXR Service (used in order to diagnose problems) is not currently running.

User action
No action is required.

AMQCO2003I
Trace is enabled.

Explanation
Tracing the MQXR Service (used in order to diagnose problems) is currently running.

User action
No action is required.

AMQCO2004I
"{0}" instances of message "{1}" were suppressed.

Explanation
The number {0} of message identifier "{1}" were suppressed from the log since the last message with this identifier was written.

User action
No additional action is required beyond that for the suppressed message.

AMQCO9999E
{0}

Explanation
If the message does not give sufficient information, check previous messages for further help.

User action
See previous messages for further information.

AMQHT1001E
Invalid text={0}(String) was found in an HTTP request or response.

Explanation
An HTTP request or response contained unexpected data not described in "http://www.w3.org/pub/WWW/Protocols/".

User action
Check that the originator or source of the HTTP request or response is producing valid requests or responses.

AMQHT1002E
HTTP header text={0}(String) was invalid.

Explanation
An HTTP request or response contained unexpected text.

User action
Check that the originator or source of the HTTP request or response is producing valid requests or responses.

AMQHT1003E
Invalid text at location={0} in string={1}(String).

Explanation
A Java Script Object Notation (JSON) string contained unexpected data not described in "http://www.json.org/".

User action
Check that the originator or JSON is producing valid data.

AMQHT2001E
WebSocket Close, status code= {0}

Explanation
The websocket was closed by the remote end.

User action
Examine the Websocket status code and determine why the Websocket was closed if this was not expected.

AMQHT9999E
{0}

Explanation
If the message does not give sufficient information, check previous messages for further help.

User action
See previous messages for further information.

AMQXR0001I
Client {0} disconnected normally.

Explanation
An MQTT disconnect flow was received and processed.

User action
None.

AMQXR0002E
On channel {2}, a throwable {1} resulted when the MQXR service received a message from an MQTT client {0}.

Explanation
Bad data was received from a network connection and could not be processed, the connection is closed by the server.

User action
Determine why the client sent the uninterpretable data.

AMQXR0003I
MQXR JAAS {0} : {1}.

Explanation
The JAAS callback in the MQXR service requested that the message is displayed to the user.

User action
Determine the cause of the security problem described in the text of the message issued by JAAS.

AMQXR0004E
MQSeries verb={0}(String) returned cc={1}(int) {2} rc={3}(int) {4}.

Explanation
A WebSphere MQ verb returned an unexpected reason and completion code.

User action
Look up the reason code to determine what caused the error.

AMQXR0005I
Running {0} version {1}.

Explanation
The command is running.

User action
None.

AMQXR0006E
Invalid argument {0} Usage: runMQXRService -m *queueManagerName* -d *QmgrDataDirectory* -g *MQGlobalDataDirectory*

Explanation
The runMQXRService command arguments are incorrect.

User action
Correct the command.

AMQXR0007E
Invalid argument {0} Usage: endMQXRService -m *queueManagerName* -d *QmgrDataDirectory* -g *MQGlobalDataDirectory*

Explanation
The endMQXRService command arguments are incorrect.

User action
Correct the command.

AMQXR0008E
Exception during start of MQXR service: {0}

Explanation
The MQXR service was starting but encountered a problem. Previous errors or FDCs will provide more detail.

User action
Use previous errors or FDCs to diagnose and address the problem then restart the MQXR service.

AMQXR0009E
Exception during shutdown of MQXR service: {0}

Explanation
The MQXR service was shutting down but encountered a problem. Previous errors or FDCs will provide more detail.

User action
Use previous errors or FDCs to diagnose and address the problem.

AMQXR0010E
An invalid ClientIdentifier {0} was received from "{1}" in an MQTT CONNECT packet on channel {2}.

Explanation
The MQXR service received a ClientIdentifier that is not valid because it contains too few, or too many characters, or the characters are not valid in a queue manager name.

User action
Change the ClientIdentifier to use valid characters.

AMQXR0011E

An error occurred during a publish on topic "{3}" from ClientIdentifier "{0}" UserName "{1}" on channel "{2}". A reason code of "{5}" "{6}" was received during an "{4}" operation.

Explanation

The publication from the client was not able to be completed

User action

Using the reason code, diagnose the cause of the problem, alter the configuration (of the client or the server as appropriate) and then retry the publish.

AMQXR0012E

An error occurred whilst subscribing on topic(s) "{3}" for ClientIdentifier "{0}" userNamer "{1}"on channel "{2}". A reason code of "{5}" "{6}" was received during an "{4}" operation.

Explanation

The subscription from the client was not able to be completed

User action

Using the reason code, diagnose the cause of the problem, alter the configuration (of the client or the server as appropriate) and then reconnect the client and retry the subscription.

AMQXR0013E

Error starting channel "{0}" (on host: "{1}" and port "{2}"). The exception was "{3}".

Explanation

The service was unable to listen for connections on the specified port

User action

Use the exception to diagnose and rectify the problem then restart the affected channel.

AMQXR0014E

Error starting channel "{0}". See earlier errors or FDCs for more details.

Explanation

The service was unable to listen for connections on the specified port because of problems that have been reported in earlier errors or FDCs.

User action

Use the preceding errors or FDCs to diagnose and rectify the problem then restart the affected channel.

AMQXR0015I

MQXR Service started successfully ({0} channels running, {1} channels stopped)

Explanation

The MQXR service has completed the processing that occurs on startup

User action

No action is required.

AMQXR0016I

Channel "{0}" has started

Explanation

This channel is now available for client connections

User action

No action is required

AMQXR0017I

A new channel (called "{0}") has been created

Explanation

In response to a request from a user, a new channel has been created

User action
No action is required

AMQXR0018I
Channel "{0}" has been altered

Explanation
In response to a request from a user, some settings on the channel were changed. Some settings do not take effect until the channel is restarted.

User action
No action is required

AMQXR0019I
Channel "{0}" has been deleted

Explanation
In response to a request from a user, a new channel has been deleted

User action
No action is required

AMQXR0020I
Channel "{0}" has been purged

Explanation
Clients have been disconnected from this channel and state associated with them has been removed

User action
No action is required

AMQXR0021W
Client "{0}" at network address "{1}" disconnected abnormally with exception "{2}".

Explanation
An MQTT client was disconnected from the network for the reason shown by the exception.

User action
Look into the exception cause to determine if action is required.

AMQXR0022I
Client "{0}" previously connected at network address "{1}" now connected at "{2}".

Explanation
A new connection has been made for the client taking over from an existing one.

User action
None, if this was intentional.

AMQXR0023I
Unsupported MQTT protocol version on channel {1}, the exception {0} was thrown.

Explanation
An MQTT client attempted to connect using an unsupported protocol version, the connection is closed by the server.

User action
Reconfigure the client to use a supported protocol version.

AMQXR0030W
Invalid Will Message from ClientIdentifier "{0}"

Explanation
The Will Message in the Connect packet is malformed, the client connection is closed by the server.

User action
Check the client application and make sure the will message has a non zero length topic name, and a valid Qos.

AMQXR1001E
MQTTV3Exception message={0}(String).

Explanation
An instance of com.ibm.mqttv3.internal.MQTTException has been caught and wrapped.

User action
Contact your IBM support center.

AMQXR1002E
MQTTV5Exception message={0}(String).

Explanation
An instance of com.ibm.mqtt.encoding.internal.MQTTException has been caught and wrapped.

User action
Contact your IBM support center.

AMQXR1003E
An invalid message type={0}(byte) was received.

Explanation
An invalid MQTT message type was received. The connection is disconnected.

User action
The client connected to the MQXR service is sending invalid MQTT messages. \ Find out what client has connected to the MQXR service and what data it has sent. Contact the provider of the client code. If you are using a client provided in the WebSphere MQ installation, \ contact your IBM support center.

AMQXR1004E
An invalid message version={0}(byte) subVersion={1}(byte) was received.

Explanation
An invalid MQTT message version was received. The connection is disconnected.

User action
The client connected to the MQXR service is sending invalid MQTT messages. Find out what client has connected to the MQXR service and what data it has sent. Contact the provider of the client code. If you are using a client provided in the WebSphere MQ installation, contact your IBM support center.

AMQXR1005E
An invalid message message={0}(Hex) was received.

Explanation
An invalid MQTT message was received. The connection is disconnected.

User action
The client connected to the MQXR service is sending invalid MQTT messages. Find out what client has connected to the MQXR service and what data it has sent. Contact the provider of the client code. If you are using a client provided in the WebSphere MQ installation, contact your IBM support center.

AMQXR10006E
An MQTT message with an invalid MultiByteLength={0}(long) was received.

Explanation
An invalid MQTT message containing an invalid multi-byte length was received. The connection is disconnected.

User action

The MQTT client application might have sent incorrect data, which is interpreted as an incorrect length. Check your MQTT client application, and verify that it is sending correct data. Contact the provider of the client code. If you are using a client provided in the WebSphere MQ installation, contact your IBM support center.

AMQXR1007E

An invalid Attribute type={0}(int) was found.

Explanation

An invalid MQTT attribute was found processing of this message is abandoned and the connection closed.

User action

Gather diagnostics and contact your IBM support center.

AMQXR1008E

An invalid mapped message was detected because of {0}(String).

Explanation

An invalid Mapped message was found, it cannot be processed.

User action

Determine where the message came from and correct the messages so that they are not mapped messages or are created with the correct format.

AMQXR1009E

An invalid WebSocket message was detected because of {0}(String).

Explanation

An invalid WebSocket message was found, it cannot be processed.

User action

Determine where the message came from and correct the messages so that they are correctly formed.

AMQXR1010E

An invalid message qos={0}(int) was received.

Explanation

An invalid MQTT qos was received.

User action

The client connected to the MQXR service is sending invalid MQTT messages. Find out what client has connected to the MQXR service and what data it has sent. Contact the provider of the client code. If you are using a client provided in the WebSphere MQ installation, contact your IBM support center.

AMQXR2001E

The command to end the MQXR service failed connecting to queue manager {0}. Exception: {1}

Explanation

The administrative layer could not connect to the queue manager.

User action

If the queue manager is no longer running, no action is required. If the queue manager is still running, check why the administrative layer is unable to connect.

AMQXR2002E

The command to end the MQXR service failed opening queue {0}. Exception: {1}

Explanation

The administrative layer could not open the queue that is required to send a request end the MQXR service.

User action
Determine why the queue could not be opened and retry stopping the service.

AMQXR2003E
The command to end the MQXR service failed: Failed Operation: {0} Exception ({1}): {2}

Explanation
The administrative layer failed to put or get a message that is required to stop the MQXR service.

User action
Correct the problem and then try stopping the service again.

AMQXR2004E
An error occurred while stopping the MQXR service. Completion Code: {0} Reason: {1}

Explanation
An error occurred while the MQXR service was shutting down.

User action
Use the reason code to diagnose the problem.

AMQXR2005E
An error occurred while releasing queue manager resources. Object: {0} Exception: {1}

Explanation
While cleaning up resources the EndMQXRService command encountered a transient problem.

User action
None.

AMQXR2010E
The MQXR service could not access the file: {0}. Exception: {1}

Explanation
The file is invalid, has an invalid format, or incorrect permissions.

User action
Check the file permissions and ensure the file is valid.

AMQXR2011I
Property {0} value {1}

Explanation
The runMQXRService command has read a property with the assigned value.

User action
None.

AMQXR2012E
Invalid property key={0} value={1}

Explanation
The runMQXRService command read an incorrect properties file.

User action
Look at the property in error, correct it, and reissue the command.

AMQXR2014E
Failed to rename {0} to {1}

Explanation
The file could not be renamed

User action
Look at the permissions on the target file and directory and alter them if necessary

AMQXR2013E
Duplicate authentication methods specified for channel={0}, previous={1} duplicate={2}

Explanation
The runMQXRService command read a properties file that specifies two authentication methods, only one is allowed.

User action
Look at the properties file and locate the definition of the named channel. Correct the file to specify a single authentication method and restart the channel.

AMQXR2014E
The following exception was thrown during the starting of an MQXR channel, channelName = "{0}" : {1}

Explanation
An MQXR channel was starting up but encountered a problem. Previous errors or FDCs will provide more detail.

User action
Use earlier errors or FDCs to diagnose and address the problem then restart the MQXR channel.

AMQXR2015E
The following exception was thrown during the stopping of an MQXR channel, channelName = "{0}" : {1}

Explanation
An MQXR channel was stopping but encountered a problem. Previous errors or FDCs will provide more detail.

User action
Use earlier errors or FDCs to diagnose and address the problem then restart the MQXR channel.

AMQXR2020E
Client {0} attempted to unsubscribe from the topic "{1}" which it is not subscribed to.

Explanation
An MQTT client attempted to unsubscribe from a topic it is not subscribed to.

User action
Check that the application logic is correct, and check for earlier errors that could have caused the application to get into an inconsistent state.

AMQXR2021E
Client {0} attempted to unsubscribe from the queue "{1}" which it is not subscribed to.

Explanation
An MQTT client attempted to unsubscribe from a queue it is not subscribed to.

User action
Check that the application logic is correct, and check for earlier errors that could have caused the application to get into an inconsistent state.

AMQXR2050E
Unable to load JAAS config: {0}. The following exception occurred {1}

Explanation
The JAAS configuration tried to authenticate a user on a connection that was unable to load

User action
Check that the JAAS config selected by the channel exists in the jaas.config file and is valid.

AMQXR2051E
Login failed for ClientIdentifier {0} with exception {1}.

Explanation
The JAAS login failed with the exception shown.

User action
Check that the username and password sent by the client are correct.

AMQXR2053E
Error in a trace factory. The following exception occurred {1}

Explanation
There was a problem starting or stopping trace.

User action
Use the exception to diagnose and rectify the problem and then restart trace.

AMQXR9999E
{0}

Explanation
If the message does not give sufficient information, check previous messages for further help.

User action
See previous messages for further information.

REST API messages

V 9.0.1 CD

Reference information to help you identify and interpret the messages for the IBM MQ REST API. The messages are listed in numerical order, grouped according to the part of the API from which they originate.

For details of these messages, see IBM Knowledge Center:

- MQWB00xx: REST API messages
- MQWB01xx: REST API messages
- MQWB02xx: REST API messages
- MQWB03xx: REST API messages
- V 9.0.5 MQWB04xx: REST API messages
- MQWB09xx: REST API messages
- V 9.0.5 MQWB20xx: REST API messages

Reading a message

The following information is provided for each message:

Message identifier
The message identifier is in two parts:

1. The characters "MQWB", which identify the message as being from the REST API
2. A four-digit decimal code

Message text
A summary of the message

Severity
0: Information
10: Warning
20: Error

30: Severe error
40: Stop Error
50: System Error

Explanation
An explanation of the message giving further information.

Response
The response required from the user. In some cases, particularly for information messages, this might be "No action is required".

Message variables

Some messages display text or numbers that vary according to the circumstances giving rise to the message; these are known as *message variables*. The message variables are indicated as <insert_1>, <insert_2>, and so on.

In some cases a message might have variables in the Explanation or Response. Find the values of the message variables by looking in the error log. The complete message, including the Explanation and the Response, is recorded there.

IBM MQ Console messages

V 9.0.1 CD

Reference information to help you identify and interpret the messages for the IBM MQ Console.

For details of these messages, see IBM Knowledge Center:

- IBM MQMQWB2xxx: IBM MQ Console messages

Reading a message

The following information is provided for each message:

Message identifier
The message identifier is in two parts:

1. The characters "MQWB", which identify the message as being from the IBM MQ Console
2. A four-digit decimal code

Message text
A summary of the message

Severity
0: Information
10: Warning
20: Error
30: Severe error
40: Stop Error
50: System Error

Explanation
An explanation of the message giving further information.

Response
The response required from the user. In some cases, particularly for information messages, this might be "No action is required".

Message variables

Some messages display text or numbers that vary according to the circumstances giving rise to the message; these are known as *message variables*. The message variables are indicated as <insert_1>, <insert_2>, and so on.

In some cases a message might have variables in the Explanation or Response. Find the values of the message variables by looking in the error log. The complete message, including the Explanation and the Response, is recorded there.

IBM MQ Bridge to blockchain diagnostic messages

Linux | V 9.0.4 | MQ Adv.

Reference information to help you identify and interpret the diagnostic messages for the IBM MQ Bridge to blockchain.

For details of these messages, see IBM Knowledge Center:

- IBM MQ Bridge to blockchainAMQBCxxx: IBM MQ Bridge to blockchain messages

Reading a message

The following information is provided for each message:

Message identifier
The message identifier is in two parts:

1. The characters "AMQBC", which identify the message as being from the IBM MQ Bridge to blockchain
2. A three-digit decimal code

Message text
A summary of the message

Message variables
Some messages display text or numbers that vary according to the circumstances giving rise to the message; these are known as *message variables*. The message variables are indicated as <insert_1>, <insert_2>, and so on.

IBM MQ Bridge to Salesforce diagnostic messages

V 9.0.4 | CD

Reference information to help you identify and interpret the diagnostic messages for the IBM MQ Bridge to Salesforce.

For details of these messages, see IBM Knowledge Center:

- AMQSFxxx: IBM MQ Bridge to Salesforce messages

Reading a message

The following information is provided for each message:

Message identifier
The message identifier is in two parts:

1. The characters "AMQSF", which identify the message as being from the IBM MQ Bridge to Salesforce
2. A three-digit decimal code

Message text
A summary of the message

Message variables
Some messages display text or numbers that vary according to the circumstances giving rise to the message; these are known as *message variables*. The message variables are indicated as <insert_1>, <insert_2>, and so on.

MQJMS Messages

List of messages with message numbers beginning with MQJMS.

Table 421. MQJMS Messages.

List of message numbers, constants and explanatory text for messages beginning with MQJMS.

Message identifier	Message constant	Description
MQJMS0000	MQJMS_EXCEPTION_ILLEGAL_STATE	Method "{0}" has been invoked at an illegal or inappropriate time or if the provider is not in an appropriate state for the requested operation.
MQJMS0002	MQJMS_EXCEPTION_INVALID_CLIENTID	IBM MQ classes for JMS attempted to set invalid connection's client ID.
MQJMS0003	MQJMS_EXCEPTION_INVALID_DESTINATION	Destination not understood or no longer valid.
MQJMS0004	MQJMS_EXCEPTION_INVALID_SELECTOR	IBM MQ classes for JMS has given JMS Provider a message selector with invalid syntax.
MQJMS0005	MQJMS_EXCEPTION_MESSAGE_EOF	Unexpected end of stream has been reached when a StreamMessage or BytesMessage is being read.
MQJMS0006	MQJMS_EXCEPTION_MESSAGE_FORMAT	IBM MQ classes for JMS attempts to use a data type not supported by a message or attempts to read data in the wrong type.
MQJMS0007	MQJMS_EXCEPTION_MESSAGE_NOT_READABLE	IBM MQ classes for JMS attempts to read a write-only message.
MQJMS0008	MQJMS_EXCEPTION_MESSAGE_NOT_WRITABLE	IBM MQ classes for JMS attempts to write a read-only message.
MQJMS0009	MQJMS_EXCEPTION_RESOURCE_ALLOCATION	IBM MQ classes for JMS is unable to allocate the resources required for a method.
MQJMS0010	MQJMS_EXCEPTION_TRANSACTION_IN_PROGRESS	Operation invalid because a transaction is in progress.
MQJMS0011	MQJMS_EXCEPTION_TRANSACTION_ROLLED_BACK	Call to Session.commit resulted in a rollback of the current transaction.
MQJMS1000	MQJMS_EXCEPTION_MSG_CREATE_ERROR	Failed to create JMS message.
MQJMS1001	MQJMS_EXCEPTION_UNKNOWN_ACK_MODE	Unknown acknowledge mode "{0}".
MQJMS1004	MQJMS_EXCEPTION_CONNECTION_CLOSED	Connection closed.
MQJMS1005	MQJMS_EXCEPTION_BAD_STATE_TRANSITION	Unhandled state transition from "{0}" to "{1}".
MQJMS1006	MQJMS_EXCEPTION_BAD_VALUE	invalid value for "{0}": "{1}".
MQJMS1007	MQJMS_E_BAD_EXIT_CLASS	failed to create instance of exit class "{0}".
MQJMS1008	MQJMS_E_UNKNOWN_TRANSPORT	unknown value of transportType: "{0}".
MQJMS1009	MQJMS_E_NO_STR_CONSTRUCTOR	no constructor with string argument.
MQJMS1010	MQJMS_E_NOT_IMPLEMENTED	not implemented.
MQJMS1011	MQJMS_E_SECURITY_CREDS_INVALID	security credentials cannot be specified when using MQ bindings.

Table 421. MQJMS Messages (continued).

List of message numbers, constants and explanatory text for messages beginning with MQJMS.

Message identifier	Message constant	Description
MQJMS1012	MQJMS_E_NO_MSG_LISTENER	no message listener.
MQJMS1013	MQJMS_E_SESSION_ASYNC	operation invalid whilst session is using asynchronous delivery.
MQJMS1014	MQJMS_E_IDENT_PRO_INVALID_OP	operation invalid for identified producer.
MQJMS1015	MQJMS_E_UNKNOWN_TARGET_CLIENT	unknown value of target client: "{0}".
MQJMS1016	MQJMS_E_INTERNAL_ERROR	an internal error has occurred. Please contact your system administrator. Detail: "{0}".
MQJMS1017	MQJMS_E_NON_LOCAL_RXQ	non-local MQ queue not valid for receiving or browsing.
MQJMS1018	MQJMS_E_NULL_CONNECTION	no valid connection available.
MQJMS1019	MQJMS_E_SESSION_NOT_TRANSACTED	invalid operation for non-transacted session.
MQJMS1020	MQJMS_E_SESSION_IS_TRANSACTED	invalid operation for transacted session.
MQJMS1021	MQJMS_E_RECOVER_BO_FAILED	recover failed: unacknowledged messages might not get redelivered.
MQJMS1022	MQJMS_E_REDIRECT_FAILED	failed to redirect message.
MQJMS1023	MQJMS_E_ROLLBACK_FAILED	rollback failed.
MQJMS1024	MQJMS_E_SESSION_CLOSED	session closed.
MQJMS1025	MQJMS_E_BROWSE_MSG_FAILED	failed to browse message.
MQJMS1026	MQJMS_E_EXCP_LSTNR_FAILED	ExceptionListener threw exception: "{0}".
MQJMS1027	MQJMS_E_BAD_DEST_STR	failed to reconstitute destination from "{0}".
MQJMS1028	MQJMS_EXCEPTION_NULL_ELEMENT_NAME	element name is null.
MQJMS1029	MQJMS_EXCEPTION_NULL_PROPERTY_NAME	property name is null.
MQJMS1030	MQJMS_EXCEPTION_BUFFER_TOO_SMALL	buffer supplied by application is too small.
MQJMS1031	MQJMS_EXCEPTION_UNEXPECTED_ERROR	an internal error has occurred. Please contact your system administrator.
MQJMS1032	MQJMS_E_CLOSE_FAILED	close() failed because of "{0}".
MQJMS1033	MQJMS_E_START_FAILED	start() failed because of "{0}".
MQJMS1034	MQJMS_E_MSG_LSTNR_FAILED	MessageListener threw: "{0}".
MQJMS1042	MQJMS_E_DELIVERY_MODE_INVALID	invalid Delivery Mode.
MQJMS1044	MQJMS_E_INVALID_HEX_STRING	String is not a valid hexadecimal number - "{0}".
MQJMS1045	MQJMS_E_S390_DOUBLE_TOO_BIG	Number outside of range for double precision S/390 Float "{0}".
MQJMS1046	MQJMS_E_BAD_CCSID	The character set "{0}" is not supported.
MQJMS1047	MQJMS_E_INVALID_MAP_MESSAGE	The map message has an incorrect format.
MQJMS1048	MQJMS_E_INVALID_STREAM_MESSAGE	The stream message has an incorrect format.
MQJMS1049	MQJMS_E_BYTE_TO_STRING	The IBM MQ classes for JMS attempted to convert a byte array to a String.
MQJMS1050	MQJMS_E_BAD_RFH2	The MQRFH2 header has an incorrect format.
MQJMS1051	MQJMS_MSG_CLASS	JMS Message class.
MQJMS1052	MQJMS_E_BAD_MSG_CLASS	Unrecognizable JMS Message class.

Table 421. MQJMS Messages (continued).

List of message numbers, constants and explanatory text for messages beginning with MQJMS.

Message identifier	Message constant	Description
MQJMS1053	MQJMS_E_INVALID_SURROGATE	Invalid UTF-16 surrogate detected "{0}".
MQJMS1054	MQJMS_E_INVALID_ESCAPE	Invalid XML escape sequence detected "{0}".
MQJMS1055	MQJMS_E_BAD_TYPE	The property or element in the message has incompatible datatype "{0}".
MQJMS1056	MQJMS_E_UNSUPPORTED_TYPE	Unsupported property or element datatype "{0}".
MQJMS1057	MQJMS_E_NO_SESSION	Message has no session associated with it.
MQJMS1058	MQJMS_E_BAD_PROPERTY_NAME	Invalid message property name: "{0}".
MQJMS1059	MQJMS_E_NO_UTF8	Fatal error - UTF8 not supported.
MQJMS1060	MQJMS_E_SERIALISE_FAILED	Unable to serialize object.
MQJMS1061	MQJMS_E_DESERIALISE_FAILED	Unable to deserialize object.
MQJMS1062	MQJMS_EXCEPTION_HAPPENED	Exception occurred reading message body: "{0}".
MQJMS1063	MQJMS_CHARS_OMITTED	Another "{0}" character(s) omitted.
MQJMS1064	MQJMS_ENCODINGS	Integer encoding: "{0}"=Floating point encoding "{1}".
MQJMS1065	MQJMS_E_COULD_NOT_WRITE	Exception occurred writing message body.
MQJMS1066	MQJMS_E_BAD_ELEMENT_NAME	Invalid message element name: "{0}".
MQJMS1067	MQJMS_E_BAD_TIMEOUT	timeout invalid for MQ.
MQJMS1068	MQJMS_E_NO_XARESOURCE	failed to obtain XAResource.
MQJMS1069	MQJMS_E_NOT_ALLOWED_WITH_XA	Not allowed with XASession.
MQJMS1072	MQJMS_E_QMGR_NAME_INQUIRE_FAILED	Could not inquire upon Queue Manager name.
MQJMS1073	MQJMS_E_QUEUE_NOT_LOCAL_OR_ALIAS	Specified MQ Queue is neither a QLOCAL nor a QALIAS.
MQJMS1074	MQJMS_E_NULL_MESSAGE	Unable to process null message.
MQJMS1075	MQJMS_E_DLH_WRITE_FAILED	Error writing dead letter header.
MQJMS1076	MQJMS_E_DLH_READ_FAILED	Error reading dead letter header.
MQJMS1077	MQJMS_E_CONN_DEST_MISMATCH	Connection/destination mismatch.
MQJMS1078	MQJMS_E_INVALID_SESSION	Invalid Session object.
MQJMS1079	MQJMS_E_DLQ_FAILED	Unable to write message to dead letter queue.
MQJMS1080	MQJMS_E_NO_BORQ	No Backout-Requeue queue defined.
MQJMS1081	MQJMS_E_REQUEUE_FAILED	Message requeue failed.
MQJMS1082	MQJMS_E_DISCARD_FAILED	Failure while discarding message.
MQJMS1085	MQJMS_E_RFH_WRITE_FAILED	Error writing RFH.
MQJMS1086	MQJMS_E_RFH_READ_FAILED	Error reading RFH.
MQJMS1087	MQJMS_E_RFH_CONTENTS_ERROR	Unrecognizable or invalid RFH content.
MQJMS1088	MQJMS_E_CC_MIXED_DOMAIN	Mixed-domain consumers acting on the same input is forbidden.
MQJMS1089	MQJMS_E_READING_MSG	Exception occurred reading message body: "{0}".
MQJMS1091	MQJMS_E_UNIDENT_PRO_INVALID_OP	operation invalid for unidentified producer.

Table 421. MQJMS Messages (continued).

List of message numbers, constants and explanatory text for messages beginning with MQJMS.

Message identifier	Message constant	Description
MQJMS1093	MQJMS_E_NULL_PARAMETER	A null parameter was passed to the constructor: "{0}".
MQJMS1094	MQJMS_E_INVALID_QUANTITY_HINT	Invalid quantityHint.
MQJMS1096	MQJMS_E_INVALID_MESSAGE_REFERENCE	Invalid MessageReference.
MQJMS1098	MQJMS_E_INVALID_MSG_REF_VERSION	Invalid MessageReference version.
MQJMS1099	MQJMS_E_INVALID_THREAD_VERSION	Invalid MQQueueAgentThread version.
MQJMS1102	MQJMS_E_MULTICAST_NOT_AVAILABLE	Multicast connection cannot be established.
MQJMS1103	MQJMS_E_MULTICAST_LOST_MESSAGES	Lost "{0}" messages in reliable multicast mode.
MQJMS1104	MQJMS_E_MULTICAST_HEARTBEAT_TIMEOUT	Multicast connection disconnected due to timeout.
MQJMS1105	MQJMS_E_MULTICAST_PORT_INVALID	Cannot connect with a specific local port for disthub multicast.
MQJMS1106	MQJMS_DIR_PGM_LIB_NOT_FOUND	Unable to load the native library required for PGM/IP.
MQJMS1110	MQJMS_E_11_NOTSUPPORTED	JMS1.1 Operation not supported by this type.
MQJMS1111	MQJMS_E_11_SERVICES_NOT_SETUP	JMS1.1 The required Queues/Publish Subscribe services are not set up.
MQJMS1112	MQJMS_E_11_INVALID_DOMAIN_SPECIFIC	JMS1.1 Invalid operation for domain specific object.
MQJMS1113	MQJMS_E_11_INVALID_CROSS_DOMAIN	JMS1.1 Invalid operation for cross domain object.
MQJMS2000	MQJMS_EXCEPTION_MQ_Q_CLOSE_FAILED	failed to close MQ queue.
MQJMS2001	MQJMS_EXCEPTION_MQ_NULL_Q	MQ Queue reference is null.
MQJMS2002	MQJMS_EXCEPTION_GET_MSG_FAILED	failed to get message from MQ queue.
MQJMS2003	MQJMS_EXCEPTION_QMDISC_FAILED	failed to disconnect queue manager.
MQJMS2004	MQJMS_EXCEPTION_MQ_NULL_QMGR	MQQueueManager reference is null.
MQJMS2005	MQJMS_EXCEPTION_QMGR_FAILED	failed to create MQQueueManager for "{0}".
MQJMS2006	MQJMS_EXCEPTION_SOME_PROBLEM	MQ problem: "{0}".
MQJMS2007	MQJMS_EXCEPTION_PUT_MSG_FAILED	failed to send message to MQ queue.
MQJMS2008	MQJMS_EXCEPTION_MQ_Q_OPEN_FAILED	failed to open MQ queue "{0}".
MQJMS2009	MQJMS_EXCEPTION_MQ_QM_COMMIT_FAILED	MQQueueManager.commit() failed.
MQJMS2010	MQJMS_EXCEPTION_MQ_UNKNOWN_DEFTYPE	unknown value for MQ queue definitionType: "{0}".
MQJMS2011	MQJMS_EXCEPTION_MQ_Q_INQUIRE_FAILED	failed to inquire MQ queue depth.
MQJMS2012	MQJMS_EXCEPTION_XACLOSE_FAILED	XACLOSE failed.
MQJMS2013	MQJMS_EXCEPTION_AUTHENTICATION_FAILED	invalid security authentication supplied for MQQueueManager.
MQJMS2014	MQJMS_EXCEPTION_XACLIENT_FAILED	Queue manager rejected XA client connection.
MQJMS3000	MQJMS_E_TMPQ_FAILED	failed to create a temporary queue from "{0}".
MQJMS3001	MQJMS_E_TMPQ_CLOSED	temporary queue already closed or deleted.
MQJMS3002	MQJMS_E_TMPQ_INUSE	temporary queue in use.
MQJMS3003	MQJMS_E_TMPQ_DEL_STATIC	cannot delete a static queue.
MQJMS3004	MQJMS_E_TMPQ_DEL_FAILED	failed to delete temporary queue.

Table 421. MQJMS Messages (continued).

List of message numbers, constants and explanatory text for messages beginning with MQJMS.

Message identifier	Message constant	Description
MQJMS3005	MQJMS_PS_GENERAL_ERROR	Publish/Subscribe failed due to "{0}".
MQJMS3006	MQJMS_PS_TOPIC_NULL	Topic reference is null.
MQJMS3008	MQJMS_PS_COMMAND_MSG_BUILD	Failed to build command "{0}".
MQJMS3009	MQJMS_PS_COMMAND_MSG_FAILED	Failed to publish command to MQ queue.
MQJMS3010	MQJMS_PS_PUBLISH_MSG_BUILD	Failed to build publish message.
MQJMS3011	MQJMS_PS_PUBLISH_MSG_FAILED	Failed to publish message to MQ queue.
MQJMS3013	MQJMS_PS_STORE_ADMIN_ENTRY	Failed to store admin entry.
MQJMS3014	MQJMS_PS_SUB_Q_OPEN_FAILED	Failed to open subscriber queue "{0}".
MQJMS3017	MQJMS_PS_SUB_Q_DELETE_FAILED	Failed to delete subscriber queue "{0}".
MQJMS3018	MQJMS_PS_UNKNOWN_DS	Unknown durable subscription "{0}".
MQJMS3019	MQJMS_E_TMPT_DELETED	TemporaryTopic already deleted.
MQJMS3020	MQJMS_E_TMPT_OUTOFSCOPE	TemporaryTopic out of scope.
MQJMS3021	MQJMS_PS_INVALID_SUBQ_PREFIX	Invalid subscriber queue prefix: "{0}".
MQJMS3022	MQJMS_PS_SUBQ_REQUEUE	Durable re-subscribe must use same subscriber queue; specified:"{0}" original:"{1}".
MQJMS3023	MQJMS_PS_SUB_ACTIVE	Subscription has an active TopicSubscriber.
MQJMS3024	MQJMS_PS_NULL_CLIENTID	Illegal use of uninitialized client ID.
MQJMS3025	MQJMS_E_TMPT_IN_USE	TemporaryTopic in use.
MQJMS3026	MQJMS_ERR_QSENDER_CLOSED	QueueSender is closed.
MQJMS3028	MQJMS_PUBLISHER_CLOSED	TopicPublisher is closed.
MQJMS3031	MQJMS_CLIENTID_FIXED	Can't set clientID after connection has been used.
MQJMS3032	MQJMS_CLIENTID_NO_RESET	Resetting the clientID is not allowed.
MQJMS3033	MQJMS_QRECEIVER_CLOSED	QueueReceiver is closed.
MQJMS3034	MQJMS_SUBSCRIBER_CLOSED	TopicSubscriber is closed.
MQJMS3037	MQJMS_MESSAGEPRODUCER_CLOSED	Message Producer is closed.
MQJMS3038	MQJMS_MESSAGECONSUMER_CLOSED	Message Consumer is closed.
MQJMS3039	MQJMS_PS_NULL_NAME	Illegal use of null name.
MQJMS3040	MQJMS_E_BROKER_MESSAGE_CONTENT	Invalid broker control message content: "{0}".
MQJMS3041	MQJMS_E_ALREADY_SET	Field "{0}" already set.
MQJMS3042	MQJMS_E_UNREC_BROKER_MESSAGE	Unrecognizable message from Pub/Sub Broker.
MQJMS3043	MQJMS_E_CLEANUP_REP_BAD_LEVEL	Invalid Level for repeating Cleanup.
MQJMS3044	MQJMS_E_CLEANUP_NONE_REQUESTED	Cleanup level of NONE requested.
MQJMS3045	MQJMS_E_CLEANUP_Q_OPEN_1	Failed to open "{0}": maybe a FORCE or NONDUR level cleanup is running?
MQJMS3046	MQJMS_E_CLEANUP_Q_OPEN_2	Failed to open "{0}": maybe another JMS application is using Pub/Sub with this queue manager?
MQJMS3047	MQJMS_PS_SUBSTORE_NOT_SUPPORTED	Subscription Store type not supported by queue manager.
MQJMS3048	MQJMS_PS_INCORRECT_SUBSTORE	Incorrect Subscription Store type.

Table 421. MQJMS Messages (continued).

List of message numbers, constants and explanatory text for messages beginning with MQJMS.

Message identifier	Message constant	Description
MQJMS3049	MQJMS_PS_WRONG_SUBSCRIPTION_TYPE	MQJMS_Messages.MQJMS_PS_ WRONG_SUBSCRIPTION_TYPE = Incorrect Subscription type for this Subscription Store.
MQJMS3050	MQJMS_PS_SUBSCRIPTION_IN_USE	Subscription is already in use and cannot be updated.
MQJMS3051	MQJMS_PS_INVALID_SUB_NAME	Invalid Subscription name.
MQJMS4124	MQJMS_ADMIN_PROPVAL_NULL	Property value for "{0}" is null.
MQJMS4125	MQJMS_ADMIN_INV_PROP	Invalid property for a "{0}": "{1}".
MQJMS4131	MQJMS_ADMIN_OBJTYPE_MISMATCH	Expected and actual object types do not match.
MQJMS5053	MQJMS_UTIL_PS_NO_BROKER	*** No broker response. Please ensure that the broker is running. If you are using the IBM MQ broker check that your brokerVersion is set to V1 ***
MQJMS5087	MQJMS_UTIL_PS_INTERNALQ	Unexpected error "{1}" accessing internal queue "{0}".
MQJMS6040	MQJMS_DIR_IMB_BADSOCKNAME	Invalid socket family name: "{0}".
MQJMS6041	MQJMS_DIR_IMB_NOCLASS	An exception occurred while attempting to load socket factory class "{0}", exception: <"{1}">.
MQJMS6056	MQJMS_DIR_MIN_NOMORE	Cannot change parameter "{0}" since no more BaseConfig parameter changes are allowed.
MQJMS6057	MQJMS_DIR_MIN_BADSET	Cannot set parameter "{0}" to value "{1}".
MQJMS6058	MQJMS_DIR_MIN_BADGET	error occurred while getting BaseConfig parameter "{0}".
MQJMS6059	MQJMS_DIR_MIN_SECLDERR	An exception occurred while loading the minimal client security implementation.
MQJMS6060	MQJMS_DIR_MIN_UNXEXC	An unexpected exception in minimal client, "{0}".
MQJMS6061	MQJMS_DIR_MIN_BADTOP	A specified topic was malformed, "{0}".
MQJMS6062	MQJMS_DIR_MIN_EOF	EOF was encountered while receiving data in the minimal client.
MQJMS6063	MQJMS_DIR_MIN_BRKERR	The broker indicated an error on the minimal client connection.
MQJMS6064	MQJMS_DIR_MIN_BADMSG	Connector.send was called with an illegal message value.
MQJMS6065	MQJMS_DIR_MIN_BADFIELD	An illegal value was encountered for a field, "{0}".
MQJMS6066	MQJMS_DIR_MIN_INTERR	An unexpected internal error occurred in the minimal client.
MQJMS6067	MQJMS_DIR_MIN_NOTBYTES	A bytes message operation was requested on something that is not a bytes message.
MQJMS6068	MQJMS_DIR_MIN_NOTTEXT	A text message operation was requested on something that is not a text message.
MQJMS6069	MQJMS_DIR_MIN_NOTSTREAM	A stream message operation was requested on something that is not a stream message.
MQJMS6070	MQJMS_DIR_MIN_NOTMAP	A map message operation was requested on something that is not a map message.

Table 421. MQJMS Messages (continued).

List of message numbers, constants and explanatory text for messages beginning with MQJMS.

Message identifier	Message constant	Description
MQJMS6071	MQJMS_DIR_MIN_BADBRKMSG	The broker sent an invalid message during authentication.
MQJMS6072	MQJMS_DIR_MIN_UNVPRO	The broker requested an unavailable protocol during authentication.
MQJMS6073	MQJMS_DIR_MIN_AUTHREJ	Minimal client connection rejected because of authentication failure.
MQJMS6074	MQJMS_DIR_MIN_NOQOP	No QOP available in the minimal client.
MQJMS6079	MQJMS_DIR_JMS_NOTHDPOOL	An exception occurred while attempting to load thread pooling support, "{0}".
MQJMS6081	MQJMS_DIR_JMS_FMTINT	An attempt was made to read from a Stream message before a previous read has completed.
MQJMS6083	MQJMS_DIR_JMS_THDEXC	An exception occurred while initializing a thread pool instance, "{0}".
MQJMS6085	MQJMS_DIR_JMS_NEXCLIS	No ExceptionListener has been set.
MQJMS6088	MQJMS_DIR_JMS_KILLMON	The client-side connection monitor is terminating.
MQJMS6090	MQJMS_DIR_JMS_LSTACT	Attempted to synchronously receive on a MessageConsumer for which a listener is active.
MQJMS6091	MQJMS_DIR_JMS_TCSTSTP	An IOException occurred when starting or stopping delivery on the connection, "{0}".
MQJMS6093	MQJMS_DIR_JMS_RUNKEXC	An exception occurred during synchronous receive, "{0}".
MQJMS6096	MQJMS_DIR_JMS_INVPRI	A JMSPriority level of "{0}" is outside the range specified in JMS.
MQJMS6097	MQJMS_DIR_JMS_BADID	The specified JMSMessageID, "{0}", is invalid.
MQJMS6105	MQJMS_DIR_JMS_NOMORE	No more client parameter changes allowed.
MQJMS6106	MQJMS_DIR_JMS_BADNUM	An exception occurred when initializing parameter "{0}", exception "{1}".
MQJMS6115	MQJMS_DIR_JMS_TCFLERR	An exception occurred while creating the TopicConnection, "{0}".
MQJMS6116	MQJMS_DIR_JMS_CLOSED	This operation is not permitted on an entity that is closed.
MQJMS6117	MQJMS_DIR_JMS_BDTOPIMPL	The "{0}" implementation of Topic is not supported.
MQJMS6118	MQJMS_DIR_JMS_PBNOWLD	Topic "{0}" contains a wildcard which is invalid for publishing.
MQJMS6119	MQJMS_DIR_JMS_PBIOERR	An IOException occurred while publishing, "{0}".
MQJMS6120	MQJMS_DIR_JMS_TMPVIO	Attempted to use a temporary topic not created on the current connection.
MQJMS6121	MQJMS_DIR_JMS_TSIOERR	An IOException occurred while subscribing, "{0}".
MQJMS6232	MQJMS_DIR_JMS_TSBADMTC	While creating a TopicSubscriber, attempting to add the subscription to the matching engine resulted in the following exception: "{0}".
MQJMS6233	MQJMS_DIR_MTCH_UNKEXC	An unexpected exception was caught in the matching engine: "{0}".

Table 421. MQJMS Messages (continued).

List of message numbers, constants and explanatory text for messages beginning with MQJMS.

Message identifier	Message constant	Description
MQJMS6234	MQJMS_DIR_MTCH_NULRM	An attempt was made to remove an object with topic "{0}" from an empty matching engine: "{1}".
MQJMS6235	MQJMS_DIR_MTCH_NULCH	An attempt was made to remove an object with topic "{0}" from the matching engine, but it did not have a cache entry: "{1}".
MQJMS6236	MQJMS_DIR_MTCH_BDTYP	An unknown check type of class "{0}" was encountered in a type-specific matcher.
MQJMS6237	MQJMS_DIR_MTCH_UNKNM	An attempt was made to access an unknown field named "{0}".
MQJMS6238	MQJMS_DIR_MTCH_BDMSG	In attempting to access a field of a message=the following exception occurred: "{0}".
MQJMS6239	MQJMS_DIR_MTCH_ECPREP	An EvalCache get or put operation occurred when the cache was not loaded.
MQJMS6240	MQJMS_DIR_MTCH_ECNMIN	An EvalCache get or put operation specified an invalid ID.
MQJMS6241	MQJMS_DIR_MTCH_TOMNY	Too many content attributes were specified.
MQJMS6242	MQJMS_DIR_MTCH_DUPDET	A duplicate MatchTarget was detected in MatchSpace.
MQJMS6243	MQJMS_DIR_MTCH_NOTPK	An attempt was made to remove MatchTarget "{0}" from MatchSpace, but it has no key (topic).
MQJMS6244	MQJMS_DIR_MTCH_NOSUB	The MatchTarget "{1}" with key (topic) "{0}" could not be removed from MatchSpace because it could not be found.
MQJMS6245	MQJMS_DIR_MTCH_NLTOP	An attempt was made to add a MatchTarget to MatchSpace without a key (topic).
MQJMS6246	MQJMS_DIR_MTCH_BDWLD	An incorrect use of a the topic wildcard character "{0}" was detected.
MQJMS6247	MQJMS_DIR_MTCH_BDSEP	The topic segment separator "{0}" appears in an incorrect position.
MQJMS6248	MQJMS_DIR_MTCH_CNTLD	An error occurred while trying to load or invoke the subscription selector parser.
MQJMS6249	MQJMS_DIR_MTCH_PSTPER	The following exception occurred while parsing a subscription selector: "{0}".
MQJMS6250	MQJMS_DIR_MTCH_BDESC	The escape character was used to terminate the following pattern: "{0}".
MQJMS6251	MQJMS_DIR_MTCH_BDESCL	The escape character "{0}" passed to the pattern tool is longer than one character.
MQJMS6252	MQJMS_DIR_MTCH_UNXTYP	A message field was expected to contain a value of type "{0}" but contained one of type "{1}".
MQJMS6228	MQJMS_DIR_MIN_AUTHEXC	Minimal client authentication failed because exception "{0}".
MQJMS6229	MQJMS_DIR_MIN_QOPDIS	QOP required but disabled for this minimal client.
MQJMS6312	MQJMS_DIR_MIN_NOSUB	Non-authorized subscription to topic "{0}".

Table 421. MQJMS Messages (continued).

List of message numbers, constants and explanatory text for messages beginning with MQJMS.

Message identifier	Message constant	Description
MQJMS6311	MQJMS_DIR_MIN_NOXASUP	Transport type 'DIRECT' within a transaction is not supported.
MQJMS6350	MQJMS_DIR_MIN_NOTOBJECT	An object message operation was requested on something that is not an object message.
MQJMS6351	MQJMS_DIR_MIN_TSBADSYN	An exception occurred when creating subscription to <''{0}'',''{1}''>, ''{2}''.
MQJMS6401	MQJMS_DIR_MIN_PER_NOT_SUPPORTED	Persistent messages not supported for transport type 'DIRECT'.
MQJMS6402	MQJMS_DIR_MIN_TTL_NOT_SUPPORTED	Time to Live > 0 not supported for transport type 'DIRECT'.
MQJMS6403	MQJMS_DIR_MIN_EXP_NOT_SUPPORTED	Topic Expiry > 0 not supported for transport type 'DIRECT'.
MQJMS6404	MQJMS_DIR_MIN_ACK_NOT_SUPPORTED	Client Acknowledge not supported for transport type 'DIRECT'.

Related information:
WMQ JMS Exception Messages

JSON format diagnostic messages

V 9.0.5

The following table describes the name, value pairs that make up the JSON format IBM MQ diagnostic messages.

See QMErrorLog service for more information on diagnostic messages.

If you write an error log file in JSON format, each error message contains single lines of JSON,

Table 422. Name/value pairs in the message object

name	Type	Description
host	string	The host name
ibm_arithInsert1	number	The first message variable.
ibm_arithInsert2	number	The second message variable.
ibm_commentInsert1	string	The third message variable, if required.
ibm_commentInsert2	string	The fourth message variable, if required.
ibm_commentInsert3	string	The fifth message variable, if required.
ibm_datetime	string	An ISO 8601 formatted timestamp indicating when the message was generated. Of the form YYYY-MM-DDTHH:MM:SS.mmmZ, always in UTC.
ibm_installationDir	string	The installation path. Included because it allows a parsing program on the machine to run appropriate commands from the installation.
ibm_installationName	string	The installation name.

Table 422. Name/value pairs in the message object (continued)

name	Type	Description
ibm_messageID	string	The diagnostic message identifier including the severity character, for example, AMQ6209W.
ibm_processID	number	The process identifier.
ibm_processName	string	The process, or job name on IBM i, for example, `amqzxma0`.
ibm_qmgrId	string	An identifier for the queue manager.
ibm_remoteHost	string	IP address of the associated client program, if there is one.
ibm_sequence	string	Sequence number of message; intended to differentiate between messages produced at the same time.
ibm_servername	string	The name of the queue manager.
ibm_threadId	number	The IBM MQ thread identifier within the process.
ibm_userName	string	The real name of the user under which the process is running.
ibm_version	string	IBM MQ Version, Release, Modification, Fix pack (VRMF) information.
loglevel	string	Either, INFO, WARNING, or ERROR.
message	string	A summary of the message, including the identifier, with inserts expanded.
module	string	The source file and line number where the message was generated, for example, `amqxerrx.c:243`.
type	string	mq_log

Example message

The following message is displayed on multiple lines, but IBM MQ typically writes the message on a single line.

```
{
  "ibm_messageId":"AMQ9209E",
  "ibm_arithInsert1":0,
  "ibm_arithInsert2":0,
  "ibm_commentInsert1":"localhost (127.0.0.1)",
  "ibm_commentInsert2":"TCP/IP",
  "ibm_commentInsert3":"SYSTEM.DEF.SVRCONN",
  "ibm_datetime":"2018-02-22T06:54:53.942Z",
  "ibm_serverName":"QM1",
  "type":"mq_log",
  "host":"0df0ce19c711",
  "loglevel":"ERROR",
  "module":"amqccita.c:4214",
  "ibm_sequence":"1519282493_947814358",
  "ibm_remoteHost":"127.0.0.1",
  "ibm_qmgrId":"QM1_2018-02-13_10.49.57",
  "ibm_processId":4927,
  "ibm_threadId":4,
  "ibm_version":"9.0.5.0",
  "ibm_processName":"amqrmppa",
  "ibm_userName":"johndoe",
  "ibm_installationName":"Installation1",
  "ibm_installationDir":"/opt/mqm",
  "message":"AMQ9209E: Connection to host 'localhost (127.0.0.1)' for channel 'SYSTEM.DEF.SVRCONN' closed."
}
```

IBM MQ for z/OS messages, completion, and reason codes

z/OS

Use this topic to interpret and understand the messages and codes issued by IBM MQ for z/OS.

The information in this topic can be used to understand a message or code produced by the IBM MQ for z/OS product. The topic is divided into the following parts:

"Messages for IBM MQ for z/OS" on page 4355
Describes all IBM MQ messages in alphanumeric order.

All IBM MQ message identifiers are eight characters long. The first three characters are always CSQ. If you get a message with a different prefix, see "Messages from other products" on page 5226 to find out which product issued the message.

The fourth character is the component identifier; this identifies the component of IBM MQ that issued the message. These are shown in "IBM MQ component identifiers" on page 5210. The fifth through seventh characters represent the numeric identifier, which is unique within the component. The last character is the message type code; this indicates the type of response that the message requires. Table 423 shows the four type codes used by IBM MQ for z/OS.

Table 423. Message type codes

Type code	Response type	Response required
A	Immediate action	System operator action is required immediately. The associated task does not continue until the requested action has been taken.
D	Immediate decision	System operator decision or action is required immediately. The operator is requested to select from specific options, such as **retry** or **cancel**. The associated task does not continue until the requested decision has been made or action has been taken.
E	Eventual action	System operator action *will* be required; however, the associated task continues independently of system operator action.
I	Information only	No operator action is required. However, certain messages may be significant - please review Console message monitoring for further information.

In messages issued by the queue manager itself and the mover, the message identifier is normally followed by the *command prefix* (CPF); this indicates which IBM MQ queue manager issued the message. These messages have prefixes starting CSQE, CSQH, CSQI, CSQM, CSQN, CSQP, CSQR, CSQV, CSQX, CSQY, CSQ2, CSQ3, CSQ5, and CSQ9; some messages with prefixes CSQJ and CSQW also have the CPF. In certain exceptional cases, the CPF might show as blank.

Messages from CICS-related components (CSQC) show the CICS application ID or transaction ID if applicable.

Messages from other components, that is messages with prefixes CSQO, CSQQ, CSQU, and CSQ1 (and some with prefixes CSQJ and CSQW) have no indicator.

"IBM MQ for z/OS codes" on page 5019
Describes all IBM MQ abend reason codes, and subsystem termination reason codes, in alphanumeric order.

The codes are four bytes long. The first byte is always 00; this is the high-order byte. The second byte is the hexadecimal identifier (Hex ID) of the IBM MQ component. These are shown in "IBM MQ component identifiers" on page 5210. The last two bytes are the numeric identifier, which is unique within the component.

"IBM MQ CICS adapter abend codes" on page 5209 and "IBM MQ CICS bridge abend codes" on page 5209 Describe the CICS abend codes issued by the IBM MQ CICS adapter, and the IBM MQ CICS bridge.

Accompanying each message and code is the following information, when applicable:

Explanation:
This section tells what the message or code means, why it occurred, and what caused it.

Severity:
Severity values have the following meanings:

0 An information message. No error has occurred.

4 A warning message. A condition has been detected of which the user should be aware. The user might need to take further action.

8 An error message. An error has been detected and processing could not continue.

12 A severe error message. A severe error has been detected and processing could not continue.

System action:
This part tells what is happening as a result of the condition causing the message or code. If this information is not shown, no system action is taken.

User response:
If a response by the user is necessary, this section tells what the appropriate responses are, and what their effect is. If this information is not shown, no user response is required.

Operator response:
If an operator response is necessary, this section tells what the appropriate responses are, and what their effect is. If this information is not shown, no operator response is required.

System programmer response:
If a response by the system programmer is required, this part tells what the appropriate responses are, and what their effect is. If this information is not shown, no system programmer response is required.

Programmer response:
If a programmer response is necessary, this part tells what the appropriate responses are, and what their effect is. If this information is not shown, no programmer response is required.

Problem determination:
This section lists the actions that can be performed to obtain adequate data for support personnel to diagnose the cause of the error. If this information is not shown, no problem determination is required.

Related reference:

"IBM MQ messages on Multiplatforms" on page 4328
IBM MQ diagnostic messages are listed in this section in numerical order, grouped according to the part of IBM MQ from which they originate.

"Communications protocol return codes" on page 5211
The communication protocols used by IBM MQ for z/OS can issue their own return codes. Use these tables to identify the return codes used by each protocol.

"Distributed queuing message codes" on page 5224
Distributed queuing is one of the components of IBM MQ for z/OS. Use this topic to interpret the message codes issued by the distributed queuing component.

"Transport Layer Security (TLS) return codes for z/OS" on page 5221
IBM MQ for z/OS can use TLS with the various communication protocols. Use this topic to identify the error codes that can be returned by TLS.

Related information:

API completion and reason codes

PCF reason codes

Transport Layer Security (TLS) return codes

WCF custom channel exceptions

Messages for IBM MQ for z/OS

z/OS

Each component of IBM MQ for z/OS can issue messages and each component uses a unique four character prefix for its messages. Use this topic to identify and interpret the messages for IBM MQ for z/OS components.

The following message types are described:

Batch adapter messages (CSQB...): z/OS

CSQB001E

Language environment programs running in z/OS batch or USS must use the DLL interface to IBM MQ

Severity

4

Explanation

Application programs using IBM MQ and Language Environment® services from z/OS Batch or Unix System Services must use the DLL interface to IBM MQ. This message is issued once per connection. The program which caused this message to be issued is using the stub interface to IBM MQ.

System action

Processing continues. The Async Consume feature of IBM MQ is not supported when using the non-DLL stub interface to IBM MQ.

CICS adapter and Bridge messages (CSQC...): z/OS

All the CICS versions supported by IBM MQ Version 9.0.0, and later, use the CICS supplied version of the bridge. See the DFHMQnnnn messages section of the CICS documentation for these messages.

Coupling Facility manager messages (CSQE...): z/OS

The value shown for *struc-name* in the coupling facility manager messages that follow is the 12-character name as used by IBM MQ. The external name of such CF structures for use by z/OS is formed by prefixing the IBM MQ name with the name of the queue-sharing group to which the queue manager is connected.

CSQE005I
Structure *struc-name* connected as *conn-name*, version=*version*

Explanation

The queue manager has successfully connected to structure *struc-name*.

System action

Processing continues. The queue manager can now access the CF structure.

CSQE006I
Structure *struc-name* connection name *conn-name* disconnected

Explanation

The queue manager has disconnected from CF structure *struc-name*.

System action

Processing continues.

CSQE007I
event-type event received for structure *struc-name* connection name *conn-name*

Explanation

The queue manager has received XES event *event-type* for CF structure *struc-name*.

System action

Processing continues.

System programmer response

Examine the event code to determine what event was issued. The event codes are described in the *z/OS MVS Programming: Sysplex Services Reference* manual.

CSQE008I
Recovery event from *qmgr-name* received for structure *struc-name*

Explanation

The queue manager issued a peer level recovery event for CF structure *struc-name*.

System action

Processing continues. The queue manager will begin peer level recovery processing.

CSQE011I
Recovery phase 1 started for structure *struc-name* connection name *conn-name*

Explanation

Peer level recovery has started phase one of its processing, following the failure of another queue manager in the queue-sharing group.

System action

Processing continues.

System programmer response

Determine why a queue manager within the queue-sharing group failed.

CSQE012I
Recovery phase 2 started for structure *struc-name* connection name *conn-name*

Explanation

Peer level recovery has started phase two of its processing.

System action

Processing continues.

CSQE013I
Recovery phase 1 completed for structure *struc-name* connection name *conn-name*

Explanation

Peer level recovery has completed phase one of its processing.

System action

Processing continues.

CSQE014I
Recovery phase 2 completed for structure *struc-name* connection name *conn-name*

Explanation

Peer level recovery has completed phase two of its processing.

System action

Processing continues.

CSQE015I
Recovery phase 2 not attempted for structure *struc-name* connection name *conn-name*

Explanation

Phase two of peer level recovery processing was not attempted because of a previous error in phase one on one of the participating queue managers.

System action

Processing continues. The connection will be recovered by the failed queue manager when it restarts.

System programmer response

Investigate the cause of the error, as reported in the preceding messages.

CSQE016E
Structure *struc-name* connection name *conn-name* disconnected, RC=*return-code* reason=reason

Explanation

The queue manager has disconnected from CF structure *struc-name*.

System action

Processing continues.

System programmer response

Examine the return and reason codes to determine why the CF structure was disconnected. The codes are described in the *z/OS MVS Programming: Sysplex Services Reference* manual.

CSQE018I
Admin structure data building started

Explanation

The queue manager is building its own data for the administration structure.

System action

Processing continues.

CSQE019I
Admin structure data building completed

Explanation

The queue manager has built its own data for the administration structure.

System action

Processing continues.

CSQE020E
Structure *struc-name* connection as *conn-name* failed, RC=*return-code* reason= *reason* codes=*s1 s2 s3*

Explanation

The queue manager failed to connect to CF structure *struc-name*.

System action

This depends on the component that caused the connection request (queue manager or channel initiator) and the reason for connecting to the CF structure. The component might terminate, or might continue processing but with functions that require the structure inhibited.

System programmer response

Examine the return and reason codes to determine why the connect failed. Codes *s1 s2 s3* are the XES IXLCONN diagnosis codes, which are described in the *z/OS MVS Programming: Sysplex Services Reference* manual.

CSQE021I
Structure *struc-name* connection as *conn-name* warning, RC=*return-code* reason=*reason* codes=*s1 s2 s3*

Explanation

The queue manager has successfully connected to CF structure *struc-name*, but the XES IXLCONN call returned with a warning.

System action

Processing continues.

System programmer response

Examine the return and reason codes to determine why the connect warning message was issued. Codes *s1 s2 s3* are the XES IXLCONN diagnosis codes, which are described in the *z/OS MVS Programming: Sysplex Services Reference* manual.

CSQE022E
Structure *struc-name* unusable, size is too small

Explanation

The queue manager cannot use the named (coupling facility) (CF) structure because its size is less than the minimum that IBM MQ requires.

System action

The queue manager disconnects from the coupling facility (CF) structure, which becomes unusable. If it is an application structure, the queues that use the structure are not usable. If it is the administration structure, the queue manager terminates with completion code X'6C6' and reason code X'00C53000'.

System programmer response

Increase the size of the CF structure to at least the minimum size required. See Planning your coupling facility and offload storage environment for guidance on required structure sizes.

If the structure is allocated and the coupling facility Resource Manager policy allows the size of it to be increased, use the z/OS command SETXCF START,ALTER,STRNAME=*ext-struc-name*,SIZE=*newsize*. If the policy does not so allow, or there is insufficient space in the coupling facility that hosts the structure, the policy must be altered; then the structure can be rebuilt using the z/OS command SETXCF START,REBUILD,STRNAME=*ext-struc-name*. (In these commands, *ext-struc-name* is formed by prefixing *struc-name* with the queue-sharing group name.)

If the structure is not allocated, alter the policy to specify a larger INITSIZE for the structure.

CSQE024E
Incorrect coupling facility level *level1*, required *level2*

Explanation

The queue manager cannot join the queue-sharing group because the version of z/OS being used supports only CF level *level1*, but IBM MQ requires at least level *level2*.

System action

CF support is not active.

System programmer response

Upgrade z/OS and the coupling facility as necessary.

CSQE025E
Invalid UOW for *qmgr-name* in list *list-id* cannot be recovered, key=*uow-key*

Explanation

A unit-of-work descriptor was read during recovery processing that contained unexpected data. The descriptor was for the indicated queue manager; it was in the coupling facility list *list-id* and had key *uow-key* (shown in hexadecimal).

System action

The unit-of-work in error cannot be processed and the descriptor is marked as being in error. Processing continues.

System programmer response

Take a memory dump of the indicated list in your coupling facility administration structure for queue manager *qmgr-name* and contact your IBM support center.

CSQE026E
Structure *struc-name* unusable, incorrect coupling facility level *level1*, required *level2*

Explanation

The queue manager cannot use the named CF structure because it has been allocated in a CF which supports level *level1*, but MQ requires at least level *level2*.

System action

The queues that use the CF structure are not usable.

System programmer response

Either upgrade the coupling facility, or use a CF structure which is in a CF running level *level2* or above.

CSQE027E
Structure *struc-name* unusable, vector size *n1* incorrect, required *n2*

Explanation

The queue manager cannot use the named CF structure because it has been allocated a list notification vector of size *n1*, but IBM requires at least size *n2*. This is probably because there is not enough available hardware storage area (HSA) for the vector.

System action

The queues that use the CF structure are not usable.

System programmer response

You cannot adjust the amount of HSA defined for your processor. Instead, retry the application (or other process) which was attempting to open the shared queue. If the problem persists, contact your IBM support center for assistance.

CSQE028I
Structure *struc-name* reset, all messages discarded

Explanation

When it tried to connect to the named CF structure, the queue manager detected that the structure had been deleted, so a new empty structure has been created.

System action

All the messages on the queues that use the CF structure are deleted.

CSQE029E
Structure *struc-name* unusable, version *v1* differs from group version *v2*

Explanation

The queue manager cannot use the named CF structure because the version number of the structure differs from that of the queue-sharing group.

System action

The queue manager disconnects from the CF structure, which becomes unusable. If it is an application structure, the queues that use the structure are not usable. If it is the administration structure, the queue manager terminates with completion code X'6C6' and reason code X'00C51057'.

System programmer response

Check that the configuration of your queue manager, queue-sharing group, and data-sharing group is correct. If so, deallocate the CF structure using the z/OS commands **SETXCF FORCE,CON** and **SETXCF FORCE,STRUCTURE**. When you use these commands, the structure name is formed by prefixing *struc-name* with the queue-sharing group name.

You might need to stop and restart the queue manager(s).

Note:

You can also use the **D XCF** command, for example **D XCF,STR,STRNAME=MQ7@CSQ_ADMIN** to show information about the structure and any connections.

CSQE030I
Serialized application cannot start, admin structure data incomplete

Explanation

A serialized application attempted to start, but it could not do so because one or more queue managers in the queue-sharing group has not completed building its data for the administration structure. Messages CSQE031I and CSQE032I precede this message to identify such queue managers.

System action

The application is not started. The MQCONNX call that it issued to connect to the queue manager fails with a completion code of MQCC_FAILED and a reason code of MQRC_CONN_TAG_NOT_USABLE. (See API completion and reason codes for more information about these codes.)

System programmer response

The administration structure is automatically rebuilt. The rebuild can occur on any member of the QSG. Restart the application after the administration structure is successfully rebuilt, which is shown by message CSQE037I on the system performing the rebuild.

CSQE031I
Admin structure data from *qmgr-name* incomplete

Explanation

Some functions are not yet available because the indicated queue manager has not completed building its data for the administration structure.

System action

Processing continues. The functions will be available when all the queue managers identified by messages CSQE031I and CSQE032I have issued message CSQE019I.

CSQE032I
Admin structure data from *qmgr-name* unavailable

Explanation

Some functions are not yet available because the indicated queue manager is not active and therefore its data for the administration structure is not available.

System action

Processing continues.

System programmer response

The rebuild of the administration structure can occur on any member of the QSG. The functions will be available after the administration structures have been successfully rebuilt. Check the log for the messages CSQE036I and CSQE037I, which will indicate the start and completion of the administration structure rebuild.

CSQE033E
Recovery phase 1 failed for structure *struc-name* connection name *conn-name*, RC=*return-code* reason=*reason*

Explanation

An error occurred during phase one of peer level recovery processing. The recovery attempt is terminated. *return-code* and *reason* are the diagnosis codes (in hexadecimal) from an XES IXL call.

System action

Processing continues. The connection will be recovered by the failed queue manager when it restarts.

System programmer response

See the *z/OS MVS Programming: Sysplex Services Reference* manual for information about the XES IXL diagnosis codes. Restart the queue manager that failed; if it is unable to recover, contact your IBM support center.

CSQE034E
Recovery phase 2 failed for structure *struc-name* connection name *conn-name*, RC=*return-code* reason=*reason*

Explanation

An error occurred during phase two of peer level recovery processing. The recovery attempt is terminated. *return-code* and *reason* are the diagnosis codes (in hexadecimal) from an XES IXL call.

System action

Processing continues. The connection will be recovered by the failed queue manager when it restarts.

System programmer response

See the *z/OS MVS Programming: Sysplex Services Reference* manual for information about the XES IXL diagnosis codes. Restart the queue manager that failed; if it is unable to recover, contact your IBM support center.

CSQE035E
csect-name Structure *struc-name* in failed state, recovery needed

Explanation

The queue manager attempted to use CF structure *struc-name*, but it is in a failed state. The failure occurred previously; it was not caused by the current use of the structure.

System action

Processing continues, but queues that use this CF structure will not be accessible.

System programmer response

Check the console for messages from XES relating to the earlier failure, and investigate the cause. See the *z/OS MVS Programming: Sysplex Services Reference* manual for information about diagnosing problems in XES.

When the problem is resolved, issue a RECOVER CFSTRUCT command specifying TYPE(NORMAL) for this and any other failed CF structure.

CSQE036I
Admin structure data building started for *qmgr-name*

Explanation

The queue manager is building the indicated queue manager's data for the administration structure.

System action

Processing continues.

CSQE037I
Admin structure data building completed for *qmgr-name*

Explanation

The queue manager has built the indicated queue manager's data for the administration structure.

System action

Processing continues.

CSQE038E
Admin structure is full

Explanation

The queue manager cannot write to the administration structure in the coupling facility (CF) because it is full.

System action

The queue manager periodically retries the write attempt. If after a number of retries the structure is still full, this message is reissued and the queue manager terminates with a completion code X'5C6' and a reason code 00C53002.

System programmer response

Increase the size of the CF structure to at least the minimum size required. See the Defining coupling facility resources for guidance on required structure sizes.

If the structure is allocated and the coupling facility Resource Manager policy allows the size of it to be increased, use the z/OS command SETXCF START,ALTER,STRNAME=*ext-struc-name*,SIZE=*newsize*. If the policy does not allow this change, or there is insufficient space in the coupling facility that hosts the structure, the policy must be altered, then the structure can be rebuilt using the z/OS command SETXCF START,REBUILD,STRNAME=*ext-struc-name*. (In these commands, *ext-struc-name* is formed by prefixing CSQ_ADMIN with the queue-sharing group name.)

If the structure is not allocated, alter the policy to specify a larger INITSIZE for the structure.

CSQE040I
Structure *struc-name* should be backed up

Explanation

The latest backup for the named CF structure is more than two hours old. Unless backups are taken frequently, the time to recover persistent messages on shared queues may become excessive.

The message is issued at checkpoint time if the queue manager was the one that took the last backup, or if it has used the structure since the last backup was taken.

System action

Processing continues.

System programmer response

Use the BACKUP CFSTRUCT command (on any queue manager in the queue-sharing group) to make a new CF structure backup. You are recommended to set up a procedure to take frequent backups automatically.

CSQE041E
Structure *struc-name* backup is more than a day old

Explanation

The latest backup for the named CF structure is more than one day old. Unless backups are taken frequently, the time to recover persistent messages on shared queues might become excessive.

The message is issued at checkpoint time if the queue manager was the one that took the last backup, or if it has used the structure since the last backup was taken.

System action

Processing continues.

System programmer response

Use the BACKUP CFSTRUCT command (on any queue manager in the queue-sharing group) to make a new CF structure backup. It is suggested you set up a procedure to take frequent backups automatically.

CSQE042E

csect-name Structure *struc-name* unusable, no EMC storage available

Explanation

The queue manager cannot use the named CF structure because its size is less than the minimum that IBM MQ requires. Specifically, the coupling facility allocation algorithms were unable to make any event monitor control (EMC) storage available during the allocation.

System action

The queue manager disconnects from the CF structure, and the CF structure becomes unusable. If it is an application structure, the queues that use the structure are not usable. If it is the administration structure, the queue manager terminates with completion code X'6C6' and reason code X'00C53003'.

System programmer response

Disconnect all connectors from the structure, and then issue

```
SETXCF FORCE,STR,STRNAMEname
```

to get the structure deallocated from the CF before you resize the structure.

Increase the size of the CF structure to at least the minimum size required. See Planning your coupling facility and offload storage environment for further information.

If the structure is allocated and the Coupling Facility Resource Manager policy allows the size of it to be increased, use the z/OS system command:

```
SETXCF START,ALTER,STRNAME=ext-struc-name,SIZE=newsize
```

If the CFRM policy does not allow an increase in size, or there is insufficient space in the coupling facility that hosts the structure, the policy must be altered. The structure can then be rebuilt using the z/OS system command:

```
SETXCF START,REBUILD,STRNAME=ext-struc-name
```

In these commands, *ext-struc-name* is formed by prefixing *struc-name* with the queue-sharing group name.

If the structure is not allocated, alter the CFRM policy to specify a larger INITSIZE for the structure.

CSQE101I

csect-name Unable to back up or recover structure *struc-name*, structure in use

Explanation

A BACKUP or RECOVER CFSTRUCT command was issued, or automatic recovery started, for a CF structure that is in use by another process. The most likely cause is that another BACKUP or RECOVER CFSTRUCT command, or automatic recovery, is already in progress on one of the active queue managers in the queue-sharing group.

This message can also be issued when new connections to the CF structure are being prevented by the system.

System action

Processing of the command, or automatic recovery for the identified structure, is terminated.

System programmer response

Check that the correct CF structure name was entered on the command. If so, wait until the current process ends before reissuing the command if required.

If there is no other BACKUP or RECOVER CFSTRUCT already in progress, check for previous messages that indicate why connections to the CF structure are being prevented.

CSQE102E
csect-name Unable to recover structure *struc-name*, not in failed state

Explanation

A RECOVER CFSTRUCT command was issued for a CF structure that is not in a failed state. Only a CF structure that has previously failed can be recovered.

System action

Processing of the command is terminated.

System programmer response

Check that the correct CF structure name was entered on the command.

CSQE103E
csect-name Unable to recover structures, admin structure data incomplete

Explanation

A RECOVER CFSTRUCT command was issued, but recovery could not be performed because one or more queue managers in the queue-sharing group has not completed building its data for the administration structure.

System action

Messages CSQE031I and CSQE032I are sent to the z/OS console to identify such queue managers. Processing of the command is terminated.

System programmer response

The administration structure is automatically rebuilt. The rebuild can occur on any member of the QSG. Reissue the command after the administration structure is successfully rebuilt, which is shown by message CSQE037I on the system performing the rebuild.

CSQE104I
csect-name RECOVER task initiated for structure *struc-name*

Explanation

The queue manager has successfully started a task to process the RECOVER CFSTRUCT command for the named CF structure.

System action

Processing continues.

CSQE105I
csect-name BACKUP task initiated for structure *struc-name*

Explanation

The queue manager has successfully started a task to process the BACKUP CFSTRUCT command for the named CF structure.

System action

Processing continues.

CSQE106E
csect-name Unable to back up structure *struc-name*, reason=*reason*

Explanation

A BACKUP CFSTRUCT command was issued for a CF structure, but the backup could not be performed.

System action

Processing of the command is terminated.

System programmer response

Examine the reason code to determine why the CF structure could not be backed-up. The codes are described in "IBM MQ for z/OS codes" on page 5019 and the *z/OS MVS Programming: Sysplex Services Reference* manual.

CSQE107E
csect-name Unable to back up or recover structure *struc-name*, structure has never been used

Explanation

A BACKUP or RECOVER CFSTRUCT command was issued, or automatic recovery started, for a CF structure that has never been used, and so does not contain any messages or data.

System action

Processing of the command, or automatic recovery for the identified structure, is terminated.

System programmer response

Check that the correct CF structure name was entered on the command.

CSQE108E
csect-name Unable to back up or recover structure *struc-name*, structure does not support recovery

Explanation

A BACKUP or RECOVER CFSTRUCT command was issued, or automatic recovery started, for a CF structure with a functional capability that is incompatible with this process; for example, the CF structure level is not high enough to support recovery, or the RECOVER attribute is set to NO.

System action

Processing of the command, or automatic recovery for the identified structure, is terminated.

System programmer response

Ensure that the CF structure is at a level of functional capability that allows the use of the BACKUP or RECOVER CFSTRUCT command and that its MQ RECOVER attribute is set to YES. You can check the values using the DIS CFSTRUCT(*) ALL command. Check that the correct CF structure name was entered on the command.

CSQE109E
csect-name Unable to recover structure *struc-name*, no backup information available

Explanation

A RECOVER CFSTRUCT command was issued or automatic recovery started for a CF structure, but no backup information could be found.

System action

Processing of the command, or automatic recovery for the identified structure, is terminated.

System programmer response

Check that the correct CF structure name was entered on the command. If so, issue a BACKUP CFSTRUCT command to ensure that backup information is available.

CSQE110E
csect-name PURGE not allowed for structure *struc-name*

Explanation

A RECOVER CFSTRUCT command was issued for CF structure *struc-name* using TYPE(PURGE). This CF structure is a system application structure. To prevent loss of messages on system queues TYPE(PURGE) is not allowed for system application structures.

System action

Processing of the command is terminated.

System programmer response

Reissue the command without the TYPE(PURGE) option.

If structure recovery fails contact your IBM support center.

CSQE111I

csect-name Structure *struct-name* will be set to failed state to allow recovery of failed SMDS data sets

Severity
0

Explanation

The **RECOVER CFSTRUCT** command was issued for a structure which is not in the failed state, but at least one of the related SMDS data sets is currently marked as failed, requiring recovery. The structure will be put into the failed state to make it unavailable for normal use so recovery can proceed.

System action

The structure is marked as failed and recovery processing continues.

CSQE112E
csect-name Unable to recover structure *struct-name*, failed to read required logs.

Explanation

A RECOVER CFSTRUCT command or automatic structure recovery was unable to read the logs required to recover a structure.

System action

Processing of the command is terminated.

Automatic recovery of the structure will not be attempted.

System programmer response

Check that the logs containing the RBA range indicated in message CSQE130I are available, and reissue the command.

Check for any prior errors or abends reporting problems using the logs.

Issue RECOVER CFSTRUCT(*struct-name*) to retry structure recovery.

CSQE120I
Backup of structure *struc-name* started at RBA=*rba*

Explanation

The named CF structure is being backed-up in response to a BACKUP CFSTRUCT command. The backup begins at the indicated RBA.

System action

Processing continues.

CSQE121I

csect-name Backup of structure *struc-name* completed at RBA=*rba*, size *n* MB

Explanation

The named CF structure has been backed-up successfully. The backup ends at the indicated RBA, and *n* is its approximate size in megabytes.

System action

Processing continues.

CSQE130I

Recovery of structure *struc-name* started, using *qmgr-name* log range from RBA=*from-rba* to RBA=*to-rba*

Explanation

CF structure recovery is starting in response to a RECOVER CFSTRUCT command. It must read the log range shown to determine how to perform recovery. The logs are read backwards, from the latest failure time of the structures to be recovered to the earliest last successful backup time of those structures.

System action

Processing continues.

CSQE131I

csect-name Recovery of structure *struc-name* completed

Explanation

The named CF structure has been recovered successfully. The structure is available for use again.

CF structure recovery was started in response to a RECOVER CFSTRUCT command. The log range determined how to perform recovery. The logs are read backwards, from the latest failure time of the structures to be recovered to the earliest last successful backup time of those structures.

System action

Processing continues.

CSQE132I

Structure recovery started, using log range from LRSN=*from-lrsn* to LRSN=*to-lrsn*

Explanation

CF structure recovery is starting in response to a RECOVER CFSTRUCT command. It must read the log range shown to determine how to perform recovery. The logs are read backwards, from the latest failure time of the structures to be recovered to the earliest last successful backup time of those structures.

See Recovering a CF structure for more information.

System action

Processing continues.

CSQE133I

Structure recovery reading log backwards, LRSN= *lrsn*

Explanation

This is issued periodically during log reading by CF structure recovery to show progress. The log range that needs to be read is shown in the preceding CSQE132I message.

CF structure recovery is starting in response to a RECOVER CFSTRUCT command. It must read the log range shown to determine how to perform recovery. The logs are read backwards, from the latest failure time of the structures to be recovered to the earliest last successful backup time of those structures.

System action

Processing continues.

System programmer response

If this message is issued repeatedly with the same LRSN value, investigate the cause; for example, IBM MQ might be waiting for a tape with an archive log data set to be mounted.

CSQE134I

Structure recovery reading log completed

Explanation

CF structure recovery is started in response to a RECOVER CFSTRUCT command. It must read the log range shown to determine how to perform recovery. The logs are read backwards, from the latest failure time of the structures to be recovered, to the earliest last successful backup time of those structures.

CF structure recovery has completed reading the logs. The individual structures can now be recovered.

System action

Each CF structure is recovered independently, as shown by messages CSQE130I and CSQE131I.

CSQE135I

Recovery of structure *struc-name* reading log, RBA=*rba*

Explanation

This is issued periodically during log reading for recovering the named CF structure to show progress. The log range that needs to be read is shown in the preceding CSQE130I message.

System action

Processing continues.

System programmer response

If this message is issued repeatedly with the same RBA value, investigate the cause; for example, MQ might be waiting for a tape with an archive log data set to be mounted.

CSQE136I

Error returned by Db2 when clearing queue *queue-name*, list header number=*list header number*, structure number=*strucnum*

Severity

4

Explanation

Shared queue messages greater than 63 KB in size have their message data held as one or more binary large objects (BLOBs) in a Db2 table. An error was returned by Db2 when clearing these messages from the table.

Note that the list header number, and structure number, are output in hexadecimal format.

System action

Processing continues.

System programmer response

The messages have been deleted from the coupling facility but message data might remain in Db2 as orphaned BLOBs. This message is normally preceded by message CSQ5023E. Examine the Db2 job log to determine why the error occurred. The orphaned messages can be deleted by issuing the '**DISPLAY GROUP OBSMSGS(YES)**' command after 24 hours.

CSQE137E

/cpf *csect-name* DB2 and CF structure out of sync for queue *queue-name*, list header number=*list header number*, structure number=*strucnum*

Severity

4

Explanation

The queue manager has identified a discrepancy between the information stored about a queue in the coupling facility and the corresponding information in Db2.

Note that the list header number, and structure number, are output in hexadecimal format.

System action

Processing continues, but applications are unable to open the affected queue until the discrepancy is resolved by the System Programmer.

System programmer response

If the queue manager has recently been recovered from a backup then the recovery process should be reviewed to ensure that everything was correctly restored, including any Db2 tables associated with the queue manager.

If the cause of the problem cannot be determined then contact your IBM support center for assistance.

CSQE138I

csect-name Structure struc-name is already in the failed state

Explanation

A **RESET CFSTRUCT ACTION(FAIL)** command was issued for a CF structure that is already in the failed state.

System action

Processing of the command is terminated.

CSQE139I

csect-name Unable to fail structure struc-name, structure in use

Explanation

A **RESET CFSTRUCT ACTION(FAIL)** command was issued for a CF structure that is in use by another process

System action

Processing of the command is terminated.

System programmer response

Check that the correct CF structure name was entered on the command. If so, wait until the process ends before reissuing the command if required.

CSQE140I

csect-name Started listening for ENF 35 events for structure *structure-name*

Severity

0

Explanation

The queue manager has registered to receive ENF 35 events and will attempt to reconnect to the identified structure if it is notified that a coupling facility resource has become available.

System action

Processing continues.

CSQE141I

csect-name Stopped listening for ENF 35 events for structure *structure-name*

Explanation

The queue manager has de-registered from receiving ENF 35 events for the identified structure, and will not attempt to reconnect to it if notified that a coupling facility resource has become available.

System action

Processing continues.

CSQE142I

csect-name Total loss of connectivity reported for structure *structure-name*

Explanation

The queue manager has been notified that no systems in the sysplex have connectivity to the coupling facility in which the identified structure is allocated.

System action

If automatic recovery has been enabled for the identified structure one of the queue managers in the queue-sharing group will attempt to recover the structure in an alternative coupling facility, if one is available.

System programmer response

Investigate and resolve the loss of connectivity to the coupling facility on which the structure is allocated.

CSQE143I

csect-name Partial loss of connectivity reported for structure *structure-name*

Explanation

The queue manager has lost connectivity to the coupling facility in which the identified structure is allocated, and has been notified that the coupling facility is still available on other systems in the sysplex.

System action

A system-managed rebuild will be scheduled to rebuild the structure in an alternative coupling facility, if one is available.

System programmer response

Investigate and resolve the loss of connectivity to the coupling facility on which the structure is allocated.

CSQE144I

csect-name System-managed rebuild initiated for structure *structure-name*

Explanation

The queue manager has initiated a system-managed rebuild for the identified structure on an alternative coupling facility.

System action

Processing continues and when the process has completed, you receive message CSQE005I.

CSQE145E

csect-name Auto recovery for structure *structure-name* is not possible, no alternative CF defined in CFRM policy

Severity

8

Explanation

The queue manager has lost connectivity to the coupling facility in which the identified structure is allocated, but cannot automatically recover the structure because there is no alternative coupling facility in the CFRM preference list.

System action

Processing continues without connectivity to the structure. Any queues that reside on the application structure remain unavailable.

System programmer response

Investigate and resolve the loss of connectivity to the Coupling Facility on which the structure is allocated.

CSQE146E

csect-name System-managed rebuild for structure *structure-name* failed, reason=*reason*

Severity

8

Explanation

The queue manager attempted to initiate a system-managed rebuild for the identified structure but the rebuild could not be performed.

System action

Processing continues without connectivity to the structure. Any queues that reside on the application structure remain unavailable.

System programmer response

Examine the reason code to determine why the system-managed rebuild could not be completed. The codes are described in the *z/OS MVS Programming: Sysplex Services Reference* manual.

CSQE147I

csect-name System-managed rebuild for structure *structure-name* is already in progress

Explanation

The queue manager attempted to initiate a system-managed rebuild for the identified structure but determined that another queue manager in the queue-sharing group has initiated it already.

System action

Processing continues.

CSQE148I

csect-name Loss of connectivity processing for structure *structure-name* deferred

Explanation

The queue manager has lost connectivity to the coupling facility in which the identified structure is allocated, but MVS has requested that the queue manager should not take action until a subsequent notification is received.

System action

Processing continues without connectivity to the structure. Any queues that reside on the application structure remain unavailable.

CSQE149I

csect-name Waiting for other queue managers to disconnect from structure *structure-name*

Explanation

The queue manager has lost connectivity to the coupling facility, in which the identified structure is allocated, but cannot delete the structure or initiate a system-managed rebuild because one or more queue managers that also lost connectivity remain connected to it.

System action

The queue manager will periodically retry the attempted operation until all of the queue managers have disconnected.

CSQE150I

csect-name System-managed rebuild already completed for structure *structure-name*

Explanation

A system-managed rebuild for the identified structure is unnecessary as another request to rebuild the structure has been completed.

System action

Processing continues.

CSQE151I

csect-name Loss of admin structure connectivity toleration enabled

Explanation

If any queue manager in the queue-sharing group loses connectivity to the administration structure the structure will be rebuilt in an alternative CF, if one is available.

If the structure cannot be rebuilt, some shared queue functions on queue managers that have lost connectivity will be unavailable until connectivity to the structure has been restored. Access to private queues will not be affected.

System action

Processing continues.

CSQE152I

csect-name Loss of admin structure connectivity toleration disabled

Explanation

If the queue manager loses connectivity to the administration structure no attempt to rebuild it is made. The queue manager terminates with abend code 5C6-00C510AB.

This can occur if the CFCONLOS queue manager attribute is set to TERMINATE.

System action

Processing continues.

CSQE153I

csect-name Auto recovery for structure *struct-name* has been scheduled

Explanation

The queue manager has detected that the identified structure which has automatic recovery enabled, has failed, or connectivity to it has been lost on all systems in the sysplex.

The queue manager has scheduled an attempt to recover the structure.

System action

One of the active queue managers in the queue-sharing group will attempt to recover the identified structure.

CSQE154I

csect-name Structure *struct-name* has been deleted

Explanation

The queue manager has successfully deleted the identified structure from the coupling facility.

System action

Processing continues.

CSQE155I

csect-name Structure *struct-name* has already been deleted

Explanation

The queue manager attempted to delete the identified structure from the coupling facility. It could not be deleted because it was not allocated.

System action

Processing continues.

CSQE156I

csect-name Structure *struct-name* has already been reallocated

Explanation

The queue manager lost connectivity to the identified structure. When attempting to delete the structure the queue manager found that the structure had been reallocated since connectivity was lost.

System action

Processing continues.

CSQE157E

csect-name Unable to recover structure *struc-name*, no suitable CF available

Severity

8

Explanation

A RECOVER CFSTRUCT command was issued or automatic recovery started for the identified structure, but there was no suitable Coupling Facility available in which to allocate it.

System action

Processing of the command, or automatic recovery for the identified structure, is terminated.

System programmer response

Ensure that a suitable Coupling Facility in the CFRM preference list for the identified structure is available, then reissue the command.

CSQE158E

csect-name Recovery of structure *struc-name* failed, reason=*reason*

Severity

8

Explanation

Recovery of the identified (coupling facility) CF structure has failed.

System action

Processing continues, but queues that use the identified (coupling facility) CF structure will not be accessible.

System programmer response

Refer to coupling facility codes (X'C5') for information about the reason code. Use this information to solve the problem, then reissue the RECOVER CFSTRUCT command for structures that do not have automatic recovery enabled.

CSQE159I

csect-name Waiting for structure rebuild to complete for structure *structure-name*

Explanation

The queue manager has lost connectivity to the coupling facility, in which the identified structure is allocated, but cannot delete the structure or initiate a system-managed rebuild, because a structure rebuild is currently in progress.

System action

The queue manager will periodically retry the attempted operation, until the structure rebuild is finished.

CSQE160I

csect-name Auto recovery for structure *struc-name* is suspended

Explanation

The queue manager detected that recovery for structure *struc-name* is not possible. Automatic recovery of the structure is suspended.

System action

Automatic recovery for structure *struc-name* is suspended. Automatic recovery is resumed when a successful connection to the structure is established.

System programmer response

Check for any previous errors or abends reporting problems recovering the structure.

Issue RECOVER CFSTRUCT(*struct-name*) to retry structure recovery.

CSQE161E

csect-name queue-sharing group state is inconsistent, no XCF data for queue manager *qmgr-number*

Explanation

A RECOVER CFSTRUCT command or automatic structure recovery could not read all the log data required for recovery, because there was no XCF data for one of the queue managers in the QSG. *qmgr-number* is the number of the affected queue manager in the MQ Db2 tables.

System action

Processing of the command is terminated. Automatic recovery of the structure will not be attempted.

System programmer response

If the queue manager with number *qmgr-number* in the MQ Db2 tables has been force removed from the queue-sharing group then added back into the QSG, start the queue manager and issue the RECOVER CFSTRUCT command again. Otherwise, reset the structure to an empty state by issuing the RECOVER CFSTRUCT TYPE(PURGE) command.

CSQE162E

csect-name Structure *struc-name* could not be deleted, RC=*return-code* reason=*reason*

Severity

8

Explanation

The queue manager failed to delete structure *struc-name* from the Coupling Facility when processing a DELETE CFSTRUCT command.

System action

Processing continues.

System programmer response

Examine the return and reason codes to determine why the Coupling Facility structure could not be deleted by the IXLFORCE macro. The codes are described in the *z/OS MVS Programming: Sysplex Services Reference* manual.

Correct the problem that caused the failure, then delete the structure by issuing the SETXCF FORCE,STRUCTURE z/OS command.

CSQE201E

Media manager request failed with return code *ccccffss* processing *req* request for control interval *rci* in SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname*

Severity

8

Explanation

An error occurred when attempting the indicated media manager request (READ, UPDATE or FORMAT) for the data set.

ccccffss
is the media manager return code in hexadecimal. The last byte *ss* indicates the overall type of error:
08 Extent error
0C Logic error
10 Permanent I/O error
14 Undetermined error

The *cccc* field identifies the specific error and the *ff* field identifies the function which returned the error. See the *z/OS DFSMSdfp Diagnosis* manual for further details of media manager return codes.

req
specifies the type of request:
READ Read one or more control intervals.
UPDATE
Rewrite one or more control intervals.
FORMAT
Format one or more control intervals.

rci
identifies the relative control interval (RCI) number of the control interval being accessed, in hexadecimal.

qmgr-name
identifies the queue manager which owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

System action

This typically results in the **SMDS** status being set to **FAILED** (if it is the data set owned by the current queue manager) or the **SMDSCONN** status being set to **ERROR** (if it is a data set owned by a different queue manager).

System programmer response

If the problem is a permanent I/O error caused by damage to the data set and recovery logging was enabled, the data set can be recovered by the recreating it from a backup and reapplying the logged changes using the **RECOVER CFSTRUCT** command.

If the data set is temporarily unavailable (for example because of a device connectivity problem) but is not damaged, then when the data set is available again, it can be put back into normal use by using the **RESET SMDS** command to set the status to **RECOVERED**.

CSQE202E

Media manager service failed with return code *ret-code*, feedback code *feedback-code*, processing *function* for SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname*

Severity
8

Explanation

A media manager support services (MMGRSRV) function gave an unexpected error.

ret-code
indicates the MMGRSRV return code, in hexadecimal.
08 Media Manager Services error.
14 Indeterminate error

feedback-code
indicates the 8-byte MMGRSRV internal feedback code, in hexadecimal.

For CONNECT processing, the first byte of this feedback code is the same as the VSAM OPEN error information returned in ACBERFLG.

function
indicates the type of function requested, which can be any of the following:

CONNECT
Open the data set.

DISCONNECT
Close the data set.

EXTEND
Extend the data set being written by the current queue manager, or obtain access to recently added extents for a data set which has been extended by another queue manager.

CATREAD
Obtain the highest allocated and highest used control interval numbers from the catalog entry for the current data set.

CATUPDT
Update the highest used control interval in the catalog entry for the current data set, after formatting new extents.

qmgr-name
identifies the queue manager which owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

System action

This typically results in the **SMDS** status being set to **FAILED** (if it is the data set owned by the current queue manager) or the **SMDSCONN** status being set to **ERROR** (if it is a data set owned by a different queue manager).

System programmer response

This message is normally preceded by a system message such as IEC161I from VSAM or DFP indicating the nature of the error.

If the problem is a permanent I/O error caused by damage to the data set and recovery logging was enabled, the data set can be recovered by the recreating it from a backup and reapplying the logged changes using the **RECOVER CFSTRUCT** command.

If the data set is temporarily unavailable (for example because of a device connectivity problem) but is not damaged, then when the data set is available again, it can be put back into normal use by using the **RESET SMDS** command to set the status to **RECOVERED**.

CSQE211I

Formatting is in progress for *count* pages in SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname*

Severity
0

Explanation

The data set is being formatted from the current highest used page to the highest allocated page. This message occurs either when a new extent has been allocated or immediately after opening an existing data set which has not been fully formatted (that is, the highest used page is less than the highest allocated page).

count
indicates the number of pages which need to be formatted (in decimal).

qmgr-name
identifies the queue manager which owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

System action

Formatting continues.

CSQE212I

Formatting is complete for SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname*

Severity

0

Explanation

Formatting of the data set has completed and the highest used page has been successfully updated in the catalog.

dsname
identifies the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

System action

The newly formatted space is made available for use.

CSQE213I

SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* is now *percentage*% full

Severity

0

Explanation

The data set is nearly full.

qmgr-name
identifies the queue manager which owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

percentage
shows the percentage of data blocks in the data set which are currently in use.

This message is issued when the data set becomes 90% full, 92% full, and so on, up to 100%. After this message has been issued for a particular percentage, it is not issued again until the usage has changed in either direction by at least 2%. If the usage then decreases to 88% or less (as a result of messages being deleted or as a result of the data set being expanded) a final message is issued to indicate the new usage percentage.

System action

If expansion is allowed, the data set is expanded. If the data set reaches 100% full, then requests to put new messages that require space in the data set are rejected with return code MQRC_STORAGE_MEDIUM_FULL.

System programmer response

You can check the usage in more detail using the **DISPLAY USAGE** command with the **SMDS** keyword.

CSQE215I

Further expansion of SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* is not possible because the maximum number of extents have been allocated

Severity
0

Explanation

The media manager interface has indicated that the data set has reached the maximum number of extents, and cannot be expanded any further.

qmgr-name
identifies the queue manager that owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

This message can be issued when the data set is opened, or following an expansion attempt, which might have been successful, as indicated by previous messages.

System action

The expansion option for the data set is changed to **DSEXPAND(NO)** to prevent further expansion attempts.

System programmer response

The only way to expand the data set further is to make it temporarily unavailable by using the **RESET SMDS** command to mark the status as **FAILED**, copy it to a new location using larger extents, then make it available again using the **RESET SMDS** command to mark the status as **RECOVERED**.

CSQE217I

Expansion of SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* was successful, *count* pages added, total pages *total*

Severity
0

Explanation

The data set was expanded, and one or more new extents have been successfully added.

qmgr-name
identifies the queue manager, which owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

count
indicates the number of new pages that have been allocated (in decimal).

total
indicates the total number of pages currently allocated (in decimal).

System action

The queue manager formats the newly allocated space.

CSQE218E

Expansion of SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* was unsuccessful

Severity
8

Explanation

An attempt was made to expand the data set, but it was unsuccessful, typically because insufficient space was available.

qmgr-name
identifies the queue manager, which owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

System action

The expansion option for the data set is changed to **DSEXPAND(NO)** to prevent further expansion attempts.

System programmer response

Check for messages from VSAM or DFP that explain why the request was unsuccessful, and do the required actions.

If space is made available later, change the expansion option back to allow expansion to be tried again.

CSQE219I

Extents refreshed for SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname*, *count* pages added, total pages *total*

Severity
0

Explanation

The data set was extended by another queue manager. The current queue manager used media manager services to update the extent information for the open data set to read message data within the new extents.

qmgr-name
identifies the queue manager that owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

count
indicates the number of new page that have been allocated (in decimal).

total
indicates the total number of pages currently allocated (in decimal).

System action

The new extents are made visible to the current queue manager.

CSQE222E

Dynamic allocation of SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* failed with return code *ret-code*, reason code *eeeeiiii*

Severity
8

Explanation

An attempt was made to allocate the data set using the data set name formed by taking the generic **DSGROUP** name and inserting the queue manager name, but the DYNALLOC macro returned an error.

qmgr-name
identifies the queue manager which owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

ret-code
shows the return code from DYNALLOC, in decimal.

eeeeiiii
shows the reason code, consisting of the error and information codes returned by DYNALLOC, in hexadecimal.

System action

This typically results in the **SMDS** status being set to **FAILED** (if it is the data set owned by the current queue manager) or the **SMDSCONN** status being set to **ERROR** (if it is a data set owned by a different queue manager).

System programmer response

Check the job log for dynamic allocation error messages giving more details about the problem.

After any changes, use the **START SMDSCONN** command to trigger a new attempt to use the data set.

When the reason code is '02540000', indicating that the allocation failed due to a required ENQ being unavailable, the queue manager will automatically retry the allocation request on subsequent attempts to access the SMDS.

CSQE223E

Dynamic deallocation of SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* failed with return code *ret-code*, reason code *eeeeiiii*

Severity
8

Explanation

An attempt was made to deallocate the data set but the DYNALLOC macro returned an error.

qmgr-name
identifies the queue manager which owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

ret-code
shows the return code from DYNALLOC, in decimal.

eeeeiiii
shows the reason code, consisting of the error and information codes returned by DYNALLOC, in hexadecimal.

System action

No further action is taken, but problems can occur if an attempt is made to use the data set, either from another job or from the same queue manager.

System programmer response

Check the job log for dynamic allocation error messages giving more details about the problem.

CSQE230E

csect-name SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* saved space map cannot be used the time stamp *time1* does not match the last CLOSE time stamp *time2* in the SMDS object

Severity

8

Explanation

The shared message data set owned by this queue manager appears to have been closed normally last time it was used, with a saved space map, but the time stamp in the data set does not match the time stamp stored in the SMDS object in Db2 the last time this queue manager closed the data set. This means that the saved space map may not be consistent with the current messages in the coupling facility, so it needs to be rebuilt.

The most probable cause for this message is that the data set has been copied or restored from a copy which was not completely up to date.

qmgr-name
identifies the queue manager that owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

time1
shows the time stamp found in the data set header.

time2
shows the time stamp found in the SMDS object in DB2.

System action

The existing saved space map is ignored and the space map is rebuilt by scanning the messages in the coupling facility structure which refer to the data set.

The rebuild scan process keeps track of the most recent message in the coupling facility that refers to the data set, and at the end of the scan it checks that the matching message data is found in the data set. If so, it is assumed that all changes up to at least that time are present in the data set, so no data has been lost, and the data set can be opened normally. Otherwise, message CSQI034E is issued and the data set is marked as failed.

CSQE231E

SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* cannot be used because it is not a VSAM linear data set with control interval size 4096 and SHAREOPTIONS(2 3)

Severity

8

Explanation

The specified data set is not a VSAM linear data set, or the control interval size is not the default value 4096, or the wrong sharing options have been specified.

qmgr-name
identifies the queue manager that owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

If the data set was initially empty, the sharing options are not checked until the data set has been initialized, closed, and reopened.

System action

The data set is closed and the **SMDS** status is set to **FAILED**.

System programmer response

Delete the incorrect data set, and create a one of the same name with the correct attributes.

After any changes, use the **START SMDSCONN** command to trigger a new attempt to use the data set.

CSQE232E

csect-name SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* cannot be used because the identification information (*field-name*) in the header record is incorrect

Severity
8

Explanation

When the data set was opened, there was existing information in the header record (so the data set was not newly formatted) but the information did not match the expected data set identification. The identification information includes a marker "CSQESMDS" for a shared message data set followed by the names of the queue-sharing group, the application structure and the queue manager which owns the shared message data set.

qmgr-name
identifies the queue manager that owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

field-name
identifies the first header identification field which did not have the expected value.

System action

The data set is closed and the connection is marked as **AVAIL(ERROR)**. If the data set status is **ACTIVE** or **RECOVERED**, indicating that it was currently in use, the status is changed to **FAILED**.

System programmer response

If the data set was already in use, this probably indicates that it has been overwritten in some way, in which case any persistent messages can be recovered using the **RECOVER CFSTRUCT** command.

If the data set was not yet in use, or was currently empty, ensure that it is either formatted or emptied before trying to use it again. After any changes, use the **START SMDSCONN** command to trigger a new attempt to use the data set.

To display the data set header record, you can use the Access Method Services **PRINT** command, for example as follows:

```
PRINT INDATASET('dsname') TOADDRESS(4095)
```

The format of the identification information within the data set header record is as follows:

Table 424. Format of identification information within the data set header record.

Offset: Dec	Offset: Hex	Type	Length	Field	Description
8	8	Character	8	MARKER	Marker 'CSQESMDS'
16	10	Character	4	Queue-sharing group	Queue-sharing group name
20	14	Character	12	CFSTRUCT	Structure name
3	20	Character	4	SMDS	Owning queue manager
36	24	Integer	4	VERSION	Header version 1

CSQE233E

SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* cannot be used because the header record indicates a newly formatted data set but it was already being used

Severity

8

Explanation

When the data set was opened, the identification information in the header record was zero, indicating a new empty data set, but the data set was already in use, so it should not now be empty.

qmgr-name
identifies the queue manager that owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

System action

The data set is closed and marked as **FAILED**.

System programmer response

Any persistent messages can be recovered using the **RECOVER CFSTRUCT** command.

CSQE234I

SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* was empty so it requires formatting

Severity

0

Explanation

When the data set was opened, it was found to be empty, with no existing data and no pre-formatted space. In this case, VSAM does not allow shared access to the data set. The queue manager needs to initialize the data set.

qmgr-name
identifies the queue manager that owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

System action

The data set is pre-formatted up to the end of the existing extents. There is a short delay before the data set is fully available.

CSQE235I

SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* was not fully formatted so it requires additional formatting

Severity
0

Explanation

This occurs if the existing data set extents have not been fully formatted when the data set is opened.

qmgr-name
identifies the queue manager that owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

System action

The data set is formatted up to the end of the existing extents. There is a short delay before the data set is fully available.

CSQE236I

SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* cannot be used because there is not enough main storage available to build the space map

Severity

8

Explanation

The queue manager needs to build a space map in main storage to manage the free space in the data set, but it was unable to obtain sufficient main storage.

qmgr-name
identifies the queue manager which owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

System action

The data set is not opened.

System programmer response

Consider increasing the queue manager's MEMLIMIT.

If necessary, use the `START SMDSCONN` command to request another attempt to open the data set.

For more details see Address space storage.

CSQE237I

SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* cannot be extended because there is not enough main storage available to build the space map

Severity

8

Explanation

The queue manager needs to build space map blocks in main storage to manage the additional space in the extended data set, but it was unable to obtain sufficient main storage.

qmgr-name
identifies the queue manager which owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

System action

The new extents of the data set are not available for use.

System programmer response

Consider increasing the queue manager's MEMLIMIT.

If necessary, use the `START SMDSCONN` command to request another attempt to open the data set.

For more details see Address space storage.

CSQE238I

SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* is too small to use because the initial space allocation is less than two logical blocks

Severity

8

Explanation

The minimum supported data set size requires at least one logical block for control information and one logical block for data, but the data set is smaller than two logical blocks.

qmgr-name
identifies the queue manager which owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

System action

The data set is not opened.

System programmer response

Delete the data set and re-create it with a larger space allocation.

After making changes, use the **START SMDSCONN** command to request another attempt to open the data set.

CSQE239I

SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* has become full so new large messages can no longer be stored in it

Severity

8

Explanation

A message written to a shared queue contains data which is large enough to require offloading to a data set, but there is insufficient space in the data set. Further requests are likely to fail until existing messages have been read and deleted from the data set.

qmgr-name
identifies the queue manager which owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

System action

Any requests encountering this problem are rejected with MQRC_STORAGE_MEDIUM_FULL. This message is not issued again until the data set has been below 90% full since the previous time it was issued.

System programmer response

This problem means that the backlog of unprocessed large shared messages exceeds the size of the data set, but the data set could not be extended in time to avoid the problem.

Ensure that applications to remove large messages from the shared queues are running. Check also for previous problems relating to extending the data set, for example if there was insufficient space on eligible volumes.

CSQE241I

SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) now has STATUS(*status*)

Severity

0

Explanation

The status of the shared message data set for the specified queue manager and application structure has been changed to the indicated value, either by automatic status management or by a **RESET SMDS** command.

qmgr-name
identifies the queue manager that owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

status
shows the new status value. For details of specific status values, see the DISPLAY CFSTATUS command with the **TYPE(SMDS)** option.

System action

All queue managers connected to the structure are notified of the status change. The queue managers take appropriate action if necessary, for example opening or closing the data set.

CSQE242I

SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) now has ACCESS(*access*)

Severity
0

Explanation

The access availability setting for the shared message data set for the specified queue manager, and application structure has been changed to the indicated value, either by automatic status management or by a **RESET SMDS** command.

qmgr-name
identifies the queue manager, which owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

access
shows the new access availability setting. For details of specific settings, see the **DISPLAY CFSTATUS** command with the **TYPE(SMDS)** option.

System action

All queue managers connected to the structure are notified of the change. The queue managers take appropriate action if necessary, for example opening or closing the data set.

CSQE243I

SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) now has DSBUFS(*value*)

Severity
0

Explanation

The number of shared message data set buffers to be used by the specified queue manager for this application structure has been changed to the indicated value. This message can either occur as a result of an **ALTER SMDS** command or when a previously specified **DSBUFS** target value cannot be achieved, in which case a warning message is issued, and the **DSBUFS** option is automatically set to the actual value achieved.

qmgr-name
identifies the queue manager, which owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

value
shows the new **DSBUFS** setting, which can either be a decimal number, giving the number of buffers to be used, or **DEFAULT**, indicating that the default **DSBUFS** value specified on the **CFSTRUCT** definition for the application structure is to be used. For more information, see the **ALTER SMDS** and **DISPLAY SMDS** commands.

System action

The queue manager identified by the **SMDS** keyword is notified, if active, and adjusts the size of its buffer pool as indicated.

CSQE244I

csect-name SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) now has DSEXPAND(*value*)

Severity
0

Explanation

The option to allow automatic expansion of a specific shared message data set has been changed as indicated. This message can occur either as a result of an **ALTER SMDS** command or when

expansion was attempted but failed, in which case the option is automatically changed to **DSEXPAND(NO)** to prevent further expansion attempts. In the latter case, when the problem has been fixed, the **ALTER SMDS** command can be used to turn automatic expansion on again.

qmgr-name
identifies the queue manager which owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

value
shows the new **DSEXPAND** setting, which is **DEFAULT**, **YES** or **NO**. For more information, see the **ALTER SMDS** and **DISPLAY SMDS** commands.

System action

The queue manager identified by the **SMDS** keyword is notified, if that queue manager is active. If the change results in expansion being enabled, and the data set is already in need of expansion, an immediate expansion is attempted.

CSQE245I

CFSTRUCT(*struc-name*) now has OFFLDUSE(*offload-usage*)

Severity
0

Explanation

The **OFFLOAD** method for an application structure was recently changed and the queue manager has now determined that there are no more messages stored using the old offload method, so there is no longer any need for the old offload method to remain active. The offload usage indicator, displayed as the **OFFLDUSE** keyword on the **DISPLAY CFSTATUS** command, has been updated to indicate that only the new offload method is now in use.

For a transition from **OFFLOAD(SMDS** to **OFFLOAD(DB2)**, this message occurs when all active data sets have been changed to the **EMPTY** state, which occurs if the data set is closed normally at a time when it does not contain any messages. In this case, the offload usage indicator is changed from **BOTH** to **DB2**, and the queue managers will no longer use the SMDS data sets, which can be deleted if no longer required.

For a transition from **OFFLOAD(DB2)** to **OFFLOAD(SMDS)**, this message occurs when the queue manager disconnects normally from the structure at a time when there are no large messages for the structure stored in Db2. In this case, the offload usage indicator is changed from **BOTH** to **SMDS**.

struc-name
identifies the application structure.

offload-usage
shows the new offload usage indicator.

System action

All queue managers connected to the structure are notified of the change. The queue managers take appropriate action if necessary, for example opening or closing data sets.

CSQE246I

csect-name SMDSCONN(*qmgr-name*) CFSTRUCT(*struc-name*) now has STATUS(*status*)

Severity
0

Explanation

The current queue manager was unable to connect to a shared message data set, usually for reasons indicated by a previous message. The error status for the data set connection has now been set to indicate the type of problem which occurred. It will be reset next time an attempt is made to open the data set.

This message is only issued for error status values, which are shown instead of normal status if the data set has been closed because of an error. No message is issued for normal status values (**CLOSED**, **OPENING**, **OPEN** or **CLOSING**).

qmgr-name
identifies the queue manager that owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

status
shows the new error status. For details of the possible status values, see the **STATUS** keyword on the **DISPLAY SMDSCONN** command.

System action

The **SMDSCONN** availability is set to **AVAIL(ERROR)** and message CSQE247I is issued.

No further attempt is made to connect to the data set until the availability value is changed back to **AVAIL(NORMAL)**. This can occur as a result of the queue manager being restarted, or data set availability changing, or in response to the **START SMDSCONN** command. If this happens while the queue manager is running, another message CSQE247I is issued showing **AVAIL(NORMAL)**.

CSQE247I

csect-name SMDSCONN(*qmgr-name*) CFSTRUCT(*struc-name*) now has AVAIL(*availability*)

Severity
0

Explanation

The availability setting for the connection between the current queue manager and a shared message data set has been changed to the indicated value. This can be changed either by automatic status management, for example if the queue manager is unable to open the data set, or by one of the commands **STOP SMDSCONN** or **START SMDSCONN**.

qmgr-name
identifies the queue manager that owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

availability
shows the new availability setting. For details of the possible values, see the **AVAIL** keyword on the **DISPLAY SMDSCONN** command.

System action

The current queue manager takes appropriate action if necessary, for example opening or closing the data set.

CSQE252I

SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* space map will be rebuilt by scanning the structure

Severity
0

Explanation

The data set space map needs to be reconstructed either following queue manager abnormal termination or data set recovery, so there will be a delay while this scan is completed.

qmgr-name
identifies the queue manager which owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

System action

The queue manager will scan the contents of the structure to determine which blocks in the data set are being referenced so that it can reconstruct the space map.

CSQE255I

SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* space map has been rebuilt, message count *msg-count*

Severity
0

Explanation

The scan to rebuild the data set space map has completed.

qmgr-name
identifies the queue manager which owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

msg-count
indicates the number of large messages currently stored in the data set.

System action

The data set is made available for use.

CSQE256E

SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* space map rebuild processing failed because a referenced message data block is beyond the end of the data set

Severity
8

Explanation

During the scan to rebuild the data set space map, a message was found in the structure which referenced a message data block with a control interval number greater than the size of the current data set. It is likely that the data set has been truncated.

qmgr-name
identifies the queue manager that owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

System action

The data set is closed and marked as **FAILED**.

System programmer response

This message indicates that the data set has been damaged, for example by copying it to a smaller data set, causing one or more message data blocks to be lost.

If the original copy is still available, the problem can be fixed without loss of data by reallocating the data set at the original size, copying in the original data, and then using the **RESET SMDS** command to mark the data set as **RECOVERED**.

Otherwise, any persistent messages can be recovered by recreating the data set at the original size and recovering the structure and the data set using the **RECOVER CFSTRUCT** command.

CSQE257E

SMDS(*qmgr-name*) CFSTRUCT(*struc-name*) data set *dsname* is smaller than the size recorded in the space map. The saved space map cannot be used

Severity

8

Explanation

The data set contained a saved space map, but the current size of the data set is smaller than the size recorded in the space map. It is likely that the data set has been truncated.

qmgr-name
identifies the queue manager that owns the shared message data set.

struc-name
identifies the application structure associated with the shared message data set.

dsname
shows the full name of the shared message data set.

System action

The saved space map is ignored and an attempt is made to rebuild the space map for the truncated data set. If all active message data is within the current extents of the data set the rebuild attempt will be successful, otherwise it will fail with message **CSQE256E**.

CSQE274E

The SMDS buffer pool for CFSTRUCT(*struc-name*) could not be created because insufficient storage was available

Severity

8

Explanation

Insufficient main storage was available to allocate the SMDS data buffer pool for the structure.

struc-name
identifies the application structure associated with the shared message data set.

System action

The data sets for this structure cannot be opened.

System programmer response

Consider increasing the queue manager's MEMLIMIT.

For more details about address space storage, see Address space storage.

CSQE275E

The SMDS buffer pool for CFSTRUCT(*struc-name*) has been created with *actual-buffers* rather than the requested *buffer-count* because insufficient storage was available

Severity

8

Explanation

Insufficient main storage was available to allocate the requested number of buffers in the SMDS data buffer pool for the structure. A smaller number of buffers were successfully allocated.

struc-name
identifies the application structure associated with the shared message data set.

actual-buffers
shows the number of buffers allocated.

buffer-count
shows the requested number of buffers.

System action

The buffer pool is created with a smaller number of buffers.

System programmer response

If the specified number of buffers is enough, change the requested value to match, to avoid similar problems in future.

Consider increasing the queue manager's MEMLIMIT.

For more details see Address space storage.

CSQE276I

The SMDS buffer pool for CFSTRUCT(*struc-name*) has been increased to *buffer-count* buffers

Severity

0

Explanation

The request to alter the **SMDS** buffer pool size has completed normally.

struc-name
identifies the application structure associated with the shared message data set.

buffer-count
shows the requested number of buffers.

System action

The additional buffers are made available for use.

CSQE277I

The SMDS buffer pool for CFSTRUCT(*struc-name*) has been increased to *actual-buffers* buffers rather than the requested *buffer-count* because insufficient storage was available

Severity

0

Explanation

The request to alter the **SMDS** buffer pool size has completed but the target number of buffers was not reached because insufficient main storage was available

struc-name
identifies the application structure associated with the shared message data set.

actual-buffers
shows the number of buffers allocated.

buffer-count
shows the requested number of buffers.

System action

The additional buffers are made available for use.

CSQE278I

The SMDS buffer pool for CFSTRUCT(*struc-name*) has been decreased to *buffer-count* buffers

Severity
0

Explanation

The request to reduce the **SMDS** buffer pool size has completed normally.

struc-name
identifies the application structure associated with the shared message data set.

buffer-count
shows the requested number of buffers.

System action

The storage for the excess buffers is released back to the system.

CSQE279I

The SMDS buffer pool for CFSTRUCT(*struc-name*) has been decreased to *actual-buffers* buffers rather than the requested *buffer-count* because the rest of the buffers are in use

Severity
0

Explanation

The request to reduce the **SMDS** buffer pool size could not reach the target number of buffers because the current number of buffers in use exceeded that number, and active buffers cannot be released.

struc-name
identifies the application structure associated with the shared message data set.

actual-buffers
shows the number of buffers allocated.

buffer-count
shows the requested number of buffers.

System action

If the number of buffers was at least partly reduced, the storage for the excess buffers is released back to the system.

CSQE280I

SMDS usage ...

Severity
0

Explanation

This message is issued in response to a **DISPLAY USAGE** command with **TYPE(SMDS)**. It shows the data set space usage information for the shared message data sets owned by the current queue manager for each application structure which is currently using SMDS support. The information is in the following format:

```
Application  Offloaded  Total  Total data  Used data  Used structure  messages  blocks  blocks  blocks  part _name
n          n          n          n          n%
:
End of SMDS report
```

The columns of information are as follows:

Application structure
This is the name of the application structure.

Offloaded messages
This shows the number of shared messages in the structure for which the message data has been stored in the data set owned by this queue manager.

Total blocks
This is the current total size of the owned data set in logical blocks, including blocks used to store the space map.

Total data blocks
This is the number of blocks in the owned data set which can be used to store data, excluding those used to store the space map.

Used data blocks
This is the number of blocks in the owned data set which are currently in use (that is, one or more pages of those blocks contain active message data).

Used part
This is the ratio of the number of used data blocks to the total data blocks, expressed as a percentage.

CSQE285I

SMDS buffer usage ...

Severity
0

Explanation

This message is issued in response to a **DISPLAY USAGE** command with **TYPE(SMDS)**. It shows the shared message data set buffer pool usage information for each application structure which is currently using SMDS support. The information is in the following format:

```
Application  Block  --------- Buffers ---------  Reads  Lowest  Wait  structure  size  Total  In use  Saved  Empty
nK         n          n          n          n         n%          n          n%
End of SMDS buffer report
```

The columns of information are as follows:

Application structure
This is the name of the application structure.

Block size
This shows the size of each buffer in Kbytes. This is equal to the logical block size of the shared message data set.

Buffers: Total
This is the actual number of buffers in the pool.

Buffers: In use
This is the number of buffers which are currently being used by requests to transfer data to or from the data set.

Buffers: Saved
This is the number of buffers which are free but currently contain saved data for recently accessed blocks.

Buffers: Empty
This is the number of buffers which are free and empty. When a new buffer is required, empty buffers are used first, but if there are no empty buffers, the least recently used saved buffer is reset to empty and used instead.

Reads saved
This is the percentage of read requests (during the current statistics interval) where the correct block was found in a saved buffer, avoiding the need to read the data from the data set.

Lowest free
This is the smallest number of free buffers during the current statistics interval, or zero if all buffers were used but no request had to wait for an empty buffer, or a negative number indicating the maximum number of requests which were waiting for a free buffer at the same time. If this value is negative, it indicates the number of additional buffers that would have been needed in order to avoid waits for a free buffer.

Wait rate
This is the fraction of requests to acquire a buffer which had to wait for a free buffer, expressed as a percentage. The numbers are reset when statistics are collected.

Security manager messages (CSQH...): z/OS

CSQH001I

Security using uppercase classes

Severity
0

Explanation

This message is issued to inform you that security is currently using the uppercase classes MQPROC, MQNLIST, MQQUEUE and MQADMIN.

CSQH002I

Security using mixed case classes

Severity
0

Explanation

This message is issued to inform you that security is currently using the mixed case classes MXPROC, MXNLIST, MXQUEUE and MXADMIN.

CSQH003I

Security refresh did not take place for class *class-name*

Severity
4

Explanation

This message follows message CSQH004I when an attempt to refresh class MQPROC, MQNLIST, or MQQUEUE was unsuccessful because of a return code from a SAF RACROUTE REQUEST=STAT call. The return code is given in message CSQH004I.

System action

The refresh does not occur.

System programmer response

Check that the class in question (*class-name*) is set up correctly. See message CSQH004I for the reason for the problem.

CSQH004I

csect-name STAT call failed for class *class-name*, SAF return code= *saf-rc*, ESM return code=*esm-rc*

Severity

8

Explanation

This message is issued as a result of a SAF RACROUTE REQUEST=STAT call to your external security manager (ESM) returning a non-zero return code at one of the following times:

- During initialization, or in response to a REFRESH SECURITY command

  If the return codes from SAF and your ESM are not zero, and are unexpected, this will cause abnormal termination with one of the following reason codes:

  - X'00C8000D'
  - X'00C80032'
  - X'00C80038'
- In response to a REFRESH SECURITY command.

  If the return codes from SAF and your ESM are not zero (for example, because a class is not active because you are not going to use it) this message is returned to the issuer of the command to advise that the STAT call failed.

Possible causes of this problem are:

- The class is not installed
- The class is not active
- The external security manager (ESM) is not active
- The RACF z/OS router table is incorrect

System programmer response

To determine if you need to take any action, see the *Security Server External Security Interface (RACROUTE) Macro Reference* for more information about the return codes.

CSQH005I

csect-name resource-type In-storage profiles successfully listed

Severity

0

Explanation

This message is issued in response to a REFRESH SECURITY command that caused the in-storage profiles to be RACLISTED (that is, rebuilt); for example, when the security switch for a resource is set on, or a refresh for a specific class is requested that requires the in-storage tables to be rebuilt.

System programmer response

This message is issued so that you can check the security configuration of your queue manager.

CSQH006I

Error returned from CSQTTIME, security timer not started

Severity

8

Explanation

An error was returned from the MQ timer component, so the security timer was not started.

System action

The queue manager terminates abnormally, with a reason code of X'00C80042'.

System programmer response

See “Security manager codes (X'C8')” on page 5035 for an explanation of the reason code.

CSQH007I

Reverify flag not set for user-id *userid*, no entry found

Severity

0

Explanation

A user identifier (*user-id*) specified in the RVERIFY SECURITY command was not valid because there was no entry found for it in the internal control table. This could be because the identifier was entered incorrectly in the command, or because it was not in the table (for example, because it had timed-out).

System action

The user identifier (*user-id*) is not flagged for reverify.

System programmer response

Check that the identifier was entered correctly.

CSQH008I

Subsystem security not active, no userids processed

Severity

0

Explanation

The RVERIFY SECURITY command was issued, but the subsystem security switch is off, so there are no internal control tables to flag for reverification.

CSQH009I

Errors occurred during security timeout processing

Severity

8

Explanation

This message is sent to the system log either:

- If an error occurs during security timeout processing (for example, a nonzero return code from the external security manager (ESM) during delete processing)
- Prior to a message CSQH010I if a nonzero return code is received from the timer (CSQTTIME) during an attempt to restart the security timer

System action

Processing continues.

System programmer response

Contact your IBM support center to report the problem.

CSQH010I

csect-name Security timeout timer not restarted

Severity
8

Explanation

This message is issued to inform you that the security timeout timer is not operational. The reason for this depends on which of the following messages precedes this one:

CSQH009I
An error occurred during timeout processing

CSQH011I
The timeout interval has been set to zero

System action

If this message follows message CSQH009I, the queue manager ends abnormally with one of the following reason codes:

csect-name
Reason code

CSQHTPOP
X'00C80040'

CSQHPATC
X'00C80041'

System programmer response

See "Security manager codes (X'C8')" on page 5035 for information about the reason code.

CSQH011I

csect-name Security interval is now set to zero

Severity
0

Explanation

The ALTER SECURITY command was entered with the INTERVAL attribute set to 0. This means that no user timeouts will occur.

System programmer response

This message is issued to warn you that no security timeouts will occur. Check that this is what was intended.

CSQH012I

Errors occurred during ALTER SECURITY timeout processing

Severity
8

Explanation

This message is issued in response to an ALTER SECURITY command if errors have been detected during timeout processing (for example, a nonzero return code from the external security manager (ESM) during timeout processing).

System action

Processing continues.

System programmer response

Contact your IBM support center to report the problem.

CSQH013E

csect-name Case conflict for class *class-name*

Severity

8

Explanation

A REFRESH SECURITY command was issued, but the case currently in use for the class *class-name* differs from the system setting and if refreshed would result in the set of classes using different case settings.

System action

The refresh does not occur.

System programmer response

Check that the class in question (*class-name*) is set up correctly and that the system setting is correct. If a change in case setting is required, issue the REFRESH SECURITY(*) command to change all classes.

CSQH015I

Security timeout = *number* minutes

Severity

0

Explanation

This message is issued in response to the DISPLAY SECURITY TIMEOUT command, or as part of the DISPLAY SECURITY ALL command.

CSQH016I

Security interval = *number* minutes

Severity

0

Explanation

This message is issued in response to the DISPLAY SECURITY INTERVAL command, or as part of the DISPLAY SECURITY ALL command.

CSQH017I

Security refresh completed with errors in signoff

Severity

8

Explanation

This message is issued when an error has been detected in refresh processing; for example, a nonzero return code from the external security manager (ESM) during signoff or delete processing.

System action

Processing continues.

System programmer response

Contact your IBM support center to report the problem.

CSQH018I

csect-name Security refresh for *resource-type* not processed, security switch set OFF

Severity

0

Explanation

A REFRESH SECURITY command was issued for resource type *resource-type*. However, the security switch for this type or the subsystem security switch is currently set off.

Note: This message is issued only for resource types MQQUEUE, MQPROC, and MQNLIST, because MQADMIN is always available for refresh.

System programmer response

Ensure that the REFRESH SECURITY request was issued for the correct resource type.

CSQH019I

Keyword values are incompatible

Severity

8

Explanation

The REFRESH SECURITY command was issued, but the command syntax is incorrect because a keyword value that is specified conflicts with the value for another keyword.

System action

The command is not executed.

System programmer response

See REFRESH SECURITY for more information.

CSQH021I

csect-name switch-type security switch set OFF, profile '*profile-type*' found

Severity

0

Explanation

This message is issued during queue manager initialization and in response to a REFRESH SECURITY command for each security switch that is set OFF because the named security profile has been found.

System action

If the subsystem security switch is set off, you will get only one message (for that switch).

System programmer response

Messages CSQH021I through CSQH026I are issued so that you can check the security configuration of your queue manager. See Switch profiles for information about setting security switches.

CSQH022I

csect-name switch-type security switch set ON, profile '*profile-type*' found

Severity

0

Explanation

This message is issued during queue manager initialization and in response to a REFRESH SECURITY command for each security switch that is set ON because the named security profile has been found.

System programmer response

Messages CSQH021I through CSQH026I are issued so that you can check the security configuration of your queue manager. See Switch profiles for information about setting security switches.

CSQH023I

csect-name switch-type security switch set OFF, profile '*profile-type*' not found

Severity

0

Explanation

This message is issued during queue manager initialization and in response to a REFRESH SECURITY command for each security switch that is set OFF because the named security profile has not been found.

System action

If the subsystem security switch is set off, you will get only one message (for that switch).

System programmer response

Messages CSQH021I through CSQH026I are issued so that you can check the security configuration of your queue manager. See Switch profiles for information about setting security switches.

CSQH024I

csect-name switch-type security switch set ON, profile '*profile-type*' not found

Severity

0

Explanation

This message is issued during queue manager initialization and in response to a REFRESH SECURITY command for each security switch that is set ON because the named security profile has not been found.

System programmer response

Messages CSQH021I through CSQH026I are issued so that you can check the security configuration of your queue manager. See Switch profiles for information about setting security switches.

CSQH025I

csect-name switch-type security switch set OFF, internal error

Severity
0

Explanation

This message is issued during queue manager initialization and in response to a REFRESH SECURITY command for each security switch that is set OFF because an error occurred.

System action

The message might be issued with message CSQH004I when an unexpected setting is encountered for a switch.

System programmer response

See message CSQH004I for more information.

Messages CSQH021I through CSQH026I are issued so that you can check the security configuration of your queue manager.

CSQH026I

csect-name switch-type security switch forced ON, profile '*profile-type*' overridden

Severity
0

Explanation

This message is issued during queue manager initialization and in response to a REFRESH SECURITY command for each security switch that was forced ON. This happens when an attempt was made to turn off both the queue manager and queue-sharing group security switches for the named profile, which is not allowed.

System programmer response

Correct the profiles for the queue manager and queue-sharing group security switches, and refresh security if required.

Messages CSQH021I through CSQH026I are issued so that you can check the security configuration of your queue manager. See Switch profiles for information about setting security switches.

CSQH030I

Security switches ...

Severity
0

Explanation

This is issued in response to a DISPLAY SECURITY ALL or DISPLAY SECURITY SWITCHES command and is followed by messages CSQH031I through CSQH036I for each security switch to show its setting and the security profile used to establish it.

System action

If the subsystem security switch is set off, you will get only one message (for that switch). Otherwise, a message is issued for each security switch.

CSQH031I

switch-type OFF, '*profile-type*' found

Severity
0

Explanation

This message is issued in response to a DISPLAY SECURITY ALL or DISPLAY SECURITY SWITCHES command for each security switch that is set OFF because the named security profile has been found.

System action

If the subsystem security switch is set off, you will get only one message (for that switch).

CSQH032I

switch-type ON, '*profile-type*' found

Severity
0

Explanation

This message is issued in response to a DISPLAY SECURITY ALL or DISPLAY SECURITY SWITCHES command for each security switch that is set ON because the named security profile has been found.

CSQH033I

switch-type OFF, '*profile-type*' not found

Severity
0

Explanation

This message is issued in response to a DISPLAY SECURITY ALL or DISPLAY SECURITY SWITCHES command for each security switch that is set OFF because the named security profile has not been found.

System action

If the subsystem security switch is set off, you will get only one message (for that switch).

CSQH034I

switch-type ON, '*profile-type*' not found

Severity
0

Explanation

This message is issued in response to a DISPLAY SECURITY ALL or DISPLAY SECURITY SWITCHES command for each security switch that is set ON because the named security profile has not been found.

CSQH035I

switch-type OFF, internal error

Severity
0

Explanation

This message is issued in response to a DISPLAY SECURITY ALL or DISPLAY SECURITY SWITCHES command for each security switch that is set OFF because an error occurred during initialization or when refreshing security.

System action

The message is be issued when an unexpected setting is encountered for a switch.

System programmer response

Check all your security switch settings. Review the z/OS system log file for other CSQH messages for errors during IBM MQ startup or when running RUNMQSC security refresh commands.

If required, correct them and refresh your security.

CSQH036I

switch-type ON, '*profile-type*' overridden

Severity
0

Explanation

This message is issued in response to a DISPLAY SECURITY ALL or DISPLAY SECURITY SWITCHES command for each security switch that was forced ON. This happens when an attempt was made to turn off both the queue manager and queue-sharing group security switches for the named profile, which is not allowed.

System programmer response

Correct the profiles for the queue manager and queue-sharing group security switches, and refresh security if required.

CSQH037I

Security using uppercase classes

Severity
0

Explanation

This message is issued in response to a DISPLAY SECURITY ALL or DISPLAY SECURITY SWITCHES command to inform you that security is currently using the uppercase classes MQPROC, MQNLIST, MQQUEUE and MQADMIN.

CSQH038I

Security using mixed case classes

Severity
0

Explanation

This message is issued in response to a DISPLAY SECURITY ALL or DISPLAY SECURITY SWITCHES command to inform you that security is currently using the mixed case classes MXPROC, MXNLIST, MXQUEUE and MXADMIN.

CSQH040I

Connection authentication ...

Severity
0

Explanation

This message is issued during queue manager initialization, in response to a DISPLAY SECURITY command, and in response to a REFRESH SECURITY TYPE(CONNAUTH) command. It is followed by messages CSQH041I and CSQH042I to show the value of the connection authentication settings.

CSQH041I

Client checks: *check-client-value*

Severity
0

Explanation

This message is issued during queue manager initialization, in response to a DISPLAY SECURITY command, and in response to a REFRESH SECURITY TYPE(CONNAUTH) command. It shows the current value of connection authentication client checks.

If the value shown is '????' this means that the connection authentication settings were not able to be read. Preceding error messages will explain why. Any applications which connect while the queue manager is in this state will result in error message CSQH045E.

CSQH042I

Local bindings checks: *check-local-value*

Severity
0

Explanation

This message is issued during queue manager initialization, in response to a DISPLAY SECURITY command, and in response to a REFRESH SECURITY TYPE(CONNAUTH) command. It shows the current value of connection authentication local bindings checks.

If the value shown is '????' this means that the connection authentication settings were not able to be read. Preceding error messages will explain why. Any applications which connect while the queue manager is in this state will result in error message CSQH045E.

CSQH043E

csect-name Object AUTHINFO(*object-name*) does not exist or has wrong type

Severity
8

Explanation

During queue manager initialization or while processing a REFRESH SECURITY TYPE(CONNAUTH) command, the authentication information object named in the queue manager's CONNAUTH field was referenced. It was found to either not exist, or not have AUTHTYPE(IDPWOS).

System action

If this message is issued in response to a REFRESH SECURITY TYPE(CONNAUTH) command, the command fails and the connection authentication settings remain unchanged.

If this message is issued during queue manager initialization, all connection attempts are refused with reason MQRC_NOT_AUTHORIZED until the connection authentication settings have been corrected.

System programmer response

Ensure the authentication information object *object-name* has been defined correctly. Ensure the queue manager's CONNAUTH field is referencing the correct object name. Correct the configuration, then issue a REFRESH SECURITY TYPE(CONNAUTH) command for the changes to become active.

CSQH044E

csect-name Access to AUTHINFO(*object-name*) object failed, reason=*mqrc (mqrc-text)*

Severity
8

Explanation

During queue manager initialization or while processing a REFRESH SECURITY TYPE(CONNAUTH) command, the authentication information object named in the queue manager's CONNAUTH field could not be accessed for the reason given by *mqrc* (*mqrc-text* provides the MQRC in textual form).

System action

If this message is issued in response to a REFRESH SECURITY TYPE(CONNAUTH) command, the command fails and the connection authentication settings remain unchanged.

If this message is issued during queue manager initialization, all connection attempts are refused with reason MQRC_NOT_AUTHORIZED until the connection authentication settings have been corrected.

System programmer response

Ensure the authentication information object *object-name* has been defined correctly. Ensure the queue manager's CONNAUTH field is referencing the correct object name. Refer to API completion and reason codes for information about *mqrc* to determine why the object cannot be accessed. Correct the configuration, then issue a REFRESH SECURITY TYPE(CONNAUTH) command for the changes to become active.

CSQH045E

csect-name application did not provide a password

Severity

8

Explanation

An application connected without supplying a user ID and password for authentication and the queue manager is configured to require this type of application to supply one.

If this is a client application, the configuration attribute CHCKCLNT is set to REQUIRED. *application* is identified by *channel name/connection details*.

If this is a locally bound application, the configuration attribute CHCKLOCL is set to REQUIRED. *application* is identified by *user id/application name*.

If the connection authentication configuration was unable to be read, this message will also be seen. See messages CSQH041I and CSQH042I.

System action

The connection fails and the application is returned MQRC_NOT_AUTHORIZED.

System programmer response

Ensure all applications are updated to supply a user ID and password, or alter the connection authentication configuration to OPTIONAL instead of REQUIRED, to allow applications to connect that have not supplied a user ID and password.

If the connection authentication configuration was unable to be read, check for earlier error messages and make corrections based on what is reported.

After making configuration changes, issue a REFRESH SECURITY TYPE(CONNAUTH) command for the changes to become active.

If the application is a client application, the user ID and password can be supplied without changing the application code, by using a security exit, such as mqccred, which is supplied with the IBM MQ MQI client.

CSQH046E

csect-name application supplied a password for user ID *userid* that has expired

Severity

8

Explanation

An application connected and supplied a user ID *userid* and password for authentication. The password supplied has expired.

If this is a client application, *application* is identified as 'channel name'/'connection details'.

If this is a locally bound application, *application* is identified as 'running user id'/'application name'.

System action

The connection fails and the application is returned MQRC_NOT_AUTHORIZED.

System programmer response

Set a new password for *userid* using O/S facilities and retry the connect from the application using the new password.

Data manager messages (CSQI...): z/OS

CSQI002I

csect-name Page set *psid* value out of range

Severity

8

Explanation

One of the following commands has been issued:

- DEFINE STGCLASS
- DISPLAY STGCLASS
- DISPLAY USAGE

The value given for the page-set identifier was not in the range 0 through 99.

System action

The command is ignored.

System programmer response

Reissue the command using the correct syntax. (See MQSC commands for information about the command.)

CSQI003I

csect-name 'PSID' not allowed with TYPE (*usage-type*)

Severity

8

Explanation

A DISPLAY USAGE command was issued specifying both the PSID keyword and either TYPE(DATASET), or TYPE(SMDS), which is not allowed.

System action

The command is ignored.

System programmer response

Reissue the command using the correct syntax; see DISPLAY USAGE for additional information.

CSQI004I

csect-name Consider indexing *queue-name* by *index-type* for *connection-type* connection *connection-name*, *num-msgs* messages skipped

Severity

0

Explanation

The queue manager has detected an application receiving messages by message ID or correlation ID from a queue that does not have an index defined.

The type of index that should be established for the queue is indicated by *index-type*, and is either MSGID or CORRELID. The type of application that is affected is identified by *connection-type*, and is either BATCH, CHIN, CICS or IMS.

- For batch applications *connection-name* contains the job name.
- For the channel initiator *connection-name* contains the channel name.
- For CICS applications *connection-name* contains the region and transaction names.
- For IMS applications *connection-name* contains the IMS sysid, PSTID and PSB names.

The number of messages skipped while searching for the requested message, shown as *num-msgs*, is an indication of the impact of not having an index defined.

System action

Processing continues.

System programmer response

Investigate the application to determine whether an index is required for the queue.

The parameter to use with the DEFINE QLOCAL or ALTER QLOCAL command is **INDXTYPE**. Set it to *MSGID* or *CORRELID*, as indicated by the output you received for this message.

Applications that receive messages by message ID or correlation ID might encounter a performance degradation if an index is not defined and the depth of the queue is large.

CSQI005I

csect-name PAGE SET *nn* OFFLINE. RECOVERY RBA = *rba*

Severity

0

Explanation

This message indicates that the page set *nn* is currently not accessible by the queue manager. This might be because the page set has not been defined to the queue manager with the DEFINE PSID command.

This message can also be issued if the pageset has been marked suspended. If this is the case a CSQP059E: Page set *n* is suspended because it uses suspended buffer pool *n* message is issued.

Note: *rba* is the restart RBA for page set *nn*.

This situation can cause problems, so you should take action to correct it as soon as possible.

System action

Processing continues.

System programmer response

If the page set is required, bring it online; this can be done without stopping the queue manager. Use the FORMAT function of the utility program CSQUTIL, specifying TYPE(REPLACE). Then issue a DEFINE PSID command to bring the page set back into use. Note that all units of recovery (except those that are indoubt) that involved the offline page set will have been backed out by the queue manager when the page set was last used. These indoubt units of recovery may be resolved once the page set is back in use by the queue manager.

CSQI006I

csect-name COMPLETED IN-STORAGE INDEX FOR QUEUE *q-name*

Severity

0

Explanation

During restart, in-storage indexes are built for non-shared queues that have the INDXTYPE attribute, which might take some time. This message records that index-building has been completed for the specified queue.

System action

Processing continues.

CSQI007I

csect-name BUILDING IN-STORAGE INDEX FOR QUEUE *q-name*

Severity

0

Explanation

During restart, in-storage indexes are built for non-shared queues that have the INDXTYPE attribute, which might take some time. This message records that an index is being built for the specified queue.

System action

The in-storage index is built.

CSQI010I

Page set usage ...

Severity

0

Explanation

This message is the response to the DISPLAY USAGE command. It provides information about the page set usage, as follows:

```
Page ...  set          _ n   page-set-information
⋮
End of page set report
```

where *n* is the page set identifier. The columns of *page-set-information* are:

Buffer pool
The buffer pool used by the page set.

Total pages
The total number of 4 KB pages in the page set (this relates to the records parameter on the VSAM definition of the page set).

Unused pages
The number of pages that are not used (that is, available page sets).

Persistent data pages
The number of pages holding persistent data (these pages are being used to store object definitions and persistent message data).

Nonpersistent data pages
The number of pages holding nonpersistent data (these pages are being used to store nonpersistent message data).

Expansion count
The type of expansion used for the page set (SYSTEM, USER, or NONE), and the number of times the page set has been dynamically expanded since restart. (The maximum number of times the page set can be expanded is constrained by the maximum number of extents allowable for the type of VSAM data set allocation and your operating system version.) If the count is large, your page set allocation might be wrong, or you might have some message processing problem.

Note: The page numbers are approximate because other threads might be altering the status of pages in this page set while the command is being processed.

If a page set is unavailable, *page-set-information* is one of:

has never been online
if the page set has been defined, but has never been used.

OFFLINE, recovery RBA=*rba*
if the page set is currently not accessible by the queue manager, for example because the page set has not been defined to the queue manager with the DEFINE PSID command; *rba* is the restart RBA for the page set.

is not defined
if the command was issued for a specific page set that is not defined to the queue manager.

is suspended, buffer pool *buffer pool number*, **recovery RBA=***rba*
if the page set is suspended; *rba* is the restart RBA for the page set. For further information about suspended page sets see message CSQP059E: Page set *n* is suspended because it uses suspended buffer pool *n*.

Exceptionally, the last line of the report might be:

```
Page set report terminated
```

if there was an error in obtaining the information. The error is described in the following messages.

CSQI012E

csect-name COULD NOT COMPLETE COMMAND. STORAGE EXHAUSTED

Severity
8

Explanation

A display of page set usage could not complete because all the available storage was exhausted.

System action

The output terminates at this point. There might be more information that has not been displayed. If this is in response to a DISPLAY USAGE command without the PSID keyword, try it again, specifying a page set identifier. This could decrease the amount of information produced, enabling it all to be displayed.

CSQI020I

MAXSMSGS(*number*)

Severity
0

Explanation

This message is issued in response to a DISPLAY MAXSMSGS command, and displays the maximum number of messages that a task can get or put within a single unit of recovery.

CSQI021I

csect-name PAGE SET *psid* IS EMPTY. MEDIA RECOVERY STARTED

Severity
0

Explanation

The queue manager has recognized a page set with a recovery RBA of zero. It will update the page set using information in the log data sets.

System action

The queue manager rebuilds the page set.

CSQI022I

csect-name PAGE SET *psid* NEWLY ADDED

Severity
0

Explanation

The queue manager has recognized that page set *psid* is new to the system.

CSQI023I

csect-name PAGE SET *psid* ONLINE AGAIN. MEDIA RECOVERY STARTED

Severity
0

Explanation

A page set has been redefined to the queue manager after a period offline or suspended.

System action

Any updates to the page set that are necessary are applied.

CSQI024I

csect-name Restart RBA for system as configured = *restart-rba*

Severity
0

Explanation

This message gives the restart RBA (relative byte address) for the queue manager, but does not include any offline or suspended page sets in the calculation of this restart point.

This value can be used to determine where to truncate logs, if you have no offline or suspended page sets.

If you have offline or suspended page sets that you want to add to your system at some time in the future, you must use the restart RBA given in message CSQI025I. If you truncate your logs at *rba* you might make it impossible to add the offline or suspended page sets back to the system.

CSQI025I

csect-name Restart RBA including offline page sets = *restart-rba*

Severity

0

Explanation

This message gives the restart RBA (relative byte address) for the queue manager, including any offline or suspended page sets.

This value can be used to determine where to truncate logs, if you have offline or suspended page sets that you want to add to the system in the future.

CSQI026I

csect-name PAGE SET *nn* DEFINED, BUT HAS NEVER BEEN ONLINE

Severity

0

Explanation

This message indicates that the page set *nn* has been defined, but it has never been used. Consequently, there is no restart RBA for the page set.

System action

Processing continues.

CSQI027I

csect-name PAGE SET *nn* TREATED AS A NEW PAGE SET

Severity

0

Explanation

This message indicates that the page set *nn* has been formatted using TYPE(NEW). It is treated as if it has been newly-added to the system, so all historical information relating to this page set is discarded. In particular, all queues that use storage classes that reference the page set will be cleared of all messages.

System action

Processing continues.

CSQI028E

csect-name PAGE SET CONFLICT FOR QUEUE queue

Severity

8

Explanation

The named queue contains messages that are on a different page set from that associated with the storage class for the queue.

System action

This message might be issued more than once, each occurrence naming a different queue. The queue manager ends abnormally with reason code X'00C93800'.

System programmer response

Contact your IBM support center for assistance.

CSQI029I

csect-name PAGE SET *psid* IS AN OLD COPY. MEDIA RECOVERY STARTED

Severity
0

Explanation

The queue manager has recognized that the media recovery RBA held within the page set is older than the media recovery RBA checkpointed for the page set. This is because the queue manager was started with an old copy of the page set.

System action

Any updates to the page set that are necessary are applied. Restart processing continues.

CSQI030I

csect-name PAGE SET *nn* TREATED AS A REPLACEMENT PAGE SET

Severity
0

Explanation

This message indicates that the page set *nn* has been formatted using TYPE(REPLACE). No media recovery will be performed on the page set.

System action

Processing continues.

CSQI031I

csect-name THE NEW EXTENT OF PAGE SET *psid* HAS FORMATTED SUCCESSFULLY

Severity
0

Explanation

Following the dynamic extension of page set *psid*, the new extent has been formatted successfully.

System action

Processing continues.

CSQI032I

csect-name NEW EXTENT(S) OF *nnn* PAGES DISCOVERED ON PAGE SET *psid* WILL NOW BE FORMATTED

Severity
0

Explanation

During restart, it was discovered that page set *psid* had been extended dynamically, but that *nnn* pages had not been formatted. This formatting will now be done.

System action

Processing continues.

CSQI033E

csect-name Block *block-number* of the message data for entry ID *entry-id* in CFSTRUCT(*struc-name*) was not found in Db2

Severity
8

Explanation

A shared message was read which referred to message data in Db2, but the corresponding data was not found in the Db2 table.

block-number
identifies the block number within the message of the data block which was not found.

entry-id
identifies the coupling facility entry for the shared message.

struc-name
identifies the application structure.

System action

If the message was persistent, the structure is marked as failed, requiring recovery, and messages CSQI036I and CSQE035E are issued.

If the message was nonpersistent, the damaged message is deleted and message CSQI037I is issued.

In both cases, a dump is produced.

CSQI034E

csect-name Block *block-number* of the message data for entry ID *entry-id* in CFSTRUCT(*struc-name*) refers to SMDS(*qmgr-id*) control interval *rci* but the stored data does not match the entry id

Severity
8

Explanation

A shared message was read which referred to message data stored in a shared message data set (SMDS), but when the data was read from the referenced location in the data set, the entry ID in the block prefix did not match the entry ID of the message.

block-number
identifies the block number within the message of the data block which was not found.

entry-id
identifies the coupling facility entry for the shared message.

struc-name
identifies the application structure.

qmgr-ide>
identifies the queue manager which owns the shared message data set.

rci
identifies the relative control interval number within the data set where the message block was expected to start.

System action

If the message was being retrieved for backup purposes, a dump is produced and the queue manager terminates.

Otherwise, action is taken as follows:

- If the message was persistent, the shared message data set and the structure are marked as failed, requiring recovery, and messages CSQI036I and CSQE035E are issued.

- If the message was nonpersistent, the damaged message is deleted and message CSQI037I is issued.

In both cases, a dump is produced.

CSQI035E

csect-name Block *block-number* of the message data for entry ID *entry-id* in CFSTRUCT(*struc-name*) refers to SMDS but the data set ID is not valid

Severity

8

Explanation

A shared message was read which referred to message data stored in a shared message data set (SMDS), but the relevant queue manager id (identified by the last byte of the entry id) is not one which currently owns a shared message data set.

block-number
identifies the block number within the message of the data block which could not be read.

entry-id
identifies the coupling facility entry for the shared message.

struc-name
identifies the application structure.

System action

If the message was persistent, the structure is marked as failed, requiring recovery, and messages CSQI036I and CSQE035E are issued.

If the message was nonpersistent, the damaged message is deleted and message CSQI037I is issued.

In both cases, a dump is produced.

CSQI036I

csect-name CFSTRUCT(*struc-name*) has been marked as failed because the data for persistent message with entry ID *entry-id* could not be retrieved

Severity

0

Explanation

A damaged persistent message was found, so the structure has been marked as failed, requiring recovery.

struc-name
identifies the application structure.

entry-id
identifies the coupling facility entry for the shared message.

System action

The structure is marked as failed and message CSQE035E is issued.

CSQI037I

csect-name The nonpersistent message with entry ID *entry-id* has been deleted from CFSTRUCT(*struc-name*) because the data could not be retrieved

Severity

0

Explanation

A damaged nonpersistent message was found which could not be successfully retrieved, so it has been deleted.

entry-id
identifies the coupling facility entry for the shared message.

struc-name
identifies the application structure.

System action

The damaged message is deleted. No attempt is made to delete any associated SMDS message data.

CSQI038I

csect-name The damaged message with entry id *entry-id* in CFSTRUCT(*struct-name*) is for queue *queue-name*

Severity
0

Explanation

A damaged shared message entry has been found, as indicated by a previous message, and this message indicates the corresponding queue name.

struc-name
identifies the application structure.

entry-id
identifies the coupling facility entry for the shared message.

queue-name
identifies the queue for which the message cannot be retrieved.

System action

Processing continues. This message will be followed by message CSQI036I or CSQI037I, depending on whether the damaged message was persistent or not.

CSQI039E

csect-name LRSN required for structure recovery not available for one or more CF structures

Explanation

The LRSN required for structure recovery for one or more CF structures could not be located within the logs indexed in the BSDS.

Previous CSQE040I and CSQE041E messages might indicate which CF structures are causing this error to occur.

System action

Processing continues.

System programmer response

Use the **BACKUP CFSTRUCT** command, on any queue manager in the queue-sharing group, to make a new CF structure backup. You might consider setting up a procedure to take frequent backups automatically.

CSQI041I

csect-name JOB *jobname* USER *userid* HAD ERROR ACCESSING PAGE SET *psid*

Severity
0

Explanation

This message is issued when there is an error on a page set. The message identifies the job name, user ID, and page set identifier associated with the error.

CSQI042E

csect-name WLM IWMCONN request failed, rc=*rc* reason=*reason*

Severity
8

Explanation

A Workload Management Services (WLM) connect call failed. *rc* is the return code and *reason* is the reason code (both in hexadecimal) from the call.

System action

Processing continues, but WLM services are not available.

System programmer response

See the *MVS Programming: Workload Management Services* manual for information about the return and reason codes from the WLM call. When you have resolved the problem, you will need to restart the queue manager. If you are unable to solve the problem, contact your IBM support center for assistance.

CSQI043E

csect-name WLM *call-name* request for process *process-name* failed, rc=*rc* reason=*reason*

Severity
8

Explanation

A Workload Management Services (WLM) call failed. *rc* is the return code and *reason* is the reason code (both in hexadecimal) from the call.

System action

Processing continues, but WLM services are not available.

System programmer response

See the *MVS Programming: Workload Management Services* manual for information about the return and reason codes from the WLM call. When you have resolved the problem, you will need to restart the queue manager. If you are unable to solve the problem, contact your IBM support center for assistance.

CSQI044I

csect-name Process *process-name* used by queue *q-name* was not found

Severity
0

Explanation

The named queue is indexed by message tokens. An action was being performed for the queue that required the use of the Workload Management Services (WLM) IWMCLSFY service. However, the process specified by the queue does not exist, so the service name for WLM cannot be determined.

System action

A blank service name is passed to the Workload Management Services (WLM) IWMCLSFY service.

System programmer response

Correct the queue or process definitions.

CSQI045I

csect-name Log RBA has reached *rba*. Plan a log reset

Severity

4

Explanation

The current log RBA is approaching the end of the log RBA.

System action

Processing continues, unless the RBA value reaches FFF800000000 (if 6-byte log RBAs are in use) or FFFFFFC000000000 (if 8-byte log RBAs are in use) when the queue manager terminates with reason code 00D10257.

System programmer response

Plan to stop the queue manager at a convenient time and reset the logs. See RESETPAGE for information on how to reset the logs using the CSQUTIL utility program.

If your queue manager is using 6-byte log RBAs, consider converting the queue manager to use 8-byte log RBAs. See Planning to increase the maximum addressable log range for further information.

CSQI046E

csect-name Log RBA has reached *rba*. Perform a log reset

Severity

8

Explanation

The current log RBA is approaching the end of the log RBA.

System action

Processing continues, unless the RBA value reaches FFF800000000 (if 6-byte log RBAs are in use) or FFFFFFC000000000 (if 8-byte log RBAs are in use) when the queue manager terminates with reason code 00D10257.

System programmer response

Stop the queue manager as soon as it is convenient and reset the logs. See RESETPAGE for information on how to reset the logs using the CSQUTIL utility program.

If your queue manager is using 6-byte log RBAs, consider converting the queue manager to use 8-byte log RBAs. See Planning to increase the maximum addressable log range for further information.

CSQI047E

csect-name Log RBA has reached *rba*. Stop queue manager and reset logs

Severity

8

Explanation

The current log RBA is too close to the end of the log RBA range.

System action

Processing continues, unless the RBA value reaches FFF800000000 (if 6-byte log RBAs are in use) or FFFFFFC000000000 (if 8-byte log RBAs are in use) when the queue manager terminates with reason code 00D10257.

System programmer response

Stop the queue manager immediately and reset the logs. See RESETPAGE for information on how to reset the logs using the CSQUTIL utility program.

If your queue manager is using 6-byte log RBAs, consider converting the queue manager to use 8-byte log RBAs. See Planning to increase the maximum addressable log range for further information.

CSQI048I

csect-name WLM reached maximum enclave limit

Severity

4

Explanation

Workload Management Services (WLM) reported that no more enclaves could be created, so a message could not be notified to WLM. (An IWMECREA call gave a return code of 8 with a reason code of X'xxxx0836'.)

Note: This message might be issued repeatedly during the scan of the indexes for WLM-managed queues.

System action

The queue manager will attempt to notify the message to WLM again on the next scan of the indexes for WLM-managed queues. This will be after the interval specified by the WLMTIME system parameter. For information about the system parameters for the CSQ6SYSP macro, see Using CSQ6SYSP.

System programmer response

See the *MVS Programming: Workload Management Services* manual for information about the return and reason codes from the WLM call.

CSQI049I

Page set *psid* has media recovery RBA=*rcvry-rba*, checkpoint RBA= *chkpt-rba*

Severity

0

Explanation

During restart, the queue manager opened the indicated page set. The media recovery RBA from the page set itself and the check pointed RBA from the logs are as shown.

If the RBAs differ, it indicates that an old copy of the page set is being used. If the checkpoint RBA and the prior checkpoint RBA shown in message CSQR003I differ, it indicates that the page set has been offline or suspended.

System action

Processing continues. Media recovery is performed if necessary to bring the page set up to date.

CSQI059E

Unable to increase cluster cache

Severity
8

Explanation

The dynamic cluster cache cannot be increased because the queue manager cluster cache task encountered an error.

System action

The cluster cache task terminates. The channel initiator will probably terminate.

System programmer response

Investigate the problem reported in any preceding messages.

CSQI060E

QSG names differ, log=*log-name* queue manager=*qmgr-name*

Severity
8

Explanation

The queue-sharing group name recorded in the log does not match the name being used by the queue manager.

Possible causes are:

- The queue manager was restarted using the log from another queue manager.
- The queue manager was restarted with the wrong QSGDATA system parameter.
- The queue manager was not removed correctly from its previous queue-sharing group.

System action

Restart is terminated abnormally with completion code X'5C6' and reason code X'00C94505'.

System programmer response

Restart the queue manager using the correct logs and BSDS, or change the QSGDATA system parameter. Note that you cannot change the name of the queue-sharing group that a queue manager uses, or remove it from a queue-sharing group, unless it has been shut down normally and the further procedures for removal described in Managing queue-sharing groups have been followed.

CSQI061E

Queue manager queue-sharing group numbers differ, log=*log-num* queue manager=*qmgr-num*

Severity
8

Explanation

The queue manager was restarted using the log from another queue manager. The queue-sharing group queue manager number recorded in the log does not match that being used by the queue manager.

System action

Restart is terminated abnormally with completion code X'5C6' and reason code X'00C94506'.

System programmer response

Restart the queue manager using the correct logs and BSDS. If the correct logs are being used, correct the entry for the queue manager in the Db2 CSQ.ADMIN_B_QMGR table. If you cannot resolve the problem, contact your IBM support center for assistance.

CSQI062I

Queue *q-name* deleted by another queue manager during restart

Severity

0

Explanation

During restart processing the queue manager detected that the named queue has been deleted by another queue manager in the queue-sharing group.

System action

Processing continues.

CSQI063E

Queue *q-name* is both PRIVATE and SHARED

Severity

0

Explanation

During restart processing the queue manager detected that the named queue exists both as a locally-defined queue on this queue manager and as a shared queue in the queue-sharing group. Opening a queue with this name will therefore not be allowed.

System action

Processing continues.

System programmer response

Delete one of the instances of the queue. See Shared queue problems for more information.

CSQI064E

Cannot get information from Db2. *obj-type* COPY objects not refreshed

Severity

8

Explanation

During queue manager or channel initiator startup, objects of type *obj-type* with a disposition of COPY were being refreshed from those with a disposition of GROUP. However, the necessary information could not be obtained from Db2; this may be because Db2 is not available or no longer available, or because the connection to Db2 is suspended, or because there was an error in accessing Db2, or because a Db2 table was temporarily locked.

System action

The COPY objects of type *obj-type* are not refreshed. Startup continues.

System programmer response

Refer to the console log for messages giving more information about the error.

When the error condition has cleared, refresh the objects manually, or restart the queue manager or channel initiator.

CSQI065I

Buffer pool attributes ...

Severity

0

Explanation

This message displays the current state of buffer pool attributes, based on the page set number passed into the **DISPLAY USAGE PSID** command. It provides information about the number of available buffers, buffers free (stealable), shown as a number and as a percentage of the buffers in the pool, and the memory LOCATION for the specified buffer pool.

```
16.17.45 STC03474  CSQI065I !MQ21 Buffer pool attributes ...  818
Buffer   Available   Stealable    Stealable   Location   818                pool     buffers     buffers   percenta
     0       50000       49958           99      BELOW   818             _     1       20000       20000          1
 50000                   49995           99      BELOW   818             _     3       20000       19999
End of buffer pool attributes
```

Buffer pool
The number of the buffer pool.

Available buffers
The total number of available buffers defined for a specified buffer pool.

If location is SWITCHING_ABOVE or SWITCHING_BELOW, the value is the sum of the numbers above and below.

Stealable buffers
The number of buffers free (stealable) for a defined buffer pool.

Stealable percentage
The amount of buffers free (stealable), as a percentage, for a defined buffer pool.

Location
The location value of the memory used by individual buffer pools. The location value is one of the following:

ABOVE
ABOVE is displayed when OPMODE(NEWFUNC, 800) is in effect. Memory is used above the bar for buffer pools.

BELOW
BELOW is the default. Memory is used below the bar for buffer pools.

SWITCHING_ABOVE
The buffer pool is in the process of switching to a location ABOVE the bar.

SWITCHING_BELOW
The buffer pool is in the process of switching to a location BELOW the bar.

Page class
The type of virtual storage pages used for backing the buffers in the buffer pool.

CSQI070I

Data set usage ...

Severity
0

Explanation

This message is the response to the DISPLAY USAGE command. It provides information about the data sets relating to various circumstances, as follows:

```
Data set  RBA/LRSN  DSName  data-set-type:            rrr      dsname
⋮
End of data set report
```

where:

data-set-type
The type of data set and circumstance, which can be:

Log, oldest with active unit of work
The log data set containing the beginning RBA of the oldest active unit of work for the queue manager.

Log, oldest for page set recovery
The log data set containing the oldest restart RBA of any page set for the queue manager.

Log, oldest for CF structure recovery
The log data set containing the LRSN which matches the time of the oldest current backup of any CF structure in the queue-sharing group. If the oldest current backup is not found, you must back up all of your structures.

rrr The RBA or LRSN corresponding to the circumstance.

dsname
The name of the copy 1 data set. If no data set relates to a circumstance, this is shown as `None`; if the data set name cannot be determined, this is shown as `Not found`.

System programmer response

This information can be used to help manage data sets; see Tips for backup and recovery for more information.

CSQI965I
modulename Backward migration required for msgs on pageset *ps-name*

Explanation
During queue manager restart it has been detected that one or more of the page sets that have been connected has been used at a higher version of queue manager code.

System action

The queue manager will automatically perform special processing during restart to alter any messages stored on the indicated page set so they can be read by the current version of the queue manager.

CSQI966I
modulename Backward migration failed for msgs on Queue *qname*, pageset *ps-name*. Reason *reason-code*

Explanation
During backward migration of messages on the indicated queue and page set a problem was encountered which prevents further backward migration of messages.

The type of problem, for example page set full, is indicated by the reason code.

System action

The indicated page set is taken offline. Queues and messages on that page set will not be available while the queue manager is running at Version 6.0.

CSQI967I
modulename Backward migration completed for msgs on *ps-name*

Explanation

The indicated page set has had all messages successfully migrated to a format where they can be processed by applications running on an IBM WebSphere MQ Version 6.0 queue manager.

The following limitations still applies:

- If `SYSTEM.RETAINED.PUB.QUEUE` is defined on this pageset, all messages on that queue will have been deleted.
- If the queue manager is subsequently restarted at Version 7.0, then all retained publications are lost.

CSQI968I

modulename Alias queue *aq-name* to TARGQ *tq-name* has TARGTYPE *ttype* which is not supported. *aq-name* has been deleted

Explanation

During object migration, an alias queue was found which had an invalid **TARGTYPE**, for example an alias queue to a topic object.

System action

The alias queue indicated is deleted.

CSQI969I

Data set *ds-name* for page set *ps-name* was used for a higher version of IBM MQ and cannot be added dynamically

Explanation

During dynamic connection to a pageset which was offline at queue manager restart, it has been detected that it requires backward migration processing.

The pageset is not dynamically added.

CSQI970E

csect-name object-type(*object-name*) COULD NOT BE MIGRATED

Explanation

Migration of the identified object could not be performed because of locks held by in-doubt transactions.

Some functions will not be available until migration of the object can be performed. For example, the object cannot be altered or deleted, and if it is a transmission queue, the associated channel may not start.

System action

The object is not migrated.

System programmer response

Use the DISPLAY CONN or the DISPLAY THREAD command to identify the list of in-doubt transactions and then resolve them via either the transaction coordinator or the RESOLVE INDOUBT command. Once the in-doubt transactions are resolved, either restart the queue manager or issue an ALTER command against the object to re-attempt its migration.

Message CSQI971I will be issued when the object has been successfully migrated.

CSQI971I

csect-name object-type(*object-name*) MIGRATED

Explanation

The identified object could not be migrated when the queue manager was first started at the current version because of locks held by in-doubt transactions (see message CSQI970E for more information).

This message is issued during a subsequent restart of the queue manager, or when the object is subsequently altered, to indicate that migration of the object has now occurred.

System action

The object is migrated.

System programmer response

none.

Recovery log manager messages (CSQJ...): z/OS

CSQJ001I

CURRENT COPY *n* ACTIVE LOG DATA SET IS DSNAME=*dsname*, STARTRBA=*sss* ENDRBA=*ttt*

Explanation

This message is generated for one of two reasons:

1. When the queue manager starts, this information message is sent to identify the current active log data sets (copy 1 and, if dual logging is used, copy 2).
2. When the current active log data set is full (or when an ARCHIVE LOG command is issued), MQ will switch to the next available active log data set. This message identifies the next available active log data set that will be used for logging.

The value specified by STARTRBA is the RBA of the first byte of log data in the named data set. The value specified by ENDRBA is the RBA of the last possible byte in the data set.

System programmer response

None required. However, if recovery is required, information from this message might be required as input to the change log inventory utility (CSQJU003).

CSQJ002I

END OF ACTIVE LOG DATA SET DSNAME=*dsname*, STARTRBA=*sss* ENDRBA=*ttt*

Explanation

This message is sent when logging switches to a new empty data set. The message shows the name and log RBA range of the full data set.

System programmer response

None required. However, if recovery is required, information from this message might be required as input to the change log inventory utility (CSQJU003).

CSQJ003I

FULL ARCHIVE LOG VOLUME DSNAME=*dsname*, STARTRBA=*sss* ENDRBA=*ttt*, STARTTIME=*ppp* ENDTIME=*qqq*, UNIT=*unitname*, COPY*n*VOL=*vvv* VOLSPAN=*xxx* CATLG=*yyy*

Explanation

Offloading for the specified archive log data set was successfully completed for the given volume. If the data set spans multiple tape volumes, this message is generated for each tape volume.

System action

An archive log data set has been created, and the archive log data set inventory in the BSDS has been updated with the information in the message:

DSNAME
The name of the archive log data set

STARTRBA
The starting RBA contained in the volume

ENDRBA
The ending RBA contained in the volume

STARTTIME
The starting store-clock value of the log records in the volume

ENDTIME
The ending store-clock value of the log records in the volume

UNIT The device unit to which the data set was allocated

COPY*n*VOL
The name of the volume; this is displayed as COPY1VOL if this is the copy-1 archive log data set, and as COPY2VOL if this is the copy-2 archive log data set

VOLSPAN
An indicator to denote one of four conditions:

NO The data set is entirely contained on the volume specified by COPYnVOL

FIRST This is the first entry of a multivolume data set

MIDDLE
This is the middle entry of a multivolume data set

LAST This is the last entry of a multivolume data set

CATLG
An indicator to denote one of two conditions:

NO The archive log data set is uncataloged

YES The archive log data set is cataloged

The BSDS is automatically updated with the information contained in this message; however, if recovery is required, information from this message might be required as input to the change log inventory utility (CSQJU003).

CSQJ004I
ACTIVE LOG COPY *n* INACTIVE, LOG IN SINGLE MODE, ENDRBA=*ttt*

Explanation

This message is sent when the dual active logging option is selected and copy *n* becomes inactive. A log copy becomes inactive when the next active log data set is not ready when required. ENDRBA is the last byte of log data written on copy *n*. This is usually caused by a delay in offload.

System action

The log is switched to single mode until the next data set for copy *n* is ready for logging.

If the queue manager is shut down or terminates abnormally while in single mode with the system parameter option still set for dual active data sets, the previous state of the active log data sets determines what happens when the queue manager is started, as follows:

- If fewer than two data sets are available (not flagged as STOPPED) for each set of active logs, queue manager startup terminates and message CSQJ112E is issued.
- If an active log data set is in NOTREUSABLE state, the queue manager can be started in single logging mode, but dual mode takes effect when the other active log data set becomes available after offloading.

System programmer response

Perform a display request to ensure that there are no outstanding requests that are related to the log offload process. Take the necessary action to satisfy any requests, and permit offload to continue.

If the switch to single mode was caused by the lack of a resource required for offload, the necessary resource should be made available to allow offload to complete and thus permit dual logging to proceed. If recovery is required, information from this message might be required as input to the change log inventory utility (CSQJU003).

CSQJ005I
ACTIVE LOG COPY *n* IS ACTIVE, LOG IN DUAL MODE, STARTRBA=*sss*

Explanation

This message is sent when copy *n* of the log becomes active after previously being flagged as inactive. STARTRBA is the RBA of the first byte of log data written on copy *n* after it was activated.

System programmer response

None required. However, if recovery is required, information from this message might be required as input to the change log inventory utility (CSQJU003).

CSQJ006I
ALLOCATION FOR NEW ARCHIVE LOG DATA SET HAS BEEN CANCELED BY OPERATOR

Explanation

This message is sent if the operator answers 'N' to message CSQJ008E.

System action

If the allocation is for the first copy of the archive log data set, offload terminates processing until the next time it is activated. If the first copy has already been allocated and this request is for the second copy, offload switches to single offload mode for this data set only.

CSQJ007I
ALLOCATION FOR ARCHIVE VOL SER=*volser* HAS BEEN CANCELED BY OPERATOR

Explanation

If the operator answers 'N' to message CSQJ009E, this message is issued. *volser* is the volume serial of an archive log volume required to satisfy the read request. The name of the archive data set is given by message CSQJ022I which follows.

System action

The read request that needed the archive volume is unsuccessful. If the request was issued with the *COND=YES* parameter, the log manager returns to its invoker with return code 12 and reason code X'00D1032B'. Otherwise, the log manager's invoker ends abnormally with the same reason code.

CSQJ008E
nn OF *mm* ACTIVE LOGS ARE FULL, *qmgr-name* NEEDS ARCHIVE SCRATCH

Explanation

IBM MQ needs a scratch volume for offloading an active log data set. *qmgr-name* is the name of the queue manager. *nn* is the number of full active log data sets. *mm* is the total number of active log data sets.

System action

The offload task issues message CSQJ021D and waits for the operator's reply.

CSQJ009E
qmgr-name NEEDS VOL SER= *nnnnnn*

Explanation

MQ needs the specified archive volume for a read operation. *qmgr-name* is the name of the queue manager.

System action

The archive log read service task issues message CSQJ021D and waits for the operator's reply. This wait affects the agent for which the log read was issued and any other agents that might be waiting on the log read service task queue.

CSQJ010I
INVALID RESPONSE - NOT Y OR N

Explanation

During archive data set allocation, a reply message was issued. The user did not respond correctly to the reply message. Either 'Y' or 'N' must be entered.

System action

The original message is repeated.

CSQJ011D
RESTART CONTROL *rrr* CREATED AT *date time* FOUND. REPLY Y TO USE, N TO CANCEL

Explanation

During queue manager initialization, a conditional restart control record was found in the BSDS data set. Both the record identifier (a 4-byte hexadecimal number) and the creation time stamp are displayed to help identify the conditional restart record which will be used. If you want a conditional restart using that record, reply 'Y' to the message. Otherwise, reply 'N'.

System action

If 'Y' is the response, the queue manager is started conditionally, using the record found. If 'N' is the response, startup is terminated.

System programmer response

Respond as indicated.

If a normal restart has failed and you have created a conditional restart record with the change log inventory utility, check whether the time and date in the message agree with when you created that record. If they do, reply 'Y'. If they do not, reply 'N' and investigate the discrepancy.

CSQJ012E
ERROR *ccc* READING RBA *rrr* IN DATA SET *dsname*, CONNECTION-ID=*xxxx* THREAD-XREF=*yyyyyy*

Explanation

While scanning log records read into a buffer, IBM MQ detected a logical error with reason code *ccc*. *rrr* is the log RBA of the segment in the buffer at which the error was detected. *dsname* is the name of the active or archive log data set from which the record was read. If *dsname* is blank, the data was read from an active log output buffer.

The connection ID and thread-xref identify the user or application that encountered the problem. Messages that have the same connection ID and thread-xref relate to the same user.

System action

The application program is terminated with reason code *ccc*. However, information in this message might be useful in diagnosing the abnormal termination that will follow.

System programmer response

See Active log problems for information about dealing with problems on the log.

CSQJ013E
TERMINAL ERROR *ccc* IN BUFFER *rrr* BEFORE ACTIVE LOG WRITE

Explanation

A scan of the log output buffer, just prior to writing the buffer, detected an inconsistency in the log data. *ccc* is the reason code associated with the SDUMP that is produced. *rrr* is the log RBA at which the error was detected.

System action

The queue manager will terminate with a dump, and will not write the damaged buffer to either COPY 1 or COPY 2 active log data set.

System programmer response

Restart the queue manager after it terminates.

Because the damaged buffer has not been written to a log data set, the queue manager can be restarted. No corrective action is required.

CSQJ014E
TERMINAL ERROR *ccc* IN BUFFER *rrr* AFTER ACTIVE LOG WRITE

Explanation

A scan of the log output buffer, after writing to the first copy of the active log data set and before writing to the second copy, detected an inconsistency in the log data. *ccc* is the reason code associated with the SDUMP that is produced. *rrr* is the log RBA at which the error was detected.

System action

The queue manager terminates with a dump, and does not write the damaged buffer to the COPY 2 data set.

System programmer response

The block containing the indicated log RBA might be damaged. The buffer was found to be in error at the completion of the write to the COPY 1 data set of the active log.

If dual active logs are being used, use the print log map utility (CSQJU004) to list the active log data sets for both copies of the active log. Find the COPY 2 data set with the corresponding RBA, and copy that data set (using Access Method Services REPRO) to the COPY 1 data set. Start the queue manager.

If only a single active log is used, contact the IBM support center for assistance. An attempt to start the queue manager might succeed if the damage to the buffer occurred after completion of the write to DASD.

CSQJ020I
csect-name RECEIVED REPLY OF N TO *msg-num*. QUEUE MANAGER STARTUP IS TERMINATED

Explanation

The operator chose to terminate queue manager startup by answering 'N' to *msg-num*.

System action

The queue manager will not restart.

CSQJ021D
REPLY Y WHEN DEVICE READY OR N TO CANCEL

Explanation

An archive log data set needs allocating, as indicated in the preceding CSQJ008E or CSQJ009E message.

System action

The log service task waits for the operator's reply.

CSQJ022I
DSNAME=*dsname*

Explanation

dsname is the name of the archive data set to which the preceding message refers.

CSQJ030E
RBA RANGE *startrba* TO *endrba* NOT AVAILABLE IN ACTIVE LOG DATA SETS

Explanation

Previous errors have made the active log data sets (that contain the RBA range reported in the message) unavailable. The status of these logs is STOPPED in the BSDS.

System action

The queue manager terminates with a dump.

System programmer response

The log RBA range must be available for the queue manager to be recoverable. Correct the previous errors and restore the active log data sets that contain the RBA range reported in the message.

- If the log data sets are recoverable, the active log data set inventory in the BSDS must be modified to reset the STOPPED status. Use the print log map utility (CSQJU004) to obtain a copy of the BSDS log inventory. Next, use the change log inventory utility (CSQJU003) to delete the active log data sets marked STOPPED (use the DELETE statement), then add them again (use the NEWLOG statement). The starting and ending RBA for each active log data set must be specified on the NEWLOG statement when the logs are added back to the BSDS using the change log inventory utility.
- If the log data sets are not recoverable, see Active log problems for information about dealing with problems on the log.

CSQJ031D

csect-name, THE LOG RBA RANGE MUST BE RESET. REPLY 'Y' TO CONTINUE STARTUP OR 'N' TO SHUTDOWN.

Explanation

If, during queue manager initialization, the current log RBA value is equal or higher than FF8000000000 (if 6-byte log RBAs are in use) or FFFFFC0000000000 (if 8-byte log RBAs are in use) this message is issued for the operator to confirm if the restart of the queue manager should continue.

System action

If 'Y' is the response, the queue manager startup continues.

If 'N' is the response, the queue manager startup terminates.

System programmer response

Stop the queue manager and reset the logs as soon as possible. See RESETPAGE for information on how to reset the logs using the CSQUTIL utility program.

If your queue manager is using 6-byte log RBAs, consider converting the queue manager to use 8-byte log RBAs. See Planning to increase the maximum addressable log range for further information.

CSQJ032E

csect-name alert-lvl - APPROACHING END OF THE LOG RBA RANGE OF *max-rba*. CURRENT LOG RBA IS *current-rba*.

Explanation

The current log RBA is approaching the end of the log RBA range. *current-rba* is the current log RBA value. The current log RBA should not be allowed to advance to the maximum log RBA value of *max-rba*.

This message is issued during queue manager initialization, or after the active log data set is full and the queue manager switches to the next available log data set.

`alert-lvl` indicates one of the following:

WARNING
Issued when the current log RBA reaches the F80000000000 value (if 6-byte log RBAs are in use) or FFFFC00000000000 (if 8-byte log RBAs are in use).

CRITICAL
Issued after the log RBA value reaches FF8000000000 (if 6-byte log RBAs are in use) or FFFFFC0000000000 (if 8-byte log RBAs are in use).

System action

Processing continues, unless the RBA value reaches FFF800000000 (if 6-byte log RBAs are in use) or FFFFFFC000000000 (if 8-byte log RBAs are in use) when the queue manager terminates with reason code 00D10257.

System programmer response

Plan to stop the queue manager and reset the logs as soon as possible; see RESETPAGE for information on how to reset the logs using the CSQUTIL utility program.

If your queue manager is using 6-byte log RBAs, consider converting the queue manager to use 8-byte log RBAs. See Planning to increase the maximum addressable log range for further information.

CSQJ033I
FULL ARCHIVE LOG VOLUME DSNAME=*dsname*, STARTRBA= *sss* ENDRBA=*ttt*, STARTLRSN=*ppp* ENDLRSN=*qqq*, UNIT=*unitname*, COPYnVOL=*vvv* VOLSPAN=*xxx* CATLG=*yyy*

Explanation

Offloading for the specified archive log data set was successfully completed for the given volume. If the data set spans multiple tape volumes, this message is generated for each tape volume.

This message is issued in place of CSQJ003I for queue-sharing groups.

System action

See message CSQJ003I. STARTTIME and ENDTIME are replaced by the following:

STARTLRSN
The starting LRSN contained in the volume for queue-sharing groups.

ENDLRSN
The ending LRSN contained in the volume for queue-sharing groups.

CSQJ034I
csect-name END OF LOG RBA RANGE IS *max-rba*

Explanation

This message is issued during queue manager startup, to indicate the end of the log RBA range that can be addressed using the current log RBA size.

A *max-rba* value of 0000FFFFFFFFFFFF indicates that the queue manager is configured to use 6 byte RBAs, while a value of FFFFFFFFFFFFFFFF indicates that the queue manager is configured to use 8 byte RBAs.

You must reset the queue manager's log before the highest used log RBA reaches the end of the log RBA range.

System action

Processing continues

System programmer response

If *max-rba* is 0000FFFFFFFFFFFF, consider converting the queue manager to use an 8-byte log RBA, to maximize the period of time until a reset of the queue manager's log is required. See Planning to increase the maximum addressable log range for further information.

CSQJ060E
parm-name system parameters are unusable

Explanation

The format of the parameters set by *parm-name* in the system parameter load module is invalid, so they cannot be used.

System action

The queue manager is terminated with abnormally with reason code X'00E80084'.

System programmer response

Ensure that the queue manager is started with a correct system parameter module, for example CSQZPARM. If necessary, reassemble the module that uses the indicated parameters, and relink-edit your system parameter load module.

CSQJ061I
parm-name system parameters are obsolete

Explanation

The parameters set by *parm-name* in the system parameter load module use some values which are now obsolete.

System action

Processing continues. The obsolete parameters are ignored, and default values are used for new parameters.

System programmer response

Review your system parameter settings. If necessary, reassemble the module that uses the indicated parameters, and relink-edit your system parameter load module.

CSQJ070E
csect-name ARCHIVE LOG DSN PREFIX NOT IN PROPER FORMAT TO RECEIVE TIME STAMP DATA. TIME STAMPING OF *dsname* BYPASSED

Explanation

The system parameters (set by CSQ6ARVP) specify that the date and time of creation of an archive log data set be included as part of the archive log data set name (DSN). To accomplish this, IBM MQ requires that the length of the archive log data set name prefix is limited. If the prefix requirement is not met, this message is issued just prior to the allocation of the archive log data set specified in the message.

System action

The archive log data set will be allocated using the archive log prefix. However, the archive log DSN will not contain the date and time as the user requested.

System programmer response

The system parameters for the log archive function must be changed. Specifically, the TSTAMP and ARCPFXn fields are not consistent with one another. For information about the actions required to eliminate this problem, see Using CSQ6ARVP.

CSQJ071E
csect-name TIMER FAILURE CAUSED TIME STAMPING OF ARCHIVE *dsname* TO BE BYPASSED

Explanation

The system parameters (set by CSQ6ARVP) specify that the date and time of creation of an archive log data set be included as part of the archive log data set name (DSN). However, an attempt to get the current date and time from the system was unsuccessful. This message is issued just prior to the allocation of the archive log data set specified in the message.

System action

The archive log data set will be allocated using the archive log prefix. However, the archive log DSN will not contain the date and time as the user requested.

CSQJ072E
ARCHIVE LOG DATA SET *dsname* HAS BEEN ALLOCATED TO NON-TAPE DEVICE AND CATALOGED, OVERRIDING CATALOG PARAMETER

Explanation

The system parameters (set by CSQ6ARVP) specify that all archive log data sets should be uncataloged (CATALOG=NO). However, MQ requires that all archive log data sets allocated to non-tape devices must be cataloged. The archive log data set specified by *dsname* has been allocated to a non-tape device, and has thus been cataloged. The user's system parameter CATALOG setting of NO has been overridden.

System action

The archive log data set has been allocated to a nontape device, and has been cataloged. The system parameter CATALOG=NO setting has been overridden. The BSDS reflects that the data set has been cataloged.

System programmer response

The archive system parameters must be changed. Specifically, the CATALOG and UNIT parameters are not consistent with one another. For information about the actions required to eliminate this problem, see Using CSQ6ARVP.

CSQJ073E
LOG ARCHIVE UNIT ALLOCATION FAILED, REASON CODE= *ccc*. ALLOCATION OR OFFLOAD OF ARCHIVE LOG DATA SET MAY FAIL

Explanation

While building the SVC99 text entries to allocate a new archive log data set dynamically, a unit allocation error was detected. The reason code, indicated by *ccc* in the message, further clarifies the problem as follows:

4-28 (X'4'-X'1C')
Return code from z/OS IEFGB4UV macro. Common values are:

4 (X'04')
Invalid unit name

8 (X'08')
Unit name has incorrect units assigned

16 (X'10')
No storage available

20 (X'14')
Device numbers not valid

32 (X'20')
MQ was able to obtain a list of devices corresponding to the device type (unit name) specified in the system parameters. However, it was determined that this list contained a mixture of tape and nontape devices.

36 (X'24')
Nonfetch-protected storage could not be obtained to build a parameter list for a z/OS service.

40 (X'28')
The device type (unit name) specified by the user in the system parameters is valid. However, no devices are currently associated with the given device type (unit name).

44 (X'2C')
The device type (unit name) specified by the user in the system parameters is valid. However, no DASD volumes are available with a volume use attribute of *storage*.

System action

This message is issued after the SVC99 text entries are built, but prior to the allocation of the new archive log data set. As a result of the error, the dynamic allocation of the archive log data set will be attempted using standard default values. The standard default values are generally acceptable; however, the allocation might be unsuccessful or the subsequent offload might produce undesirable processing results. For example:

- A reason code of 4 or 44 (X'2C') indicates an allocation error (CSQJ103E) when the SVC99 is issued for the archive data set.
- Offload processing to tape might be unsuccessful. IBM MQ uses a volume count of 20 when allocating to tape, and uses the standard z/OS volume count default of 5 volumes when writing to non-tape devices. In the case of most of the above errors, it would be impossible for IBM MQ to determine the device type on which the data set is to be allocated. Therefore, the standard z/OS default is assumed for the volume count. If the data set is successfully allocated to a tape device, and the volume of data is such that more than five volumes will be used for the archive data set, the offload processing will receive a z/OS completion code X'837-08' with message IEC028I when attempting to write to the sixth tape volume.
- Offload processing to a direct access device might be unsuccessful. When allocating a new archive log data set on a direct access device, IBM MQ will use a unit count to facilitate multivolume archive data sets. With most of the above errors, it might be impossible for IBM MQ to correctly determine the type of device on which the data set is to be allocated. Therefore, the standard default (1) is assumed for the unit count. If the data set is successfully allocated to a direct access device, and during the offload processing it becomes necessary to extend the data set to another device, the offload processing will receive a z/OS X'B37' (out of space) completion code, and the archive log data set will be deallocated.

System programmer response

The required action is based on the reason code indicated in the message:

4-28 (X'4'-X'1C')
See the *MVS Authorized Assembler Services Guide* for more info about the return code from the z/OS IEFGB4UV macro. The most likely causes for the common values are:

4 (X'04')
Incorrect specification in the archive system parameters. Correct the UNIT parameter. If the UNIT parameter from the archive system parameters appears to be correct, check the EDT to ensure that the esoteric or generic unit name specified in the parameters is actually in the EDT. Subsequent offload processing will archive the log data which could not be previously archived due to the allocation error (CSQJ103E).

8 (X'08')
Incorrect specification in archive system parameters, incorrect operational setup.

16 (X'10')
This is usually a temporary problem. If the allocation of the archive log data set is successful, no action is required to correct this situation. If this is a recurring

problem, sufficient page space is not available, and the region size for the queue manager address space might have to be increased, or standard z/OS diagnostic procedures might have to be used to correct the problem.

20 (X'14')
Incorrect specification in archive system parameters, incorrect operational

32 (X'20') or 40 (X'28')
To correct this situation, change the archive system parameter UNIT to use a device type (unit name) that contains homogenous devices, or modify the device list associated with the device type (unit name) using a system generation to supply a list of homogenous devices.

44 (X'2C')
To correct this situation, issue the z/OS command MOUNT to change the volume use attribute of a mounted private volume to storage. If this is a recurring problem, you might have to do one of the following:

- Perform a system generation to add permanently resident volumes with a volume use attribute of storage to the esoteric or generic unit
- Change the archive system parameters to use a different esoteric or generic unit name for the UNIT

CSQJ077E
LOG OR BSDS READ ERROR FOR QMGR *qmgr-name*, REASON CODE=*ccc*

Explanation

This message identifies a queue manager with log data that cannot be accessed. The logs or BSDSs of other queue managers in a queue-sharing group might be accessed during a RECOVER CFSTRUCT operation or during the rebuild of peer administration structures that might occur on a queue manager in a queue-sharing group.

System action

The execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

Look for earlier messages which might identify more specifically the data set being accessed and the problem.

If you are unable to solve the problem, note the reason code, collect the following items, and contact your IBM support center:

- System dump
- Console output for the issuing queue manager
- Console output for the other queue manager
- Printout of SYS1.LOGREC

CSQJ098E
csect-name RESTART CONTROL ENDLRSN *rrr* IS NOT IN KNOWN LRSN RANGE. QUEUE MANAGER STARTUP IS TERMINATED

Explanation

A conditional restart control record requests truncation, but it cannot take place because the end LRSN was not in the range of LRSN values known to either the active or archive logs. *rrr* is the end LRSN specified in the active record. The end LRSN is either higher than the end LRSN of the most recent active log data set, or lower than the starting LRSN of the oldest archive log data set.

System action

Queue manager startup is terminated.

System programmer response

Check the ENDLRSN value specified in the conditional restart control record. If it is not correct, run the change log inventory utility (CSQJU003) using CRESTART CANCEL cancel the conditional restart, and a new CRESTART specifying the correct ENDLRSN.

CSQJ099I
LOG RECORDING TO COMMENCE WITH STARTRBA= *sss*

Explanation

This message is generated during queue manager startup. The value specified by STARTRBA is the RBA of the next byte of log data to be recorded in the active log data sets.

This message is preceded by one (if single logging) or two (if dual logging) CSQJ001I messages.

System programmer response

None required. However, if recovery is required, information from this message might be required as input to the change log inventory utility (CSQJU003).

CSQJ100E
csect-name ERROR OPENING BSDS*n* DSNAME=*dsname*, ERROR STATUS=*eeii*

Explanation

During startup, or while processing a RECOVER BSDS command, MQ could not open the specified BSDS. BSDS*n* matches the DDname in the queue manager started task JCL procedure (xxxxMSTR) of the data set that cannot be opened. The value of *n* is 1 or 2. The error status contains the VSAM open return code in *ee*, and the VSAM open reason code in *ii*.

System action

When this error occurs at initialization time, startup must be terminated, because the log data sets cannot be determined and allocated without the BSDS. When this error occurs during RECOVER BSDS processing, the command is terminated, and the queue manager continues in single BSDS mode.

System programmer response

Recover the BSDS that cannot be opened. See Active log problems for information about dealing with problems on the BSDS or the log.

CSQJ101E
csect-name RESTART CONTROL ENDRBA *rrr* IS NOT IN KNOWN RBA RANGE. QUEUE MANAGER STARTUP IS TERMINATED

Explanation

A conditional restart control record requests truncation, but it cannot take place because the end RBA was not in the range of RBA values known to either the active or archive logs. *rrr* is the end RBA specified in the active record. The end RBA is either higher than the end RBA of the most recent active log data set, or lower than the starting RBA of the oldest archive log data set.

System action

Queue manager startup is terminated.

System programmer response

Check the ENDRBA value specified in the conditional restart control record. If it is not correct, run the change log inventory utility (CSQJU003) using CRESTART CANCEL cancel the conditional restart, and a new CRESTART specifying the correct ENDRBA.

Otherwise, then most likely, the archive log data set that contained the requested RBA has been deleted from the BSDS data set by the change log inventory utility. Locate the output from an old print log map utility and identify the data set that contains the missing RBA. If the data set has not been reused, run the change log inventory utility to add this data set back into the inventory of log data sets. Restart the queue manager.

CSQJ102E
LOG RBA CONTENT OF LOG DATA SET DSNAME= *dsname*, STARTRBA= *sss* ENDRBA=*ttt*, DOES NOT AGREE WITH BSDS INFORMATION

Explanation

The log RBA range shown in the BSDS for the specified data set does not agree with the content of the data set.

System action

Startup processing is terminated.

System programmer response

Use the print log map and change log inventory utilities to make the BSDS consistent with the log data sets.

CSQJ103E
csect-name LOG ALLOCATION ERROR DSNAME=*dsname*, ERROR STATUS=*eeeeiiii*, SMS REASON CODE=*ssssssss*

Explanation

An error occurred while attempting to allocate the active or archive log data set indicated by DSNAME. STATUS indicates the error reason code returned by z/OS dynamic allocation (SVC99).

This message might be preceded by message CSQJ073E.

System action

Subsequent actions depend on the type of data set involved.

For active log data sets, if the error is encountered during queue manager initialization, startup is terminated. If two copies of the active log data sets are defined, this message appears only once.

For archive log data sets, if two copies of the archive log data sets are defined, processing continues on the remaining archive log data set.

System programmer response

The error status portion of this message contains a 2-byte error code (*eeee*, S99ERROR) followed by the 2-byte information code (*iiii*, S99INFO) from the SVC99 request block. If the S99ERROR code indicates an SMS allocation error ('97xx'), then *ssssssss* contains additional SMS reason code information obtained from S99ERSN. See the *MVS Authorized Assembler Services Guide* manual for a description of these codes.

For active log data sets, if the problem occurred during queue manager initialization, you can resolve the problem by doing one of the following:

- Resolve the error associated with the active log data set as indicated by STATUS
- Provide another copy of the active log data set, using Access Method Services
- Update the BSDS with the change log inventory utility (CSQJU003)
- Restart the queue manager

For archive log data sets:

- If the problem occurred during allocation with the intent to write the data set, no immediate action is required. However, if you do not resolve the SVC99 error (indicated by the STATUS

value in the message), the available space in the active log could eventually be exhausted (CSQJ111A) because all future offloads might be unsuccessful because of the same error.

- If the problem occurred during allocation with the intent to read the data set, determine the problem, and use the change log inventory utility (CSQJU003) DELETE function to delete the archive log data set from the BSDS archive log inventory. Then use the NEWLOG function to add the data set back into the archive log inventory, pointing to the correct volume and device.

See Active log problems for information about dealing with problems on the log.

This message might also be issued as the result of a user error. If STATUS displays a value of '17080000', you might have one or more active log data sets defined in the BSDS, but not allocated on DASD. To correct the situation, print the contents of the current active log data set inventory using the print log map utility (CSQJU004), then either:

- Use Access Method Services to allocate the active log data set for each active log data set listed in the BSDS, but not actually allocated on DASD. You can find the Access Method Services command syntax for active log data sets in the CSQ4BSDS sample JCL.
- Use the change log inventory utility (CSQJU003) DELETE statement to delete the errant active log data set name, and the NEWLOG statement to add the correct name to the active log data set inventory. The name specified on the NEWLOG statement must be the same as the name of the actual active log data set allocated on DASD.

CSQJ104E

csect-name RECEIVED ERROR STATUS *nnn* FROM *macro-name* FOR DSNAME *dsname*

Explanation

An error ocurred while issuing macro *macro-name*. Error status is the return code from the specified macro:

- For an OPEN of a VSAM data set, the return code in the error field of the Access Method Services control block is included in this message as the error status value. See the *DFSMS/MVS Macro Instructions for Data Sets* manual for a description of these values.
- If the OPEN was for a non-VSAM data set, the error status is zero.
- For MMSRV errors, error status contains the error information returned by media manager services. If an MMSRV CATUPDT error occurs attempting to truncate an active log data set, the log data set will be unavailable and the status of the log data set will be flagged as STOPPED in the BSDS.
- For VSAM OPEN and MMSRV errors, this message is preceded by an IEC161I message that defines the error that occurred.
- For a PROTECT of an archive log data set, the return code is from DADSM PROTECT. See the *MVS/ESA System - Data Administration* manual for details of the return code.

See Active log problems for information about dealing with problems on the log.

System action

Subsequent actions depend on when the error occurred.

During queue manager initialization, startup is terminated.

When using the data set either for offload or for input operations, processing continues. If a second copy of the data is available, IBM MQ attempts to allocate and open the second data set.

When using the data set as an active log data set, IBM MQ attempts to retry the request. If the retry is unsuccessful, the queue manager is terminated.

During checkpoint processing, where IBM MQ attempts to locate the oldest active or archive log data sets that are required for restart recovery of page sets and restart and media recovery of CF structures, processing continues. The message is a warning that either restart recovery would fail

or media recovery of CF structures would fail. It is most likely to occur when all CF application structures are not being regularly backed up, thereby requiring excessively old log data sets for recovery.

System programmer response

If the error occurred during initialization, either correct the problem so that the data set is available or provide another copy of the data set and change the BSDSs to point to the new data set.

If the error occurred after startup, the return code should be reviewed and the appropriate action taken to correct the problem, so that the data set can be used at a later time, or the data set entry can be removed from the BSDS using the change log inventory utility.

If the error was received from PROTECT, there might be a problem with the PASSWORD data set. See the appropriate DADSM publication to determine the cause of the problem. When the problem has been corrected, ensure the archive log data sets receiving the error are added to the PASSWORD data set. If these archive log data sets are not added to the PASSWORD data set, archive read will not be able to OPEN these data sets. If you do not have information about the named macro, note the macro name and the return code and contact your IBM support center for help.

If the error occurred during checkpoint processing, issue the DISPLAY USAGE TYPE(DATASET) command to show which log data sets are currently required for page set and media recovery, and ensure that they are available. If applicable, use the BACKUP CFSTRUCT command for your CF structures, and institute a procedure to back up your CF structures frequently.

CSQJ105E

csect-name LOG WRITE ERROR DSNAME= *dsname*, LOGRBA=*rrr*, ERROR STATUS=*ccccffss*

Explanation

An error occurred while writing a log data set. If *csect-name* is CSQJW107, the error occurred writing the log buffers to an active log data set. If *csect-name* is CSQJW207, the error occurred while preformatting the next control area before writing log data into it.

Error status contains the error information returned by media manager in the form *ccccffss*, where *cccc* is a 2-byte return code that describes the error, *ff* is a 1-byte code that defines the functional routine that detected the error, and *ss* is the 1-byte status code that defines a general category of error.

System action

If the dual active logging option is selected, then IBM MQ switches to the next data set for this copy. If the next data set is not ready, IBM MQ temporarily enters single logging mode and allocates a replacement data set for the one that encountered the error. Dual logging is resumed as soon as possible.

If single active logging option is selected and the next data set is not ready, IBM MQ waits for that data set to be available. In this case, log writing is inhibited until the replacement is ready for output.

System programmer response

See the *MVS/DFP Diagnosis Reference* manual for information about return codes from the media manager. If you are unable to resolve the problem, note the return code, and contact your IBM support center.

CSQJ106E

LOG READ ERROR DSNAME=*dsname*, LOGRBA=*rrr*, ERROR STATUS=*ccccffss*

Explanation

An error occurred while reading an active log data set. The error status contains the error information returned by the media manager in the form *ccccffss*, where *cccc* is a 2-byte return code that describes the error, *ff* is a 1-byte code that defines the functional routine that detected the error, and *ss* is the 1-byte status code that defines a general category of error. (See the *MVS/DFP Diagnosis Reference* manual for information about return codes from the media manager.)

System action

If another log data set contains the data, IBM MQ attempts to read the data from the alternate source. If an alternate source is not available, a read error return code is sent to the program requesting the log data. Depending on the circumstances under which the failure occurred, the queue manager might continue with the alternate log data set if dual logging is used, or end abnormally.

System programmer response

If you are using dual logging, the requested RBA was probably retrieved from the corresponding dual active log data set, and no immediate response is necessary. However, if this error occurs frequently, or if you are using single logging, immediate attention might be required. If so, note the contents of the error status field, and contact your IBM support center for help.

It might be necessary to replace the data set in error with a new data set containing the log data, and to update the BSDSs to reflect the new data set using the change log inventory (CSQJU003) NEWLOG operation.

See Active log problems for information about dealing with problems on the log.

This message might also be issued as the result of a user error. If the data set name specified by DSNAME is missing, and STATUS displays a value of '00180408' or '00100408', you are using dual logging, but only one set of active log data sets is defined in the BSDS. To resolve this condition, do either of the following:

- Define a second set of active log data sets using Access Method Services (if they are not defined already), and update the BSDS log inventory, using the change log inventory (CSQJU003) NEWLOG operation.
- Reset the log system parameters to indicate single logging. You can do this by setting TWOACTV to 'NO' in the CSQ6LOGP system parameters.

CSQJ107E
READ ERROR ON BSDS DSNAME=*dsname* ERROR STATUS=*eee*

Explanation

An error occurred while reading the specified BSDS. Error Status contains the VSAM return and feedback codes. It is a 2-byte field with the first byte containing the hexadecimal return code and the second containing the hexadecimal feedback code. See the *DFSMS/MVS Macro Instructions for Data Sets* manual for a description of VSAM return and reason codes.

See Active log problems for information about dealing with problems on the BSDS or the log.

System action

If dual BSDSs are available, MQ attempts to read from the other BSDSs. If the read from the second BSDS fails or if there is only one BSDS, an error code is returned to the log request that caused access to the BSDS.

If the read error is detected during startup, the queue manager terminates.

System programmer response

It might be necessary to replace or repair the BSDS, depending on what conditions resulted from the read error. To replace a BSDS, first delete the BSDS in error, then define the new BSDS with

the same name and attributes. If a new name is used for the new BSDS, change the queue manager started task JCL procedure (xxxxMSTR) to specify the new BSDS name.

CSQJ108E
WRITE ERROR ON BSDS DSNAME=*dsname* ERROR STATUS=*eee*

Explanation

An error occurred while writing to the specified BSDS. Error Status contains the VSAM return and feedback codes. It is a 2-byte field with the first containing the hexadecimal return code and the second containing the hexadecimal feedback code. See the *DFSMS/MVS Macro Instructions for Data Sets* manual for a description of VSAM return and reason codes.

System action

If dual BSDSs are available, MQ enters single BSDS mode using the remaining good BSDS. Otherwise, an error code is returned to the log request that caused access to the BSDS.

System programmer response

If dual BSDS mode is being used, run an offline Access Method Services job to rename the error BSDS and define a new BSDS with the same name. Then enter the RECOVER BSDS command to reestablish dual BSDS mode.

If dual BSDS mode is not being used, the queue manager must be shut down, and the BSDS must be recovered from a backup copy. To recover the BSDS, use the change log inventory utility.

CSQJ109E
OUT OF SPACE IN BSDS DSNAME=*dsname*

Explanation

There is no more space in the specified BSDS. The operation that encountered the out-of-space condition did not complete properly.

System action

If dual BSDSs are available, IBM MQ enters single BSDS mode using the remaining good BSDS. Otherwise, an error code is returned to the log request that caused access to the BSDS.

System programmer response

If dual BSDS mode is being used, run an offline Access Method Services job to rename the full BSDS and define a new, larger BSDS with the same name. Enter the RECOVER BSDS command to reestablish dual BSDS mode.

If dual BSDS mode is not being used, the queue manager must be shut down and the BSDS recovered offline. In this case, run the same Access Method Services job mentioned above to rename the full data set and define a larger data set. Next, run an Access Method Services REPRO job to copy the full BSDS into the new BSDS.

CSQJ110E
LAST COPY*n* ACTIVE LOG DATA SET IS *nnn* PERCENT FULL

Explanation

This message is issued when the last available active log data set is 5% full, and is reissued after each additional 5% of the data set space is filled.

System action

Each time the message is issued, the offload processing will be re-attempted. If the situation is not corrected, the active log data set will fill to capacity, message CSQJ111A will be issued, and IBM MQ processing will stop.

System programmer response

To clear this condition, you must take steps to complete other waiting offload tasks. Once an active log data set is made available (reusable) by completing the offload process for it, the IBM MQ logging activity can continue.

Perform a display request to determine the outstanding requests related to the log offload process. Take the necessary action to satisfy any requests, and permit offload to continue.

Consider whether there are sufficient active log data sets. If necessary, additional log data sets can be added dynamically using the DEFINE LOG command.

If offload does not complete normally or cannot be initiated, either correct the problem that is causing the offload process error, increase the size of the allocated data sets, or add more active log data sets. Note that the latter action requires the queue manager to be inactive and the change log inventory utility to be run.

Possible causes for the shortage of active log data space are:

- Excessive logging. For example, there is a lot of persistent message activity.
- Delayed or slow offloading. For example, failure to mount archive volumes, incorrect replies to offload messages, or slow device speeds.
- Excessive use of the ARCHIVE LOG command. Each invocation of this command causes IBM MQ to switch to a new active log data set and to initiate an offload of the active log. Although the command will not be processed when only one active log data set remains in a copy of the active log (see CSQJ319I), excessive use of the command could have consumed all space in the active log except the current active log data sets.
- Offloads were unsuccessful.
- Insufficient active log space.

CSQJ111A
OUT OF SPACE IN ACTIVE LOG DATA SETS

Explanation

Due to delays in offload processing, all available space in all active log data sets has been exhausted. Recovery logging cannot continue.

System action

MQ waits for an available data set. Any tasks performing MQ API calls that require logging will wait.

System programmer response

Perform a display request to ensure that there are no outstanding requests that are related to the log offload process. Take the necessary action to satisfy any requests, and permit offload to continue.

Consider whether there are sufficient active log data sets. If necessary, additional log data sets can be added dynamically using the DEFINE LOG command.

If the delay was caused by the lack of a resource required for offload, the necessary resource must be made available to allow offload to complete and thus permit logging to proceed. For information about recovery from this condition, see Archive log problems.

If the problem occurred because archiving was set off, or because archive data sets could not be allocated, or for any other reason that requires the system parameters to be changed, the queue manager must be canceled as neither STOP MODE(QUIESCE) nor STOP MODE(FORCE) commands will work.

To free any tasks that are waiting because they were performing MQ API calls that require logging, you must solve the underlying problem, or cancel the queue manager.

If the offload process has stalled because some resource is not available or for some other reason, it may be possible to resolve the problem by canceling the currently executing offload task using

the ARCHIVE LOG CANCEL OFFLOAD command, and then starting another. If there are hardware problems, it may be necessary to use z/OS commands to cancel the devices with problems.

CSQJ112E

csect-name INSUFFICIENT ACTIVE LOG DATA SETS DEFINED IN BSDS

Explanation

There are not enough active log data sets defined in the BSDS to start the queue manager. This condition usually exists for one of the following reasons:

- Fewer than two data sets are defined for one of the active log copy sets.
- The CSQ6LOGP system parameters specified TWOACTV=YES but data sets for two copies of active log are not defined in BSDS.
- Fewer than two data sets are available (not flagged as STOPPED) for one of the active log copy sets.

System action

Startup is terminated.

System programmer response

Use the change log inventory utility to make the number of active log data sets defined in the BSDS consistent with the system parameters specified in CSQ6LOGP, or to add further active log data sets so that there are two or more active log data sets available for use in each active log copy. Restart the queue manager.

Note: Log data sets that are flagged as STOPPED will not be reused by IBM MQ. Once the queue manager has been restarted you might need to recover STOPPED log data sets. To clear the STOPPED status:

1. Stop the queue manager
2. Recover the log data set (either redefined or recovered from the other copy of the log)
3. Delete and re-add to the BSDS (using the change log inventory utility) with the appropriate RBAs

CSQJ113E

RBA *log-rba* NOT IN ANY ACTIVE OR ARCHIVE LOG DATA SET, CONNECTION-ID=*xxxx* THREAD-XREF=*yyyyyy*

Explanation

There was a request to read the log record starting at this RBA. However, this log record cannot be found in any active or archive log data set. The connection ID and thread-xref identify the user or application that encountered the problem (this could be an internal IBM MQ task). See Active log problems for information about dealing with problems on the log.

System action

Depending upon what log record is being read and why, the requestor might end abnormally with a reason code of X'00D1032A'.

System programmer response

Probable user error. Most likely, the archive log data set that contained the requested RBA has been deleted from the BSDS by the change log inventory utility. Locate the output from an old print log map run, and identify the data set that contains the missing RBA. If the data set has not been reused, run the change log inventory utility to add this data set back into the inventory of log data sets. Restart the queue manager.

CSQJ114I
ERROR ON ARCHIVE DATA SET, OFFLOAD CONTINUING WITH ONLY ONE ARCHIVE DATA SET BEING GENERATED

Explanation

An error occurred while accessing one of the archive data sets being created by offload. Because the dual archive option is specified, offload is continuing with the other archive data set. For the RBA range being offloaded, there is only one copy of archive instead of the usual two copies.

System action

Offload produces a single archive data set.

System programmer response

A second copy of this archive log data set can be made, and the BSDSs can be updated with the change log inventory utility.

CSQJ115E
OFFLOAD FAILED, COULD NOT ALLOCATE AN ARCHIVE DATA SET

Explanation

Offload could not allocate an archive log data set. The offload was not performed. This message is preceded by message CSQJ103E or CSQJ073E.

Note: If you are using the dual archiving option, neither copy is made.

System action

Offload will be tried at a later time.

System programmer response

Review the error status information of message CSQJ103E or CSQJ073E. Correct the condition that caused the data set allocation error so that, on retry, the offload can take place.

CSQJ116E
ERROR ADDING ARCHIVE ENTRY TO BSDS

Explanation

Offload could not add an archive entry to the BSDS. The offload is considered incomplete. The active log data set is not marked as reusable for new log data. This message is preceded by message CSQJ107E, CSQJ108E, or CSQJ109E.

System action

Offload will be retried at a later time.

System programmer response

See the specific preceding message for action.

CSQJ117E
INITIALIZATION ERROR READING BSDS DSNAME= *dsname*, ERROR STATUS=*eee*

Explanation

An error occurred during initialization reading from the specified BSDS. Error Status contains the VSAM return and feedback codes. It is a 2-byte field with the first containing the hexadecimal return code and the second byte containing the hexadecimal feedback code. See the *DFSMS/MVS Macro Instructions for Data Sets* manual for a description of VSAM return and reason codes.

System action

Startup is terminated.

System programmer response

Determine the cause of the read error using the VSAM error status information provided. Restart the queue manager.

CSQJ118E
MACRO *xxx* FAILED IN LOG INITIALIZATION, RC=*ccc*

Explanation

Log initialization received a return code from the named macro.

System action

Startup is terminated.

System programmer response

Determine the problem from the documentation on the named macro and return code. Then take appropriate steps, and restart the queue manager. If you do not have information about the named macro, note the macro name and the return code and contact your IBM support center for help.

CSQJ119E
BOOTSTRAP ACCESS INITIALIZATION PROCESSING FAILED

Explanation

During queue manager initialization, the BSDS access function was unable to complete its initialization process. See Active log problems for information about dealing with problems on the BSDS or the log.

System action

Startup is terminated.

System programmer response

One or more error messages describing the specific error have preceded this message. See the specific messages for error analysis and the appropriate action to take.

CSQJ120E
DUAL BSDS DATA SETS HAVE UNEQUAL TIME STAMPS, SYSTEM BSDS1=*sys-bsds1*, BSDS2=*sys-bsds2*, UTILITY BSDS1=*uty-bsds1*, BSDS2=*uty-bsds2*

Explanation

When the queue manager was initialized, the time stamps of the dual BSDS did not agree. The time stamps from the system and from the change log inventory utility are shown for each BSDS. The time stamps have the format `date hh:mm:ss.th`.

System action

The queue manager attempts to re-synchronize the BSDS data sets to restore dual BSDS mode. If re-synchronization is successful, message CSQJ130I is issued and startup continues. Otherwise, startup is terminated.

System programmer response

If startup fails, run the print log map utility against each BSDS. From the output, determine which data set is obsolete, delete it, define a replacement for it, and copy the remaining BSDS to the replacement.

If output from the print log map utility for both data sets is similar, delete the data set with the oldest time stamp, and copy the data set with the most recent time stamp.

CSQJ121E
INITIALIZATION ERROR READING JFCB, DDNAME=*ddd*

Explanation

During queue manager initialization (if dual BSDS data sets are specified), the job file control block (JFCB) in z/OS is read to obtain the data set names associated with DDnames BSDS1 and BSDS2. This error is caused by a missing DD statement.

System action

Startup is terminated.

System programmer response

Ensure that a DD statement exists in the queue manager started task JCL procedure xxxxMSTR for DDname BSDS1. If dual BSDS data sets are used, ensure that a DD statement also exists in the queue manager started task JCL procedure xxxxMSTR for DDname BSDS2.

CSQJ122E
DUAL BSDS DATA SETS ARE OUT OF SYNCHRONIZATION

Explanation

During queue manager initialization, or when running a utility, the dual BSDSs were found to differ in content.

System action

The program or queue manager startup is terminated.

System programmer response

If the error occurred during queue manager initialization, run the print log map utility against each BSDS to determine which data set was last used as the first copy. Delete the second copy data set, define a replacement for the deleted data set, and copy the remaining BSDS to the replacement.

If the error occurred when running the BSDS conversion utility after the queue manager terminated abnormally, first attempt to restart the queue manager and shut it down cleanly before attempting to run the BSDS conversion utility again. If this does not solve the problem, run the print log map utility against each BSDS to determine which data set was last used as the first copy. Change the JCL used to invoke the BSDS conversion utility to specify this BSDS in the `SYSUT1 DD` statement, and remove the `SYSUT2 DD` statement, before submitting the job again.

CSQJ123E
CHANGE LOG INVENTORY FAILURE DETECTED

Explanation

During queue manager initialization, the BSDSs was found to have been incompletely processed by the change log inventory utility.

System action

Startup is terminated.

System programmer response

Run the print log map utility to determine what operation against the BSDS did not complete. Run the change log inventory utility against the BSDSs to allow any unfinished processing to be completed.

CSQJ124E
OFFLOAD OF ACTIVE LOG SUSPENDED FROM RBA *xxxxxx* TO RBA *xxxxxx* DUE TO I/O ERROR

Explanation

During offload, an unrecoverable input/output error was encountered on an active log data set. The data set experiencing the error is marked unusable, and no further logging is done to that data set.

System action

Active log data sets continue to be offloaded as they become full.

System programmer response

Recover the data manually from the data set, copy it to an archive data set, run the change log inventory utility to make the new archive data set available to the queue manager, and remove the error-prone active log data set.

CSQJ125E
ERROR COPYING BSDS, OFFLOAD CONTINUING WITHOUT THE BSDS COPY

Explanation

An error occurred while copying the BSDS data set during the offload process. The data set is not produced, and the volume containing the offloaded data set does not contain a BSDS for recovery use.

System action

The queue manager continues the offload process without producing a copy of the BSDS.

System programmer response

When archiving occurs, both a copy of the active log data set, and the BSDS at that time, are dumped. The BSDS is not critical because it will be copied again with the next archive log (the missing one simply means an extended restart time). However, the underlying data management problem (for example, not enough space allocated) should be resolved for subsequent BSDS offloads to occur.

CSQJ126E
BSDS ERROR FORCED SINGLE BSDS MODE

Explanation

An input/output error or a VSAM logical error occurred on a BSDS. This message is preceded by message CSQJ107E or CSQJ108E.

System action

IBM MQ enters single BSDS mode using the remaining BSDS.

System programmer response

Run an offline Access Method Services job to rename the error BSDS and define a new BSDS with the same name. Then enter the RECOVER BSDS command to reestablish dual BSDS mode.

CSQJ127I
SYSTEM TIME STAMP FOR BSDS=*date time*

Explanation

When the queue manager is initialized, the system time stamp for the BSDS is displayed. The time stamp is of the format `date hh:mm:ss.th`. This time stamp should be close to the last time at which this queue manager was stopped. If not, it might indicate a restart is being attempted with the wrong BSDS.

The time stamp will show as '****' if the BSDS has not been used before.

System action

Startup continues.

System programmer response

If the time displayed is not close to the time this queue manager was last stopped, and you cannot explain any time discrepancy, cancel the queue manager. From the queue manager started task JCL procedure xxxxMSTR, determine the data set names of the BSDSs and run the print log map utility. Check whether the active and archive log data sets all belong to this queue manager. If not, then change the started task JCL procedure xxxxMSTR for the queue manager to use the correct BSDSs.

CSQJ128E
LOG OFFLOAD TASK FAILED FOR ACTIVE LOG *dsname*

Explanation

The offload task ended abnormally while attempting to offload the RBA range in active log data set *dsname*.

System action

The offload task terminates and the archive data sets allocated to the offload task are deallocated and deleted. The status of the active log data sets involved in the unsuccessful offload processing remains set to 'not reusable'.

The log offload task will be reinitiated by one of several events. The most common are:

- All the available space in the current active log data set has been used (normal case)
- A CSQJ110E message is issued
- The queue manager address space is started, but data in the active log has not been archived
- An I/O error occurs on the active log, which will force the queue manager to truncate and offload the active log data set, and switch to a new active log data set

System programmer response

This message is the result of an offload error, and will be preceded by one or more IBM MQ messages (for example, CSQJ073E) and z/OS messages (for example, IEC030I, IEC031I, IEC032I). If the queue manager is operating with restricted active log resources (see message CSQJ110E), quiesce the system to restrict logging activity until the abnormal termination or the CSQJ110E condition can be resolved.

Investigate and correct the cause of the abnormal termination before the offload is attempted again by the queue manager.

CSQJ129E
END OF LOG RBA *eol-rba* COULD NOT BE FOUND IN ANY ACTIVE LOG DATA SET, HIGHEST RBA FOUND WAS *hi-rba*

Explanation

There was a request to find *eol-rba*, the log record that has been recorded in the BSDS as the highest RBA written. This RBA cannot be found in any active log data set. The highest RBA which could be found in any active data set was *hi-rba*.

System action

Startup processing is terminated.

System programmer response

Most likely, the active log data set containing the requested RBA has been deleted from the BSDS by the change log inventory utility. If the data set has not been reused, run the change log inventory utility to add this data set back into the BSDS. Restart the queue manager.

If the data set is not available, contact your IBM support center.

CSQJ130I
DUAL BSDS MODE RESTORED FROM BSDS*n*

Explanation

Dual BSDS mode was restored using BSDS copy *n*. This is the BSDS data set with the most recent system time stamp.

System action

Startup continues.

CSQJ131E
csect-name ERROR WRITING QUEUE MANAGER INFORMATION TO Db2

Explanation

During command processing, a failure occurred attempting to write queue manager information to Db2.

System action

Processing of the command is terminated.

System programmer response

Check the console for messages relating to the problem.

CSQJ132E
csect-name ERROR READING QUEUE MANAGER INFORMATION FROM Db2

Explanation

During command processing, a failure occurred attempting to read queue manager information from Db2.

System action

Processing of the command is terminated.

System programmer response

Check the console for messages relating to the problem.

CSQJ133E
LRSN *rrr* NOT IN ANY ACTIVE OR ARCHIVE LOG DATA SET, CONNECTION-ID=*xxxx* THREAD-XREF= *yyyyyy*, QMGR=*qmgr-name*

Explanation

There was a request to read the log record starting at this LRSN for the indicated queue manager (which might not be the issuer of the message). However, this log record cannot be found in any active or archive log data set. The connection ID and thread-xref identify the user or application that encountered the problem (this could be an internal IBM MQ task). See Active log problems for information about dealing with problems on the log.

System action

Depending upon what log record is being read and why, the requestor might end abnormally with a reason code of X'00D1032A'.

System programmer response

This is probably a user error. Most likely, the archive log data set that contained the requested RBA has been deleted from the BSDS by the change log inventory utility. Locate the output from an old print log map run, and identify the data set that contains the missing LRSN. If the data set has not been reused, run the change log inventory utility to add this data set back into the inventory of log data sets. Restart the queue manager.

CSQJ134E
RBA *log-rba* NOT IN ANY ACTIVE OR ARCHIVE LOG DATA SET, CONNECTION-ID=*xxxx* THREAD-XREF=*yyyyyy*, QMGR=*qmgr-name*

Explanation

There was a request to read the log record starting at this RBA for the indicated queue manager. However, this log record cannot be found in any active or archive log data set. The connection ID and thread-xref identify the user or application that encountered the problem (this could be an internal IBM MQ task). See Active log problems for information about dealing with problems on the log.

System action

Depending upon what log record is being read and why, the requestor might end abnormally with a reason code of X'00D1032A'.

System programmer response

This problem can occur for the following reasons:

1. The entry with the log range in the BSDS has been deleted from the BSDS
2. The entry with the log range is in BSDS, but the archive log data set has been deleted. When an archive log is created, the CSQ6ARVP parameter ARCRETN is used to specify when the data set can be deleted. When this date has passed MVS deletes the data set, so if you are trying to use this data set after this date, the data set cannot be found.

See BSDS problems for further information.

CSQJ136I
UNABLE TO ALLOCATE TAPE UNIT FOR CONNECTION-ID=*xxxx* CORRELATION-ID=*yyyyyy*, *m* ALLOCATED *n* ALLOWED

Explanation

An attempt to allocate a tape unit for the indicated connection ID failed. The current maximum tape unit specified is *n*, but only *m* are physically available.

System action

The process for the connection ID and correlation ID is held until either an allocated tape unit becomes free or more tape units are varied online and made available to the archive read task. This situation rectifies itself over time as currently allocated tape units become available.

CSQJ139I
LOG OFFLOAD TASK ENDED

Explanation

Processing of the active log offload ended.

System action

This message is written to the z/OS console.

CSQJ140I
Data set *dsname* successfully added to active log copy *n*

Explanation

A DEFINE LOG command has dynamically added a new log data set, *dsn*, and added it to either the LOGCOPY1 or LOGCOPY2 ring of active log data sets, as indicated by *n*.

The new active log data set is eligible to be used when the current active log data set fills and logging switches to the next active log data set in the ring.

Information about the data set is stored in the BSDS and will persist over a restart of the queue manager.

CSQJ141E
Error adding new active log data set *dsname*

Explanation

A DEFINE LOG command failed to add a new log data set. Further information about the failure is given in the preceding messages.

System programmer response

Investigate and correct the cause of the failure, then enter the command again.

CSQJ142I
Data set *dsname* has been used previously

Explanation

IBM MQ checks that a data set being added by a DEFINE LOG command has not been previously used as a log data set, as this might be an indication of operator error. The requested data set *dsname* was found to have been previously so used.

System action

The data set is closed and deallocated. Dynamic addition of a new active log data set fails.

System programmer response

Ensure that the data set being added as an active log data set is newly allocated, or has been formatted with the active log preformat utility, CSQJUFMT.

CSQJ143I
BSDS active log data set record is full

Explanation

The maximum number of active log data sets is fixed. No further entries can be inserted in the BSDS after the maximum has been reached.

System action

Dynamic addition of a new active log data set fails.

CSQJ144I
Active log data set allocation error

Explanation

It was not possible for IBM MQ to dynamically allocate the requested data set (named in the following CSQJ141E message) for use as a new active log data set.

System action

Dynamic addition of a new active log data set fails.

System programmer response

Ensure that the data set being added as a new active log data set is a VSAM linear data set with SHAREOPTIONS(2 3) and that it is not in use by any other jobs.

CSQJ150E
LOG CAPTURE EXIT ABEND, EXIT DEACTIVATED

Explanation

An abnormal program interrupt was detected while executing in the installation-supplied log capture exit code (that is entry point CSQJW117 in load module CSQJL004). As a result of this, the log capture exit will no longer be active; log data will no longer be available for exit capture/processing.

This message can only occur when an *installation-supplied* log capture exit (entry CSQJW117) is active for this queue manager.

System action

The log capture exit (entry point CSQJW117) is terminated. No further calls will be attempted for this queue manager. A full dump is provided for diagnostic purposes.

System programmer response

Determine the cause of the CSQJL004 load module (CSQJW117 entry point) abend and take corrective action.

Note: A correctly-functioning copy of load module CSQJL004/entry CSQJW117 must be available to start the queue manager. If the problem that caused this error cannot be corrected, ensure that the default CSQJW117 entry (load module CSQJL004 - supplied with IBM MQ) is available during the next queue manager start.

CSQJ151I

csect-name ERROR READING RBA *rrr*, CONNECTION-ID=*xxxx* CORRELATION-ID=*yyyyyy* REASON CODE=*ccc*

Explanation

The queue manager could not successfully complete the read of the indicated RBA due to reason code *ccc*. The user or application that encountered the error is identified by the connection and correlation IDs. Messages that have the same connection ID and correlation ID relate to the same application. Correlation IDs beginning with '0nn', where nn is a number from 01 to 28, identify system agents.

System action

The queue manager attempts to recover from the error.

System programmer response

If the queue manager was able to recover from the error and successfully complete the application, no further action is required. If the application abnormally terminated or the queue manager could not recover successfully, this message is followed by one or more messages. Refer to the information in this message and the subsequent messages to determine the appropriate corrective action. .

CSQJ152I

csect-name ERROR BUILDING ARCHIVE LOG VOLUME REPORT, CONNECTION-ID=*xxxx* CORRELATION-ID=*yyyyyy* REASON CODE=*ccc*

Explanation

An error occurred while attempting to create the archive log volume report. An RBA range could not be successfully mapped into one or more archive data sets due to reason code *ccc*. The user or application that encountered the error is identified by the connection and correlation IDs. This message might be preceded by one or more related error messages. Messages that have the same connection ID and correlation ID relate to the same application. Correlation IDs beginning with '0nn', where nn is a number from 01 to 28, identify system agents.

This failure could be caused by one or more missing archive log data sets, or a system error (for example, an I/O error reading the BSDS).

System action

The archive log volume report (see message CSQJ330I) is not produced. In addition, no premounting of tapes is possible.

The user or application continues processing. The physical read process for the user or application continues until the job completes normally or terminates abnormally. The job can terminate abnormally if the error is encountered again when the data set is physically required for the read process.

System programmer response

If the user or application completes successfully, no further action is necessary. If the user or application does not complete successfully, refer to the messages related to the actual failure to determine the appropriate corrective action.

CSQJ153I

csect-name ERROR READING LRSN *rrr*, CONNECTION-ID=*xxxx* CORRELATION-ID=*yyyyyy* REASON CODE=*ccc*, QMGR=*qmgr-name*

Explanation

The queue manager could not successfully complete the read of the indicated LRSN for the indicated queue manager (which might not be the issuer of the message) due to reason code *ccc*. The user or application that encountered the error is identified by the connection and correlation IDs. Messages that have the same connection ID and correlation ID relate to the same application. Correlation IDs beginning with '0nn', where nn is a number from 01 to 28, identify system agents.

System action

The queue manager attempts to recover from the error.

System programmer response

If the queue manager was able to recover from the error and successfully complete the application, no further action is required. If the application abnormally terminated or the queue manager could not recover successfully, this message is followed by one or more messages. Refer to the information in this message and the subsequent messages to determine the appropriate corrective action..

CSQJ154I

csect-name ERROR READING RBA *rrr*, CONNECTION-ID=*xxxx* CORRELATION-ID=*yyyyyy* REASON CODE=*ccc*, QMGR=*qmgr-name*

Explanation

The queue manager could not successfully complete the read of the indicated RBA for the indicated queue manager due to reason code *ccc*. The user or application that encountered the error is identified by the connection ID and correlation ID. Messages that have the same connection ID and correlation ID relate to the same application. Correlation IDs beginning with '0nn', where nn is a number from 01 to 28, identify system agents.

System action

The queue manager attempts to recover from the error.

System programmer response

If the queue manager was able to recover from the error and successfully complete the application, no further action is required. If the application abnormally terminated or the queue manager could not recover successfully, this message is followed by one or more messages. Refer to the information in this message and the subsequent messages to determine the appropriate corrective action.

CSQJ160I

LONG-RUNNING UOW FOUND, URID=*urid* CONNECTION NAME=*name*

Explanation

During log switch processing an uncommitted unit of recovery, spanning more than two active log switches, has been encountered. The unit of recovery identifier *urid* together with the connection name *name* identify the associated thread.

System action

Processing continues.

System programmer response

Consult with the application programmer to determine if there is a problem with the unit of recovery, and to ensure that the application commits work frequently enough. Uncommitted units of recovery can lead to difficulties later.

CSQJ161I
UOW UNRESOLVED AFTER *n* OFFLOADS, URID=*urid* CONNECTION NAME=*name*

Explanation

During log switch processing, an uncommitted unit of recovery was encountered that now has activity spanning several log data sets. The unit of recovery identifier *urid* together with the connection name *name* identify the associated thread.

System action

Processing continues.

System programmer response

Consult with the application programmer to determine if there is a problem with the unit of recovery, and to ensure that the application commits work frequently enough. Uncommitted units of recovery can lead to difficulties later.

CSQJ163E
COPY(2) specified but TWOACTV(NO)

Explanation

A DEFINE LOG command specified the COPY(2) parameter but the dual logging parameter (TWOACTV=YES) was not specified in CSQ6LOGP at queue manager startup.

System action

Dynamic addition of the new active log data set fails.

System programmer response

Either specify COPY(1) on the DEFINE LOG command or configure the queue manager to use dual logging.

CSQJ164I
csect-name Log archiving delayed, all available offload tasks in use

Explanation

The offload of one or more active logs has been delayed because all available offload tasks are in use.

A maximum of 31 offload tasks can concurrently write new archive log data sets. The number of offload tasks can be tuned using the MAXCNOFF parameter, which is set using either CSQ6LOGP or the SET LOG command. MAXCNOFF is provided to tailor the offloading of IBM MQ logs to match system constraints, such as the number of available tape units.

System action

Processing continues. The offload will complete when an offload task becomes available. Message CSQJ168I will be issued when the offload of active logs is no longer being delayed.

System programmer response

This is most likely a transient situation as a result of IBM MQ suddenly being able to archive a large number of full active logs, for example after problems with archiving have been resolved.

In other circumstances, review the MAXCNOFF parameter setting.

Consider increasing the active log capacity to match the active and archive log rates. The DEFINE LOG command can be used to provide additional active log capacity.

CSQJ165I

zHyperWrite bypassed for active log data set *dsname*

Explanation

The system parameter **ZHYWRITE** has been enabled, but zHyperWrite has not been used for some write requests to the active log data set identified by *dsname*.

System action

Processing continues. If Basic Metro Mirror (PPRC) has been configured for the active log volumes then traditional PPRC is used.

Using traditional PPRC means that the queue manager does not benefit from performance gains that are available when using zHyperWrite.

System programmer response

Review the configuration for the active log volumes and the zHyperWrite feature. The most likely reason for bypassing zHyperWrite is that the HyperSwap relationships have not been set up correctly.

CSQJ166E

PPRC configuration is inconsistent for active log copy *n*

Explanation

The data sets for each copy of the active log should be consistently configured for Basic Metro Mirror (PPRC).

This means that, either all of the data sets that comprise an active log copy should be configured on PPRC-enabled volumes, or none of the data sets should be configured on PPRC-enabled volumes.

The queue manager has detected an inconsistency in the PPRC configuration for the data sets that comprise log copy *n*..

System action

Processing continues, but an inconsistent logging rate might be observed due to the inconsistent configuration of the active log volumes. There is also a risk of losing data on the remote site, because not all of the active log volumes have a remote copy.

System programmer response

Review the configuration for the active log volumes.

CSQJ167E

zHyperWrite enabled but no active logs have PPRC configured

Explanation

The system parameter **ZHYWRITE** has been enabled, but the queue manager cannot exploit zHyperWrite because no active log copy resides on a PPRC-enabled volume.

System action

Processing continues without the Metro Mirror (PPRC) function.

System programmer response

Review the configuration for the active log volumes and the zHyperWrite feature.

CSQJ168I

csect-name Log archiving is no longer delayed

Explanation

The offload of active logs is no longer being delayed by a shortage of available offload tasks.

System action

Processing continues.

CSQJ200I

csect-name UTILITY PROCESSING COMPLETED SUCCESSFULLY

Explanation

The utility completed successfully.

CSQJ201I

csect-name UTILITY PROCESSING WAS UNSUCCESSFUL

Explanation

The utility was unable to complete processing successfully.

System action

The current utility is terminated.

System programmer response

Review other messages produced by the utility to determine the appropriate action to be taken.

CSQJ202E

INSUFFICIENT STORAGE AVAILABLE TO CONTINUE

Explanation

A request for storage was unsuccessful because no more storage is available.

System action

The current utility is terminated.

System programmer response

Rerun the utility after increasing the storage available.

CSQJ203E

oper OPERATION IS INVALID

Explanation

The user entered a utility control statement operation (*oper*) that is invalid.

System action

The current utility is terminated.

System programmer response

Correct the control statement, and rerun the utility.

CSQJ204E
xxxx PARAMETER IS INVALID

Explanation

The user specified a utility control statement parameter (*xxxx*) that is invalid.

System action

The current utility is terminated.

System programmer response

Correct the control statement, and rerun the utility.

CSQJ205E
xxxx PARAMETER HAS NO ARGUMENT

Explanation

xxxx contains the name of a parameter that requires an argument.

System action

The current utility is terminated.

System programmer response

Specify an argument for the identified parameter and then rerun the utility.

CSQJ206E
xxxx PARAMETER REQUIRES NO ARGUMENT

Explanation

xxxx contains the name of the parameter that has been incorrectly followed by an = symbol.

System action

The current utility is terminated.

System programmer response

Correct the control statement, and rerun the utility.

CSQJ207E
PARAMETERS INCONSISTENT WITH SPECIFIED OPERATION

Explanation

The user has specified utility control statement parameters that are inconsistent with the specified utility operation.

System action

The current utility is terminated.

System programmer response

Correct the control statement, and rerun the utility.

CSQJ211E
UNEXPECTED END OF DATA ON SYSIN DATA SET

Explanation

Additional control statements were expected, but could not be found.

System action

The current utility is terminated.

System programmer response

Correct the control statements, and rerun the utility.

CSQJ212E
ERROR RETURNED FROM BSDS READ, RPLERRCD= *yy*, DDNAME=*ddd*

Explanation

A VSAM GET was issued that resulted in a nonzero return code. *yy* contains the error code returned by VSAM. *ddd* contains the DDname of the BSDS encountering the error.

System action

The current utility is terminated.

System programmer response

The action taken is dictated by the reason code. See z/OS DFSMS Macro Instructions for Data Sets for information about the reason code in RPLERRCD. The BSDS might have to be recovered by use of a backup copy.

CSQJ213E
ERROR RETURNED FROM BSDS WRITE, RPLERRCD= *yy*, DDNAME=*ddd*

Explanation

A VSAM PUT was issued that resulted in a nonzero return code. *yy* contains the error code returned by VSAM. *ddd* contains the DDname of the BSDS encountering the error.

System action

The current utility is terminated.

System programmer response

The action to be taken is dictated by the return code. See z/OS DFSMS Macro Instructions for Data Sets for information about the reason code in RPLERRCD. The BSDS might have to be recovered by use of a backup copy.

If this error occurs when running the BSDS conversion utility (CSQJUCNV), and RPLERRCD indicates that the reason was an attempt to store a record with a duplicate key, ensure that the output BSDS is empty before running the utility.

CSQJ214E
SPECIFIED DSNAME ALREADY EXISTS IN BSDS, DDNAME=*ddd*

Explanation

You attempted a NEWLOG operation with a data set name that already exists in the BSDS. An entry is never made in a BSDS if the specified DSNAME currently exists in either the active or archive records of that BSDS. *ddd* contains the DDname of the subject BSDS.

System action

The current utility is terminated.

System programmer response

Either correct the control statement and rerun the utility, or delete the existing DSNAME from the BSDS and rerun the utility.

CSQJ215I
modname timestamps formatted with no local correction

Explanation

The parameter TIME(RAW) was specified on the invocation of utility *modname*. Where possible, timestamps formatted as date and time in the output will have no local timezone, or leapsecond adjustment performed so will be the UTC time of the event on the source system.

This mode of processing is most useful when the log, or BSDS being formatted has been produced on a remote system in a different timezone, or in a different daylight saving regime.

System action

Processing continues.

System programmer response

Either correct the control statement and rerun the utility, or delete the existing DSNAME from the BSDS and rerun the utility.

CSQJ216E
BSDS ACTIVE LOG DATA SET RECORD IS FULL, DDNAME=*ddd*

Explanation

The maximum number of active log data sets is fixed. No further entries can be inserted in the BSDS after the maximum has been reached. *ddd* contains the DDname of the subject BSDS.

System action

The current utility is terminated.

System programmer response

Run the print log map utility to determine the current status of the BSDS. Subsequent actions can then be formulated, depending upon the status of the BSDS.

CSQJ217E
SPECIFIED DSNAME DOES NOT EXIST IN BSDS, DDNAME=*ddd*

Explanation

The DELETE operation specifies a DSNAME that cannot be found in the BSDS. *ddd* contains the DDname of the subject BSDS.

System action

The current utility is terminated.

System programmer response

Correct the control statement, and rerun the utility.

CSQJ218E
SPECIFIED VOLUME DOES NOT EXIST IN BSDS, DDNAME=*ddd*

Explanation

The DELETE operation specifies a COPY1VOL or COPY2VOL argument that cannot be found in the BSDS. *ddd* contains the DDname of the subject BSDS.

System action

The current utility is terminated.

System programmer response

Correct the control statement, and rerun the utility.

CSQJ219E
OPEN ERROR, DDNAME=*ddd*

Explanation

An error occurred when *csect-name* tried to open a data set named *ddd*.

This error can be caused by a number of different conditions. The most probable conditions are:

1. The DDname of the SYSPRINT, SYSIN, or SYSUT1 data set was not specified in the user's job control language (JCL)
2. The queue manager is currently active
3. The BSDS has been allocated by another job with a disposition (DISP) that conflicts with the DISP specified in the user's JCL
4. The data set associated with *ddd* is already open, possibly due to an earlier error
5. The user is not authorized to access the data set associated with *ddd*
6. Insufficient storage is available to perform the OPEN operation
7. The catalog indicates that the data set associated with *ddd* has an invalid physical record size

System action

The current utility is terminated.

System programmer response

The user's action depends on the condition that caused the OPEN error. The following is a list of appropriate actions corresponding to the conditions listed in the explanation:

1. Provide the missing data definition (DD) statements, and then rerun the utility. See the section Preparing your program to run for further information.
2. Wait until the queue manager is inactive before running the utility again because the log utility cannot run while it is active.
3. Correct the disposition conflict and then rerun the utility.
4. Submit an Access Method Services (IDCAMS) VERIFY job against the data set associated with *ddd*. Rerun the log utility job.
5. In the case of an authorization problem, a separate message is usually generated from the authorization facility (RACF, for example). Investigate the authorization messages and obtain the proper authorization before running the utility again.
6. Insufficient storage is usually accompanied by a separate error from z/OS. Increase the storage available and rerun the utility.
7. Reallocate the data set with a suitable physical record size.

CSQJ220E
BSDS IN CREATE MODE. NO DATA TO MAP, DDNAME=*ddd*

Explanation

A utility found the BSDS to be in create mode, so it cannot contain data to map. *ddd* contains the DDname of the data set.

System action

The current utility is terminated.

System programmer response

Correct the JCL so that a non-null data set can be processed.

CSQJ221I
PREVIOUS ERROR CAUSED *oper* OPERATION TO BE BYPASSED

Explanation

Errors were encountered errors during utility processing. These errors subsequently caused *oper* to be bypassed.

This message is a warning only and is displayed after messages that specify the error or errors that occurred. Note that the error or errors might not be associated with the current *oper* operation; rather, under log utility processing, a significant error in any operation causes the control statements for this and any subsequent operations to be checked for syntax only. BSDS updates do not occur for any operation specified in this message.

System action

The log utility continues to process. However, for this and all subsequent operations, the BSDS is not updated and the utility only checks the syntax of the control statements.

System programmer response

Consult the previous messages and correct any errors that caused this message to be generated. Resubmit the log utility job for all operations that have been bypassed.

CSQJ222E
INVALID SPECIFICATION OF *xxxx* PARAMETER ARGUMENT

Explanation

You specified the parameter *xxxx*. This parameter is not valid for the argument.

System action

The current utility is terminated.

System programmer response

Correct the parameter argument on the control statement, and rerun the utility.

CSQJ223E
xxxx PARAMETER ARGUMENT EXCEEDS MAXIMUM ALLOWABLE LENGTH

Explanation

xxxx specifies the name of the parameter with an argument value that exceeded the maximum length allowed.

System action

The current utility is terminated.

System programmer response

Correct the parameter argument on the control statement, and rerun the utility.

CSQJ224E
xxxx PARAMETER APPEARS TOO OFTEN

Explanation

xxxx gives the name of the parameter that you have specified more than once on the same control statement.

System action

The current utility is terminated.

System programmer response

Remove the redundant parameter, and rerun the utility.

CSQJ225I
oper OPERATION SUCCESSFULLY COMPLETED

Explanation

The *oper* specified in the message identifies the name of the change log inventory utility operation that has been successfully completed.

CSQJ226E
SPECIFIED VOLUME ALREADY EXISTS IN BSDS, DDNAME=*ddd*

Explanation

The specified volume currently exists in the archive log records of the BSDS. *ddd* specifies the DDname of the subject BSDS.

System action

The current utility is terminated.

System programmer response

Either correct the parameter argument on the control statement, or delete the specified volume and rerun the utility.

CSQJ227E
NO SPACE IN BSDS FOR ADDITIONAL ARCHIVE ENTRIES, DDNAME=*ddd*

Explanation

The maximum number of archive volumes has been exceeded, and no more space is available for volume entries in the copy specified.

System action

The current utility is terminated.

System programmer response

Delete some of the archive entries in the specified copy number, and rerun the utility.

CSQJ228E
csect-name LOG DEALLOCATION ERROR DSNAME=*dsname*, ERROR STATUS=*eeeeiiii*, SMS REASON CODE=*ssssssss*

Explanation

An error occurred when trying to dynamically deallocate the data set. Error status is the error reason code returned by z/OS dynamic allocation.

System action

Processing continues.

System programmer response

The error status portion of this message contains a 2-byte error code (*eeee*, S99ERROR) followed by the 2-byte information code (*iiii*, S99INFO) from the SVC99 request block. If the S99ERROR code indicates an SMS allocation error ('97xx'), then *ssssssss* contains additional SMS reason code information obtained from S99ERSN. See the *MVS Authorized Assembler Services Guide* manual for a description of these codes.

CSQJ230E
LOG OFFLOAD INITIALIZATION PROCESSING FAILED

Explanation

During queue manager initialization, the offload function was unable to complete its initialization process.

System action

Startup is terminated.

System programmer response

One or more error messages describing the specific error preceded this message. See the specific messages for error analysis and the appropriate actions to take.

CSQJ231E
LOG COMMAND INITIALIZATION PROCESSING FAILED

Explanation

During queue manager initialization, the command function was unable to complete its initialization process.

System action

Startup is terminated.

System programmer response

One or more error messages describing the specific error preceded this message. See the specific messages for error analysis and the appropriate action to take.

CSQJ232E
OUTPUT DATA SET CONTROL INITIALIZATION PROCESSING FAILED

Explanation

During queue manager initialization, the output data set control function was unable to complete its initialization process.

System action

Startup is terminated.

System programmer response

One or more error messages describing the specific error preceded this message. See the specific message for error analysis and the appropriate action to take.

CSQJ233E
ARCHIVE LOG READ INITIALIZATION PROCESSING FAILED

Explanation

During queue manager initialization, the archive log read function was unable to complete its initialization process.

System action

Startup is terminated.

System programmer response

One or more error messages describing the specific error preceded this message. See the specific messages for error analysis and the appropriate action to take.

CSQJ234E
ARCHIVE LOG COMMAND QUIESCE INITIALIZATION PROCESSING FAILED

Explanation

During queue manager initialization, the quiesce function which supports the ARCHIVE LOG MODE(QUIESCE) command processing was unable to complete its initialization process.

System action

Startup is terminated.

System programmer response

One or more error messages describing the specific error preceded this message. See the specific messages for error analysis and the appropriate action to take.

CSQJ235E
OUTPUT BUFFER WRITER INITIALIZATION PROCESSING FAILED

Explanation

During queue manager initialization, the output buffer writer function was unable to complete its initialization process.

System action

Startup is terminated.

System programmer response

One or more error messages describing the specific error preceded this message. See the specific messages for error analysis and the appropriate action to take.

CSQJ236E
BOOTSTRAP ACCESS TERMINATION PROCESSING FAILED

Explanation

During queue manager termination, the BSDS access function was unable to complete its termination process.

System action

Termination processing continues.

System programmer response

One or more error messages describing the specific error preceded this message. See the specific messages for error analysis and the appropriate action to take.

CSQJ238E
LOG OFFLOAD TERMINATION PROCESSING FAILED

Explanation

During queue manager termination, the offload function was unable to complete its termination process.

System action

Termination processing continues.

System programmer response

One or more error messages describing the specific error preceded this message. See the specific messages for error analysis and the appropriate action to take.

CSQJ239E
LOG COMMAND TERMINATION PROCESSING FAILED

Explanation

During queue manager termination, the command function was unable to complete its termination process.

System action

Termination processing continues.

System programmer response

One or more error messages describing the specific error preceded this message. See the specific messages for error analysis and the appropriate action to take.

CSQJ240E
OUTPUT DATA SET CONTROL TERMINATION PROCESSING FAILED

Explanation

During queue manager termination, the output data set control function was unable to complete its termination process.

System action

Termination processing continues.

System programmer response

One or more error messages describing the specific error preceded this message. See the specific messages for error analysis and the appropriate action to take.

CSQJ241E
ARCHIVE LOG READ TERMINATION PROCESSING FAILED

Explanation

During queue manager termination, the archive log read function was unable to complete its termination process.

System action

Termination processing continues.

System programmer response

One or more error messages describing the specific error preceded this message. See the specific messages for error analysis and the appropriate action to take.

CSQJ242E
ARCHIVE LOG COMMAND QUIESCE TERMINATION PROCESSING FAILED

Explanation

During queue manager termination, the quiesce function which supports the ARCHIVE LOG MODE(QUIESCE) command processing was unable to complete its termination process.

System action

Termination processing continues.

System programmer response

One or more error messages describing the specific error preceded this message. See the specific messages for error analysis and the appropriate action to take.

CSQJ243E
OUTPUT BUFFER WRITER TERMINATION PROCESSING FAILED

Explanation

During queue manager termination, the output buffer writer function was unable to complete its termination process.

System action

Termination processing continues.

System programmer response

One or more error messages describing the specific error preceded this message. See the specific messages for error analysis and the appropriate action to take.

CSQJ244E
MACRO *xxx* FAILED IN LOG TERMINATION, RC=*ccc*

Explanation

During termination, there was a return code from the named macro that indicated an error.

System action

Termination processing continues.

System programmer response

If the problem persists, contact your IBM support center for assistance.

CSQJ245D
RESTART CONTROL INDICATES TRUNCATION AT RBA *rrr*. REPLY Y TO CONTINUE, N TO CANCEL

Explanation

The conditional restart control record in use indicates that the log should be truncated at the specified RBA.

System action

If 'Y', queue manager startup continues. If 'N', startup is terminated.

System programmer response

Run the change log inventory utility (CSQJU003) to modify the conditional restart record.

CSQJ246D
RESTART CONTROL INDICATES COLD START AT RBA *rrr*. REPLY Y TO CONTINUE, N TO CANCEL

Explanation

The conditional restart control record in use indicates that the queue manager is to be restarted and that logging is to begin at the specified RBA.

System action

If 'Y', queue manager startup continues. If 'N', startup is terminated.

System programmer response

Run the change log inventory utility (CSQJU003) to modify the conditional restart record.

CSQJ247E
csect-name I/O ERROR PROCESSING BSDS ARCHIVE LOG RECORD, RC=*rc* REASON=*reason*

Explanation

An input/output error occurred while processing a BSDS record. *rc* indicates the return code received from the input/output operation. *reason* indicates the reason code received from the operation.

Return code 4 indicates that IBM MQ detected a problem. Return code 8 indicates a VSAM error.

System action

Startup is terminated.

System programmer response

For a return code of 4, if the problem persists, contact your IBM support centre for assistance. For a return code of 8, run an offline Access Method Services job to determine the cause of the VSAM error.

CSQJ250I
csect-name DATA SET *dsname* HAS SHAREOPTIONS LESS THAN (2 3) - CF STRUCTURE RECOVERY NOT POSSIBLE

Explanation

An active log data set was detected with share options that do not permit CF structure recovery in a queue-sharing group environment. All active log data sets must be have SHAREOPTIONS(2 3) at least to allow CF structure recovery.

This can occur when the queue manager's own log data sets are checked during startup, or when a RECOVER CFSTRUCT command is issued that requires to access another queue manager's log data sets.

System action

If this is a result of a RECOVER CFSTRUCT command, the command is terminated. Otherwise, startup continues, but CF structure recovery will not be possible.

System programmer response

If you want CF structure recovery, use the Access Method Services ALTER function to correct the SHAREOPTIONS for the data set; for example

```
ALTER dsname.DATA SHAREOPTIONS(2 3)
```

Then restart the queue manager that owns the data set.

CSQJ295D
RESTART CONTROL INDICATES TRUNCATION AT LRSN *rrr*. REPLY Y TO CONTINUE, N TO CANCEL

Explanation

The conditional restart control record in use indicates that the log should be truncated at the specified LRSN.

System action

If 'Y', queue manager startup continues. If 'N', startup is terminated.

System programmer response

Run the change log inventory utility (CSQJU003) to modify the conditional restart record.

CSQJ301E
csect-name ERROR USING ONLINE BOOTSTRAP DATA SET (ACTION CODE *a*)

Explanation

During command processing for the RECOVER BSDS command or the ARCHIVE LOG command, an error occurred while performing an operation on the BSDS. The type of operation is specified by the code *a*:

1 Unable to OPEN the BSDS

2 Unable to read a required record from the BSDS

3 Unable to write a required record to the BSDS

4 The contents of the stable BSDS was successfully copied to the replacement BSDS; however, the queue manager was unable to successfully restore dual BSDS operation

System action

If this message was received during processing of the RECOVER BSDS command, then the queue manager will continue in single BSDS mode. If this message was received during processing of the ARCHIVE LOG command, the archive log history record in the BSDS will not be updated to reflect the occurrence of an ARCHIVE LOG command; logging and the offload processing will continue.

System programmer response

If this message was received during processing of the RECOVER BSDS command, recovery action must be performed on the BSDS before re-entering the command. If this message was received during processing of the ARCHIVE LOG command, no action is necessary.

CSQJ302E
ALLOCATION ERROR ON REPLACEMENT BSDS DSNAME=*dsname* ERROR STATUS=*eee*

Explanation

The RECOVER BSDS command encountered an error while trying to allocate the specified data set dynamically. DSNAME is the data set name. Error Status is the error code and information code returned by z/OS dynamic allocation.

System action

Processing of the command is terminated. The queue manager continues in single BSDS mode.

System programmer response

Determine the cause of the error from the error status contained in the message, and correct the condition. Then re-enter the RECOVER BSDS command.

The error status portion of this message contains the 2-byte error code (S99ERROR) followed by the 2-byte information code (S99INFO) from the SVC request block. See the *MVS Authorized Assembler Services Guide* manual for a description of these codes.

CSQJ303E
WRITE ERROR ON REPLACEMENT BSDS DSNAME=*dsname* ERROR STATUS=*eee*

Explanation

The RECOVER BSDS command encountered an error while attempting to write to the specified BSDS. Error status contains the VSAM return and feedback codes. It is a 2-byte field with the first containing the hexadecimal return code and the second containing the hexadecimal feedback code.

System action

Processing of the command is terminated. The queue manager continues in single BSDS mode.

System programmer response

Run an offline Access Method Services job to delete or rename the replacement BSDS and define a new BSDS with the same name. Re-enter the RECOVER BSDS command to reestablish dual BSDS mode.

CSQJ304E
ERROR CLOSING REPLACEMENT BSDS DSNAME=*dsname* ERROR STATUS=*eee*

Explanation

The RECOVER BSDS command encountered an error while attempting to close the specified BSDS. Error Status contains the VSAM return and feedback codes. It is a 2-byte field with the first containing the hexadecimal return code and the second containing the hexadecimal feedback code.

System action

Processing of the command is terminated. The queue manager continues in single BSDS mode.

System programmer response

Run an offline Access Method Services job to delete or rename the replacement BSDS and define a new BSDS with the same name. Re-enter the RECOVER BSDS command to reestablish dual BSDS mode.

CSQJ305E
REPLACEMENT BSDS NOT EMPTY DSNAME=*dsname*

Explanation

The RECOVER BSDS command was issued, but the replacement BSDS was not empty; that is, it contained data.

System action

Processing of the command is terminated. The queue manager continues in single BSDS mode.

System programmer response

Run an offline Access Method Services job to delete or rename the error BSDS and define a new BSDS with the same name. Re-enter the RECOVER BSDS command to reestablish dual BSDS mode.

CSQJ306I
DUAL BSDS MODE ALREADY ESTABLISHED

Explanation

The RECOVER BSDS command was issued, but the queue manager was already in dual BSDS mode.

System action

The command is ignored.

CSQJ307I
LOG INITIALIZED IN SINGLE BSDS MODE

Explanation

The RECOVER BSDS command was issued, but the queue manager was initialized in single BSDS mode.

System action

Processing of the command is terminated. The queue manager continues in single BSDS mode.

CSQJ308I
LOG NOT OFFLOADED FOR ARCHIVE LOG COMMAND, ARCHIVING IS OFF

Explanation

The ARCHIVE LOG command was issued, but archiving is off (that is, OFFLOAD is set to 'NO' in the CSQ6LOGP system parameters).

System action

The current active log data set is not offloaded. However, it is truncated and logging continues using the next active log data set.

CSQJ309I
QUIESCING FOR ARCHIVE LOG COMMAND WITH WAIT(YES) STARTED FOR MAXIMUM OF *xxx* SECONDS

Explanation

An ARCHIVE LOG command with the MODE(QUIESCE) and WAIT(YES) options has been accepted by the queue manager. The quiesce processing has commenced.

WAIT(YES) means that quiesce processing will be synchronous to the user; that is, the user can enter additional commands, but they will not be processed until the quiesce processing has ended.

System action

The queue manager attempts to stop all updates to IBM MQ resources within the time period specified in the message. Users and jobs using the queue manager are allowed to reach a point of consistency (commit point) before being blocked from further update activity. Users and jobs are suspended until they are released by the queue manager following the initiation of the offload processing. If the queue manager can effectively block all users from performing updates before the maximum specified time, the offload is initiated immediately, and normal processing is resumed.

This message will be followed by message CSQJ311I or CSQJ317I.

CSQJ310I
QUIESCING FOR ARCHIVE LOG COMMAND WITH WAIT(NO) STARTED FOR MAXIMUM OF *xxx* SECONDS

Explanation

An ARCHIVE LOG command with the MODE(QUIESCE) and WAIT(NO) by the queue manager. The quiesce processing has commenced.

WAIT(NO) means that quiesce processing will be asynchronous to the user; that is, control will be returned to the invoker as soon as the quiesce task has been started. Thus, the queue manager will accept, and process, any new commands while the quiesce task is running.

System action

The queue manager attempts to stop all updates to IBM MQ resources within the time period specified in the message. Users and jobs using the queue manager are allowed to reach a point of consistency (commit point) before being blocked from further update activity. Users and jobs are suspended until they are released by the queue manager following the initiation of the offload processing. If the queue manager can effectively block all users from performing updates before the maximum specified time, the offload is initiated immediately, and normal processing is resumed.

This message will be followed by message CSQJ311I or CSQJ317I.

CSQJ311I
csect-name LOG ARCHIVE (OFFLOAD) TASK INITIATED

Explanation

A user-initiated ARCHIVE LOG command has been accepted by the queue manager. A task to archive (offload) the active log data set has been started.

System action

The current active log data sets will be truncated and switched to the next available active log data sets. The task has been started will archive the active log data sets asynchronously, allowing the queue manager to continue processing.

This message will be followed by the CSQJ312I message if the MODE(QUIESCE) option was used with the ARCHIVE LOG command.

CSQJ312I
ARCHIVE LOG QUIESCE ENDED. UPDATE ACTIVITY IS NOW RESUMED

Explanation

An ARCHIVE LOG command with the MODE(QUIESCE) option was processed by the queue manager. As part of the MODE(QUIESCE) processing, an attempt was made to stop all new update activity against IBM MQ resources. This message signals the end of the quiesce processing, and the resumption of normal activity for all users and jobs which were blocked during the quiesce period.

This message will follow the CSQJ311I message or CSQJ317I message.

System action

The queue manager has now resumed all normal activity for all users and jobs which were blocked during the quiesce period.

CSQJ314E
'*kwd1*' requires '*kwd2*' to be specified too

Explanation

A command was entered that specified the *kwd1* keyword. However, use of this keyword requires that the *kwd2* keyword is also used.

System action

Processing for the command is terminated.

CSQJ315I
STOP QMGR MODE(FORCE) IN PROGRESS

Explanation

An attempt was made to issue an ARCHIVE LOG command when a STOP QMGR MODE(FORCE) command was already in progress.

System action

Command processing will terminate for the ARCHIVE LOG command. The STOP QMGR MODE(FORCE) processing will continue.

CSQJ316I
SYSTEM QUIESCE ALREADY IN PROGRESS

Explanation

An ARCHIVE LOG command with the MODE(QUIESCE) option or a SUSPEND QMGR LOG command was issued when a system quiesce was already in progress. The system quiesce could be the result of processing by another ARCHIVE LOG MODE(QUIESCE) command, or by a STOP QMGR MODE(QUIESCE) command.

System action

Command processing will terminate. The system quiesce currently in progress will continue.

CSQJ317I
QUIESCE PERIOD EXPIRED WITH *nn* OUTSTANDING URS AT *time*. ARCHIVE LOG PROCESSING TERMINATED

Explanation

An ARCHIVE LOG MODE(QUIESCE) command was processed by the queue manager. However, the queue manager was not able to quiesce all update activity in the user-specified quiesce time interval.

System action

This message is for information only. The queue manager determined that *nn* units of recovery did not reach a point of consistency during the quiesce period, and therefore could not be stopped from continuing their associated update processing.

Consequently, the ARCHIVE LOG processing will be terminated. The current active log data sets will not be truncated, and will not be switched to the next available active log data sets. The log archive (offload) task will not be created. All jobs and users suspended during the quiesce will be resumed, and normal update activity against IBM MQ resources will be commenced.

This message will be followed by the CSQJ312I message.

System programmer response

You must decide whether the outstanding (non-quiesced) units of recovery represent significant work.

Each user on the system has a unit of recovery if they are modifying IBM MQ resources. Units of recovery are also created by the queue manager itself for internal processing. Because the purpose of the MODE(QUIESCE) option is to have all units of recovery reach a point of consistency (commit point) before the active log data set is truncated and offloaded, determine all outstanding non-queued jobs and users by using DISPLAY THREAD and the z/OS command DISPLAY ACTIVE,LIST.

Note that units of recovery might be outstanding due to lock contention between a user or job that holds a resource (and has reached a point of consistency), and a user or job that wants a lock (and therefore cannot reach a point of consistency).

Before resubmitting the ARCHIVE LOG command with the MODE(QUIESCE) option, either:

- Wait until the threads have been deallocated
- Wait until the queue manager is less busy
- Force the offending threads to terminate
- Use the TIME option to override and extend the maximum quiesce time period specified in the system parameters
- If having all units of recovery reach a point of consistency in the active log is no longer critical, issue the ARCHIVE LOG command without the MODE(QUIESCE) option

Note: If you decide to use the ARCHIVE LOG command without the MODE(QUIESCE) option, the active log data sets will be truncated without regard to quiescing activity on the queue manager. If the resulting archive log data set is used for recovery, it is possible that some units of recovery might be found to be in-flight, in-backout, in-commit, or in-doubt during queue manager initialization.

If expiration of the quiesce period before all units of recovery reach a consistent point is a problem, you might have to adjust the QUIESCE value in the CSQ6ARVP system parameters. For more information, see Using CSQ6ARVP.

CSQJ318I
ARCHIVE LOG COMMAND ALREADY IN PROGRESS

Explanation

An attempt was made to issue an ARCHIVE LOG command when another ARCHIVE LOG command was already in progress.

System action

Command processing will terminate. The ARCHIVE LOG command currently in progress will continue.

CSQJ319I
csect-name CURRENT ACTIVE LOG DATA SET IS THE LAST AVAILABLE ACTIVE LOG DATA SET. ARCHIVE LOG PROCESSING WILL BE TERMINATED

Explanation

The ARCHIVE LOG command was rejected because the current active log is the last available active log data set. To process the command when these conditions exist would cause the queue manager to exhaust its available active log resources and immediately halt processing.

System action

Processing for the command is terminated.

If the situation is not corrected, the queue manager will issue the CSQJ110E message (if it has not already done so) when the available active log data space reaches critically low levels. Ultimately,

message CSQJ111A will be issued when the available active log data space is exhausted, and processing will stop until active log space is made available.

System programmer response

To clear this condition, steps must be taken to complete other waiting offload tasks. Once another active log is made available (re-usable) by completing the offload process for it, the command processing for the current active log can proceed.

Perform a display request to determine the outstanding requests related to the log offload process. Take the necessary action to satisfy any requests, and permit offload to continue.

If offload does not complete normally, or cannot be initiated, either correct the problem that is causing the offload problem, or consider whether there are sufficient active log data sets. If necessary, additional log data sets can be added dynamically using the DEFINE LOG command.

Possible causes for the shortage of active log data space are:

- Excessive logging. For example, there is a lot of persistent message activity.
- Delayed or slow offloading. For example, failure to mount archive volumes, incorrect replies to offload messages, or slow device speeds.
- Excessive use of the ARCHIVE LOG command. Each invocation of the command causes the queue manager to switch to a new active log data set. Excessive use could consume the available active log data space if the resulting offloads were not processed in a timely manner.
- Offloads unsuccessful.
- Insufficient active log space.

CSQJ320E

csect-name UNABLE TO PROCESS LOG TRUNCATION REQUEST DUE TO INTERNAL ERROR. (ERROR DATA=*ddd*)

Explanation

While processing an ARCHIVE LOG command, an internal request was made of the log buffer output routine to force-write the log buffers and to truncate and switch the active log to the next available active log data sets.

System action

Processing for the command is terminated.

System programmer response

This is an internal error detected by the queue manager. The error might be caused by an unrelated error in the log buffer writer component (CSQJWxxx), by a STOP QMGR MODE(FORCE) command, or by abnormal termination. See any messages that precede this message.

CSQJ321E

UNABLE TO CONTINUE ARCHIVE LOG QUIESCE DUE TO INTERNAL ERROR. ARCHIVE LOG PROCESSING TERMINATED

Explanation

An ARCHIVE LOG command with the MODE(QUIESCE) option was processed by the queue manager. As part of the MODE(QUIESCE) processing, an attempt was made to stop all new update activity against IBM MQ resources. During the processing, an internal error occurred.

System action

The ARCHIVE LOG MODE(QUIESCE) processing is terminated. This message will be followed by message CSQJ312I after all users and jobs quiesced by the MODE(QUIESCE) processing are resumed.

System programmer response

This error is an internal error detected by the queue manager. Retry the ARCHIVE LOG MODE(QUIESCE) command. If the error persists, the active log data sets can be switched using the ARCHIVE LOG command without the MODE(QUIESCE) option.

CSQJ322I
DISPLAY parm-type report ...

Explanation

This message is part of the response to the DISPLAY and SET *parm-type* commands (where *parm-type* is SYSTEM, LOG, or ARCHIVE). It provides information about the corresponding system parameters. For example:

```
Parameter  Initial value  SET value
LOGLOAD      500000       400000
CMDUSER      CSQOPR
EXCLMSG      X500,X501,X528,X208,
             X519,X599
End of parm-type report
```

where:

LOGLOAD
was set in CSQ6SYSP and changed using the SET SYSTEM LOGLOAD command.

CMDUSER
was set in CSQ6SYSP and has not been changed.

EXCLMSG
was set to the default in CSQ6SYSP, and has been changed using the SET SYSTEM EXCMSG command.

System action

Processing continues.

CSQJ325I
ARCHIVE tape unit report ...

Explanation

This message is part of the response to the DISPLAY and SET ARCHIVE commands. It provides information about tape units used for archive logging, as follows:

```
Addr St CorrelID VolSer DSName  addr st correlid volser dsname|  End of tape unit report
```

where:

addr The physical address of a tape unit allocated to read the archive log.

st The status of the tape unit:

B Busy, actively processing an archive log data set.

P Premount, active and allocated for premounting.

A Available, inactive and waiting for work.

***** Unknown.

correlid
The correlation ID associated with the user of the tape being processed; '*****' if there is no current user.

volser The volume serial number of the tape that is mounted.

dsname
The data set name on the tape volume that is being processed or was last processed.

If no tape units are allocated, the list is replaced by:

```
No tape archive reading activity
```

System action

Processing continues.

CSQJ330I
ARCHIVE LOG VOLUMES required for connection-ID *xxxx*, correlation-ID *yyyyyy*:

Explanation

This message lists the names of the archive log volumes needed by the indicated correlation ID for the given connection ID. The archive log volumes are listed with a maximum of six on each line. It is generated automatically by the archive read process at the first archive log tape mount for that correlation ID. The connection ID is an identifier representing the connection name used to establish the thread; the correlation ID is an identifier associated with a specified thread, such as a job name.

A volume name prefixed with an '*' signifies that the data on the archive log volume is also mapped by an active log data set. As such, the volume might not be required for the read process, because the data is read from the active log if possible.

The following is an example of the output produced by message CSJ330I::

```
CSQJ330I: ARCHIVE LOG VOLUMES required for connection-ID xxxx,
 correlation-ID yyyyyy: volume1, volume2, volume3, volume4, volume5, volume6 End of ARCHIVE LOG VOLUMES report
```

System action

Processing continues.

CSQJ334E
Parameter value is unacceptable for ' *kwd*'

Explanation

The parameter value specified is not an acceptable value for the named keyword, or is incompatible with values set for other keywords.

System action

Processing for the command is terminated.

CSQJ335E
Invalid command syntax

Explanation

No keywords or an unacceptable combination of keywords was specified on a command.

System action

Processing for the command is terminated.

CSQJ337I
parm-type parameters set

Explanation

The SET command completed successfully, setting system parameter values for the indicated *parm-type* (SYSTEM, LOG, or ARCHIVE).

CSQJ364I
IMS Bridge facility suspended for XCFGNAME=*gname* XCFMNAME=*mname*

Explanation

This is issued as part of the response to a DISPLAY SYSTEM command if the IBM MQ-IMS Bridge facility to the partner IMS system identified by *gname* and *mname* is suspended.

System programmer response

Use the RESUME QMGR FACILITY(IMSBRIDGE) command when ready to resume the IBM MQ-IMS Bridge.

CSQJ365I
Db2 connection suspended

Explanation

This is issued as part of the response to a DISPLAY SYSTEM command if the connection to Db2 is suspended.

System programmer response

Use the RESUME QMGR FACILITY(Db2) command when ready to resume the connection to Db2.

CSQJ366I
Logging already suspended

Explanation

A SUSPEND QMGR LOG command was issued, but logging was already suspended by a previous command.

System action

The command is ignored.

CSQJ367I
Queue manager stopping

Explanation

A SUSPEND QMGR LOG command was issued, but the queue manager is stopping.

System action

The command is ignored.

CSQJ368I
Logging not suspended

Explanation

A RESUME QMGR LOG command was issued, but logging was not suspended.

System action

The command is ignored.

CSQJ369E
csect-name Failure while suspending logging

Explanation

A SUSPEND QMGR LOG command was issued, but it terminated abnormally.

System action

The command is ignored, and logging is not suspended.

System programmer response

Verify the command entry, and reissue the command. If it fails again, collect the items listed in the Problem Determination section, and contact your IBM support center.

CSQJ370I
LOG status report ...

Explanation

This message is part of the response to the DISPLAY and SET LOG commands. It provides information about the status of the log data sets, as follows:

```
 Copy %Full PPRC DSName   1    k  p  dsname   2    k   p dsname  Restarted at date time using RBA=sss
Latest RBA=rrr  Offload task is xxx  Full logs to offload - m of n
```

where:

1, 2 Information for the current active log copy 1 and copy 2 data sets.

k The percentage of the active log data set that has been used.

p Indicates whether zHyperWrite is used for this log data set.

NO The zHyperWrite feature is not used for this log data set.

YES The zHyperWrite feature is used for this log data set.

dsname
The data set name of the active log data set. If the copy is not currently active, this is shown as `Inactive`.

date time
The time that the queue manager was started.

sss The RBA from which logging began when the queue manager was started.

rrr The RBA of the most recently written log record. If logging is suspended, this line is replaced by

```
Logging suspended at RBA=rrr
```

xxx The status of the offload task, which can be:

BUSY, allocating archive data set
This could indicate that a tape mount request is pending.

BUSY, copying BSDS
Copying the BSDS data set.

BUSY, copying active log
Copying the active log data set.

BUSY Other processing.

AVAILABLE
Waiting for work.

m, n The number of full active log data sets that have not yet been archived, and the total number of active log data sets.

System action

Processing continues.

CSQJ372I
Logging suspended for *qmgr-name* at RBA=*rrr*

Explanation

This is issued in response to a SUSPEND QMGR LOG command if it completed successfully.

It is also issued in response to other commands if logging is suspended, indicating that the command cannot be processed while logging is suspended.

System action

All log update activity is suspended for the queue manager named. *rrr* is the RBA of the last log record written.

For commands other than SUSPEND QMGR LOG, the command is ignored.

System programmer response

Use the RESUME QMGR LOG command when ready to resume logging.

CSQJ373I
Logging resumed for *qmgr-name*

Explanation

The RESUME QMGR LOG command completed successfully.

System action

All log update activity is resumed for the queue manager named.

CSQJ401E
RECORD NOT FOUND - *rrr*

Explanation

An attempt was made to read the *rrrr* record from the BSDS. In doing so, the read routine (CSQJU01B) could not find the record.

This is not necessarily an error; for example, if you have never used CSQJU003 CRESTART, there will not be any CRCR records, so you will get this message from CSQJU004 for the RESTART CONTROL records.

System action

Utility processing continues.

CSQJ404E
kwd NOT ALLOWED FOR *oper* OPERATION

Explanation

An invalid keyword was used during the *oper* operation.

System action

The current utility processing is terminated.

CSQJ405E
KEYWORDS *kwd1* AND *kwd2* CANNOT BOTH BE SPECIFIED

Explanation

Keywords *kwd1* and *kwd2* cannot appear on the same control statement.

System action

The current utility processing is terminated.

CSQJ406E
EITHER KEYWORD *kwd1* OR *kwd2* MUST BE SPECIFIED

Explanation

A required keyword was not used on the control statement. Use either *kwd1* or *kwd2* with that control statement type.

System action

The current utility processing is terminated.

CSQJ407E
NO VALID CHECKPOINT RBA FOUND

Explanation

After completing its search through the resource manager status table and the checkpoint queue, no valid checkpoint RBA was found within the specified range.

System action

The current utility processing is terminated.

System programmer response

The last 100 checkpoints are recorded in the BSDS, including the log STARTRBA and log ENDRBA of the checkpoint range. The utility attempts to locate a valid checkpoint in the range. In this case the utility was unsuccessful in finding a valid checkpoint.

Use the Print Log Map Utility (CSQJU004) to determine the valid RBA ranges, and rerun the job with a suitable RBA specification.

CSQJ408I
CHECKPOINT RBA FOUND, RBA=*rba*, TIME=*date time*

Explanation

After completing its search through the resource manager status table and the checkpoint queue, *rba* was the most recent checkpoint RBA in the specified range, and *date time* was the time of the checkpoint.

System action

Utility processing continues.

CSQJ409E
I/O ERROR DURING READ PROCESSING OF RECORD - *yyy*

Explanation

An input/output error occurred during a READ of a record. *yyy* specifies the record in question.

System action

The current utility processing is terminated. This message is accompanied by message CSQJ212E.

System programmer response

Determine the cause of the error based on the error status information provided in message CSQJ212E.

CSQJ410E
I/O ERROR DURING WRITE PROCESSING OF RECORD - *yyy*

Explanation

An input/output error occurred during a WRITE of a record. *yyy* specifies the record in question.

System action

The current utility processing is terminated. This message is accompanied by message CSQJ213E.

System programmer response

Determine the cause of the error based upon the error status information provided in message CSQJ213E.

CSQJ411I
CRESTART CREATE FOR CRCRID=*yyyy*, DDNAME=*ddd*

Explanation

A CRESTART CREATE request has just completed. *yyyy* is the restart control record hexadecimal identifier and *ddd* is the BSDS data set (SYSUT1 or SYSUT2) associated with the request.

System action

Current® utility processing continues.

System programmer response

Note the record identifier for future reference.

CSQJ412E
RESTART CONTROL RECORD NOT FOUND IN BSDS

Explanation

A CRESTART CANCEL keyword was specified but the conditional restart control record does not exist in the BSDS data set.

System action

Current utility processing is terminated.

System programmer response

None necessary, if CANCEL was the intended action. Otherwise, correct the control statement and rerun the utility.

CSQJ413E
INVALID LOG RANGE SCOPE OR CHECKPOINT SPECIFIED

Explanation

The values specified through the STARTRBA and ENDRBA keywords are invalid.

System action

Current utility processing is terminated.

System programmer response

Ensure that the log range values are correct and correspond to the other log range values either specified or defaulted. The STARTRBA must be less than or equal to the ENDRBA.

CSQJ414I
COLD START WILL RESULT FROM THIS RESTART CONTROL RECORD. FORWARD AND BACKOUT SET TO NO

Explanation

STARTRBA and ENDRBA are equal. A cold start will result if this restart control record is used during restart. No forward or backout processing will be performed.

System action

CRESTART processing continues.

System programmer response

No additional actions are needed if a cold start of the queue manager is required. If a cold start is not required, reissue the CRESTART and either CANCEL the current restart control record, or CREATE a new restart control record.

CSQJ415E
ENDRBA=*rba* IS INVALID, MUST BE A MULTIPLE OF 4K

Explanation

The specified ENDRBA at *rba* is not a multiple of 4K.

System action

CRESTART processing is terminated.

System programmer response

Correct the ENDRBA value on the CRESTART statement and rerun the utility.

CSQJ416I
WARNING - BSDS UTILITY TIME STAMP MISMATCH DETECTED. PROCESSING CONTINUES

Explanation

As a result of a change log inventory update, it was discovered that the SYSUT1 BSDS and SYSUT2 BSDS time stamps are unequal. Their inequality indicates the possibility of a BSDS mismatch.

System action

Current utility processing continues.

System programmer response

Run the print log map utility against the SYSUT1 BSDS and SYSUT2 BSDS. Determine if each BSDS is current. If each BSDS is current, this warning can be ignored. If either BSDS is not current, delete the obsolete data set and define a replacement data set, then copy the current BSDS into the replacement data set.

CSQJ417E
REQUIRED *xxxx* PARAMETER FOR *oper* OPERATION IS MISSING

Explanation

Required parameter *xxxx* for a log utility operation was missing from the log utility control statement. The attempted operation is *oper*.

System action

The log utility *oper* operation does not perform its function. All subsequent log utility control statements are processed. A nonzero return code is issued by the utility.

System programmer response

Add the missing parameter to the control statements associated with the specified operation and rerun the utility.

CSQJ418I
NOTREUSABLE ACTIVE LOG DELETED FROM THE BSDS LOG INVENTORY, STARTRBA=*sss* ENDRBA=*ttt*

Explanation

The data set name specified on the DSNAME parameter of the change log inventory utility DELETE statement was a NOTREUSABLE active log.

System action

The change log inventory utility processing continues. It will terminate with a return code of 4.

System programmer response

No additional actions are required if you want to delete a NOTREUSABLE active log. If not, re-create the deleted log by using the NEWLOG statement with the RBA values specified in the warning message.

CSQJ421I
CRESTART CANCEL FOR CRCRID=*yyyy*, DDNAME=*ddd*

Explanation

A CRESTART CANCEL request has just completed. *yyyy* is the restart control record hexadecimal identifier and *ddd* is the BSDS data set (SYSUT1 or SYSUT2) associated with the request.

System action

Current utility processing continues.

System programmer response

Note the record identifier for future reference.

CSQJ425E
INVALID VALUE OR FORMAT FOR *xxxx* PARAMETER (YYYYDDDHHMMSST)

Explanation

The *xxxx* parameter contains an incorrect value or incorrect format for the date and time.

System action

The current utility is terminated.

System programmer response

Correct the control statement and rerun the utility.

CSQJ426E
ENDTIME VALUE CANNOT BE LESS THAN STARTIME VALUE

Explanation

The STARTIME and ENDTIME parameters specify a time range. Therefore, the ENDTIME value must be equal to or greater than STARTIME value.

System action

The current utility is terminated.

System programmer response

Correct the control statement and rerun the utility.

CSQJ427I
CHECKPOINT RECORD ADDED TO QUEUE

Explanation

The checkpoint record specified has been added to the checkpoint queue in the BSDS.

System action

Processing continues.

CSQJ428I
CHECKPOINT RECORD DELETED FROM QUEUE, STARTRBA= *ssss* ENDRBA=*ttt*

Explanation

The checkpoint record specified has been deleted from the checkpoint queue in the BSDS. *sss* and *ttt* was the RBA range indicated in the deleted checkpoint record.

System action

Processing continues.

CSQJ429E
RBA RANGE CONFLICTS WITH EXISTING CHECKPOINT RECORD RBA RANGE

Explanation

The specified RBA range for the new checkpoint record either exists, or overlaps an existing RBA range in the checkpoint queue in the BSDS.

System action

The current utility is terminated.

System programmer response

Run the print log map utility against the SYSUT1 BSDS and SYSUT2 BSDS. Determine the correct RBA range, correct the STARTRBA and ENDRBA parameters, and rerun the utility.

CSQJ430E
SPECIFIED ENTRY CANNOT BE ADDED WITHOUT OVERLAYING EXISTING LOWEST ENTRY

Explanation

The specified RBA range for the new checkpoint record is less than the lowest existing entry. The checkpoint queue in the BSDS is currently full and cannot add the new entry without overlaying the lowest entry.

System action

The current utility is terminated.

System programmer response

Run the print log map utility against the SYSUT1 BSDS and SYSUT2 BSDS. Determine the lowest existing entry, either change the STARTRBA and ENDRBA parameters or delete the lowest existing entry and add a new low checkpoint entry, and rerun the utility.

CSQJ431E
STARTRBA SPECIFIED CANNOT BE FOUND IN CHECKPOINT QUEUE

Explanation

The specified STARTRBA could not be located in the checkpoint queue in the BSDS.

System action

The current utility is terminated.

System programmer response

Run the print log map utility against the SYSUT1 BSDS and SYSUT2 BSDS. Determine the correct STARTRBA value, correct the STARTRBA parameter, and rerun the utility.

CSQJ432E
kwd VALUE MUST END WITH ' *xxx*'

Explanation

The value specified for keyword *kwd* is not valid. It must end with '*xxx*'.

System action

The current utility is terminated.

System programmer response

Correct the control statement and rerun the utility.

CSQJ440I
csect-name IBM MQ for z/OS version

Explanation

This message is issued as part of the header to reports issued by the utility programs.

CSQJ443I
csect-name CHANGE LOG INVENTORY UTILITY - *date time*

Explanation

This message is issued as a header to the report issued by the utility program.

CSQJ444I
csect-name PRINT LOG MAP UTILITY - *date time*

Explanation

This message is issued as a header to the report issued by the utility program.

CSQJ445I
csect-name BSDS CONVERSION UTILITY - *date time*

Explanation

This message is issued as a header to the report issued by the utility program.

CSQJ450E
csect-name VERSION *n* BSDS NOT SUPPORTED BY ALL QSG MEMBERS

Explanation

The BSDS conversion utility detected that at least one queue manager in the queue-sharing group does not support the version of BSDS that will be produced as a result of the conversion.

System action

The current utility is terminated with no action taken.

System programmer response

Migrate all queue managers in the queue-sharing group to a level that supports the new BSDS version and change the setting of OPMODE if necessary, then run the conversion utility again.

CSQJ451E
csect-name BSDS CI SIZE NOT CORRECT, DDNAME=*ddd*

Explanation

A data set provided to the BSDS conversion utility is unusable because the CI size is not correct. The CI size of the BSDS must be 4096. The variable *ddd* contains the DD name of the data set..

System action

The current utility is terminated with no action taken.

System programmer response

Ensure that the DD statement refers to a valid BSDS. If the DD name refers to an output data set, delete and redefine the output BSDS, then rerun the utility.

CSQJ452E
csect-name BSDS UTILITY TIMESTAMP MISMATCH DETECTED

Explanation

A mismatch was detected in the timestamps for the SYSUT1 and SYSUT2 BSDS copies during execution of the BSDS conversion utility. This mismatch indicates the possibility that the dual BSDSs are out of sync.

System action

The current utility is terminated with no action taken.

System programmer response

Run the print log map utility (CSQJU004) against each BSDS. From the output, determine which data set is obsolete, delete it, and define a replacement for it. Then copy the remaining data set into the replacement and try the utility again.

If output from the print log map utility for both data sets is similar, delete the data set with the oldest timestamp, then copy the data set with the most recent timestamp into the replacement.

CSQJ453E
csect-name INPUT BSDS NOT IN CORRECT FORMAT, DDNAME=*ddd*

Explanation

The BSDS conversion utility detected that the input BSDS was not in the correct format to be converted. The input BSDS must be in version 1 format. The variable *ddd* contains the DD name of the data set.

System action

The current utility is terminated with no action taken.

System programmer response

Run the print log map utility (CSQJU004) against the BSDS to determine its version. Ensure that the DD statement refers to an input BSDS in version 1 format, then rerun the utility if necessary.

CSQJ454E
csect-name UNRECOGNIZED BSDS RECORD, KEY=*key-value*

Explanation

During conversion of the BSDS, a record was found that is not a known format. The *key-value* is the VSAM KSDS key of the BSDS record that was not recognized.

System action

The current utility is terminated.

System programmer response

To determine the operation that inserted the record into the BSDS, use IDCAMS PRINT and specify this key value. If the record is not needed, delete it then rerun the BSDS conversion.

CSQJ455E
INVALID BSDS CONVERSION

Explanation

This message is issued when a utility, attempting to access the BSDS data set, encounters an invalid BSDS. An invalid BSDS is the result of a failure during a prior attempt to run the BSDS conversion utility.

System action

The current utility is terminated with no action taken.

System programmer response

The procedure for running the BSDS conversion utility involves renaming the original BSDS. Restore the BSDS to the original pre-conversion copy, by renaming the data sets, then try the conversion again.

CSQJ456E
xxxx PARAMETER ARGUMENT EXCEEDS MAXIMUM VALUE FOR BSDS VERSION *n*

Explanation

The *xxxx* parameter specifies the name of the parameter with a value that exceeds the maximum that can be specified for a BSDS in version *n* format.

System action

The current utility is terminated.

System programmer response

Correct the parameter argument on the control statement, then rerun the utility.

CSQJ491I
csect-name Log Data Set Preformatter Utility - *date time*

Explanation

This message is issued as a header to the report issued by the utility program.

CSQJ492I
Log data set name = *dsname*

Explanation

This identifies the name of the log data set to be preformatted.

CSQJ493I
Log data set is not VSAM

Explanation

The input log data set is not a VSAM data set.

System action

Utility processing is terminated.

System programmer response

Check that the SYSUT1 DD statement and the data set name is specified correctly. Use Access Method Services to define the data set as a VSAM linear data set.

CSQJ494E
VSAM OPEN failed, ACBERRFLG=*ee*

Explanation

Opening the log data set failed with the indicated ACB error code.

System action

Utility processing is terminated if the error code is 128 or more; otherwise processing continues.

System programmer response

See the *DFSMS/MVS Macro Instructions for Data Sets* for information about the VSAM error code.

CSQJ495E
VSAM PUT failed, RPLERREG=*ee* reason code=*reason*

Explanation

Writing the log data set failed with the indicated RPL error code and reason code.

System action

Utility processing is terminated.

System programmer response

See the *DFSMS/MVS Macro Instructions for Data Sets* for information about the VSAM error code and reason code.

CSQJ496I
Log preformat completed successfully, *n* records formatted

Explanation

The active log data set has been preformatted successfully.

System action

Utility processing is complete.

CSQJ497I
Log preformat terminated

Explanation

Preformatting the active log data set did not complete successfully.

System action

Utility processing is terminated.

System programmer response

See the preceding error messages for more information.

CSQJ498I
Log data set is not empty

Explanation

The input log data set is not an empty data set.

System action

Utility processing is terminated.

System programmer response

Check that the SYSUT1 DD statement and the data set name is specified correctly. Use Access Method Services to define the data set as a VSAM linear data set.

CSQJ499I

Log data set is larger than 4GB

Severity
0

Explanation

The log preformat utility, CSQJUFMT, detected that the VSAM data set to be formatted is greater than 4 GB in size.

System action

Processing continues. The entire data set will be pre-formatted, but IBM MQ for z/OS log data sets are restricted to a maximum of 4 GB. Any additional space in the data set is not used to hold log data.

System programmer response

Check that the data set name is specified correctly. Use Access Method Services to define the data set with a maximum size of 4 GB.

Message manager messages (CSQM...): z/OS

CSQM001E

csect-name MSTR user ID cannot invoke USS callable services

Severity

8

Explanation

The IBM MQ queue manager MSTR address space is running under a user ID that has not been configured with authority to execute callable Unix System Services (USS).

In RACF, the user ID requires an OMVS segment with a UID assigned.

System action

This message is issued and the process of Unix System Services calls, for reverse DNS host name lookup, are disabled in the MSTR address space.

System programmer response

Refer to Planning your z/OS UNIX or UNIX System Services environment, where queue manager MSTR and CHIN address spaces require user IDs with OMVS segments defined with a valid UID.

Correct the configuration of the queue manager MSTR address space user ID and restart the queue manager.

CSQM050I

csect-name Intra-group queuing agent starting, TCB=*tcb-name*

Severity

0

Explanation

The intra-group queuing (IGQ) agent was started during the initialization of a queue manager that is in a queue-sharing group. The agent uses TCB *tcb-name*.

The IGQ agent handles SYSTEM.QSG.TRANSMIT.QUEUE.

System action

Processing continues. The IGQ agent starts asynchronously.

CSQM051I

csect-name Intra-group queuing agent stopping

Severity

0

Explanation

The intra-group queuing (IGQ) agent is stopping because:

- the queue manager is stopping
- it has retried a failing request repeatedly without success
- it was unable to recover from an abnormal ending

System action

The IGQ agent stops.

System programmer response

If the queue manager is not stopping, investigate the cause of the error as reported in the preceding messages. To restart the IGQ agent, issue an ALTER QMGR command specifying IGQ(ENABLED).

CSQM052I

csect-name Shared channel recovery completed for *qmgr-name*, *n* channels found, *p* FIXSHARED, *r* recovered

Severity

0

Explanation

The queue manager successfully recovered some shared channels that were owned by queue manager *qmgr-name* in the queue-sharing group when it or its channel initiator terminated abnormally. This recovery process might occur when:

- another queue manager or its channel initiator terminates abnormally
- the channel initiator is started, for channels that were owned by other queue managers
- the channel initiator is started, for channels that were owned by itself

n channels were found that needed recovery, of which *p* were originally started as FIXSHARED. The number recovered, *r*, might be less than *n* (or even 0) because other active queue managers are also recovering the channels and because FIXSHARED channels cannot be recovered by another queue manager.

For more information about shared channel recovery, see Shared channels.

System action

Processing continues.

CSQM053E

csect-name Shared channel recovery terminated, DB2 not available

Severity

8

Explanation

Because Db2 is not available or no longer available, the queue manager was unable to recover some shared channels that were owned by a queue manager in the queue-sharing group when it or its channel initiator terminated abnormally. This recovery process might occur when:

- another queue manager or its channel initiator terminates abnormally
- the channel initiator is started, for channels that were owned by other queue managers
- the channel initiator is started, for channels that were owned by itself

System action

The recovery process is terminated; some channels might have been recovered, while others have not.

System programmer response

Use the preceding messages on the z/OS console to investigate why Db2 is not available, and resume the connection or restart Db2 if necessary. Any channels that were not recovered will be recovered when the recovery process next runs; alternatively, they can be restarted manually.

CSQM054E

csect-name Shared channel recovery terminated, error accessing DB2

Severity

8

Explanation

Because there was an error in accessing Db2, the queue manager was unable to recover some shared channels that were owned by a queue manager in the queue-sharing group when it or its channel initiator terminated abnormally. This recovery process might occur when:

- another queue manager or its channel initiator terminates abnormally
- the channel initiator is started, for channels that were owned by other queue managers
- the channel initiator is started, for channels that were owned by itself

System action

The recovery process is terminated; some channels might have been recovered, while others have not.

System programmer response

Resolve the error reported in the preceding messages. Any channels that were not recovered will be recovered when the recovery process next runs; alternatively, they can be restarted manually.

CSQM055E

csect-name Shared channel recovery terminated, error putting command, MQRC=*mqrc (mqrc-text)*

Severity

8

Explanation

Because there was an error putting a message on the system-command input queue, the queue manager was unable to recover some shared channels that were owned by a queue manager in the queue-sharing group when it or its channel initiator terminated abnormally. This recovery process might occur when:

- another queue manager or its channel initiator terminates abnormally
- the channel initiator is started, for channels that were owned by other queue managers
- the channel initiator is started, for channels that were owned by itself

System action

The recovery process is terminated; some channels might have been recovered, while others have not.

System programmer response

Refer to API completion and reason codes for information about *mqrc* (*mqrc-text*provides the MQRC in textual form), and resolve the error. Any channels that were not recovered will be recovered when the recovery process next runs; alternatively, they can be restarted manually.

CSQM056E

csect-name mqapi-call failed for queue *q-name*, MQRC=*mqrc (mqrc-text)*

Severity

8

Explanation

The indicated IBM MQ API call for the named queue, failed for the specified reason, which might be an IBM MQ reason code (MQRC_) or a signal completion code (MQEC_).

System action

If the queue is SYSTEM.ADMIN.CONFIG.EVENT or SYSTEM.ADMIN.COMMAND.EVENT, processing continues but events are not generated; message CSQM071E follows to show how

many event messages have not been generated since the problem first occurred. These messages are generated on the first occurrence of the problem, and at intervals thereafter while the problem persists.

Depending on the queue involved and the type of error, it might continue processing, try the request again at regular intervals until the error is corrected, or terminate.

System programmer response

See the API completion and reason codes for information about IBM MQ reason codes. For information about signal completion codes, see Signaling. Correct the problem with the queue, or use the ALTER QMGR command to disable the events.

CSQM057E

csect-name MQPUT of trigger message failed for queue *q-name*, MQRC=*mqrc (mqrc-text)*

Severity

8

Explanation

The queue manager could not deliver a trigger message to the indicated initiation queue for the specified IBM MQ reason code (MQRC_).

System action

The queue manager attempts to put the trigger message on to the dead-letter queue if one has been defined.

System programmer response

Refer to API completion and reason codes for information about IBM MQ reason codes, and what action to take to correct the problem with the initiation queue.

CSQM058E

csect-name Unable to start channel *channel-name*

Severity

8

Explanation

An attempt was made to start cluster channel *channel-name* because a message was placed on the SYSTEM.CLUSTER.TRANSMIT.QUEUE. If the channel could not be started because of an internal queuing error this message is preceded by CSQM056E. This message is also issued if the queue manager encounters a storage shortage.

System action

The message remains queued on the SYSTEM.CLUSTER.TRANSMIT.QUEUE queue and the original MQPUT completes successfully. If the cluster channel is not already running it is not automatically started.

System programmer response

If required, manually start the channel using the START CHANNEL command. Stopping and restarting the channel initiator or the queue manager, or placing another message on the transmission queue for this cluster destination triggers another START request.

If message CSQM056E is issued because of an internal queuing error, action might be needed to ensure that future start channel requests can be processed correctly.

If there is a lack of storage and the problem persists, you might need to increase the region size used by your queue manager, or you might need to reduce the number of jobs running in your system.

CSQM059E

csect-name Queue *q-name* has incorrect attributes

Severity

8

Explanation

The named queue, used by the intra-group queuing (IGQ) agent, has incorrect attributes. For example, SYSTEM.QSG.TRANSMIT.QUEUE must have attributes USAGE(XMITQ), INDXTYPE(CORRELID), QSGDISP(SHARED).

System action

The IGQ agent retries at regular intervals until the error is corrected.

System programmer response

Redefine the queue with the correct attributes.

CSQM060E

csect-name Cluster cache is full

Severity

8

Explanation

No more space is available in the cluster cache area.

System action

The application call that resulted in the need for more space will fail with MQRC_CLUSTER_RESOURCE_ERROR. Processing continues, and existing users of clustering will be unaffected unless their actions are such as to need more cluster cache space.

System programmer response

The problem may be temporary. If it persists, the queue manager must be restarted; this will cause more space to be allocated for the cluster cache area.

Consider changing the cluster cache type system parameter CLCACHE to dynamic, so that more space for the cache will be obtained automatically as required. (If you are using a cluster workload exit, ensure that it supports a dynamic cluster cache.) For information about the system parameters for the CSQ6SYSP macro, see Using CSQ6SYSP.

CSQM061E

csect-name Cluster workload exit *exit-name* does not support dynamic cache

Severity

8

Explanation

In response to the initialization call (using ExitReason MQXR_INIT), the cluster workload exit returned the value MQCLCT_STATIC in the ExitResponse2 field, indicating that it does not support a dynamic cluster cache.

System action

The cluster workload exit is suppressed.

System programmer response

Either change the cluster cache type system parameter CLCACHE to static, or rewrite the exit to be compatible with a dynamic cache. For information about the system parameters for the CSQ6SYSP macro, see Using CSQ6SYSP.

CSQM062I

csect-name INDXTYPE(*index-type*) not allowed for shared transmission queue *shared-xmitq*

Severity

4

Explanation

A shared transmission queue is a queue that is defined with both USAGE(XMITQ) and QSGDISP(SHARED). To support recovery of messages that are in-doubt after a channel failure, the index type (INDXTYPE) for shared transmission queues must be either NONE or MSGID.

System action

Processing continues.

System programmer response

Modify the INDXTYPE attribute for the shared transmission queue to NONE or MSGID.

CSQM063E

csect-name Specified dead-letter queue name is unacceptable

Severity

4

Explanation

The intra-group queuing (IGQ) agent has attempted to put a persistent message on the dead-letter queue that is defined to the queue manager. The dead-letter queue specified is either SYSTEM.QSG.TRANSMIT.QUEUE or there is no dead-letter queue name specified.

System action

The put of the message to the dead-letter queue does not take place, the get of the message from the SYSTEM.QSG.TRANSMIT.QUEUE is backed out and the intra-group queuing (IGQ) agent goes into retry.

System programmer response

Ensure the queue manager has a dead-letter queue defined which is neither blank nor SYSTEM.QSG.TRANSMIT.QUEUE. Examine the message to determine the reason for its placement on the dead-letter queue.

CSQM064I

csect-name Intra-group queuing agent put messages to dead-letter queue

Severity

4

Explanation

The intra-group queuing (IGQ) agent was unable to deliver some messages to the required destination queue, so has put them on the dead-letter queue.

System action

Processing continues.

System programmer response

Examine the contents of the dead-letter queue. Each message is contained in a structure that describes why the message was put to the queue, and to where it was originally addressed.

CSQM065E

csect-name mqapi-call failed, MQRC=*mqrc (mqrc-text)*

Severity

8

Explanation

The indicated MQ API call failed for the specified reason, which is an IBM MQ reason code *mqrc* (mqrc-text provides the MQRC in textual form).

System action

It is the intra-group queuing (IGQ) agent that issued the call; it was unable to commit or backout a batch of messages for the specified reason. Depending on the type of error, it may retry the request at regular intervals until the error is corrected, or terminate.

System programmer response

Refer to API completion and reason codes for information about MQ reason codes. Correct the problem if required.

CSQM067E

csect-name Intra-group queuing agent ended abnormally. Restarting

Severity

8

Explanation

The intra-group queuing (IGQ) agent has ended abnormally because a severe error occurred, as reported in the preceding messages.

System action

The IGQ agent attempts to restart a number of times. If it fails persistently, it terminates.

System programmer response

Investigate the reason for the abnormal termination, as reported in the preceding messages.

CSQM070E

csect-name Queue *q-name* available again, *n* events not generated

Severity

4

Explanation

An earlier problem with putting messages on the configuration or command event queue has been corrected. *n* is the number of event messages that have not been generated since the problem first occurred.

System action

Processing continues and event messages for that queue will be generated again.

System programmer response

If the queue is SYSTEM.ADMIN.CONFIG.EVENT, and complete configuration information is required, use the REFRESH QMGR TYPE(CONFIGEV) command to generate events to replace those that were not generated; specify the INCLINT parameter to cover the period when the problem was occurring.

If the queue is SYSTEM.ADMIN.COMMAND.EVENT, a limited number of the missed event messages may be recovered automatically, as reported by message CSQM072I.

CSQM071E

csect-name Queue *q-name* unavailable, *n* events not generated

Severity

8

Explanation

There was an error putting a message on the configuration or command event queue, as reported in the preceding CSQM056E message; *n* is the number of event messages that have not been generated since the problem first occurred.

System action

Processing continues but event messages for that queue are not generated. This message is issued on the first occurrence of the problem, and at intervals thereafter while the problem persists.

System programmer response

Correct the problem with the event queue, or use the ALTER QMGR command to set the CONFIGEV or CMDEV attribute to DISABLED if events are not required.

CSQM072I

csect-name Queue *q-name*, *n* events recovered

Severity

0

Explanation

An earlier problem with putting messages on the command event queue has been corrected. *n* event messages that were not generated have been automatically recovered and generated.

Only a limited number of the missed event messages can be recovered in this way. If *n* is less than the value reported in message CSQM070E, the remaining event messages are lost, and there is no way to recover them.

System action

Processing continues.

CSQM073I

csect-name Loading of durable subscribers started

Severity

0

Explanation

Information about the durable subscribers on a queue manager is stored on the SYSTEM.DURABLE.SUBSCRIBER.QUEUE queue. During the restart of the queue manager the durable subscriptions are remade on the queue manager.

System action

Processing continues.

CSQM074I

csect-name Loading of durable subscribers finished

Severity

0

Explanation

The queue manager has finished reloading all of the durable subscribers.

System action

Processing continues.

CSQM075I

csect-name Consolidation of durable subscribers started

Severity
0

Explanation

Information about the durable subscribers on a queue manager is stored on the SYSTEM.DURABLE.SUBSCRIBER.QUEUE queue. To aid in restart processing and to speed up the time it takes to reload all of the durable subscribers, these messages are consolidated into fewer messages.

System action

Processing continues.

CSQM076I

csect-name Consolidation of durable subscribers finished

Severity
0

Explanation

The queue manager has finished consolidating the messages on the SYSTEM.DURABLE.SUBSCRIBER.QUEUE queue. The processing might be restarted at a later stage if there is a change in the number of durable subscribers.

System action

Processing continues

CSQM077I

csect-name PUBLISH/SUBSCRIBE ENGINE HAS SHUTDOWN

Severity
0

Explanation

The publish/subscribe engine has been shutdown.

System action

The publish/subscribe engine has shutdown.

System programmer response

No action is required if the queue manager is stopping. If the publish/subscribe engine has shutdown because you have disabled it, updating the PSMODE queue manager attribute from the value DISABLED will restart it.

CSQM078E

csect-name Unable to create thread structures for connection-type *connection* from *jobname*, insufficient ACE storage

Severity
8

Explanation

jobname attempted to create a new connection to IBM MQ as the result of issuing the first IBM MQ API call on a new thread. The connection-type is likely to be RRSBATCH.

There was insufficient common storage available to build the control blocks to represent the connection and the connect attempt failed.

There might be a system wide ECSA shortage, or the storage available for creating new queue manager connections might be limited by the ACELIM system parameter.

This message can be seen for CICS and the channel initiator, as well as for RRS applications; for example, Db2 stored procedures and WebSphere Application Server.

System action

IBM MQ API request fails with return code MQRC_STORAGE_NOT_AVAILABLE 2071

Queue manager processing continues

CSQM079I

csect-name Policy access attempt rejected due to incompatible AMS version, jobname *jobname*

Severity
4

Explanation

An incompatible version of Advanced Message Security (AMS), identified by *jobname*, attempted to open the policy queue, SYSTEM.PROTECTION.POLICY.QUEUE.

System action

The request to open the policy queue is rejected.

System programmer response

Update the incompatible version of AMS so it does not attempt to connect to the queue manager. From Version 8.0, AMS is provided as an integrated feature of IBM MQ for z/OS. For information about how to configure AMS as an integrated feature, see Advanced Message Security for z/OS.

CSQM084I

csect-name COMMAND INHIBITED DURING RESTART/TERMINATION

Severity
8

Explanation

A command that will affect a recoverable object was requested either too early in queue manager startup, or too late in termination.

The usual reason for receiving this message is that some prohibited command was issued in the initialization input data set CSQINP1.

System action

Message CSQM085I is also issued and the command is ignored.

System programmer response

Wait until the queue manager is in a state where it is possible to reissue the prohibited commands. If appropriate, remove the command from CSQINP1, and place it in CSQINP2, to ensure that this problem does not recur.

CSQM085I

csect-name ABNORMAL COMPLETION

Severity

8

Explanation

This message is issued with message CSQM084I, and indicates that the command requested has not been actioned.

System action

The command is not actioned.

System programmer response

Wait until the queue manager is in a state where it is possible to use the prohibited commands.

CSQM086E

QUEUE MANAGER CREATE ERROR, CODE=*reason-code*, RESTART UNSUCCESSFUL

Severity

8

Explanation

During restart, the creation of the queue manager object has failed. The reason code is of the form '00D44xxx'.

System action

The queue manager fails to restart.

System programmer response

See "Message manager codes (X'D4')" on page 5093 for an explanation of the reason code, and what action to take. Reissue the START QMGR command to restart the queue manager. If the error persists note this reason code, and contact your IBM support center.

CSQM090E

csect-name FAILURE REASON CODE *reason-code*

Severity

8

Explanation

A command has failed. The reason code is of the form '00D44xxx'. This message is accompanied by one or more other more specific messages, which indicate the reason for the failure.

System action

The command is ignored.

System programmer response

See the explanations of the accompanying messages for more information. See "Message manager codes (X'D4')" on page 5093 for an explanation of the reason code, and what action to take. If the reason code is not one of those listed, make a note of it and contact your IBM support center.

CSQM091E

csect-name FAILURE MQRC=*mqrc (mqrc-text)*

Severity

8

Explanation

A command has failed. The reason code is an IBM MQ reason code. This message is accompanied by one or more other more specific messages, which indicate the reason for the failure.

System action

The command is ignored.

System programmer response

See the explanations of the accompanying messages for more information. Refer to API completion and reason codes for an explanation of *mqrc*, (*mqrc-text* provides the MQRC in textual form), and what action to take.

CSQM092I

csect-name keyword(*value*) VALUE INVALID OR OUT OF RANGE

Severity

8

Explanation

Either:

- A keyword was entered that takes a bounded numeric value but the value specified is outside the bounds.
- A keyword was entered that takes a pair of numeric values defining a range, but only one value is specified or the values are not in ascending order.

System action

The command is ignored.

System programmer response

Reissue the command with the parameter specified correctly. For more information about the command, see MQSC commands.

CSQM093I

csect-name keyword(value) NAME CONTAINS INVALID CHARACTERS

Severity

8

Explanation

A name was specified that contains one or more invalid characters. See MQSC commands for information about validation required for the name in question to correct this.

System action

The command is ignored.

System programmer response

Reissue the command with the correct name. For more information about the command, see MQSC commands.

CSQM094I

csect-name keyword(value) WAS NOT FOUND

Severity

8

Explanation

A command was issued that refers to an object that does not exist. That is, no object could be found with the specified name and type (and subtype, for queues and channels) and with any disposition in the queue-sharing group.

System action

The command is ignored.

System programmer response

Check that you specified the correct name for the object, and the correct subtype (for queues and channels). If a queue-sharing group is in use, check that Db2 is available and not suspended. Define the object if necessary.

Note:

1. If you are dealing with a queue or channel object, an object of the same name, but of a different subtype, might already exist.
2. Remember that the object might have recently been deleted by someone else, or from another queue manager in the queue-sharing group.

CSQM095I

csect-name keyword(value) existing-disposition ALREADY EXISTS

Severity

8

Explanation

A DEFINE command was issued, but an object of that type with the specified name already exists, although it might not necessarily have the same subtype, or the same disposition in the queue-sharing group. (You cannot have a locally-defined object and a local copy of a group object with the same name; for local queues, you cannot have a shared queue with the same name as a queue with any other disposition.) Where applicable, *existing-disposition* identifies the queue-sharing group disposition of the existing object.

System action

The command is ignored.

System programmer response

Reissue the command with another name or with the REPLACE option, or use the existing object, as appropriate.

CSQM096I

csect-name keyword(value) NAME HAS INVALID LENGTH

Severity

8

Explanation

A name was specified that is of an incorrect length.

System action

The command is ignored.

System programmer response

Reissue the command with a name of the correct length. For more information about the command, see MQSC commands.

CSQM097I

csect-name keyword(value) NAME CANNOT BE COMPLETELY BLANK

Severity
8

Explanation

A name was specified that is blank. This is not allowed.

System action

The command is ignored.

System programmer response

Reissue the command with a non-blank name. For more information about the command, see MQSC commands.

CSQM098I

csect-name keyword(value) FIELD TOO LONG

Severity
8

Explanation

Either a numeric or character parameter was specified but it is too long, or (if *value* is blank) a list of character parameters was specified with a total length that is too long.

System action

The command is ignored.

System programmer response

Reissue the command with the correct field length. For more information about the command, see MQSC commands.

CSQM099I

csect-name keyword(value) NAME IN USE AS A DIFFERENT TYPE

Severity
8

Explanation

An object was specified as one particular subtype, but it already exists as another subtype, although it might not necessarily have the same disposition in the queue-sharing group. (You cannot have a locally-defined object and a local copy of a group object with the same name; for local queues, you cannot have a shared queue with the same name as a queue with any other disposition.)

System action

The command is ignored.

System programmer response

Reissue the command with the correct name and subtype. For more information about the command, see MQSC commands.

CSQM100I

csect-name keyword(value) VALUE INVALID OR OUT OF RANGE

Severity
8

Explanation

A value is invalid or out of range. This could be because:

- A keyword was entered that takes a series of character values, but the value specified is not one of them.
- A keyword was entered that takes a series of character values, but the value specified is not valid for the particular subtype of object.
- A keyword was entered that takes a bounded numeric value, but the value specified is outside the bounds.
- A keyword was entered that takes a character or hexadecimal value, but the value specified is invalid for that keyword.

System action

The command is ignored.

System programmer response

Reissue the command with the parameter specified correctly. For more information about the command, see MQSC commands.

CSQM101I

csect-name keyword(value) IS CURRENTLY IN USE

Severity

8

Explanation

The object specified is in use. This could be because:

- It is open through the API.
- A trigger message is presently being written to it.
- It is in the process of being deleted.
- When it is a storage class, there is a queue defined as using the storage class, and there are messages currently on the queue.
- When it is a CF structure, there is a queue defined as using the CF structure, and there are messages currently on the queue or the queue is open.
- When altering the index type of a queue, the necessary conditions regarding messages and uncommitted activity are not satisfied.
- When altering the default transmission queue, the old queue is currently being used as a transmission queue by default.
- Although the FORCE option was specified to overcome the object being open through the API, the object was created with a previous version of IBM MQ.
- There is no connection from the queue manager to the structure.

System action

The command is ignored.

System programmer response

Either:

- Wait until the object has been closed or deleted.

  Note: MCAs for receiver channels, or the intra-group queuing (IGQ) agent, can keep the destination queues open for a while even when messages are not being transmitted, and so such queues might appear to be in use.
- Wait until all the queues that use a storage class are empty

- Wait until the queue is empty
- Wait until use of the queue as a default transmission queue has ended

It is not possible to use the FORCE option of the ALTER command to overcome the situations that cause this message.

For more information about the command, see MQSC commands.

CSQM102E

csect-name SSLCIPH *sslciph* IS A WEAK OR BROKEN CIPHERSPEC

Severity

8

Explanation

A channel could not be defined or altered, because the specified SSLCIPH parameter contains a CipherSpec that is potentially insecure.

System action

The named channel is not defined or altered

System programmer response

Examine the CipherSpec specified in the SSLCIPH parameter and consider using a more secure CipherSpec.

If you want to re-enable the use of weak CipherSpecs, you can do so by adding a dummy Data Definition (DD) statement named CSQXWEAK to the channel initiator JCL. For example:

```
//CSQXWEAK DD DUMMY
```

If you want to re-enable the disabled SSLv3 support in IBM MQ, you can do so by adding a dummy Data Definition (DD) statement named CSQXSSL3 to the channel initiator JCL. For example:

```
//CSQXSSL3 DD DUMMY
```

You need to specify both of the preceding dummy DD statements, if you want to enable a weak SSLv3-based CipherSpec.

There are alternative mechanisms that can be used to forcibly re-enable weak CipherSpecs, and SSLv3 support,if the Data Definition change is unsuitable. Contact IBM Service for further information.

Attention: Re-enabling CipherSpecs in this manner leaves systems exposed to possible security problems. You should use CipherSpecs that use only the TLS protocol, rather than SSLv3.

CSQM103I

csect-name keyword(value) QSGDISP(*disposition*) HAS MESSAGES ASSOCIATED WITH IT

Severity

8

Explanation

A local queue specified for deletion has messages associated with it, and the DELETE request did not include the PURGE option.

System action

The command is ignored.

System programmer response

Either delete the local queue when it is empty, or reissue the request specifying the PURGE option. If the queue is a local copy of a group object, you must issue the request specifying PURGE explicitly for the local copy; specifying PURGE on the request to delete the group object has no effect.

CSQM104I

csect-name keyword(value) FLAGGED FOR DEFERRED DELETION

Severity

8

Explanation

A local dynamic queue specified on a DEFINE, ALTER, or DELETE request has been flagged for deferred deletion because it was found to be in use at the time of deletion.

System action

The queue is no longer available to new users, and will be deleted when all existing users of it have relinquished access.

CSQM105I

csect-name 'keyword' VALUE IS SAME AS QALIAS NAME

Severity

8

Explanation

An attempt was made to DEFINE or ALTER an alias queue so that the queue itself was named on the TARGQ keyword. Unless the queue is a cluster queue, this is not allowed because an alias queue can only resolve to a local or remote queue.

System action

The command is ignored.

System programmer response

Reissue the command with a different name for the TARGQ keyword.

CSQM106I

csect-name DEFXMITQ(*q-name*) IS NOT ALLOWED

Severity

8

Explanation

The specified queue is not allowed to be used as the default transmission queue because it is reserved for use exclusively by clustering.

System action

The command is ignored.

System programmer response

Reissue the command with a different DEFXMITQ name.

CSQM107I

csect-name STGCLASS ACTIVE OR QUEUE IN USE

Severity

8

Explanation

A request to ALTER or DEFINE REPLACE a local queue involving a change to the STGCLASS field is not allowed because there are messages on the queue, or other threads have the queue open.

System action

The command is ignored.

System programmer response

If there are messages on the queue, you must remove them before changing the storage class.

Note: If you remove all the messages from the queue, there might be a short delay before the command can be processed successfully.

If other threads have the queue open, wait until they have closed the queue before reissuing the command.

CSQM108I

csect-name keyword(value) NOT ALLOWED, INCOMPATIBLE NAME AND TYPE

Severity

8

Explanation

An attempt was made to issue a DEFINE command on a reserved object name, using an incorrect object type or subtype. The object is only allowed to be of the predetermined type listed in this topic:

Type	Object
Any Queue	SYSTEM.ADMIN.ACTIVITY.QUEUE SYSTEM.ADMIN.CHANNEL.EVENT SYSTEM.ADMIN.COMMAND.EVENT SYSTEM.ADMIN.CONFIG.EVENT SYSTEM.ADMIN.PERFM.EVENT SYSTEM.ADMIN.QMGR.EVENT SYSTEM.ADMIN.PUBSUB.EVENT SYSTEM.ADMIN.TRACE.ROUTE.QUEUE
Alias queue	SYSTEM.DEFAULT.ALIAS.QUEUE
Alias or local queue	SYSTEM.ADMIN.COMMAND.QUEUE SYSTEM.COMMAND.INPUT
Local queue	SYSTEM.CHANNEL.INITQ SYSTEM.CHANNEL.SYNCQ SYSTEM.CHLAUTH.DATA.QUEUE SYSTEM.CLUSTER.COMMAND.QUEUE SYSTEM.CLUSTER.REPOSITORY.QUEUE SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.DEFAULT.LOCAL.QUEUE SYSTEM.QSG.CHANNEL.SYNCQ SYSTEM.QSG.TRANSMIT.QUEUE
Model queue	SYSTEM.COMMAND.REPLY.MODEL SYSTEM.DEFAULT.MODEL.QUEUE SYSTEM.JMS.TEMPQ.MODEL SYSTEM.MQEXPLORER.REPLY.MODEL
Remote queue	SYSTEM.DEFAULT.REMOTE.QUEUE
Cluster-sender channel	SYSTEM.DEF.CLUSSDR
Cluster-receiver channel	SYSTEM.DEF.CLUSRCVR
Sender channel	SYSTEM.DEF.SENDER
Server channel	SYSTEM.DEF.SERVER
Receiver channel	SYSTEM.DEF.RECEIVER
Requester channel	SYSTEM.DEF.REQUESTER
Client-connection channel	SYSTEM.DEF.CLNTCONN

Type	Object
Server-connection channel	SYSTEM.ADMIN.SVRCONN SYSTEM.DEF.SVRCONN
Authentication information	SYSTEM.DEFAULT.AUTHINFO.CRLLDAP
Namelist	SYSTEM.DEFAULT.NAMELIST
Process	SYSTEM.DEFAULT.PROCESS
Storage class	SYSTEMST

System action

The command is ignored.

System programmer response

Ensure that reserved objects are defined with the correct object type or subtype.

CSQM109E

csect-name DYNAMIC QUEUE *value* NOT DELETED, MQRC=*mqrc (mqrc-text)*

Severity

8

Explanation

A dynamic queue could not be deleted during normal close processing, thread termination, or the end of queue manager restart, because an error occurred whilst attempting to delete it. *mqrc* gives the reason code for the error.

System action

The named dynamic queue is not deleted.

System programmer response

Refer to API completion and reason codes for information about the reason code to determine why the queue could not be deleted, and take the appropriate action as necessary. The most likely reason codes are:

- MQRC_OBJECT_IN_USE
- MQRC_PAGESET_ERROR
- MQRC_Q_NOT_EMPTY

CSQM110I

csect-name keyword(value) QSGDISP(*disposition*) HAS INCOMPLETE UNITS OF RECOVERY

Severity

8

Explanation

A command was issued that refers to a local queue that has incomplete units of recovery outstanding for it.

System action

The command is ignored.

System programmer response

Wait until all units of recovery for this queue are complete before attempting to issue the command again.

CSQM111E

csect-name COULD NOT PUT TO THE DEAD QUEUE, MQRC=*mqrc (mqrc-text)*

Severity
4

Explanation

An attempt to put a message to the dead letter queue was unsuccessful. *mqrc* gives the reason code for the error.

System action

Processing continues.

System programmer response

Refer to API completion and reason codes for information about *mqrc* (*mqrc-text* provides the MQRC in textual form) to determine the cause of the problem.

CSQM112E

csect-name ERROR ACCESSING *keyword(value)*

Severity
4

Explanation

While processing a command for an object, object information could not be accessed. This may be because of an error on page set zero, or in the coupling facility information, or because a coupling facility structure has failed, or because Db2 is not available or is suspended. This message is issued with message CSQM090E or CSQM091E, which include a reason code that gives more information about the error.

System action

The command is ignored.

System programmer response

Check for error messages on the console log that might relate to the problem. Verify that page set zero is set up correctly; for information about this, see Page sets. If a queue-sharing group is in use, check whether the coupling facility structure has failed and check that Db2 is available and not suspended. If the accompanying message is CSQM091E, refer to API completion and reason codes for an explanation of the *mqrc* in that message, and what action to take.

CSQM113E

csect-name NO SPACE FOR *keyword(value)* QSGDISP(*disposition*)

Severity
8

Explanation

A command failed because page set zero is full, or because the application structure is full, or because no more application structures are available in the coupling facility (the limit is 63).

System action

The command is not actioned.

System programmer response

Do one of the following, depending on the cause of the error:

- Increase the size of page set zero or the application structure. Refer to Managing page sets for information about how to do this.
- Reduce the number of application structures you are using.

CSQM114E

csect-name keyword(value) EXCEEDED LOCAL QUEUE LIMIT

Severity

8

Explanation

A command failed because no more local queues could be defined. There is an implementation limit of 524 287 for the total number of local queues that can exist. For shared queues, there is a limit of 512 queues in a single coupling facility structure.

System action

The command is not actioned.

System programmer response

Delete any existing queues that are no longer required.

CSQM115I

csect-name keyword(value) IS CURRENTLY IN USE, ALTER WITH FORCE NEEDED

Severity

8

Explanation

The object specified is in use. This could be because:

- It is open through the API.
- When altering the USAGE attribute of a local queue, there are messages currently on the queue.
- When altering the default transmission queue, the old queue is currently being used as a transmission queue by default.

System action

The command is ignored.

System programmer response

Either:

- Wait until the object has been closed or deleted.

  Note: MCAs for receiver channels, or the intra-group queuing (IGQ) agent, can keep the destination queues open for a while even when messages are not being transmitted, and so such queues might appear to be in use.
- Wait until the queue is emptied.
- Wait until use of the queue as a default transmission queue has ended.
- Use the ALTER command with the FORCE option.

  Note: Any subsequent API calls referencing the object will fail with a reason code of MQRC_OBJECT_CHANGED.

For more information about the command, see MQSC commands.

CSQM117E

csect-name ERROR ACCESSING *keyword(value)* QSGDISP(*disposition*)

Severity

4

Explanation

While processing a command for an object, object information could not be accessed. This may be because of an error on page set zero, or in the coupling facility information, or because a coupling facility structure has failed, or because Db2 is not available or is suspended. This message is issued with message CSQM090E or CSQM091E, which include a reason code that gives more information about the error.

System action

The command is ignored.

System programmer response

Check for error messages on the console log that might relate to the problem. If *disposition* is QMGR, COPY, or PRIVATE, verify that page set zero is set up correctly; for information about this, see Page sets. If *disposition* is GROUP or SHARED, check whether the coupling facility structure has failed and check that Db2 is available and is not suspended. If the accompanying message is CSQM091E, see API completion and reason codes for an explanation of the *mqrc* in that message, and what action to take.

CSQM118I

csect-name keyword(value) QSGDISP(*disposition*) LEVEL IS INCOMPATIBLE

Explanation

The definition level of the specified object is incompatible with that of the queue manager or other members of the queue-sharing group.

System action

Processing of the command is terminated.

System programmer response

For information about migration and compatibility between releases, see Maintaining and migrating.

CSQM119I

csect-name keyword(value) LEVEL IS INCOMPATIBLE

Explanation

The definition level of the specified object is incompatible with that of the queue manager or other members of the queue-sharing group.

System action

Processing of the command is terminated.

System programmer response

For information about migration and compatibility between releases, see Maintaining and migrating.

CSQM120I

csect-name keyword(value) NOT ALLOWED FOR SHARED QUEUE

Severity

8

Explanation

The specified value for the object name or attribute is not allowed for a local queue with a disposition that is shared or a model queue used to create a dynamic queue that is shared.

System action

Processing of the command is terminated.

System programmer response

Reissue the command correctly.

CSQM121I

csect-name keyword(value) NOT ALLOWED, NOT IN QUEUE-SHARING GROUP

Severity
8

Explanation

The specified value for the attribute requires a queue-sharing group, but the queue manager is not in a group.

System action

Processing of the command is terminated.

System programmer response

Reissue the command correctly.

CSQM122I

csect-name 'verb-name object' COMPLETED FOR QSGDISP(*disposition*)

Severity
0

Explanation

Processing for the specified command that refers to an object with the indicated disposition has completed successfully.

System action

A command is generated specifying CMDSCOPE(*) to perform further processing on all queue managers in the queue-sharing group. For example, if *disposition* is GROUP, the corresponding processing must be performed for local copies of the group object.

CSQM123I

csect-name 'keyword' VALUE CANNOT BE CHANGED

Severity
8

Explanation

The value for the specified attribute cannot be changed.

System action

Processing of the command is terminated.

System programmer response

To change the attribute, the object must be deleted and then redefined with the new value.

CSQM124I

csect-name keyword(value) ALLOWED ONLY WITH QSGDISP(*disposition*)

Severity
8

Explanation

The specified value for the attribute is allowed only for an object that has the indicated disposition.

System action

Processing of the command is terminated.

System programmer response

Reissue the command correctly.

CSQM125I

csect-name keyword(value) QSGDISP(*disposition*) WAS NOT FOUND

Severity

8

Explanation

A command was issued that refers to an object that does not exist. That is, no object could be found with the specified name and type (and subtype, for queues and channels) and disposition in the queue-sharing group.

System action

The command is ignored.

System programmer response

Check that you specified the correct name for the object, and the correct subtype (for queues and channels) or channel definition table (for deleting channels). If *disposition* is GROUP or SHARED, check that Db2 is available and is not suspended. Define the object if necessary.

Note:

1. An object of the same name and type, but of a different disposition, might already exist.
2. If you are dealing with a queue or channel object, an object of the same name, but of a different subtype, might already exist.
3. Remember that the object might have recently been deleted by someone else, or from another queue manager in the queue-sharing group.

CSQM126I

csect-name 'keyword' ONLY APPLICABLE TO LU62 PROTOCOL

Severity

8

Explanation

The named keyword can only be specified when TRPTYPE(LU62) is specified.

System action

Processing of the command is terminated.

System programmer response

Reissue the command without the named keyword.

CSQM127I

csect-name keyword(value) IS EMPTY OR WRONG TYPE

Severity

8

Explanation

A namelist used to specify a list of clusters has no names in it or does not have type CLUSTER or NONE.

System action

Processing of the command is terminated.

System programmer response

Reissue the command specifying a namelist that is not empty and has type CLUSTER or NONE.

CSQM128E

csect-name MQPUT FAILED FOR QUEUE *q-name*, MQRC=*mqrc (mqrc-text)*

Severity
8

Explanation

During the processing of a command, an attempt to put a message to the named queue failed for the specified reason.

System action

In general, the command is not actioned. If the command was REFRESH QMGR for configuration events, it might be partially completed as indicated by the preceding CSQM169I messages.

System programmer response

Refer to API completion and reason codes for information about *mqrc* (*mqrc-text*provides the MQRC in textual form). If *mqrc* is 2003, the message could not be committed.

CSQM129I

csect-name keyword(value) HAS WRONG CHANNEL TYPE

Severity
8

Explanation

The command (or the command with the particular disposition) cannot be used with the named channel because it cannot be used for channels of that type.

System action

The command is not actioned.

System programmer response

Check that the correct channel name and disposition was specified on the command. For more information about the command, see MQSC commands.

CSQM130I

csect-name CLUSTER REQUEST QUEUED

Severity
0

Explanation

Initial processing for a command completed successfully. The command requires further action by the cluster repository manager, for which a request was queued.

This message is followed by message CSQ9022I to indicate that the command has completed successfully, in that a request has been sent. It does **not** indicate that the cluster request has completed successfully. Such requests are processed asynchronously by the cluster repository manager; any errors are reported to the z/OS console, not to the command issuer.

System action

A request was queued for the cluster repository manager, which will process it asynchronously.

CSQM131I

csect-name CHANNEL INITIATOR NOT ACTIVE, CLUSTER AND CHANNEL COMMANDS INHIBITED

Severity
8

Explanation

A command was issued that required the channel initiator to be started.

System action

The command is not actioned.

System programmer response

Issue the START CHINIT command to start the channel initiator, and reissue the command.

CSQM132I

csect-name CHANNEL INITIATOR ALREADY ACTIVE

Severity
8

Explanation

The START CHINIT command was issued but the channel initiator is already active.

System action

The command is not actioned.

CSQM133I

csect-name UNABLE TO START CHANNEL INITIATOR

Severity
8

Explanation

A START CHINIT command was issued but the channel initiator could not be started.

This could be for one of the following reasons:

- The system did not allow the channel initiator address space to be created at this time due to a heavy system workload
- There was not enough storage to start the channel initiator address space
- The system tried to obtain more address spaces than the maximum number supported
- The queue manager was quiescing or shutting down.

System action

The command is not actioned.

System programmer response

Reissue the command when the system workload is reduced and when the queue manager is not shutting down.

CSQM134I

csect-name command keyword(value) COMMAND ACCEPTED

Severity
0

Explanation

Initial processing for a command has completed successfully. The command requires further action by the channel initiator, for which a request has been queued. Messages reporting the success or otherwise of the action will be sent to the command issuer subsequently.

System action

A request was queued for the channel initiator. Further messages will be produced when the command has been completed.

CSQM135I

csect-name NO CHANNEL INITIATOR AVAILABLE

Severity
8

Explanation

A command was issued for a shared channel, but there was no suitable channel initiator available for any active queue manager in the queue-sharing group. This could be because:

- no channel initiators are running
- the channel initiators that are running are too busy to allow any channel, or a channel of the particular type, to be started

System action

The command is not actioned.

System programmer response

Start a new channel initiator (on an active queue manager where there is no channel initiator running), or try again when there are fewer channels running.

CSQM136I
COMMAND NOT ALLOWED, COMMAND SERVER UNAVAILABLE

Explanation

A command for the channel initiator was entered, but the command server is not running and not enabled so the command cannot be processed.

System action

The command is not actioned.

System programmer response

Use the START CMDSERV command to start the command server, and reissue the command.

CSQM137I

csect-name command keyword COMMAND ACCEPTED

Severity
0

Explanation

Initial processing for a command has completed successfully. The command requires further action by the channel initiator, for which a request has been queued. Messages reporting the success or otherwise of the action will be sent to the command issuer subsequently.

System action

A request was queued for the channel initiator. Further messages will be produced when the command has been completed.

CSQM138I

csect-name CHANNEL INITIATOR STARTING

Severity
0

Explanation

A START CHINIT command was issued and the channel initiator address space has been started successfully.

System action

Further messages will be produced when the channel initiator itself has started.

CSQM139I

csect-name INDXTYPE(MSGTOKEN) NOT ALLOWED FOR TEMPORARY DYNAMIC QUEUE

Severity
8

Explanation

An attempt was made to define or alter a temporary-dynamic queue from which messages could be retrieved using message tokens. This combination is not allowed.

System action

Processing of the command is terminated.

System programmer response

Reissue the command with correct values.

CSQM140I

csect-name '*keyword*' NOT ALLOWED WITH TRPTYPE(*value*)

Severity
8

Explanation

The named keyword cannot be used on a START LISTENER command for the transport type shown.

System action

The command is not actioned.

System programmer response

Reissue the command with the correct keywords.

CSQM141I

csect-name 'LUNAME' REQUIRED WITH TRPTYPE(LU62)

Severity
8

Explanation

A START LISTENER command was issued specifying TRPTYPE(LU62) but without the LUNAME keyword. The LUNAME keyword is required with TRPTYPE(LU62).

System action

The command is not actioned.

System programmer response

Reissue the command with the correct keywords.

CSQM142I

csect-name CLUSTER(*cluster-name*) REPOSITORY IS NOT ON THIS QUEUE MANAGER

Severity
8

Explanation

A RESET CLUSTER command was issued, but the queue manager does not provide a full repository management service for the specified cluster. That is, the REPOS attribute of the queue manager is not *cluster_name*, or the namelist specified by the REPOSNL attribute of the queue manager does not contain *cluster_name* or is not of type CLUSTER or NONE.

System action

Processing of the command is terminated.

System programmer response

Reissue the command with the correct values or on the correct queue manager.

CSQM143I

csect-name CLUSTER TOPICS INHIBITED DUE TO PSCLUS(DISABLED)

Severity
8

Explanation

An attempt was made to define a cluster topic when the PSCLUS queue manager attribute is set to DISABLED.

System action

Processing of the command is terminated.

System programmer response

To enable publish/subscribe clustering, alter the PSCLUS attribute on all queue managers in the cluster to ENABLED.

CSQM144I

csect-name keyword(value) CANNOT BE A CLUSTER QUEUE

Severity
8

Explanation

An attempt was made to define or alter a queue to make it part of a cluster. This is not allowed if the queue is dynamic or is one of the following reserved queues:

- SYSTEM.CHANNEL.INITQ
- SYSTEM.CHANNEL.SYNCQ
- SYSTEM.CLUSTER.COMMAND.QUEUE
- SYSTEM.CLUSTER.REPOSITORY.QUEUE
- SYSTEM.COMMAND.INPUT

- SYSTEM.QSG.CHANNEL.SYNCQ
- SYSTEM.QSG.TRANSMIT.QUEUE

System action

Processing of the command is terminated.

System programmer response

Reissue the command with the correct values.

CSQM145I

csect-name '*keyword*' VALUE REQUIRED FOR SHARED QUEUE

Severity

8

Explanation

A non-blank value must be specified for the named keyword for a local queue with a disposition that is shared or a model queue used to create a dynamic queue that is shared.

System action

Processing of the command is terminated.

System programmer response

Reissue the command with a value for the keyword added.

CSQM146I

csect-name keyword(value) VALUE IS REPEATED

Severity

8

Explanation

A keyword was entered that takes a list of values, and the named value appears more than once in the list.

System action

The command is ignored.

System programmer response

Reissue the command with the parameter specified correctly. For more information about the command, see MQSC commands.

CSQM147I

csect-name '*keyword1*' AND '*keyword2*' VALUES MUST BOTH BE BLANK OR NON-BLANK

Severity

8

Explanation

An attempt was made to define or alter an object so that it had a blank value for one of the specified keywords and a non-blank value for the other. Both of those values must either be blank or non-blank.

System action

Processing of the command is terminated.

System programmer response

Reissue the command with correct values.

CSQM148I

csect-name '*keyword*' NOT ALLOWED WITH TYPE '*value*'

Severity
8

Explanation

The named keyword cannot be specified for queues or channels of the type shown.

System action

Processing of the command is terminated.

System programmer response

Reissue the command without the named keyword.

CSQM149I

csect-name '*keyword*' REQUIRED WITH TYPE '*value*'

Severity
8

Explanation

The named keyword was not specified but is required for queues or channels of the type shown.

System action

Processing of the command is terminated.

System programmer response

Reissue the command with the named keyword added.

CSQM150I

csect-name '*keyword1*' AND '*keyword2*' VALUES ARE INCOMPATIBLE

Severity
8

Explanation

An attempt was made to define or alter an object so that it had incompatible values for the specified keywords.

System action

Processing of the command is terminated.

System programmer response

Reissue the command with correct values. For information about the restrictions on the values for the keywords, see MQSC commands.

CSQM151I

csect-name '*keyword1*' AND '*keyword2*' VALUES CANNOT BOTH BE NON-BLANK

Severity
8

Explanation

An attempt was made to define or alter an object so that it had non-blank values for both of the specified keywords. At most one of those values can be non-blank.

System action

Processing of the command is terminated.

System programmer response

Reissue the command with correct values.

CSQM152I

csect-name USAGE(XMITQ) NOT ALLOWED FOR CLUSTER QUEUE

Severity
8

Explanation

An attempt was made to define or alter a queue so that it was both a transmission queue and in a cluster. This is not allowed.

System action

Processing of the command is terminated.

System programmer response

Reissue the command with correct values.

CSQM153E

csect-name Db2 NOT AVAILABLE

Severity
8

Explanation

Because Db2 is not available or no longer available, the queue manager cannot handle the command for a CF structure or shared channel.

System action

Processing of the command is terminated.

System programmer response

Use the preceding messages on the z/OS console to investigate why Db2 is not available, and resume the connection or restart Db2 if necessary.

CSQM154E

csect-name ERROR ACCESSING Db2

Severity
8

Explanation

Because there was an error in accessing Db2, the queue manager cannot handle the command for a CF structure or shared channel.

System action

Processing of the command is terminated.

System programmer response

Resolve the error reported in the preceding messages.

CSQM155I

csect-name STATUS(STOPPED) NOT ALLOWED WITH QMNAME OR CONNAME

Severity
8

Explanation

An attempt was made to stop a channel using STATUS(STOPPED), but a queue manager name or connection name was also specified. This is not allowed.

System action

Processing of the command is terminated.

System programmer response

Reissue the command with correct values.

CSQM156I

csect-name INDXTYPE(GROUPID) NOT ALLOWED FOR *keyword(value)*

Severity
8

Explanation

An attempt was made to define or alter a queue with a reserved name so that it had an index type of GROUPID. This is not allowed.

System action

Processing of the command is terminated.

System programmer response

Reissue the command with correct values.

CSQM157E

csect-name NO SPACE FOR *keyword(value)*

Severity
8

Explanation

An IBM MQ DEFINE CFSTRUCT command failed because no more application structures are available in the coupling facility (the limit is 63).

System action

The command is not actioned.

System programmer response

Reduce the number of application structures you are using.

CSQM158I

csect-name RECOVER(YES) NOT ALLOWED WITH CFLEVEL(*value*)

Severity
8

Explanation

An attempt was made to define or alter a CF structure to support recovery, but the level of the CF structure was less than 3. This is not allowed.

System action

Processing of the command is terminated.

System programmer response

Reissue the command with correct values. You cannot alter the level of a CF structure; you must delete the structure and then redefine it.

CSQM159I

csect-name verb-name object(obj-name) NOT ALLOWED, INCOMPATIBLE QUEUE MANAGER CMDLEVELS

Severity

8

Explanation

An attempt was made to alter the CF level of a CF structure, or to delete the structure. This action requires that all queue managers in the queue-sharing group must have a certain command level. Some of the queue managers have a lower level.

System action

Processing of the command is terminated.

System programmer response

Ensure all the queue managers in the queue-sharing group have the appropriate command level. For information about restrictions on the command, see MQSC commands.

CSQM160I

csect-name keyword(value) IS NOT UNIQUE

Severity

8

Explanation

A command was issued that refers to an object that exists with more that one disposition in the queue-sharing group, so the object to be used cannot be determined.

System action

The command is not executed.

System programmer response

Delete one of the objects.

CSQM161I

csect-name QUEUE ATTRIBUTES ARE INCOMPATIBLE

Severity

8

Explanation

A MOVE QLOCAL command was issued, but the queues involved have different values for one or more of these attributes: DEFTYPE, HARDENBO, INDXTYPE, USAGE. Messages cannot be moved safely if these attributes differ.

System action

The command is not executed.

System programmer response

Check that the queue names have been entered correctly. Change the queue attributes as necessary.

CSQM162I

csect-name keyword(value) MAXDEPTH IS TOO SMALL

Severity

8

Explanation

A MOVE QLOCAL command was issued, but the MAXDEPTH attribute value for the target queue is too small to allow all the messages to be moved.

System action

The command is not executed.

System programmer response

Change the MAXDEPTH value for the queue.

CSQM163I

csect-name ERROR USING *keyword(value)*, MQRC=*mqrc (mqrc-text)*

Severity

8

Explanation

During the processing of a MOVE QLOCAL command, an attempt to open the named queue or to get or put a message for it failed for the specified reason. For example, a put to the target queue will fail if a message is too long.

System action

The command stops processing. If some messages have already been moved and committed, they will remain on the target queue; the rest of the messages will not be moved.

System programmer response

Refer to API completion and reason codes for information about *mqrc* (*mqrc-text* provides the MQRC in textual form), and take the appropriate action to resolve the problem.

CSQM164I

csect-name keyword(value) HAS MESSAGES ASSOCIATED WITH IT

Severity

8

Explanation

A MOVE QLOCAL command was issued specifying TYPE(MOVE), the target queue already has messages associated with it.

System action

The command is not executed.

System programmer response

Check that the queue name was entered correctly. Determine if it is safe to add messages to the queue, then reissue the command using the TYPE(ADD) option.

CSQM165I

csect-name n MESSAGES MOVED

Severity

0

Explanation

A MOVE QLOCAL command was issued, and moved the indicated number of messages.

If the command completed successfully and moved all the messages on the queue, this confirms the number moved. If an error occurred while moving the messages, this shows how many messages were successfully moved to the target queue and committed.

System action

Processing continues.

System programmer response

If the command did not complete successfully, as shown by the following CSQ9023E message, investigate the problem reported in the preceding messages.

CSQM166I

csect-name keyword(value) NOT AUTHORIZED

Severity

8

Explanation

You do not have proper authorization to use the command for the specified object.

System action

The command is not executed for that object.

System programmer response

Check that the object name was entered correctly. If required, arrange for someone who is authorized to use the object to issue the command for you, or get the necessary authority granted to you.

CSQM167I

csect-name PERFORMANCE EVENTS DISABLED

Severity

8

Explanation

A command was issued that required performance events to be enabled.

System action

The command is not executed.

System programmer response

Use the ALTER QMGR command to set the PERFMEV attribute to ENABLED if performance events are required.

CSQM168I

csect-name CONFIGURATION EVENTS DISABLED

Severity

8

Explanation

A command was issued that required configuration events to be enabled.

System action

The command is not executed.

System programmer response

Use the ALTER QMGR command to set the CONFIGEV attribute to ENABLED if configuration events are required.

CSQM169I

csect-name object-type OBJECTS: *m* FOUND, *n* EVENTS GENERATED

Severity

0

Explanation

A REFRESH QMGR command was issued for configuration events. *m* objects of the indicated type were found that matched the specified selection criteria (such as name or time of alteration), and *n* event messages were generated. The number of event messages might be less than the number of objects found because certain objects might be excluded, such as temporary dynamic queues or objects in the process of being deleted. It might also be less than the number of objects found if there was a problem with the event queue.

System action

Processing continues.

System programmer response

If *n* is less than *m*, but message CSQ9022I follows these messages to indicate that the command completed successfully, no action is needed. Otherwise, investigate the problem with the event queue as reported in the preceding messages.

CSQM170I

csect-name REFRESHING CONFIGURATION EVENTS SINCE *date time*

Severity

0

Explanation

A REFRESH QMGR command was issued for configuration events specifying a refresh interval with the INCLINT keyword. Event messages will be generated for all objects with an alteration date and time later than *date time* (provided they match any other specified selection criteria, such as name or type). However, event messages will not be generated for objects deleted after that time.

CSQM171I

csect-name CONFIGURATION EVENTS REFRESH NEEDED

Severity

0

Explanation

An ALTER QMGR command was issued that enables configuration events. Event messages need to be generated to ensure that the configuration information is complete and up to date.

System action

Processing continues.

System programmer response

If complete configuration information is required, do one of the following, as appropriate:

- If this is the first time that configuration events have been enabled, use the REFRESH QMGR TYPE(CONFIGEV) command to generate configuration events for **all** objects. If you have many objects, it may be preferable to use several such commands each with a different selection of objects, but such that all are included.
- Otherwise, use the REFRESH QMGR TYPE(CONFIGEV) command to generate events to replace those that were not generated while configuration events were disabled; specify the INCLINT parameter to cover this period.

CSQM172I

csect-name '*keyword*' NOT ALLOWED WITH TYPE(*value*)

Severity
8

Explanation

The named keyword cannot be specified with the TYPE value shown.

System action

Processing of the command is terminated.

System programmer response

Reissue the command without the named keyword.

CSQM173I

csect-name EXPIRED MESSAGE SCAN REQUESTED FOR *m* QUEUES

Severity
0

Explanation

A REFRESH QMGR command was issued for expired message scanning. *m* queues were found that matched the specified selection criteria.

System action

Processing continues.

CSQM174E

csect-name '*keyword*' is not allowed with CFLEVEL(*cflevel*) - this keyword requires CFLEVEL(5)

Severity
8

Explanation

An attempt was made to define or alter the value of a structure attribute related to SMDS, but the level of the structure was less than CFLEVEL(5). This is not allowed.

System action

Processing for the command is terminated.

System programmer response

Issue the command again with correct values. You cannot alter the level of a CF structure; you must delete the structure, and then define it again.

CSQM175E

csect-name '*keyword*' cannot be altered because a data set is currently active for this structure

Severity
8

Explanation

The keywords DSGROUP and DSBLOCK can only be altered before the first data set has been allocated for the structure. Once an SMDS data set has become active for this structure then these attribute values cannot be changed.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry and reissue the command correctly.

CSQM176E

csect-name SMDS cannot currently be reset to *keyword*(*value*)

Severity

8

Explanation

A **RESET SMDS** command requested a change of status which is not compatible with the existing status.

- The option **STATUS(FAILED)** is only allowed when the current status is **ACTIVE** or **RECOVERED** (or already **FAILED**, in which case the command has no effect).
- The option **STATUS(RECOVERED)** is only allowed when the current status is **FAILED** (or already **RECOVERED**).

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command correctly.

CSQM177I

csect-name '*keyword*' NOT ALLOWED WITH ACTION '*value*'

Severity

8

Explanation

The named keyword cannot be specified for channel authentication settings of the action shown.

System action

Processing for the command is terminated.

System programmer response

Reissue the command without the named keyword.

CSQM178I

csect-name ACTION NOT ALLOWED FOR CHANNEL *channel-type*(*channel-name*)

Severity

8

Explanation

The MATCH(RUNCHECK) action that you requested cannot be performed on the channel with the specified parameters. This may be because either: -

- The channel is a SVRCONN and the QMNAME parameter was supplied.

- The channel is not a SVRCONN and the CLNTUSER parameter was supplied

System action

Processing of the command is terminated.

System programmer response

Either correct the specified parameters or alter the channel to the appropriate channel type and then reissue the command.

CSQM179I

csect-name CHANNEL WILL RUN USING MCAUSER(*userid*)

Severity

0

Explanation

No matching channel authentication (CHLAUTH) records were found that match the given fields.

Note:

1. The returned MCAUSER value does not take into account possible actions by a channel security exit.
2. Channel authentication rules that match the host name apply only if the queue manager REVDNS attribute is enabled and the DNS server returns a valid host name for the IP address.

CSQM181I

csect-name INSUFFICIENT STORAGE TO COMPLETE COMMAND

Severity

8

Explanation

There was insufficient storage available to complete processing for the command.

System action

The command terminates. Any processing already completed may be retained or backed out.

System programmer response

Refer to the accompanying messages to determine what processing has been done. Retry the command, if appropriate, when your queue manager is less busy. If the problem persists, you might need to increase the region size used by your queue manager, or you might need to reduce the number of jobs running in your system.

CSQM182E

csect-name DURABLE SUBSCRIPTIONS NOT ALLOWED

Severity

8

Explanation

A DEFINE SUB command was issued, but it was not possible to make a durable subscription.

This could be for one of the following reasons:

- The topic subscribed to is defined as DURSUB(NO)
- The queue named SYSTEM.DURABLE.SUBSCRIBER.QUEUE is not available
- The CSQINP2 data sets are in the wrong order, the order is:

```
//CSQINP2  DD DSN=hlq.SCSQPROC(CSQ4INYS),DISP=SHR //         DD DSN=hlq.SCSQPROC(CSQ4INSX),DISP=SHR
//         DD DSN=hlq.SCSQPROC(CSQ4INSG),DISP=SHR
```

System action

The command is not executed.

System programmer response

Durable subscriptions are stored on the SYSTEM.DURABLE.SUBSCRIBER.QUEUE. Ensure that this queue is available for use. Possible reasons for failure include the queue being full, the queue being put inhibited, or the queue not existing.

If the topic subscribed to is defined as DURSUB(NO) then it is not possible to administratively define a subscription. The topic can be altered to DURSUB(YES) to enable the subscription to be defined.

CSQM183E

csect-name SUBSCRIPTION INHIBITED

Severity

8

Explanation

A DEFINE SUB command was issued, but it was not possible to make a subscription because the topic subscribed to is defined as SUB(DISABLED).

System action

The command is not executed.

System programmer response

If the topic subscribed to is defined as SUB(DISABLED) then it is not possible to administratively define a subscription. The topic can be altered to SUB(ENABLED) to enable the subscription to be defined.

CSQM184I

csect-name '*keyword1*' AND '*keyword2*' VALUES CANNOT BOTH BE BLANK

Severity

8

Explanation

An attempt was made to define or alter an object so that it had blank values for both of the specified keywords. One of those values must be provided.

System action

Processing of the command is terminated.

System programmer response

Reissue the command with correct values.

CSQM185E

csect-name SUBSCRIPTION HAS FIXED SUBUSER

Severity

8

Explanation

An ALTER SUB command was issued, but it was not possible to ALTER the target subscription because the userid performing the ALTER did not match the SUBUSER attribute of the subscription and the subscription has had the VARUSER(FIXED) attribute set.

System action

The command is not executed.

System programmer response

The subscription can be altered only by the owning userid that is displayed in the SUBUSER attribute.

CSQM186E

csect-name DESTCLAS VALUE CANNOT BE ALTERED

Severity
8

Explanation

An ALTER SUB command was issued, but it was not possible to ALTER the target subscription because the DESTCLAS attribute specified on the request did not match the one in the existing subscription. DESTCLAS cannot be altered.

System action

The command is not executed.

System programmer response

Ensure that the DESTCLAS attribute matches the existing subscription and rerun the request.

CSQM187E

csect-name GROUPING VALUE CANNOT BE ALTERED

Severity
8

Explanation

An ALTER SUB command was issued, but it was not possible to ALTER the target subscription because the GROUPING attribute specified on the request did not match the one in the existing subscription. GROUPING attributes cannot be altered.

System action

The command is not executed.

System programmer response

Ensure that the GROUPING attribute matches the existing subscription and rerun the request.

CSQM188E

csect-name SUBSCOPE VALUE CANNOT BE ALTERED

Severity
8

Explanation

An ALTER SUB command was issued, but it was not possible to ALTER the target subscription because the SUBSCOPE attribute specified on the request did not match the one in the existing subscription. SUBSCOPE cannot be altered.

System action

The command is not executed.

System programmer response

Ensure that the SUBSCOPE attribute matches the existing subscription and rerun the request.

CSQM189E

csect-name SELECTOR VALUE CANNOT BE ALTERED

Severity

8

Explanation

An ALTER SUB command was issued, but it was not possible to ALTER the target subscription because the SELECTOR attribute specified on the request did not match the one in the existing subscription. SELECTOR cannot be altered.

System action

The command is not executed.

System programmer response

Ensure that the SELECTOR attribute matches the existing subscription and rerun the request.

CSQM190E

csect-name TOPIC STRING IS INVALID

Severity

8

Explanation

A DEFINE SUB command was issued, but it was not possible to make a subscription because the topic string was invalid.

This could be because the WSCHEMA attribute was set to CHAR and either:

- The TOPICSTR attribute contains an invalid escape character, or
- The TOPICOBJ attribute refers to a TOPIC object with a TOPICSTR attribute that contains an invalid escape character.

System action

The command is not executed.

System programmer response

Correct the TOPICSTR attribute on the **DEFINE SUB** command to correctly use escape characters. If the problem is with the TOPICSTR in a TOPIC object, correct that TOPIC object or refer to a different TOPIC object. If the TOPICSTR needs to use the characters in that way, set the WSCHEMA attribute to *TOPIC* to avoid errors with escape characters.

CSQM191E

csect-name TOPIC STRING CANNOT BE ALTERED

Severity

8

Explanation

A DEFINE TOPIC command using the REPLACE keyword was issued, providing a value for TOPICSTR that was different from the value in the existing object. This is not allowed.

System action

The command is not executed.

System programmer response

Reissue the command with correct values. You cannot alter the topic string in a topic object; you must delete the object and then redefine it.

CSQM192I

csect-name Address '*address*' is invalid.

Severity
8

Explanation

The IP address or host name *address* contains invalid characters.

System action

Processing of the command is terminated.

System programmer response

Reissue the command with the parameter specified correctly. Note that the BLOCKADDR list may contain only IP addresses: host name addresses are not permitted.

CSQM193I

csect-name IP address '*ipaddress*' contains an invalid range.

Severity
8

Explanation

The IP address *ipaddress* contains an invalid range. For example, the lower number is greater than or equal to the upper number for the range.

System action

Processing of the command is terminated.

System programmer response

Reissue the command with the parameter specified correctly.

CSQM194I

csect-name IP address '*ipaddress1*' overlaps existing IP address '*ipaddress2*'.

Severity
8

Explanation

The IP address *ipaddress1* overlaps with an existing IP address *ipaddress2*. For example, addresses 1.2.3.4-7 and 1.2.3.6-8 overlap.

System action

Processing of the command is terminated.

System programmer response

Reissue the command with the parameter specified correctly.

CSQM195I

csect-name MATCH RUNCHECK FOUND A GENERIC VALUE IN *field-name*

Severity
8

Explanation

A DISPLAY **CHLAUTH** command was issued using the MATCH(RUNCHECK) parameter and the *field-name* parameter was found to contain a generic value, which is not allowed.

System action

Processing of the command is terminated.

System programmer response

Reissue the command with a value in *field-name* which is not generic.

CSQM196I

csect-name REQUIRED KEYWORD MISSING FOR *keyword*(*value*)

Severity
8

Explanation

A required additional keyword was not specified in conjunction with *keyword* (*value*).

This message can be returned in the following scenarios:

- A **DISPLAY CHLAUTH** command, specifying **MATCH(RUNCHECK)** did not specify the **ADDRESS** keyword or one of the keywords **CLNTUSR** or **QMNAME**.
- A **SET CHLAUTH** command, the **MCAUSER** is missing when **USERSRC(MAP)** is specified or **USERSRC** is missing as **USERSRC(MAP)** is the default.

System action

Processing of the command is terminated.

System programmer response

Reissue the command specifying one of the required keywords

CSQM197I

csect-name '*keyword*' NOT ALLOWED WITH MATCH '*value*'

Severity
8

Explanation

The named keyword cannot be specified for **DISPLAY CHLAUTH** in conjunction with the identified value for the **MATCH** keyword.

System action

Processing of the command is terminated.

System programmer response

Reissue the command without the named keyword.

CSQM198I

csect-name CHANNEL AUTHENTICATION PROFILE NAME IS INVALID

Severity
8

Explanation

The channel profile name used in the command was not valid.

System action

Processing of the command is terminated.

System programmer response

Check that the characters entered for the profile are valid and reissue the command. If TYPE(BLOCKADDR) is specified, check that CHLAUTH('*') is also specified

CSQM199I

csect-name CFCONLOS (TOLERATE) NOT ALLOWED, INCOMPATIBLE QUEUE MANAGER CMDLEVELS

Severity

8

Explanation

An attempt was made to change the **CFCONLOS** queue manager attribute to a value of **TOLERATE**, which enables toleration of loss of connectivity to Coupling Facility structures. This action requires that all queue managers in the queue-sharing group must have a command level of at least 710. Some of the queue managers have a lower level.

System action

Processing of the command is terminated.

System programmer response

Ensure all the queue managers in the queue-sharing group have the appropriate command level. For information about restrictions on the command, see MQSC commands.

CSQM201I

csect-name verb-name *obj-type* DETAILS

Severity

0

Explanation

This message is the response to a command that displays attributes or other information about objects, when that command was entered from either the console, or the command server initialization server. It shows the attributes requested for *obj-type*, as follows:

```
  obj-type(name)  attribute-value  attribute-value
⋮
   END obj-type DETAILS
```

See the specific command for details of the attributes and values.

csect-name might include the command prefix (CPF), depending on how the command was entered.

Exceptionally, the last line might be:

```
  obj-type TERMINATED WITH MAX LINES
```

if the number of lines allowed in a multiple line WTO to be issued on the console (255) was exceeded. This figure includes the first and last lines of the display. The only object that might cause this message is namelist because displaying a complete namelist would require 263 lines in total. (This only occurs when the command was issued from the console.) For details of the fields reported, see the command description.

CSQM224I

csect-name verb-name obj-type DETAILS - CURRENTLY DISABLED

Severity
0

Explanation

This message is issued instead of CSQM201I for channel authentication (CHLAUTH) records if the CHLAUTH queue manager attribute has been set to DISABLED.

See the explanation of message CSQM201I for more information.

CSQM292I

csect-name PUBLISH/SUBSCRIBE ENGINE IS DISABLED

Severity
0

Explanation

The publish/subscribe engine is unavailable because it has been disabled.

System action

The command is actioned, but no results are returned because the publish/subscribe engine has been disabled.

System programmer response

This message occurs because you are attempting to query the publish/subscribe engine but you have disabled it. To use the publish/subscribe engine, set the PSMODE queue manager attribute to a value other than DISABLED.

CSQM293I

csect-name m obj-type FOUND MATCHING REQUEST CRITERIA

Severity
0

Explanation

A command that displays attributes or other information about objects has been issued. *m* objects were found that matched the specified selection criteria.

System action

For each object found, a message follows giving its details.

CSQM294I

csect-name CANNOT GET INFORMATION FROM DB2

Severity
8

Explanation

While processing a command that displays attributes or other information about objects with a disposition of GROUP or SHARED, information could not be obtained from Db2. This might be because Db2 is not available or no longer available, or because it is suspended, or because there was an error in accessing Db2, or because a Db2 table was temporarily locked.

System action

Information about objects with a disposition of GROUP or SHARED is not displayed, so the information displayed might therefore be incomplete.

System programmer response

Refer to the console log for messages giving more information about the error.

CSQM295I

csect-name UNEXPECTED ERROR DURING DISPLAY

Severity
8

Explanation

A severe error occurred while processing a command that displays attributes or other information about objects.

System action

The command is terminated.

System programmer response

Refer to the console log for messages giving more information about the error.

CSQM297I

csect-name NO *item* FOUND MATCHING REQUEST CRITERIA

Severity
0

Explanation

A command that displays attributes or other information about objects or runtime status found that there are no items that match the specified name and satisfy any other criteria requested (such as subtype or disposition in a queue-sharing group).

CSQM298I

csect-name TOTAL MESSAGE LENGTH ALLOWED ON CONSOLE EXCEEDED

Severity
8

Explanation

The total message length for the command allowed on the console (32 K) was exceeded.

System action

The command is actioned, but the display of the command is terminated.

System programmer response

This error occurs if a command that displays attributes or other information about objects is entered using a generic name (for example, DIS Q(*) ALL), and the total amount of data to be displayed exceeds 32 K. To avoid this problem, try to be more selective about the information requested (for example, DIS Q(PAY*) ALL).

CSQM299I

csect-name INSUFFICIENT STORAGE TO COMPLETE DISPLAY

Severity
8

Explanation

There was insufficient storage available to complete processing of a command that displays attributes or other information about objects.

System action

The command is actioned, but the display of the information is terminated before completion. The data returned is a subset of the requested information. Refer to message CSQM293I, which indicates how many objects have information returned. The message does not indicate how many matching objects were found.

System programmer response

If this error occurs when a generic name is used in the command (for example, DIS QUEUE(*) ALL), try to be more selective about the information requested (for example, DIS QUEUE(PAY*) ALL). If the problem persists, you might need to increase the region size used by your queue manager or channel initiator, or you might need to reduce the number of jobs running in your system.

CSQM4nnI

object details

Severity

0

Explanation

This message consists of the entire object or object status details formatted for use by applications. It is issued in response to commands entered from the command server. Message CSQ9022I follows this message.

The message number depends on the object or object status type, as follows:

Number	Object or status type
CSQM400I	Storage class object
CSQM401I	Local queue object
CSQM402I	Model queue object
CSQM403I	Alias queue object
CSQM406I	Remote queue object
CSQM407I	Namelist object
CSQM408I	Process object
CSQM409I	Queue manager object
CSQM410I	Sender channel object
CSQM411I	Server channel object
CSQM412I	Receiver channel object
CSQM413I	Requester channel object
CSQM415I	Server-connection channel object
CSQM416I	Client-connection channel object
CSQM417I	Cluster-receiver channel object
CSQM418I	Cluster-sender channel object
CSQM420I	Sender channel status
CSQM421I	Server channel status
CSQM422I	Receiver channel status
CSQM423I	Requester channel status
CSQM425I	Server-connection channel status
CSQM427I	Cluster-receiver channel status
CSQM428I	Cluster-sender channel status

Number	Object or status type
CSQM430I	CF structure object
CSQM431I	Cluster queue object
CSQM437I	Authentication information object
CSQM438I	Topic object
CSQM439I	Cluster queue manager object
CSQM440I	CF structure status
CSQM441I	Local queue status
CSQM442I	Connection information
CSQM443I	Topic status
CSQM444I	Subscription
CSQM445I	Subscription status
CSQM446I	Publish/Subscribe status
CSQM451I	Local queue statistics
CSQM452I	Shared message data set
CSQM453I	Shared message data set connection
CSQM454I	Channel authentication record

CSQM500I

csect-name GROUPUR agent starting TCB=*tcb-name*

Severity

0

Explanation

The group unit of recovery (GROUPUR) agent was started during the initialization of a queue manager that is in a queue-sharing group. The agent uses TCB *tcb-name*.

The GROUPUR agent monitors the SYSTEM.QSG.UR.RESOLUTION.QUEUE to process requests from other queue managers within the QSG.

System action

Processing continues. The GROUPUR agent is started.

CSQM501I

csect-name GROUPUR agent stopping

Severity

4

Explanation

The group unit of recovery (GROUPUR) agent is stopping because of one the following reasons:

- the queue manager is stopping
- it was unable to recover from an IBM MQ API error or an abnormal ending

System action

The GROUPUR agent stops.

If the agent has stopped due to an error it will be automatically restarted.

System programmer response

If the queue manager is not stopping, investigate the cause of the error as reported in the preceding messages.

CSQM502I

csect-name processed BACKOUT request from *qmgr-name* for in-doubt UOW, URID=*urid*, CONNECTION-NAME=*name*

Severity

0

Explanation

This message is generated during queue manager startup when the GROUPUR agent has processed a message on the SYSTEM.QSG.UR.RESOLUTION.QUEUE from another queue manager in the queue-sharing group requesting that the specified UOW be backed out.

System action

Processing continues.

CSQM503I

csect-name processed COMMIT request from *qmgr-name* for in-doubt UOW, URID=*urid*, CONNECTION-NAME=*name*

Severity

0

Explanation

This message is generated during queue manager startup when the GROUPUR agent has processed a message on the SYSTEM.QSG.UR.RESOLUTION.QUEUE from another queue manager in the queue-sharing group requesting that the specified UOW be committed.

System action

Startup continues.

CSQM504I

csect-name GROUPUR support enabled

Severity

0

Explanation

This message is generated during queue manager startup, or in response to an ALTER QMGR command, if the GROUPUR queue manager attribute is enabled and all of the configuration checks performed by the GROUPUR agent are satisfied.

System action

The queue manager permits applications to establish transactions with a GROUP unit of recovery disposition.

CSQM505I

csect-name GROUPUR support disabled

Severity

0

Explanation

This message is generated during queue manager startup or in response to an ALTER QMGR command if the GROUPUR queue manager attribute is disabled.

System action

The queue manager inhibits applications from establishing transactions with a GROUP unit of recovery disposition.

CSQM506I

csect-name GROUPUR qmgr attribute has been disabled CODE=*code*

Severity

4

Explanation

This message is generated at queue manager startup if the GROUPUR queue manager attribute is enabled but one of the configuration checks performed by the GROUPUR agent failed. CODE=*code* contains an identifier indicating which configuration check failed.

System action

The GROUPUR queue manager attribute is disabled.

System programmer response

The system programmer should use the code specified to identify what configuration check failed. If support for group units of recovery is required, they should take corrective action and then re-enable the GROUPUR queue manager attribute.

CSQM507E

csect-name GROUPUR qmgr attribute was not enabled CODE=*code*

Severity

8

Explanation

This message is generated in response to an ALTER QMGR command if an attempt to enable the GROUPUR queue manager attribute fails because one of the configuration checks performed by the GROUPUR agent are not satisfied. CODE=*code* contains an identifier indicating which configuration check failed.

System action

The GROUPUR queue manager attribute remains disabled and the ALTER QMGR command fails.

System programmer response

The system programmer should use the code specified to identify what configuration check failed. They should then take corrective action and then re-issue the ALTER QMGR command.

When you enable group units of recovery (GROUPUR support) a number of configuration checks are performed to ensure the configuration steps have been completed. You cannot enable this support if any of these checks fail.

These checks are also performed at queue manager startup if GROUPUR queue manager attribute is enabled. If one of these checks fails during startup then group units of recovery will be disabled until you correct the error and re-enable the GROUPUR queue manager attribute.

If a check fails it will be identified with a return code (number). You can use this code to identify the failing check using the following list:

1. This queue manager is not a member of a queue-sharing group.
2. The SYSTEM.QSG.UR.RESOLUTION.QUEUE does not exist.
3. The SYSTEM.QSG.UR.RESOLUTION.QUEUE does not support persistent messages.
4. The SYSTEM.QSG.UR.RESOLUTION.QUEUE is not indexed by correlation ID.

5. The SYSTEM.QSG.UR.RESOLUTION.QUEUE does not reside on the system application coupling facility structure, CSQSYSAPPL.
6. The queue manager name is the same as the name of the queue-sharing group.

CSQM508E

csect-name GROUPUR agent ended abnormally. Restarting

Severity

8

Explanation

The group unit of recovery (GROUPUR) agent has ended abnormally because a severe error occurred, as reported in the preceding messages.

System action

The group unit of recovery (GROUPUR) agent attempts to restart a number of times. If it fails persistently, it terminates.

System programmer response

Ensure the CFSTRUCT called CSQSYSAPPL is configured for GROUPUR operation. See Enabling GROUP units of recovery.

Investigate the reason for the abnormal termination, as reported in the preceding messages.

CSQM520I

csect-name PSCLUS CANNOT BE ALTERED, CLUSTER TOPICS EXIST

Severity

8

Explanation

An attempt was made to set the PSCLUS queue manager attribute to DISABLED, indicating that Publish/Subscribe activity is not expected in this cluster between queue managers, but a cluster topic exists so the setting cannot be modified.

System action

Processing of the command is terminated.

System programmer response

To disable publish/subscribe clustering delete all cluster topic objects before altering the PSCLUS attribute on all queue managers in the cluster to DISABLED.

CSQM521I

csect-name CLCHNAME MUST BE BLANK FOR DYNAMIC QUEUE

Severity

8

Explanation

An attempt was made to define or alter a dynamic queue with a non blank value for the CLCHNAME attribute, which is not allowed.

System action

Processing of the command is terminated.

System programmer response

Reissue the command with compatible attribute values.

CSQM522I

csect-name NOSHARE NOT ALLOWED WITH NON-BLANK CLCHNAME

Severity

8

Explanation

An attempt was made to define or alter a queue with a non-blank value for the CLCHNAME attribute, but NOSHARE was specified or implied.

System action

Processing of the command is terminated.

System programmer response

Reissue the command, specifying either SHARE or a blank value for the CLCHNAME attribute.

CSQM523I

csect-name CLUSTER OR CLROUTE CANNOT CURRENTLY BE ALTERED

Severity

8

Explanation

An attempt was made to alter an administered topic that is currently in a named cluster. While a topic is in a cluster it is not permitted to modify the CLROUTE attribute, or to modify the CLUSTER attribute to an alternative cluster name if CLROUTE is set to TOPICHOST.

System action

Processing of the command is terminated.

System programmer response

To alter the CLROUTE or CLUSTER attribute, perform the following actions:

1. Quiesce publish/subscribe messaging for the topic.
2. Remove the topic from the cluster by setting the value of the CLUSTER attribute to blank.
3. Set the CLROUTE and CLUSTER attributes to their required value once the topic has been removed from the cluster.
4. Resume publish/subscribe messaging once the change is visible in the cluster and the queue manager has received proxy subscriptions for any remote subscriptions.

CSQM524I

csect-name CLROUTE CONFLICT DETECTED FOR CLUSTER TOPIC

Severity

8

Explanation

An attempt was made to define a cluster topic but the value of the CLROUTE attribute conflicts with an existing topic, either above or below it, in the topic tree.

System action

Processing of the command is terminated.

System programmer response

Review the cluster routing requirements for the topic tree, then correct and reissue the command.

CSQM525I

csect-name obj-type(obj-name) DOES NOT EXIST OR IS DEFINED INCORRECTLY

Severity
8

Explanation

The queue manager could not complete a requested operation because an object named *obj-name* of type *obj-type* does not exist or is defined incorrectly.

System action

Processing of the operation is terminated.

System programmer response

Check the object has been defined correctly, then try the operation again.

For information on how to define system objects, see Sample definitions supplied with IBM MQ.

If this message has been issued for the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE, and the queue has been defined, ensure it has the following attributes:

- The **USAGE** attribute must be set to **XMITQ**
- The **QSGDISP** attribute must not be **SHARED**
- The **DEFTYPE** attribute must be set to **PERMDYN**
- The **INDXTYPE** attribute must be set to**CORRELID**
- The **SHARE** attribute must be set

CSQM526I

csect-name CERTIFICATE LABEL NOT ALLOWED FOR SSL 3.0 CHANNEL

Severity
8

Explanation

An attempt was made to specify a certificate label for an inbound channel that uses a SSL 3.0 CipherSpec, which is not allowed. Certificate labels for inbound channels are only supported for TLS channels.

System action

Processing of the command is terminated.

System programmer response

If you need to configure a certificate label, alter the channel to use a TLS CipherSpec.

CSQM550I

csect-name Switch of transmission queue for channel *channel-name* from *old-xmitq* to *new-xmitq* started

Severity
0

Explanation

A switch of transmission queue for the channel identified by *channel-name* is required due to a change to the default cluster transmission queue configuration of the queue manager, or to the cluster channel name attribute of a cluster transmission queue. This message is issued by the queue manager when the process of switching the transmission queue from *old-xmitq* to *new-xmitq* is started.

System action

Processing continues.

System programmer response

None.

CSQM551I

csect-name Switch of transmission queue for channel *channel-name* completed - *num-msgs* messages moved from *old-xmitq* to *new-xmitq*

Severity

0

Explanation

A switch of transmission queue for the channel identified by *channel-name* was required due to a change to the default cluster transmission queue configuration of the queue manager, or to the cluster channel name attribute of a cluster transmission queue. This message is issued by the queue manager when the process of switching the transmission queue from *old-xmitq* to *new-xmitq* has completed.

During the switching process the queue manager moved *num-msgs* messages from *old-xmitq* to *new-xmitq*.

System action

Processing continues.

System programmer response

None.

CSQM552E

csect-name Switch of transmission queue for channel *channel-name* from *old-xmitq* to *new-xmitq* failed

Severity

4

Explanation

A switch of transmission queue for the channel identified by *channel-name* is required due to a change to the default cluster transmission queue configuration of the queue manager, or to the cluster channel name attribute of a cluster transmission queue. This message is issued if an error occurs when attempting to start the process of switching the transmission queue from *old-xmitq* to *new-xmitq*.

System action

The process of switching the transmission queue is not started and the channel continues to use the transmission queue *old-xmitq*.

The queue manager will retry to start the switching process the next time the channel starts.

System programmer response

Investigate why the process of switching the transmission queue could not be started, as reported in the preceding messages.

CSQM553I

csect-name Moving messages for channel *channel-name* from transmission queue *old-xmitq* to *new-xmitq*

Severity

0

Explanation

A switch of transmission queue for the channel identified by *channel-name* is required due to a change to the default cluster transmission queue configuration of the queue manager, or to the cluster channel name attribute of a cluster transmission queue. This message is issued when the process of moving messages from the old transmission queue, *old-xmitq*, to the new transmission queue, *new-xmitq*, is started.

System action

Processing continues.

System programmer response

None.

CSQM554I

csect-name Moved *num-msgs* messages for channel *channel-name* from transmission queue *old-xmitq* to *new-xmitq* - *remaining-msgs* messages remaining

Severity
0

Explanation

A switch of transmission queue for the channel identified by *channel-name* is required due to a change to the default cluster transmission queue configuration of the queue manager, or to the cluster channel name attribute of a cluster transmission queue. The switch of transmission queue requires that messages be moved from the old transmission queue, *old-xmitq*, to the new transmission queue, *new-xmitq*. This message is periodically issued to report the progress of this operation.

System action

Processing continues.

System programmer response

If this message is repeatedly issued it might indicate the old transmission queue cannot be drained of messages for the channel, which means the switching process can not complete. Applications continue to put messages to the old transmission queue during the switching process to preserve ordering. If the switching process cannot complete this might indicate that messages are being put to the old transmission queue faster than they can be moved by the switching process, or uncommitted messages remain on the old transmission queue for the channel.

CSQM555E

csect-name Moving of messages for channel *channel-name* from transmission queue *old-xmitq* to *new-xmitq* failed

Severity
8

Explanation

A switch of transmission queue for the channel identified by *channel-name* is required due to a change to the default cluster transmission queue configuration of the queue manager, or to the cluster channel name attribute of a cluster transmission queue. The switch of transmission queue requires that messages for the channel be moved from the old transmission queue, *old-xmitq*, to the new transmission queue, *new-xmitq*. This message is issued if an error occurs while moving these messages.

System action

Moving of messages from the old transmission queue to the new transmission queue is stopped. Any existing messages on the old transmission queue and any new messages put by applications remain on the old transmission queue and are not available to be sent by the cluster-sender channel until action is taken to restart the switching process.

System programmer response

You can use preceding messages to identify and resolve the cause of the error, then restart the switching process by either stopping and starting the channel, or by using the CSQUTIL utility to restart the switching operation.

CSQM556E

csect-name Unable to open transmission queue *xmitq-name* for channel *channel-name*, MQRC=*mqrc (mqrc-text)*

Severity

8

Explanation

The switch of transmission queue requires that messages for the channel be moved from the old transmission queue to the new transmission queue. This message is issued if the old transmission queue, *xmitq-name*, cannot be opened due to reason *mqrc* when attempting to perform this operation (*mqrc-text* provides the MQRC in textual form).

System action

The switching operation fails because the moving of messages from the old transmission queue to the new transmission queue cannot be completed.

System programmer response

You can use the reason code to identify and resolve the cause of the error, then restart the switching process by either stopping and starting the channel, or by using the CSQUTIL utility to restart the switching operation. If the error cannot be resolved, or the old transmission queue has been deleted, the CSQUTIL utility can be used to perform the switching operation without moving messages from the old transmission queue to the new transmission queue. If this option is used it is the responsibility of the IBM MQ administrator to deal with any messages for this channel on the old transmission queue.

CSQM557E

csect-name Unable to open new transmission queue *xmitq-name* for channel *channel-name*, MQRC=*mqrc (mqrc-text)*

Severity

8

Explanation

The switch of transmission queue requires that messages for the channel be moved from the old transmission queue to the new transmission queue. This message is issued if the new transmission queue, *xmitq-name*, cannot be opened due to reason *mqrc* when attempting to perform this operation (*mqrc-text* provides the MQRC in textual form).

System action

The switching operation fails because the moving of messages from the old transmission queue to the new transmission queue cannot be completed.

System programmer response

You can use the reason code to identify and resolve the cause of the error, then restart the switching process by either stopping and starting the channel, or by using the CSQUTIL utility to restart the switching operation.

CSQM558E

csect-name Unable to persist transmission queue state for channel *channel-name*, MQRC=*mqrc (mqrc-text)*

Severity
8

Explanation

The queue manager uses persistent messages on the queue SYSTEM.CHANNEL.SYNCQ to track which transmission queue is used by each cluster-sender channel. This message is issued if state information cannot be updated on this queue due to reason *mqrc* (*mqrc-text* provides the MQRC in textual form).

System action

The operation requiring the persisted transmission queue state to be updated fails.

System programmer response

You can use the reason code to identify and resolve the cause of the error, then review subsequent messages to identify any additional actions that are required.

CSQM559I

csect-name Loading of cluster transmission queue state started

Severity
0

Explanation

The queue manager uses persistent messages on the queue SYSTEM.CHANNEL.SYNCQ to track which transmission queue is used by each cluster-sender channel. This message is issued during queue manager startup to indicate that loading of this information has started.

System action

Processing continues.

System programmer response

None.

CSQM560I

csect-name Loading of cluster transmission queue state completed - *num-records* records processed

Severity
0

Explanation

The queue manager uses persistent messages on the queue SYSTEM.CHANNEL.SYNCQ to track which transmission queue is used by each cluster-sender channel. This message is issued during queue manager startup to indicate loading of this information has completed. The number of cluster-sender channel records that were processed is identified by *num-records*.

System action

Processing continues.

System programmer response

None.

CSQM561E

csect-name Loading of cluster transmission queue state failed

Severity
8

Explanation

The queue manager uses persistent messages on the queue SYSTEM.CHANNEL.SYNCQ to track which transmission queue is used by each cluster-sender channel. This message is issued during queue manager startup to indicate that an error has occurred when loading this information.

System action

Processing continues with restricted clustering function. The queue manager is unable to determine which transmission queue should be used by each cluster-sender channel, so these channels are unable to start. Any requests to put a message to a remote cluster queue will fail with MQRC_CLUSTER_RESOURCE_ERROR.

System programmer response

Investigate why the transmission queue state information could not be loaded, as reported in preceding messages. Resolve the error, then restart the queue manager to restore clustering function. If you are unable to resolve the error contact your IBM support center.

CSQM562E

csect-name Duplicate cluster transmission queue record found for channel *channel-name*

Severity
8

Explanation

The queue manager uses persistent messages on the queue SYSTEM.CHANNEL.SYNCQ to track which transmission queue is used by each cluster-sender channel. This message is issued during queue manager startup if a duplicate record is found for a channel.

System action

The duplicate record is ignored and processing continues, but the channel may use an incorrect transmission queue if the duplicated record should not have been used.

System programmer response

This condition should not occur. Contact your IBM support center.

CSQM563E

csect-name Failed to create dynamic cluster transmission queue *xmitq-name*, MQRC=*mqrc (mqrc-text)*

Severity
4

Explanation

A channel is required to switch to a permanent-dynamic transmission queue due to a change to the default cluster transmission queue configuration of the queue manager, or to the cluster channel name attribute of a cluster transmission queue. The queue manager could not create the permanent-dynamic transmission queue, *xmitq-name*, due to reason *mqrc*.

The cluster-sender channel that is affected can be identified from the name of the transmission queue because the queue name is in the format SYSTEM.CLUSTER.TRANSMIT*channel_name*.

System action

Processing continues.

System programmer response

You can use the reason code to identify and resolve the error that has prevented the permanent-dynamic cluster transmission queue from being created. Additional messages might be issued to provide further information. If the reason code is MQRC_UNKNOWN_OBJECT_NAME this means the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE has not been defined. The definition for this model queue can be found in the supplied sample **CSQ4INSX**.

Refer to API completion and reason codes for information about *mqrc* (*mqrc-text*provides the MQRC in textual form).

CSQM564E

csect-name Cluster transmission model queue *model-xmitq* has incorrect attributes

Severity

4

Explanation

The queue manager failed to create a permanent-dynamic transmission queue for a cluster-sender channel because the model queue *model-xmitq* has been defined incorrectly.

The model queue must have the following attributes:

- The **USAGE** attribute must be set to **XMITQ**
- The **QSGDISP** attribute must not be **SHARED**
- The **DEFTYPE** attribute must be set to **PERMDYN**
- The **INDXTYPE** attribute must be set to**CORRELID**
- The **SHARE** attribute must be set

System action

Processing continues.

System programmer response

Review and correct the definition of the model transmission queue. The definition for the model queue can be found in the supplied sample **CSQ4INSX**.

CSQM565E

csect-name Delivery delay processor initialization failed, reason*reason-code*

Severity

8

Explanation

Initialization of the delivery delay processor task failed with the specified *reason-code*. As a result, any messages sent with delivery delay, using JMS 2.0, will not be processed and will be left on the delivery delay staging queue.

System action

The delivery delay processor task will end and will not restart. Messages can still be sent to the delivery delay staging queue by JMS 2.0 applications, however, the messages will not be processed until the delivery delay task is restarted. See system programmer response for instructions on how to restart the delivery delay processor.

System programmer response

The most likely reason for this message is a shortage of storage below the bar, in which case *reason-code* will be 4. Review the amount of storage used below the bar, and if possible try and reduce it. You can attempt to restart the delivery delay processor by altering the delivery delay staging queue state from 'get enabled' to 'get inhibited', and back to the 'get enabled' state again.

CSQM566I

csect-name Delivery delay processor started

Severity
0

Explanation

The delivery delay processor has started and is available to process messages from the delivery delay staging queue.

System action

Processing continues.

System programmer response

None.

CSQM567I

csect-name Delivery delay processor stopped

Severity
0

Explanation

The delivery delay processor has stopped and is no longer available to process messages from the delivery delay staging queue. This message is output in the following situations:

- The queue manager is shutting down.
- The delivery delay staging queue has been deleted, or does not exist.

System action

Processing continues.

System programmer response

None.

CSQM568E

csect-name Delivery delay processor ended abnormally, MQRC=*mqrc*

Severity
4

Explanation

The delivery delay processor has detected an error, indicated by *mqrc*, and has shut down.

System action

The delivery delay processor task ends and will not restart. Messages can still be sent to the delivery delay staging queue by JMS 2 applications, however, they will not be processed until the delivery delay task is restarted. See system programmer response for instructions on how to restart the delivery delay processor.

System programmer response

This message is output for many reasons, some of which will be expected and some will not. For example, if the delivery delay staging queue state is altered to 'get inhibited' this message will be output, and *mqrc* will be *MQRC_GET_INHIBITED*. If the message is expected then no action is required. If the message is unexpected use the value of*mqrc*, and any other messages to attempt to rectify the situation. You can attempt to restart the delivery delay processor by altering the delivery delay staging queue state from 'get enabled' to' get inhibited', and back to the 'get enabled' state again.

CSQM569I

csect-name Delivery delay processor failed to get a message with correlation ID *correlid*, MQRC=*mqrc* (*mqrc-text*)

Severity

4

Explanation

The delivery delay processor attempted to perform a destructive MQGET for the message with the specified correlation ID from the delivery delay staging queue, SYSTEM.DDELAY.LOCAL.QUEUE. The message was no longer on the queue.

System action

Processing continues.

System programmer response

Investigate whether the message was taken off the delivery delay staging queue for a valid reason, for example, it was put there by mistake. Validate the security settings for the delivery delay staging queue to ensure that only authorized users have access to it.

CSQM570E

csect-name Delivery delay processor failed to process a message with correlation ID *correlid*, for queue *q-name*, according to its report options *report-options*, MQRC=*mqrc* (*mqrc-text*)

Severity

8

Explanation

The delivery delay processor could not put the specified message to the specified queue and, so, was attempting to either send the message to the dead-letter-queue or discard the message according to the disposition options specified in the report field of the message.

The message might have also requested an exception report. Some part of the processing of the disposition options, or the report, failed with the specified return code.

System action

The message is rolled back to the delivery delay staging queue, SYSTEM.DDELAY.LOCAL.QUEUE, and reprocessed at a time decided by the system.

System programmer response

Use the information from the message to establish the cause of the problem. Some possible explanations are:

- The specified queue might no longer exist, be full, or be put disabled.
- If the message should have been put to the dead-letter-queue, check that the dead-letter-queue is defined, is not full, and is put enabled.
- If an exception report message was to be generated, check that the queue the report was to be put to is defined, is not full, is put enabled, and that the user ID in the message has access to the queue.

Otherwise, check that the dead-letter-queue is defined, is not full, and is put enabled.

CSQM571I

csect-name Delivery delay processor received an unexpected message with message ID *msgid*

Severity

4

Explanation

The delivery delay processor received a message from the delivery delay staging queue, SYSTEM.DDELAY.LOCAL.QUEUE, that was not in the correct format.

The delivery delay processor either sends the message to the dead-letter-queue, or discards it according to the disposition options specified in the report field of the message.

System action

The delivery delay processor either sends the message to the dead-letter-queue, or discards it according to the disposition options specified in the report field of the message. If the message specified an exception report, this will be generated.

System programmer response

Investigate why unexpected messages are being sent to the delivery delay staging queue. Validate the security settings for the delivery delay staging queue to ensure that only authorized users have access to it.

V 9.0.3

CSQM572E

Required key *key* is missing from stanza *name* in CSQMQIN DD card

Severity
8

Explanation

The expected key is not present in the stanza. The service relating to the stanza will not start.

System action

Correct the configuration in the CSQMQIN DD card and restart the queue manager.

V 9.0.3

CSQM573E

Unable to parse line *number* in CSQMQIN DD card

Severity
8

Explanation

The queue manager cannot parse the line in the CSQQMIN DD card. The service relating to the stanza will not start. Possible causes are:

- A missing continuation character.
- The line is too long.

System action
Processing continues.

System programmer response

Correct the configuration in the CSQMQIN DD card and restart the queue manager.

V 9.0.3

CSQM574E

csect-name Invalid value *value* for key *key* in stanza *stanza* in CSQMQMIN DD card around line *line*

Severity
8

Explanation

The specified value is not valid for the specified key. The service relating to the stanza will not start.

V 9.0.4 Possible causes are:

- The `serviceProxy` in the `ReportingService` stanza does not begin with http://
- The `serviceProxy` in the `ReportingService` stanza specifies a port, but the port is not valid.

System action

Processing continues.

System programmer response

Correct the configuration in the CSQMQIN DD card and restart the queue manager.

V 9.0.3

CSQM575E

V 9.0.4

csect-name Invalid or duplicate key *key* in stanza *stanza* in CSQMQMIN DD card around line *line*

Severity
8

Explanation

The key *key* in stanza *stanza* is not recognized by the queue manager. The service relating to the stanza will not start.

System action

Processing continues.

System programmer response

Correct the configuration in the CSQMQIN DD card and restart the queue manager.

V 9.0.3

CSQM576E

No data will be sent to the IBM Cloud Product Insights service

Severity
8

Explanation

The queue manager is configured to send data to the IBM Cloud Product Insights service, and an error has occurred.

System action

The queue manager will not attempt to send any further data to the IBM Cloud Product Insights service

System programmer response

Review the queue manager log for related messages. Correct any issues and restart the queue manager.

V 9.0.3

CSQM577E

csect-name MQPUT FAILED FOR QUEUE *q-name*, REASON=*mqrc*

Severity
8

Explanation

The queue manager is configured to send data to the IBM Cloud Product Insights service, and an error has occurred.

An attempt to put a message to the named queue for internal processing failed for the specified reason.

System action

The internal processing will not occur. The associated service might be stopped, or the MQPUT call might be retried.

System programmer response

Refer to API reason codes for more information about the return code.

V 9.0.4

CSQM578I

DD card CSQMQINI read successfully.

Severity
10

Explanation

The CSQMQINI DD card has been read successfully.

System action

None

System programmer response

None.

CSQM999E

csect-name UNRECOGNIZED RETURN CODE *ret-code* FOR *'keyword'*

Severity
8

Explanation

An unexpected return code was issued from a command, relating to the named keyword.

System action

The command is ignored.

System programmer response

Note the return code *ret-code* (which is shown in hexadecimal) and contact your IBM support center.

Command server messages (CSQN...): z/OS

CSQN001I

COMMAND SERVER STARTED

Severity

0

Explanation

A request to start the command server with the START CMDSERV command has been accepted.

System action

The command server is triggered to start.

CSQN002I

COMMAND SERVER ALREADY STARTED

Severity

0

Explanation

A START CMDSERV command has been entered, but the command server is already running.

System action

The command is ignored.

CSQN003I

COMMAND SERVER ENABLED

Severity

0

Explanation

In response to a START CMDSERV command in an initialization file, the command server has been put in to an enabled state.

System action

The command server will be started automatically when initialization finishes.

CSQN004I

COMMAND SERVER ALREADY ENABLED

Severity

0

Explanation

A START CMDSERV command has been entered, but the command server was already enabled.

System action

The command is ignored.

CSQN005I

COMMAND SERVER STOPPED

Severity

0

Explanation

A request to stop the command server with a STOP CMDSERV command has been accepted.

System action

The command server shuts down when it finishes processing the current command (or immediately if it is not processing a command). This message is followed by message CSQN201I to confirm that the stop has started.

CSQN006I

COMMAND SERVER ALREADY STOPPED

Severity

0

Explanation

A STOP CMDSERV command was entered, but the command server was not running.

System action

The command is ignored.

CSQN007I

COMMAND SERVER DISABLED

Severity

0

Explanation

In response to a STOP CMDSERV command in an initialization file, the command server has been put in to a disabled state.

System action

The command server will not start automatically when initialization finishes.

CSQN008I

COMMAND SERVER ALREADY DISABLED

Severity

0

Explanation

A STOP CMDSERV command has been entered, but the command server was already disabled.

System action

The command is ignored.

CSQN009I

csect-nameverb-namepkw-name COMMAND DISABLED

Severity

4

Explanation

The command was not processed because it was not allowed during this stage of initialization or termination. *verb-name* might include the command prefix (CPF). This depends on how the command was entered.

System action

The command is ignored.

CSQN011I

COMMAND SERVER STATUS IS ENABLED

Severity

0

Explanation

The command server is in an enabled state; that is, the command server will be started automatically when initialization finishes.

CSQN012I

COMMAND SERVER STATUS IS DISABLED

Severity

0

Explanation

The command server is in a disabled state; that is, the command server will not be started automatically when initialization finishes.

CSQN013I

COMMAND SERVER STATUS IS RUNNING

Severity

0

Explanation

The command server is in a running state; that is, the command server is currently processing a command.

CSQN014I

COMMAND SERVER STATUS IS WAITING

Severity

0

Explanation

The command server is in a waiting state; that is, the command server is waiting for a message to be put onto the system-command input queue.

CSQN015I

COMMAND SERVER STATUS IS STOPPED

Severity

0

Explanation

The command server is in a stopped state; that is, the command server will not process any commands until a START CMDSERV command is entered.

CSQN016I

COMMAND SERVER STATUS IS STARTING

Severity

0

Explanation

The command server is in a starting state; that is, a START CMDSERV command has been entered, but the command server has not yet started up.

CSQN017I

COMMAND SERVER STATUS IS STOPPING

Severity

0

Explanation

The command server is in a stopping state; that is, a STOP CMDSERV command has been entered, but the command server has not yet stopped.

CSQN018E

csect-name INTERNAL ERROR FOR *identifier*, RETURN CODE=*rc*

Severity

8

Explanation

This message could be caused by the following:

Identifier

Description

INSSRV01

During the early part of initialization, the queue manager was unable to start the task that processes commands in CSQINP1.

INSSRV02

During the later part of initialization, the queue manager was unable to start the task that processes commands in CSQINP2.

RTSSRV01

After initialization has completed with the command server enabled, or in response to a START CMDSERV command, the queue manager was unable to start the command server task that processes commands in the system-command input queue.

GRSSRV01

After initialization has completed with the command server enabled, or in response to a START CMDSERV command, the queue manager was unable to start the command server task that processes commands using CMDSCOPE.

System action

The task is not started.

System programmer response

Stop and restart the queue manager. Check the console for other messages regarding this error, and note the message number, *identifier*, and *rc*. Also collect the system dump (if one was produced). Contact your IBM support center to report the problem.

CSQN019E

csect-name INTERNAL ERROR FOR *identifier*, RETURN CODE=*rc*

Severity

8

Explanation

This message could be caused by the following:

Identifier
Description

INSSRV01
During the early part of initialization an error occurred when trying to delete the task that processes commands in CSQINP1.

INSSRV02
During the later part of initialization an error occurred when trying to delete the task that processes commands in CSQINP2.

RTSSRV01
During termination with the command server running, or in response to a START CMDSERV command, an error occurred when trying to delete the command server task that processes commands in the system-command input queue.

GRSSRV01
During termination with the command server running, or in response to a START CMDSERV command, an error occurred when trying to delete the command server task that processes commands using CMDSCOPE.

System action

If the value of *identifier* was INSSRV01 or INSSRV02, the error is ignored, and startup continues.

If the value of *identifier* was RTSSRV01 or GRSSRV01 and *csect-name* was CSQNESTP, the command server could have terminated while processing a command.

System programmer response

Check the console for other messages regarding this error. If you are unable to resolve the problem, note the message number, *identifier*, and *rc*, collect the system dump (if one was produced), and contact your IBM support center.

CSQN020E

csect-name UNABLE TO START COMMAND SERVER *identifier*

Severity
8

Explanation

csect-name was unable to start the command server task *identifier*.

System action

If *identifier* is INSSRV01 or INSSRV02, initialization is not completed and a dump might be produced. In other cases, the command server is not started.

System programmer response

Stop and restart the queue manager. Contact your IBM support center with details of this message, any previous messages pertaining to this error, and the dump (if applicable).

CSQN021E

csect-name COMMAND SERVER *identifier* ABNORMAL COMPLETION

Severity
8

Explanation

The command server task *identifier* was unable to complete its processing during startup.

System action

Queue manager startup continues.

System programmer response

Check the z/OS console for related messages (probably concerning the CSQINPx data sets). The CSQOUTx data sets should also be checked to determine how much command processing was done before the error occurred. If required, reissue any unprocessed commands, or resolve the problem and restart the queue manager.

CSQN100I

COMMAND EXCEEDS MAXIMUM SIZE, COMMAND IGNORED

Severity

4

Explanation

The command string was too long.

System action

The command is ignored, and processing of CSQINP1 or CSQINP2 continues.

System programmer response

The command in question precedes this message in the CSQOUT1 or CSQOUT2 data set. For details about forming a command string, see Initialization commands.

CSQN101I

COMMAND ENDS WITH A CONTINUATION MARK, COMMAND IGNORED

Severity

4

Explanation

The last command in the CSQINP1 or CSQINP2 data set ended with a continuation mark.

System action

The command is ignored.

System programmer response

The command in question precedes this message in the CSQOUT1 or CSQOUT2 data set. For details about forming a command string, see Initialization commands.

CSQN102I

COMMAND BUFFER INVALID, ERROR UNKNOWN, COMMAND IGNORED

Severity

4

Explanation

An internal error has occurred.

System action

This command is ignored, and the next command is processed.

System programmer response

The command in question precedes this message in the CSQOUT1 or CSQOUT2 data set. If you are unable to solve the problem, contact your IBM support center.

CSQN103I

COMMAND PROCESSOR RETURN CODE=*rc*, REASON CODE=*reason*

Severity

4

Explanation

An error occurred while processing the command preceding this message in the CSQOUT1 or CSQOUT2 data set. The possible values of *rc* are as follows:

Return code
Description

00000004
Internal error

00000008
Syntax or command preprocessor error, see the following lines in the CSQOUTx data set

0000000C
Command processor error, see the following lines in the CSQOUTx data set

00000010
Command processor abnormal termination

00000014
Command completed, but there is insufficient storage for the messages

00000018
Command preprocessor has insufficient storage (there could be further messages about this error)

0000001C
The command processor has insufficient storage (the command could be partially completed)

00000020
Security check

00D50102
See "Command server codes (X'D5')" on page 5112

Note: If the return code is '00000010', the reason code has no meaning.

If *reason* is 00000004 and *return code* is 00000000, the command has been accepted and will be completed later. Further messages will be produced when the command has been completed.

Otherwise the reason code indicates the command result as follows:

Reason
Description

00000000
Command completed

00000004
Partial completion

00000008
Command not actioned

0000000C
Command processor abend

FFFFFFFF
Command not actioned

System action

The next command is processed, if possible.

System programmer response

If *reason* indicates that the command did not complete, examine the command and all associated messages. See IBM MQ for z/OS messages, completion, and reason codes for further information about the commands.

If you are unable to solve the problem, collect the input and output data sets and contact your IBM support center.

CSQN104I

INITIALIZATION RETURN CODE=*rc*, REASON CODE=*reason*

Severity

8

Explanation

An error occurred while processing one of the initialization data sets.

System action

The system action depends on the reason code (*reason*). See "Command server codes (X'D5')" on page 5112 for information the code you have received.

System programmer response

The response you should make depends on the reason code (*reason*). See "Command server codes (X'D5')" on page 5112 for information about the code you have received.

CSQN105I

Commands from *ddname* for queue manager *qmgr-name* - *date time*

Severity

0

Explanation

This message forms the header for the output data sets CSQOUT1 and CSQOUT2.

CSQN121I

'*verb-namepkw-name*' command responses from *qmgr-name*

Explanation

The following messages are responses from queue manager *qmgr-name* to the indicated command - either entered or generated by another command - that specified CMDSCOPE.

CSQN122I

'*verb-namepkw-name*' command for CMDSCOPE(*qmgr-name*) normal completion

Explanation

Processing for the indicated command that specified CMDSCOPE(*qmgr-name*) - either entered or generated by another command - has completed successfully on all requested queue managers.

CSQN123E

'*verb-name pkw-name*' command for CMDSCOPE(*qmgr-name*) abnormal completion

Explanation

Processing for the indicated command that specified CMDSCOPE(*qmgr-name*) - either entered or generated by another command - has completed, but not successfully. If the command was sent to more than one queue manager, it might have completed successfully on some and not on others.

System programmer response

Examine the preceding responses from the command. Reissue the command correctly if necessary for the queue managers where it failed.

CSQN127E

Queue-sharing group error, reason=*reason*

Severity
8

Explanation

While processing a command that specified CMDSCOPE, the command server experienced an error while trying to send data to the coupling facility.

System action

The command is not processed.

System programmer response

The response you should make depends on the reason code (*reason*). See "Coupling Facility codes (X'C5')" on page 5023 for information about the code.

CSQN128E
Insufficient storage for CMDSCOPE(*qmgr-name*)

Explanation

While processing a command that specified CMDSCOPE, the command server was unable to obtain storage needed.

System action

The command is not processed.

System programmer response

If the problem persists, you might need to restart the queue manager after making more storage available.

CSQN129E

Error saving command reply information

Severity
8

Explanation

While processing a command that specified CMDSCOPE or a command for the channel initiator, the command server experienced an error while trying to save information about the command.

System action

The command is not processed.

System programmer response

The most likely cause is insufficient storage. If the problem persists, you may need to restart the queue manager after making more storage available.

CSQN130E
Command exceeds maximum size for CMDSCOPE(*qmgr-name*)

Explanation

A command that specified CMDSCOPE(*qmgr-name*) was too long.

System action

The command is not processed.

System programmer response

Reissue the command correctly.

CSQN131E
CMDSCOPE(qmgr-name) not allowed during restart

Explanation

A command that specified CMDSCOPE(*qmgr-name*) was issued in the initialization input data set CSQINP1. This is not allowed.

System action

The command is not processed.

System programmer response

Reissue the command later.

CSQN132E
CMDSCOPE(*qmgr-name*) not allowed with disposition *disposition*

Explanation

A command that specified CMDSCOPE(*qmgr-name*) with QSGDISP(*disposition*) or CHLDISP(*disposition*) was issued. This combination of values is not allowed.

System action

The command is not processed.

System programmer response

Reissue the command correctly.

CSQN133E
CMDSCOPE(*qmgr-name*) not allowed, command server unavailable

Explanation

A command that specified CMDSCOPE(*qmgr-name*) was entered or generated by another command, but the command server is not running and not enabled.

System action

The command is not processed.

System programmer response

Use the START CMDSERV command to start the command server, and reissue the command.

CSQN135E
Queue manager *qmgr-name* not active in queue-sharing group

Explanation

A command specifying CMDSCOPE(*qmgr-name*) was entered or generated by another command, but that queue manager is not currently active in the group.

System action

The command is not processed.

System programmer response

Start the queue manager and reissue the command if required.

CSQN136E
Not in queue-sharing group

Explanation

A command that requires a queue-sharing group was entered, but the queue manager is not in a group.

System action

The command is not processed.

System programmer response

Reissue the command correctly.

CSQN137I
'*verb-name pkw-name*' accepted for CMDSCOPE(*qmgr-name*), sent to *n*

Explanation

A command that specified CMDSCOPE was entered. It has been passed to the requested queue manager(s) for processing; *n* is the number of queue managers.

System action

Processing continues.

CSQN138I
'*verb-name pkw-name*' generated for CMDSCOPE(*qmgr-name*), sent to *n*

Explanation

A command that specified CMDSCOPE was generated in response to the command originally entered. It has been passed to the indicated queue manager(s) for processing; *n* is the number of queue managers.

System action

Processing continues.

CSQN201I

COMMAND SERVER IS SHUTTING DOWN

Severity
0

Explanation

This message confirms that the command server is shutting down after an error.

System action

The command server shuts down and will not process any more commands.

System programmer response

Correct the errors reported in the preceding messages, and use the START CMDSERV command to restart the command server.

CSQN202I

COMMAND SERVER RETURN CODE=*rc*, REASON=*reason*

Severity
8

Explanation

An error occurred in the command server, as indicated by the preceding messages.

System action

The system action depends on the reason code (*reason*). See "Command server codes (X'D5')" on page 5112 or "Coupling Facility codes (X'C5')" on page 5023 for information about the code.

System programmer response

The response you should make depends on the reason code (*reason*).

The return code *rc* is dependent on *reason*, and is of use to IBM service personnel.

CSQN203I

QUEUE *queuename*, MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Severity
8

Explanation

An API call, as indicated in the preceding message, did not complete successfully. The completion code is *mqcc*, and the reason code is *mqrc* (*mqrc-text*provides the MQRC in textual form).

System programmer response

See API completion and reason codes for information about completion codes and reason codes.

If you are unable to resolve the problem, note the numbers of any messages and codes associated with the error, and contact your IBM support center.

Reason codes above 8000 are internal queue manager error codes. If such a code persists, report it to your IBM support centre.

CSQN205I

COUNT=*count*, RETURN=*rc*, REASON=*reason*

Severity
0

Explanation

This message reports the results from the command processor (refer to the section Writing programs to administer IBM MQ for further information). *count* is the number of messages (including this one) to be written to the reply-to queue in response to the command. Possible values of *rc* are as follows:

Return code
Description

00000000
Normal completion

00000004
Internal error

00000008
Syntax or command preprocessor error, see the following messages

0000000C
Command processor error, see the following messages

00000010
Command processor abnormal termination

00000014
Command completed, but there is insufficient storage for the messages

00000018
Command preprocessor has insufficient storage, (there could be further messages about this error)

0000001C
The command processor has insufficient storage (the command could be partially completed)

00000020
Security check, check userid authority

00000024
Command too long, see the following messages

00000028
Queue-sharing group error, see the following messages

00D5xxxx
See "Command server codes (X'D5')" on page 5112

Note: If the return code is '00000010', the reason code has no meaning.

If *reason* is 00000004 and *return code* is 00000000, the set of reply messages is incomplete. Further sets of messages, each including another CSQN205I message, will be produced later. The results of the command will be shown by the codes in the CSQN205I message included with the final set of messages.

Otherwise the reason code indicates the command result as follows:

Reason
Description

00000000
Command completed

00000004
Partial completion

00000008
Command not actioned

0000000C
Command processor abend

FFFFFFFF
Command not actioned

System action

The next command is processed, if possible.

System programmer response

If *reason* indicates that the command did not complete, examine the command and all associated messages. See IBM MQ for z/OS messages, completion, and reason codes for further information about the commands.

If you are unable to solve the problem, collect the input and output data sets and contact your IBM support center.

CSQN206I

COMMAND SERVER ECBLIST, STOP=*ecb1*, WAIT=*ecb2*

Severity

8

Explanation

This message reports the ECB values associated with an error in the command server.

System action

The command server terminates.

System programmer response

This message is usually preceded by a CSQN202I message. Refer to the preceding messages for more information about the cause of the problem.

CSQN207I

COMMAND SERVER UNABLE TO OPEN REPLY TO QUEUE

Explanation

The command server was unable to open the reply-to queue while processing a command.

System action

Message CSQN203I is sent to the z/OS console reporting the completion and reason codes from the `MQOPEN` request. The command responses are discarded.

System programmer response

See API completion and reason codes for information about the completion and reason codes. Use this information to solve the problem, and restart the command server. If this does not help you to solve the problem, collect the following items, and contact your IBM support center.

- Return and reason codes from the message produced
- Any trace information collected

CSQN208E

COMMAND SERVER UNABLE TO OPEN COMMAND INPUT QUEUE

Explanation

The command server was unable to open the system-command input queue while starting.

System action

Message CSQN203I is sent to the z/OS console reporting the completion and reason codes from the `MQOPEN` request. The command server stops, without processing any commands.

System programmer response

See API completion and reason codes for information about the completion and reason codes. Use this information to solve the problem, and restart the command server. If this does not help you to solve the problem, collect the following items, and contact your IBM support center.

- Return and reason codes from the message produced
- Any trace information collected

CSQN209E

COMMAND SERVER ERROR CLOSING COMMAND INPUT QUEUE

Explanation

While the command server was shutting down, an error occurred when closing the system-command input queue.

System action

Message CSQN203I is sent to the z/OS console reporting the completion and reason codes from the `MQCLOSE` request. The shutdown procedure continues.

System programmer response

See API completion and reason codes for information about the completion and reason codes. If this does not help you to solve the problem, collect the following items, and contact your IBM support center:

- Return and reason codes from the message produced
- Any trace information collected

CSQN210E
COMMAND SERVER ERROR CLOSING REPLY TO QUEUE

Explanation

The command server was unable to close the reply-to queue while processing a command.

System action

Message CSQN203I is sent to the z/OS console reporting the completion and reason codes from the `MQCLOSE` request.

The command server continues.

System programmer response

See API completion and reason codes for information about the completion and reason codes.

CSQN211E
COMMAND SERVER ERROR GETTING FROM COMMAND INPUT QUEUE

Explanation

The command server experienced an error while trying to get a message from the system-command input queue.

System action

Message CSQN203I is sent to the z/OS console, reporting the completion and reason codes from the `MQGET` request.

The command server terminates.

System programmer response

See API completion and reason codes for information about the completion and reason codes. Use this information to solve the problem, and restart the command server. If this does not help you to solve the problem, collect the following items, and contact your IBM support center:

- Return and reason codes from the console message
- Any trace information collected

CSQN212E
COMMAND SERVER ERROR PUTTING TO REPLY TO QUEUE

Explanation

The command server was unable to put a response message onto a reply-to queue while processing a command.

System action

Message CSQN203I is sent to the z/OS console reporting the completion and reason codes from the MQPUT request. If possible, the command server sends the response message to the dead-letter queue, otherwise the response is discarded.

The command server continues.

System programmer response

See API completion and reason codes for information about the completion and reason codes. If this does not help you to solve the problem, collect the following items, and contact your IBM support center:

- Return and reason codes from the message produced
- Any trace information collected

CSQN213E
COMMAND SERVER ERROR, COMMAND INPUT QUEUE DISABLED

Explanation

While waiting for a command the system-command input queue has been disabled.

System action

Message CSQN203I is sent to the console containing the return and reason codes from the request function. The command server terminates.

System programmer response

Change the system-command input queue to be enabled, and issue the START CMDSERV command.

If the problem persists, collect the following items, and contact your IBM support center:

- Return and reason codes
- Any trace data collected
- Printout of SYS1.LOGREC

CSQN219E

Unable to find command reply information

Severity
8

Explanation

While processing responses from a command that specified CMDSCOPE or a command for the channel initiator, the command server could not find the information to determine where to send the responses.

System action

The command might not be processed; any command responses are discarded. The command server continues.

System programmer response

If the problem persists, contact your IBM support center with details of this message, any previous messages pertaining to this error, and the dump (if applicable).

CSQN220E
Error monitoring CMDSCOPE command data

Explanation

The command server experienced an error while monitoring command data in the coupling facility.

System action

Message CSQN202I is sent to the z/OS console, reporting the return and reason codes from the request.

The command server terminates.

System programmer response

See "Coupling Facility codes (X'C5')" on page 5023 for information about the reason code. Use this information to solve the problem, and restart the command server. If this does not help you to solve the problem, collect the following items, and contact your IBM support center:

- Return and reason codes from the console message
- Any trace information collected

CSQN221E
Error receiving CMDSCOPE command data

Explanation

The command server experienced an error while trying to get command data from the coupling facility.

System action

Message CSQN202I is sent to the z/OS console, reporting the return and reason codes from the request.

The command server terminates.

System programmer response

See "Coupling Facility codes (X'C5')" on page 5023 for information about the reason code. Use this information to solve the problem, and restart the command server. If this does not help you to solve the problem, collect the following items, and contact your IBM support center:

- Return and reason codes from the console message
- Any trace information collected

CSQN222E
Error sending CMDSCOPE command data

Explanation

The command server experienced an error while trying to send command data to the coupling facility.

System action

Message CSQN202I is sent to the z/OS console, reporting the return and reason codes from the request.

The command server terminates.

System programmer response

See "Coupling Facility codes (X'C5')" on page 5023 for information about the reason code. Use this information to solve the problem, and restart the command server. If this does not help you to solve the problem, collect the following items, and contact your IBM support center:

- Return and reason codes from the console message
- Any trace information collected

CSQN223E
Insufficient storage for CMDSCOPE command data

Explanation

The command server was unable to obtain storage needed for command data in the coupling facility.

System action

The command server terminates.

System programmer response

Use the START CMDSERV command to restart the command server. If the problem persists, you might need to restart the queue manager after making more storage available.

CSQN224E

GROUP COMMAND SERVER ENDED ABNORMALLY. RESTARTING

Severity

8

Explanation

The Group Command Server has ended abnormally because a severe error occurred.

System action

The Group Command Server is automatically restarted.

System programmer response

Investigate the reason for abnormal termination. If the problem persists contact your IBM support center.

Operations and control messages (CSQO...): z/OS

CSQO001I

'*' may only be final character.

Severity

8

Explanation

A character string entered in the Name field contains an asterisk character that is not in the last position. This is not allowed.

System action

The main menu is redisplayed.

CSQO002I

Action *action* is not allowed.

Severity

8

Explanation

An incorrect action number was entered in the action code field. The number must be in the range shown on the panel.

System action

The panel is redisplayed.

CSQO003I

Use the ISPF command PFSHOW to display F-key settings

Severity
0

Explanation

On entry to Operations and Control, F-key settings are not being displayed. This tells you how to display the settings; you need to use F-keys to use the Operations and Control panels.

System action

None.

CSQO004I

Object *object-type* is not allowed.

Severity
8

Explanation

The value entered in the Object type field was invalid.

System action

The main menu is redisplayed.

CSQO005I

Multiple replies returned. Press F10 to view.

Severity
4

Explanation

Several error messages were returned by the queue manager in response to an action from Operations and Control.

System action

The main menu is redisplayed.

CSQO006I

Blank name is not allowed with action queue manager *.

Severity
8

Explanation

The Define action was selected and the Name field was left blank to define a new object using default attributes. However, an asterisk (*) was entered for the action queue manager, which is not allowed in this case.

System action

The main menu is redisplayed.

CSQO007I

'*field*' must be supplied.

Severity
8

Explanation

Nothing was entered in the named field. This value is required to continue.

System action

The current panel is displayed again.

CSQO008I

F-key is not active.

Severity
4

Explanation

A function key that is not currently available was pressed.

System action

The current panel is redisplayed.

CSQO009I

Action *action* is not allowed for object type *object-type*.

Severity
8

Explanation

The action number that you entered is not allowed for *object-type* objects.

System action

The current panel is redisplayed.

CSQO010I

Queue manager or group is not available.

Severity
8

Explanation

An attempt to connect to a queue manager was unsuccessful. If a queue manager name was specified, the queue manager is not running. If a queue-sharing group name was specified, there are no queue managers running in that group.

System action

None, the panel is redisplayed.

CSQO011E

MQCONN unsuccessful. Reason code=*mqrc*.

Severity
8

Explanation

An attempt to connect to a queue manager or queue-sharing group was unsuccessful for one of the following reasons:

1. Insufficient storage is available
2. A severe error has occurred

System action

None, the panel is redisplayed.

System programmer response

Refer to API completion and reason codes for information about *mqrc*.

CSQO012I

Connect name is invalid or unknown.

Severity

8

Explanation

An attempt to connect to a queue manager or queue-sharing group was unsuccessful because the name specified is not known, or not valid. If a blank name was specified, this means that there was no default queue manager or group defined for your installation.

System action

None, the panel is redisplayed.

CSQO013I

Not authorized to use queue manager.

Severity

8

Explanation

An attempt to connect to a queue manager was unsuccessful because the connection security failed, or you are not authorized to do so.

System action

None, the panel is redisplayed.

CSQO014E

MQOPEN of *q-name* unsuccessful. Reason code=*mqrc*.

Severity

8

Explanation

An attempt to open *q-name* was unsuccessful. *mqrc* is the reason code returned by `MQOPEN`; seeAPI completion and reason codes for more information. *q-name* is one of the following:

- SYSTEM.COMMAND.INPUT
- SYSTEM.COMMAND.REPLY.MODEL; the requested dynamic queue name is appended in parentheses.
- The name of a transmission queue (if you are attempting to send commands to a remote system)

Likely causes of this problem are:

- One or both of the required queues is not defined on the queue manager that you have connected to.
- An attempt was made to send commands to a remote system, but no transport queue is defined.
- You are not authorized to open one of the required queues. If the message indicates that it is the SYSTEM.COMMAND.REPLY.MODEL queue that you are not authorized to open, it could be that you are not authorized to open the SYSTEM.CSQOREXX.* dynamic queue.
- There is insufficient storage available.

System action

The main menu is redisplayed.

CSQO015E

Command issued but no reply received.

Severity

8

Explanation

The reply to a command could not be retrieved from the reply-to queue using MQGET because the response wait time was exceeded.

System action

The panel is redisplayed. The command was sent to the queue manager, but it might not have been executed successfully.

CSQO016E

MQPUT to *q-name* unsuccessful. Reason code=*mqrc*.

Severity

8

Explanation

An attempt to put a command on a queue (*q-name*) using MQPUT was unsuccessful. *q-name* is the name of either the system-command input queue, or a transmission queue if you are sending commands to a remote queue manager. *mqrc* is the reason code returned from MQPUT; see API completion and reason codes for more information.

The most likely causes of this problem are:

1. Put requests are inhibited for the system-command input queue or the transmission queue.
2. The system-command input queue or transmission queue is full, because the command server is not running.
3. There is insufficient storage available.

System action

The command is not sent to the queue manager and the panel is redisplayed.

CSQO017E

MQGET from *reply-q* unsuccessful. Reason code=*mqrc*.

Severity

8

Explanation

The reply to a command could not be retrieved from the reply-to queue using MQGET. (The reply-to queue is a local queue generated from the model queue SYSTEM.COMMAND.REPLY.MODEL.) *mqrc* is the reason code returned from MQGET; see API completion and reason codes for more information.

A possible cause of this problem is that get requests are inhibited on the reply-to queue.

System action

The panel is redisplayed. The command was sent to the queue manager, but it might not have been executed successfully.

CSQO018E

Queue manager is invalid or unknown or unavailable.

Severity

8

Explanation

An attempt to send a command was unsuccessful because the target or action queue manager was not known or not valid or not running.

System action

The command is not sent the queue manager and the panel is redisplayed.

CSQO019E

Queue manager is no longer available.

Severity
8

Explanation

The queue manager that you were using is no longer running. The action that you requested might not have been actioned.

System action

The main menu is redisplayed.

CSQO020I

'*field*' truncated due to quotes. Press Enter to continue.

Severity
0

Explanation

The value in field *field* contains one or more quotation marks. In order that these are treated as quotation marks instead of indicators of the beginning or end of a string, each quotation mark is converted into two quotation marks (doubling up) in the command for the queue manager. However, this conversion has made the string too long, and it has been truncated.

System action

The value is truncated. The panel may be displayed again with *field-name* set to the truncated value.

CSQO021I

Generic name not allowed.

Severity
8

Explanation

You entered a name ending with an asterisk, but generic names are only allowed on the Main Menu panel.

System action

The panel is redisplayed.

CSQO022I

Filter value invalid.

Severity
8

Explanation

You asked to list objects with filtering, but the value entered for the attribute to be used was invalid.

System action

The main menu panel or an empty list panel is displayed.

CSQO023I

Command *command* not recognized.

Severity
4

Explanation

The command entered in the panel command area (or using a function key) is not valid.

System action

The panel is redisplayed.

CSQO025I

There are no messages to view.

Severity
0

Explanation

The MSGVIEW panel command was entered in the command area, or the messages function key was pressed, but there are no messages from the queue manager to view.

System action

The panel is redisplayed.

CSQO027I

Function *function* not allowed for object type *object-type*.

Severity
8

Explanation

The function number that you entered is not allowed for *object-type* objects.

System action

The current panel is redisplayed.

CSQO028I

One of '*field1*' or '*field2*' but not both must be supplied.

Severity
0

Explanation

Nothing was entered in the two named fields, or something was entered in both of them. Either one or the other must have a value.

System action

The current panel is redisplayed.

CSQO029I

Command exceeds maximum allowable length of 32762 bytes.

Severity
4

Explanation

While defining or altering a namelist, too many names are added causing the necessary command to exceed the maximum allowable length.

System action

The panel is redisplayed.

CSQO030I

No objects of type *objtype* match *name*.

Severity
0

Explanation

You asked to display or list the objects of type *objtype* and name *name*, but no matching objects were found.

System action

The current panel is redisplayed.

CSQO031E

ALLOCATE of data set *dsname* unsuccessful. Return code = *rc*.

Severity
8

Explanation

An ALLOCATE error occurred when processing the data set allocated during an attempt to edit the names in a namelist. *dsname* is the name of the data set, and is of the form *userid*.NAMELIST.NAMES*n* (where *userid* is the TSO userid involved, and *n* is a number). *rc* is the return code from the TSO command ALLOCATE.

The most likely cause of this problem is that another data set with the same name already exists, or that DDname CSQONL*n* is in use.

System action

The panel is redisplayed.

System programmer response

This message will be accompanied by one or more messages from TSO, giving more information about the cause of the problem. The return code is documented in the *TSO/E Command Reference* manual.

If you are unable to resolve the problem, contact your IBM support center.

CSQO032E

Serious error returned. Press F10 to view.

Severity
12

Explanation

A command was sent to the queue manager, but message CSQN205I was received in reply, indicating a severe error.

System action

Message CSQN205I is saved. The current panel is redisplayed.

System programmer response

Look up message CSQN205I and take the appropriate action.

CSQO033E

Format of first reply not recognized. Press F10 to view.

Severity
8

Explanation

A command was sent to the queue manager, but the first reply message received is not CSQN205I.

System action

The messages received are saved. If it is not possible to continue, the current panel is redisplayed.

CSQO034E

Reply format not recognized. Press F10 to view.

Severity
8

Explanation

A command was sent to the queue manager. The first reply message received was CSQN205I as expected, but a subsequent message was not as expected.

System action

The message that caused the problem, and any subsequent messages are saved. If it is not possible to continue, the current panel is redisplayed.

CSQO035E

Unable to get storage (return code = *rc*).

Severity
12

Explanation

An attempt to get storage was unsuccessful.

System action

The system is unable to acquire enough storage.

System programmer response

Determine why there was insufficient storage available to satisfy the request.

CSQO036I

List is not filtered.

Severity
0

Explanation

You asked for a secondary list from a list that was filtered (for example, status from a list of queues or channels). The filter condition is not applied to the secondary list; all items that match the originally requested name, type, and disposition are included.

CSQO037I

Locally-defined channel will be used.

Severity
4

Explanation

You selected an action from the List Cluster queue manager Channels panel for an auto-defined cluster channel, but there is a locally-defined channel of the same name. In such a case, if you decide to take the action, it will be performed against the locally-defined channel instead.

System action

The action panel is displayed.

CSQO038I

Function is recursive.

Severity
4

Explanation

The function you requested would cause recursion; that is, it would take you to a panel that you have previously come from. This is not allowed.

System action

The current panel is redisplayed.

CSQO039E

EDIT of data set *dsname* failed. Return code = *rc*.

Severity
8

Explanation

An EDIT error occurred when processing the data set allocated during an attempt to edit the names in a namelist. *dsname* is the name of the data set, and is of the form *userid*.NAMELIST.NAMES*n* (where *userid* is the TSO userid involved, and *n* is a number). *rc* is the return code from the ISPF command EDIT.

System action

The panel is redisplayed.

System programmer response

This message will be accompanied by one or more messages from TSO, giving more information about the cause of the problem. The return code is documented in the *TSO/E Command Reference* manual.

If you are unable to resolve the problem, contact your IBM support center.

CSQO040I

No open queues with disposition *disptype* match *name*.

Severity
0

Explanation

You asked to list the open queues with disposition (or dispositions) *disptype* and name *name*, but no matching objects were found.

System action

The empty list panel is displayed.

CSQO041I

Action requires a specific object type.

Severity
4

Explanation

A define request was issued for object type QUEUE or CHANNEL.

System action

The secondary window or main panel is redisplayed.

CSQO042I

On the first panel.

Severity
0

Explanation

A function key was pressed that requests scrolling back to the previous panel, but the first panel is already being displayed.

System action

The panel is redisplayed.

CSQO043I

On the last panel.

Severity
0

Explanation

A function key was pressed that requests scrolling forward to the next panel, but the last panel is already being displayed.

System action

The panel is redisplayed.

CSQO044I

Function not available for objects with type *objtype*.

Severity
0

Explanation

The function you requested (for example, status or cluster information) is not available for objects with type *objtype*.

System action

The panel is redisplayed.

CSQO045I

Name too long for object type *type*.

Severity
8

Explanation

You specified a name that was longer than 20 characters for a channel object or longer than 16 characters for a connection object or longer than 8 characters or longer than 12 characters for a CF structure object or longer than 8 characters for a storage class object.

System action

The panel is redisplayed.

CSQO046I

No channels with saved status for *name*.

Severity
0

Explanation

You asked to list the saved status for channel *name*, but there was none.

System action

The empty list panel is displayed.

CSQO047I

No current channels for *name*.

Severity
0

Explanation

You asked to list the current instances for channel *name*, but there were none.

System action

The empty list panel is displayed.

CSQO048I

Channel initiator is not active.

Severity
0

Explanation

The action you requested needs the channel initiator to be active on the action queue manager, but it is not.

System action

The panel is redisplayed.

CSQO049I

EXEC cannot be invoked as a TSO command.

Severity
4

Explanation

An attempt was made to issue one of the Operations and Control execs as a TSO command.

System action

The request is ignored.

System programmer response

Use CSQOREXX to invoke the Operations and Control panels.

CSQO050I

No objects of type *objtype* disposition *disptype* match *name*.

Severity

0

Explanation

You asked to display or list the objects of type *objtype*, with disposition (or dispositions) *disptype* and name *name*, but no matching objects were found.

System action

The current panel is redisplayed or the empty list panel is displayed.

CSQO051I

Like object name with disposition *disptype* not found. Name assumed to be for defining new object with default attributes.

Severity

0

Explanation

You asked to define an object of type *objtype*, using as a basis an object with disposition *disptype* and name *name*, but no such object was found.

(In earlier releases, you could specify the name of a new object to define on the Main Menu panel, and a 'like' name to use as a basis for your definition. Now, only the 'like' name can be specified for Define on the Main Menu panel; you specify the new object name on the Define panel.)

System action

The Define panel is displayed, initialized with the name you specified and the default attributes for that type of object, on the assumption that you intended to define a new object with default attributes.

CSQO052I

Queue manager names changed because connect name changed.

Severity

0

Explanation

The Connect name field was changed but the Target queue manager field was not, and the new connect name was different from the target queue manager name. It is assumed you have forgotten to change the target queue manager.

System action

The target queue manager is changed to the queue manager you are connected to; the action queue manager might also be changed. The 'Queue Manager Names' secondary window is displayed, showing the new names that will be used.

CSQO053I

Blank connect or queue manager names specified.

Severity
0

Explanation

One or more of Connect name, Target queue manager, or Action queue manager fields was blank, specifying that the default name should be used.

System action

The Queue Manager Names secondary window is displayed, showing the actual names that will be used.

CSQO054I

Function not available for objects with disposition *disptype*.

Severity
0

Explanation

The function you requested (for example, status or cluster information) is not available for objects with disposition (or dispositions) *disptype*.

System action

The panel is redisplayed.

CSQO055I

Connect name is a queue-sharing group.

Severity
0

Explanation

The Connect name field specified the name of a queue-sharing group, to connect to any queue manager in the group.

System action

The Queue Manager Names secondary window is displayed, showing the queue manager you are connected to.

CSQO056I

Queue-sharing group is needed.

Severity
0

Explanation

The action you requested needs the queue manager to be part of a queue-sharing group, but it is not.

System action

The panel is redisplayed.

CSQO057I

Function *function* is not allowed for disposition *disposition*.

Severity
8

Explanation

The function number that you entered is not allowed with the specified disposition. This is the disposition of the object you are working with if you are using the Manage action, or the disposition you chose if you are performing a channel function.

System action

The current panel is redisplayed.

CSQO058I

Action *action* is not allowed for channels with disposition *disposition*.

Severity
8

Explanation

The action number that you entered is not allowed for channel objects with the specified disposition.

System action

The current panel is redisplayed.

CSQO059I

Disposition *disposition* is not allowed for object type *object-type*.

Severity
8

Explanation

The disposition that you entered is not allowed for *object-type* objects.

System action

The current panel is redisplayed.

CSQO060I

Platform for target queue manager *qmgr-name* is not z/OS or OS/390.

Severity
4

Explanation

The target queue manager is running on a plaform that is not z/OS or OS/390. With such a queue manager, it is likely that actions will work only partially, incorrectly, or not at all, and that the replies from the queue manager will not be recognized.

System action

The Confirm Target Queue Manager secondary window is displayed.

CSQO061I

Target queue manager *qmgr-name* command level is not supported.

Severity
4

Explanation

The target queue manager has a command level which is not one of those supported by the Operations and Control panels. With such a queue manager, it is likely that actions will work only partially, incorrectly, or not at all, and that the replies from the queue manager will not be recognized.

System action

The Confirm Target Queue Manager secondary window is displayed.

CSQO062I

Action queue manager *qmgr-name* command level is not the current level.

Severity
4

Explanation

The action queue manager has a command level which is not the current level supported by the Operations and Control panels. If an action is directed to such a queue manager most actions will work, but some fields will be ignored; a few objects and actions will be disallowed.

System action

The Confirm Action Queue Manager secondary window is displayed.

CSQO063I

Command level of some queue managers in the queue-sharing group is not the current level.

Severity
4

Explanation

The action queue manager is * and one or more queue managers in the queue-sharing group has a command level which is not the current level supported by the Operations and Control panels. If an action is directed to such a queue manager or to all queue managers in the queue-sharing group, most actions will work, but some fields will be ignored; a few objects and actions will be disallowed.

System action

The Confirm Action Queue Manager secondary window is displayed.

CSQO064I

Object type *object-type* is not allowed with command level of action or target queue manager.

Severity
4

Explanation

The action or target queue manager has a command level which does not support *object-type* objects.

System action

The 'Confirm Action Queue Manager' secondary window is displayed.

CSQO065I

Object name *name* is invalid.

Severity
8

Explanation

The value entered in the Name field was invalid.

System action

The panel is redisplayed.

CSQO066I

No status of this type for CF structures matching *name*.

Severity

0

Explanation

You asked to list status for CF structures with name *name*, but there were none with status of that type.

System action

The empty list panel is displayed.

CSQO067I

Some channel initiators not active in queue-sharing group. List may be incomplete.

Severity

4

Explanation

The action you requested requires information from the channel initiators on all the queue managers in the queue-sharing group, but some of those channel initiators are not active. The information might therefore be incomplete.

System action

The list panel is displayed, but might be incomplete.

CSQO068I

No channel initiators active in queue-sharing group.

Severity

4

Explanation

The action you requested requires information from the channel initiators on all the queue managers in the queue-sharing group, but none of those channel initiators are active. No information can therefore be displayed.

System action

The empty list panel is displayed.

CSQO069I

Action or function or object type is not allowed because of queue manager command level.

Severity

4

Explanation

The action queue manager has a command level which is not the current level supported by the Operations and Control panels. The action, function, or object type you chose is not allowed at that command level.

System action

The panel is redisplayed.

CSQO070I

No field value supplied.

Severity
0

Explanation

You asked to list objects with filtering, but no value was entered into any of the fields on the filter panels. A value must be entered into one (and only one) field to specify the filtering you want.

System action

The panel is redisplayed.

CSQO071I

More than one field value supplied.

Severity
0

Explanation

You asked to list objects with filtering, but a value was entered into more than one of the fields on the filter panels. Only one field value may be entered to specify the filtering you want.

System action

The panel is redisplayed.

CSQO072I

No current channels for *name* match filter condition.

Severity
0

Explanation

You asked to list the current instances for channel *name* with a filter condition, but there were none that satisfied the condition.

System action

The empty list panel is displayed.

CSQO073I

No channels with saved status for *name* match filter condition.

Severity
0

Explanation

You asked to list the saved status for channel *name* with a filter condition, but there were none with saved status that satisfied the condition.

System action

The empty list panel is displayed.

CSQO074I

No objects of type *objtype* match *name* and filter condition.

Severity
0

Explanation

You asked to display or list the objects of type *objtype* and name *name*, with a filter condition, but no matching objects were found that satisfied the condition.

System action

The current panel is redisplayed.

CSQO075I

No objects of type *objtype* disposition *disptype* match *name* and filter condition.

Severity
0

Explanation

You asked to display or list the objects of type *objtype*, with disposition (or dispositions) *disptype* and name *name*, with a filter condition, but no matching objects were found that satisfied the condition.

System action

The current panel is redisplayed or the empty list panel is displayed.

CSQO076I

No connections match *name*.

Severity
0

Explanation

You asked to list connections with name *name*, but there were none.

System action

The empty list panel is displayed.

CSQO077I

No open handles for connection name match *name*.

Severity
0

Explanation

You asked to list the open handles for the connection *name*, but no such handles were found.

System action

The empty list panel is displayed.

CSQO078I

No connections match *name* and filter condition.

Severity
0

Explanation

You asked to list connections with name *name*, but there were none that satisfied the condition.

System action

The empty list panel is displayed.

CSQO079I

No open queues with disposition *disptype* match *name* and filter condition.

Severity
0

Explanation

You asked to list the open queues with disposition (or dispositions) *disptype* and name *name* with a filter condition, but no matching objects were found that satisfied the condition.

System action

The empty list panel is displayed.

CSQO085E

Error in *pgm-name*. TBCREATE *table-name* failed, return code = *rc*.

Severity
12

Explanation

An attempt by *pgm-name* to call the ISPF TBCREATE service was unsuccessful. *table-name* is the name of the table that *pgm-name* was attempting to create.

System action

An internal error has occurred. The current panel is redisplayed. An ISPF message giving more details about the error might be shown first.

System programmer response

An internal error has occurred, note the message number and the values contained in it, together with any associated ISPF message, and contact your IBM support center to report the problem.

CSQO086E

Error in *pgm-name*. TBDISPL *panel-name* failed, return code = *rc*.

Severity
12

Explanation

An attempt by *pgm-name* to call the ISPF TBDISPL service was unsuccessful. *panel-name* is the name of the panel that *pgm-name* was attempting to display.

System action

The system is unable to display the panel, and the last panel is redisplayed (if applicable). An ISPF message giving more details about the error might be shown first.

System programmer response

If *rc*=12, the system is unable to find the panel. If you receive this message when you are trying to display the 'Main Menu' panel it could be that you do not have the data set containing the panels in your library concatenation. Find the name of the data set containing the panels, then check your ISPPLIB library definitions. This will probably be in your TSO logon procedure unless you are calling CSQOREXX from a higher level exec or CLIST that has the ISPF LIBDEF calls in it.

If you are already using the panels when you get this message, either a panel is missing from your ISPPLIB library, or an internal error has occurred. If you are unable to solve the problem, contact your IBM support center for assistance.

If *rc*=20, the most likely cause of the problem is that the system was unable to find the key-list which goes with the panel that it is trying to display. All the key lists are in an ISPF table (CSQOKEYS) that should be in a library in your ISPTLIB concatenation.

CSQO087E

Error in *pgm-name*. SELECT *program* failed, return code = *rc*.

Severity

12

Explanation

An attempt by *pgm-name* to call the ISPF SELECT service was unsuccessful. *program* is the name of the program that *pgm-name* was attempting to select.

System action

The current panel is redisplayed. An ISPF message giving more details about the error might be shown first.

System programmer response

The system is unable to find a load module. Check your ISPLLIB library concatenation.

CSQO088E

Error in *pgm-name*. DISPLAY *panel-name* failed, return code = *rc*.

Severity

12

Explanation

An attempt by *pgm-name* to call the ISPF DISPLAY service was unsuccessful. *panel-name* is the name of the panel that *pgm-name* was attempting to display.

System action

The system is unable to display the panel, and the last panel is redisplayed (if applicable). An ISPF message giving more details about the error might be shown first.

System programmer response

If *rc*=12, the system is unable to find the panel. If you receive this message when you are trying to display the 'Main Menu' panel it could be that you do not have the data set containing the panels in your library concatenation. Find the name of the data set containing the panels, then check your ISPPLIB library definitions. This will probably be in your TSO logon procedure unless you are calling CSQOREXX from a higher level exec or CLIST that has the ISPF LIBDEF calls in it.

If you are already using the panels when you get this message, either a panel is missing from your ISPPLIB library, or an internal error has occurred. If you are unable to solve the problem, contact your IBM support center for assistance.

If *rc*=20, the most likely cause of the problem is that the system was unable to find the key-list which goes with the panel that it is trying to display. All the key lists are in an ISPF table (CSQOKEYS) that should be in a library in your ISPTLIB concatenation.

CSQO089E

Error in *pgm-name*. *service* failed, return code = *rc*.

Severity

12

Explanation

An attempt by *pgm-name* to call the ISPF service (*service*) was unsuccessful.

System action

The current panel is redisplayed. An ISPF message giving more details about the error might be shown first.

System programmer response

service=**VDEFINE, VPUT, or TBADD**
An internal error has occurred, note the message number and the values contained in it, and contact your IBM support center for assistance.

If *service* is anything else, note the message number and the values contained in it, together with any associated ISPF message, and contact your IBM support center to report the problem.

CSQO090E

Internal error in *program*. Action field is not valid.

Severity
12

Explanation

An internal error has occurred.

System action

The current panel is redisplayed.

System programmer response

Collect the following items, and contact your IBM support center:

- The number of the message, and the value of *program*
- The name of the panel involved
- A description of the actions that led to the problem

CSQO091E

Internal error in *program*. Object field is not valid.

Severity
12

Explanation

An internal error has occurred.

System action

The last panel is redisplayed.

System programmer response

Collect the following items, and contact your IBM support center:

- The number of the message, and the value of *program*
- The name of the panel involved
- A description of the actions that led to the problem

CSQO092E

Internal error in *program*. Error in reply translation.

Severity
12

Explanation

An internal error has occurred.

System action

The last panel is redisplayed.

System programmer response

Collect the following items, and contact your IBM support center:

- The number of the message, and the value of *program*
- The name of the panel involved
- A description of the actions that led to the problem

CSQO093E

Internal error in *program*. Command request is not valid.

Severity

12

Explanation

An internal error has occurred.

System action

The last panel is redisplayed.

System programmer response

Collect the following items, and contact your IBM support center:

- The number of the message, and the value of *program*
- The name of the panel involved
- A description of the actions that led to the problem

CSQO095E

Internal error in *program*. *service* failed, return code = *rc*.

Severity

12

Explanation

An internal error has occurred.

System action

The last panel is redisplayed.

System programmer response

Collect the following items, and contact your IBM support center:

- The number of the message, and the values of *program* and *service*
- The name of the panel involved
- A description of the actions that led to the problem
- Any associated ISPF message shown

CSQO096E

Internal error in *program*. *att-name* not in keyword table.

Severity

12

Explanation

An internal error has occurred.

System action

The last panel is redisplayed.

System programmer response

Collect the following items, and contact your IBM support center:

- The number of the message, and the values of *program* and *att-name*
- The name of the panel involved
- A description of the actions that led to the problem

CSQO097E

Internal error in *program*. No handle for required system queue.

Severity

12

Explanation

An internal error has occurred.

System action

The last panel is redisplayed.

System programmer response

Collect the following items, and contact your IBM support center:

- The number of the message
- The name of the panel involved
- A description of the actions that led to the problem

Buffer manager messages (CSQP...): z/OS

CSQP002I

BUFFPOOL VALUE OUT OF RANGE

Severity

8

Explanation

One of the following commands has been issued incorrectly:

- DEFINE BUFFPOOL(n)
- ALTER BUFFPOOL(n)
- DELETE BUFFPOOL(n)
- DEFINE PSID(x) BUFFPOOL(n)

The value of n depends on OPMODE and can be either 0 to 15 or 0 to 99.

System action

The command is ignored.

System programmer response

See MQSC commands for information about the command, and reissue the command correctly.

CSQP003I

PSID VALUE OUT OF RANGE

Severity

8

Explanation

One of the following commands has been issued incorrectly:

- DEFINE PSID(x)
- ALTER PSID(x)
- DELETE PSID(x)

The value of x must be in the range 0 through 99.

System action

The command is ignored.

System programmer response

See MQSC commands for information about the command, and reissue the command correctly.

CSQP004E

csect-name I/O ERROR STATUS *ret-code* PSID *psid* RBA *rba*

Severity

8

Explanation

An I/O error has occurred. *ret-code* is the return code from the Media Manager. *psid* is the identifier of the page set for which the error occurred and *rba* is the RBA (in hexadecimal) of the record on which the error occurred.

System action

The queue manager can be abended. For example, in the case of a failing MQGET or MQPUT, the queue manager is not terminated if the CSQP004E I/O error occurs during a IBM MQ API call. However, if the I/O error occurs during checkpoint processing, the queue manager is terminated.

System programmer response

See the *MVS/DFP Diagnosis Reference* manual for information about return codes from the Media Manager. If you do not have access to the required manual, contact your IBM support center, quoting the return code from the Media Manager.

CSQP005I

BUFFERS VALUE OUT OF RANGE

Severity

8

Explanation

One of the following commands has been issued incorrectly:

- DEFINE BUFFPOOL(n) BUFFERS(x)
- ALTER BUFFPOOL(n) BUFFERS(x)

If the value of the LOCATION parameter is BELOW, the minimum value of buffers is 100 and the maximum value is 500,000. If the value of the LOCATION parameter is ABOVE then valid values are in the range of 100 to 999999999 (nine nines).

System action

The command is ignored.

System programmer response

Reissue the command correctly. The total number of buffers that it is possible to define in all the buffer pools is determined by the amount of storage available in the queue manager address space.

CSQP006I

LOG CHECKPOINT NAME *log-name* DOES NOT MATCH QUEUE MANAGER NAME *qmgr-name*

Severity

8

Explanation

An attempt to restart with a log from another queue manager was detected. The name recorded in the log during checkpoint does not match the name of the queue manager using that log for restart.

System action

Restart is abnormally terminated with completion code X'5C6' and reason code X'00D70102'.

System programmer response

Change the started task JCL procedure xxxxMSTR for the queue manager to name the appropriate bootstrap and log data sets.

CSQP007I

Page set *x* uses buffer pool *n*

Severity

0

Explanation

This message gives the buffer pool used by the specified page set.

It is sent in response to a DEFINE PSID(x) command.

CSQP009I

PAGE RECOVERY STARTED FOR PAGE SET *psid* PAGE *page-number*

Severity

0

Explanation

An incomplete update operation was detected for page *page-number* of page set *psid*. The page is being restored to a consistent state from information on the log.

Message CSQP010I will be issued when the page recovery operation has completed.

CSQP010I

PAGE RECOVERY COMPLETE FOR PAGE SET *psid* PAGE *page-number*

Severity

0

Explanation

An incomplete update operation was detected for page *page-number* of page set *psid*. The page has been restored to a consistent state from information on the log.

CSQP011E

CONNECT ERROR STATUS *ret-code* FOR PAGE SET *psid*

Severity

8

Explanation

An attempt to open a page set was unsuccessful. *psid* is the page set identifier and *ret-code* is the return code from the Data Facilities Product (DFP) CONNECT function.

This can occur during queue manager startup, where the most likely cause is that there is no DD statement for the page set included in the queue manager started task JCL, or in response to a DEFINE PSID command used to add a page set dynamically.

System action

If this occurs during queue manager startup, MQ attempts to dynamically allocate the page set and retry the open, on the assumption that the DD statement for the page set is missing. Messages following message CSQI010I at the end of restart indicate whether the dynamic page set allocation was successful, or whether such page sets still remain offline.

If the page set cannot be opened, the queue manager continues running, but you will be unable to access the data in that page set. You could encounter problems during restart, or when attempting to open a queue.

System programmer response

If applicable, ensure that there is a DD statement for the page set included in the queue manager started task JCL.

If the page set cannot be opened, see the *MVS/DFP Diagnosis Reference* manual for information about return codes from the Media Manager. If you do not have access to the required manual, contact your IBM support center, quoting the return code from the Media Manager.

CSQP012I

DISCONNECT ERROR STATUS *ret-code* FOR PAGE SET *psid*

Severity

8

Explanation

An attempt to close a page set was unsuccessful. *psid* is the page set identifier and *ret-code* is the return code from the Media Manager.

System action

Queue manager shutdown continues, but some information might be missing from the page set. This will be corrected from the log during restart.

System programmer response

See the *MVS/DFP Diagnosis Reference* manual for information about return codes from the Media Manager. If you do not have access to the required manual, contact your IBM support center, quoting the return code from the Media Manager.

CSQP013I

csect-name NEW EXTENT CREATED FOR PAGE SET psid. NEW EXTENT WILL NOW BE FORMATTED

Severity

0

Explanation

Page set *psid* has been successfully dynamically expanded by creating a new extent.

System action

The new extent is formatted; message CSQI031I will be issued when formatting completes successfully.

System programmer response

The page set can only be expanded 123 times. After this you will have to reallocate the page set using larger primary and secondary extents. For information about managing page sets, see Managing page sets.

CSQP014E

csect-name EXPANSION FAILED FOR PAGE SET psid. FUTURE REQUESTS TO EXTEND IT WILL BE REJECTED

Severity

8

Explanation

An attempt to expand a page set dynamically was unsuccessful.

System action

Processing continues.

System programmer response

Look for messages from VSAM or DFP that explain why the request was unsuccessful, and do the required actions.

Determine why the page set needs to expand:

- Review Planning your page sets and buffer pools to make sure your page set allocation is large enough for your application queues.
- If there is a large depth on the Dead Letter Queue (DLQ) either implement the DLQ Handler, CSQUDLQH, or clear the queue with CLEAR QLOCAL command if you don't need to take further action with the messages. Similarly, SYSTEM.EVENT.* queues can fill a page set.
- Look in joblogs or application logs to see if an error is preventing the getting application from running.
- See if an application is failing to commit its gets or puts. You can tell if there are uncommitted messages by using the following command:

  ```
  DISPLAY QSTATUS(qname) UNCOM QDEPTH
  ```

  Notes:

  1. The display does not show how many messages are uncommitted, and whether they are for gets or puts.
  2. A message that is subject to an uncommitted MQGET still takes up space on the page set, although the message no longer contributes to the depth of the queue.
- If the getting application is a channel, is the channel starting, and is the channel able to successfully move messages? Use the command

  ```
  DISPLAY CHSTATUS(channelname) ALL
  ```

  to verify the channel status attributes including STATUS, SUBSTATE, and INDOUBT.
- If the messages use an integer in MQMD.EXPIRY, there might be expired messages that need to be cleaned up. If EXPRYINT is set to OFF in the QMGR definition, the command

  ```
  REFRESH QMGR TYPE(EXPIRY) NAME(big.queue)
  ```

  causes an EXPIRY scan of the queue that matches the name provided in the NAME() field. This command can take some time to process. Issue the command

  ```
  DISPLAY USAGE PSID(n)
  ```

  where `n` is the page set number, at regular intervals, to monitor progress.
- Check for any third party products on the system that get involved with EOV or EXTEND processing.

If you have received message IEC070I, and the *return code* (the first value in that message) is:

034(004):

End of volume - Non-extended addressable. The new allocation amount would exceed 4 GB.

If the message volume or size requires a larger page set , follow the instructions at Defining a page set to be larger than 4 GB

104 No more volumes are available on which to allocate space (no more candidate volumes).

Use the following commands to add space and switch off the internal "page not expandable" flag:

- The TSO ALTER ADDVOLUME command to add space for extents.
- ALTER PSID() EXPAND()

  You must supply valid syntax, that is, a pageset number and expand value. See ALTER PSID for more information.

203 An extend was attempted, but no secondary space allocation quantity was specified.

204 An extend was attempted, but the maximum number of extents was reached.

The maximum number of extents for a VSAM data set cataloged in an ICF catalog is between 119 and 123, depending upon the number of extents (1-5) allocated by DADSM per allocate/extend request.

209

- An extend was attempted, but no space was available on user volume.
- No secondary space quantity was specified and no candidate volumes are available.

You can follow the directions in How to increase the size of a page set as IBM MQ for z/OS allows you to enable dynamic page set expansion, or add candidate volumes using IDCAMS ALTER ADDVOL.

The data set then needs to be closed and reopened so that the TIOT is rebuilt; otherwise IEC070I 211(8,306)-221 and IGD306I UNEXPECTED ERROR DURING IEFAB4C2 PROCESSING RETURN CODE 24 REASON CODE 0 might occur.

The close can be done without a recycle of the queue manager by using the following JCL:

```
//STEP1 EXEC PGM=IDCAMS
//DSFILE DD DSN=your.dataset.name,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
VERIFY FILE(DSFILE)
/*
```

You might need to run the JCL twice to complete with a non-zero return code. Some flags might not be reset during the first run.

Note: DFP uses up to five non-contiguous areas of disk to satisfy the total space requirements of a primary or secondary extent. This means, in the worst case of badly fragmented disk space, that you might only get around 22 times the secondary space allocated before you reach the maximum space limit.

If you believe that there is sufficient free space that could be used by another secondary extent, contact your IBM support center for assistance.

CSQP016E

csect-name PAGE SET *psid* HAS REACHED THE MAXIMUM NUMBER OF EXTENTS. IT CANNOT BE EXTENDED AGAIN

Severity
8

Explanation

An attempt to expand page set *psid* dynamically was unsuccessful because the maximum number of extents had been used.

System action

The page set cannot be extended again. When the messages on the full page set are retrieved, the existing space will be reused.

System programmer response

Copy the page set to a new page set with larger primary and secondary extents. By defining the page set as a multivolume data set, you can take advantage of the free space on as many disk volumes as possible. See the Planning on z/OS. For more information about page set organization and management, see Managing page sets.

CSQP017I

csect-name EXPANSION STARTED FOR PAGE SET *psid*

Severity
0

Explanation

Page set *psid* is being expanded dynamically, by creating a new extent.

System action

All threads that are currently adding message to page set *psid* are suspended until the page set expansion completes (this is indicated by message CSQP013I).

CSQP018I

csect-name CHECKPOINT STARTED FOR ALL BUFFER POOLS

Severity
0

Explanation

A checkpoint is being taken for all defined buffer pools.

CSQP019I

csect-name CHECKPOINT COMPLETED FOR BUFFER POOL *n*, *pages* PAGES WRITTEN

Severity
0

Explanation

A checkpoint has been successfully taken for buffer pool *n*.

CSQP020E

csect-name Buffer pool *n* is too small

Severity
8

Explanation

Contention is taking place for buffers in a buffer pool. Messages will have to be read from and written to the page sets, which increases the time to process an application request and increases the amount of processor time used.

System action

Processing continues.

System programmer response

If required, use the ALTER BUFFPOOL command to add more buffers to the buffer pool. Consider first altering other buffer pools to reduce the total number of buffers in use. Refer to the latest CSQY220I message on the z/OS console to see how much virtual storage is free, and hence how many extra buffers may be safely added to a buffer pool. If you do change the number of buffers in the buffer pool, you should also change the DEFINE BUFFPOOL commands in the CSQINP1 initialization input data set used by the queue manager.

Alternatively, specify DEFINE BUFFPOOL(X) REPLACE as this option does not use the log checkpoint record.

If the buffer pool has a LOCATION value of BELOW and there is insufficient storage below the bar then consider moving the buffer above the bar by setting its LOCATION value to ABOVE. This might require altering the value of the MEMLIMIT parameter. For more information, see Address space storage.

CSQP021I

Page set *psid* new media recovery RBA=*rcvry-rba*, checkpoint RBA=*chkpt-rba*

Severity

0

Explanation

During checkpoint processing, buffers have been flushed from the buffer pools to the indicated page set, establishing a new media recovery RBA. This RBA is the point from which log data would be required to perform media recovery for the page set. It should be the same as the checkpoint RBA.

System action

Processing continues.

System programmer response

If the media recovery and checkpoint RBAs differ, contact your IBM support center.

CSQP022I

Buffer pool *n* is not defined

Severity

8

Explanation

A command has been issued specifying a buffer pool that is not defined.

System action

The command is ignored.

System programmer response

See MQSC commands for information about the command, and reissue the command correctly.

CSQP023I

Request completed for buffer pool *n*, now has *k* buffers

Severity

0

Explanation

The size of the specified buffer pool has been successfully changed.

CSQP024I

Request initiated for buffer pool *n*

Severity
0

Explanation

The request to change the buffer pool has been accepted. One of the messages CSQP023I, CSQP052I, or CSQP053I will be sent to the z/OS console when the change is complete,

CSQP025I

Page set *n* is not defined or offline

Severity
8

Explanation

A command has been issued specifying a page set that is not available to the queue manager.

System action

The command is ignored.

System programmer response

See MQSC commands for information about the command, and reissue the command correctly.

CSQP026I

Page set *n* is in use by a storage class

Severity
8

Explanation

The page set specified is referenced by a storage class, and so cannot be deleted.

System action

The command is ignored.

System programmer response

Change or delete all the storage classes that reference the page set, and then reissue the command.

CSQP027I

Page set *n* has buffers in use

Severity
8

Explanation

The page set specified has buffers that are still in use, and so cannot be deleted.

System action

The command is ignored.

System programmer response

Wait until three checkpoints have been completed, and then reissue the command.

CSQP028I

Request initiated for page set *n*

Severity
0

Explanation

The request to define or delete the page set has been accepted. Message CSQP042I or CSQP032I will be sent to the z/OS console when the change is complete. If the change fails, messages CSQP041E or CSQP031E will be sent.

CSQP030E

Deallocation failed for data set *dsname*, error status=*eeeeiiii*, SMS reason code=*ssssssss*

Severity
8

Explanation

An error occurred when trying to dynamically deallocate the page set data set. Error status is the error reason code returned by z/OS dynamic allocation.

System action

The page set is deleted and is no longer available for use.

System programmer response

The error status portion of this message contains a 2-byte error code (*eeee*, S99ERROR) followed by the 2-byte information code (*iiii*, S99INFO) from the SVC99 request block. If the S99ERROR code indicates an SMS allocation error ('97xx'), then *ssssssss* contains additional SMS reason code information obtained from S99ERSN. See the *MVS Authorized Assembler Services Guide* manual for a description of these codes.

CSQP031E

Page set *n* deletion failed

Severity
8

Explanation

An error occurred while deleting the specified page set.

System action

Processing continues.

System programmer response

See the preceding error messages for more information about the error.

CSQP032I

Page set *n* deletion completed

Severity
0

Explanation

The specified page set has been successfully deleted.

CSQP033E

Error deleting page set *n*, code=*rrr*

Severity
8

Explanation

An error occurred while deleting the specified page set.

System action

The page set is not deleted, and is still available for use.

System programmer response

Note the error code and contact your IBM support center.

CSQP034E

Page set *n* is already defined

Severity
8

Explanation

The specified page set is already in use by the queue manager, and so cannot be dynamically defined.

System action

The command is ignored.

System programmer response

See MQSC commands for information about the command, and reissue the command correctly.

CSQP035E

Allocation failed for data set *dsname*, error status=*eeeeiiii*, SMS reason code=*ssssssss*

Severity
8

Explanation

An error occurred when trying to dynamically allocate the page set data set. Error status is the error reason code returned by z/OS dynamic allocation.

System action

The page set is not defined.

System programmer response

The error status portion of this message contains a 2-byte error code (*eeee*, S99ERROR) followed by the 2-byte information code (*iiii*, S99INFO) from the SVC99 request block. If the S99ERROR code indicates an SMS allocation error ('97xx'), then *ssssssss* contains additional SMS reason code information obtained from S99ERSN. See the *MVS Authorized Assembler Services Guide* manual for a description of these codes.

CSQP036I

Data set *dsname* for page set *n* is not formatted with RECOVER or REPLACE

Severity
8

Explanation

The named page set data set was not formatted correctly. A data set that is to be used for adding a page set dynamically must be one that is newly formatted (using TYPE(RECOVER)), or one that has previously been used to hold messages and has been formatted using TYPE(REPLACE).

System action

The page set is not defined.

System programmer response

Format the data set as required. If you are adding a previously unused page set to the queue manager, use the FORMAT function of the utility program CSQUTIL, specifying TYPE(RECOVER). If the page set was previously used to hold messages, use the FORMAT function specifying TYPE(REPLACE).

In the latter case, if the queue manager terminated abnormally, the formatting may fail, and message CSQU160E will be issued. It is not possible to add such a page set data set dynamically, but the page set can be brought into use again by including it in the started task JCL procedure xxxxMSTR for the queue manager, and then restarting the queue manager.

CSQP037E

OPEN failed for page set *n*, VSAM return code=*rc* reason code=*reason*

Severity

8

Explanation

A VSAM error occurred when trying to open the page set data set.

System action

The page set is not defined.

System programmer response

See the *DFSMS/MVS Macro Instructions for Data Sets* for information about the return and reason codes from VSAM. If necessary, reissue the request.

CSQP038E

GET failed for page set *n*, VSAM return code=*rc* reason code=*reason*

Severity

8

Explanation

A VSAM error occurred when trying to get a record from the page set data set.

System action

The page set is not defined.

System programmer response

See the *DFSMS/MVS Macro Instructions for Data Sets* for information about the return and reason codes from VSAM. If necessary, reissue the request.

CSQP039E

CLOSE failed for page set *n*, VSAM return code=*rc* reason code=*reason*

Severity

8

Explanation

A VSAM error occurred when trying to close the page set data set.

System action

The page set is not defined.

System programmer response

See the *DFSMS/MVS Macro Instructions for Data Sets* for information about the return and reason codes from VSAM. If necessary, reissue the request.

CSQP041E

Page set *n* definition failed

Severity
8

Explanation

An error occurred while defining the specified page set.

System action

Processing continues.

System programmer response

See the preceding error messages for more information about the error.

CSQP042I

Page set *n* definition completed

Severity
0

Explanation

The specified page set has been successfully defined.

CSQP043I

Buffer pool *n* is in use by a page set

Severity
8

Explanation

The buffer pool specified is in use by a page set, and so cannot be deleted.

System action

The command is ignored.

System programmer response

Change or delete all the page sets that reference the buffer pool, and then reissue the command.

CSQP045I

Buffer pool *n* is not in use by any page set

Severity
8

Explanation

The buffer pool specified is not in use by any page set, and so cannot have buffers added or removed.

System action

The command is ignored.

System programmer response

Define at least one page set that references the buffer pool, and then reissue the command, or delete the buffer pool.

CSQP046I

Request already in progress for buffer pool *n*

Severity
8

Explanation

The buffer pool specified is being altered or deleted by another command.

System action

The command is ignored.

System programmer response

Wait until the other command has completed processing, and then reissue the command if appropriate.

CSQP047E

Unavailable page sets can cause problems - take action to correct this situation

Severity
4

Explanation

One or more page sets are unavailable, as reported in the preceding messages; they are either offline having been used previously, not defined, suspended, or otherwise inaccessible. For example, MQ may have attempted to open a page set at restart, but failed perhaps because it was in use by another application.

This situation can cause problems, so you should take action to correct it as soon as possible.

System action

Processing continues.

System programmer response

Use the DISPLAY USAGE command to get a list of the unavailable page sets.

If a previously-used page set is required, bring it online; this can be done without stopping the queue manager. Use the FORMAT function of the utility program CSQUTIL, specifying TYPE(REPLACE). Then issue a DEFINE PSID command to bring the page set back into use. Note that all units of recovery (except those that are indoubt) that involved the offline page set will have been backed out by the queue manager when the page set was last used. These indoubt units of recovery may be resolved once the page set is back in use by the queue manager.

If a page set is not required, issue a DELETE PSID command to remove it. Also remove any DEFINE PSID command for it from the CSQINP1 initialization input data set.

CSQP048E

PUT failed for page set *n*, VSAM return code=*rc* reason code=*reason*

Severity
8

Explanation

A VSAM error occurred when trying to get a record from the page set data set.

System action

The page set is not defined.

System programmer response

See the *DFSMS/MVS Macro Instructions for Data Sets* for information about the return and reason codes from VSAM. If necessary, reissue the request.

CSQP049I

Data set *dsname* is formatted for a different page set *n*

Severity
8

Explanation

The page set data set was formatted using TYPE(REPLACE), and as such may contain messages for a specific page set *n*. It cannot be added dynamically with a different page set identifier.

System action

The page set is not defined.

System programmer response

Reissue the command specifying the correct data set and page set. If you intended adding a previously unused page set, reformat the data set with use the FORMAT function of the utility program CSQUTIL, specifying TYPE(RECOVER).

CSQP051I

Insufficient storage for buffer pool *n* request

Severity
4

Explanation

The size of the specified buffer pool has not been changed as requested because insufficient storage is available.

System programmer response

The DISPLAY USAGE command can be used to determine the current sizes of all buffer pools defined to the system. It may be possible to reduce the size of other buffer pools, so freeing storage, which can then be assigned to this buffer pool by reissuing the command.

Message CSQY220I shows the storage information. Refer to Managing buffer pools for more information on how to alter your buffer pool.

CSQP052I

Request partially completed for buffer pool *n*, now has *k* buffers

Severity
4

Explanation

The size of the specified buffer pool has been changed. The number of buffers is not that requested because, for example, insufficient storage is available.

CSQP053I

Request completed for buffer pool *n*, buffers not changed

Severity
0

Explanation

The size of the specified buffer pool has not been changed. This could be because the number of buffers requested was the same as the existing size, or because there was insufficient storage available to change the size or location of the buffer pool (as shown by preceding message CSQP051I).

CSQP054I

Buffer pool *n* is now located above the bar

Severity
0

Explanation

The specified buffer pool has now been moved so that it is located above the bar.

CSQP055I

Buffer pool *n* is now located below the bar

Severity
0

Explanation

The specified buffer pool has now been moved so that it is located below the bar.

CSQP056E

The ALTER BUFFPOOL command for buffer pool *n* has failed

Severity
8

Explanation

An unexpected error occurred while processing the ALTER BUFFPOOL command. The buffer pool will be left with the number of buffers that were in it at the time the error occurred.

System action

Processing continues.

System programmer response

Use the DISPLAY USAGE PSID(*) command to view the current state of the buffer pool. If necessary reissue the ALTER BUFFPOOL command again.

If any abends have been issued, look at the abend code to see if the error is caused by the queue manager being short of storage. Changing the LOCATION parameter from BELOW to ABOVE for a buffer pool might require you to increase the MEMLIMIT parameter in the JCL of the queue manager stored procedure, xxxxMSTR. For more details, see Address space storage.

If switching a buffer pool from above to below the bar you might need to decrease the number of buffers in the buffer pool.

CSQP057E

Buffer pool *n* is suspended because of the current value of OPMODE

Severity
4

Explanation

The queue manager was previously running with Version 8.0 new functions enabled with OPMODE when the buffer pool was created. The buffer pool has an ID in the range of 16 to 99. Subsequently, the queue manager has been restarted with the OPMODE parameter changed to disable Version 8.0 new functions.

Buffer pools with an ID greater than 15 are not supported when Version 8.0 new functions are not enabled, because this support is available only from IBM MQ Version 8.0 onwards.

The specified buffer pool is marked as suspended. As a result, any page sets that use this buffer pool are also suspended and message CSQP059E: Page set *n* is suspended because it uses suspended buffer pool *n* is issued.

The buffer pool remains suspended and information is kept in checkpoint records until the queue manager is restarted with Version 8.0 new functions enabled. The buffer pool and any page sets that use the buffer pool are no longer suspended.

It is not possible to use the **ALTER** or **DELETE** MQSC commands on a suspended buffer pool. Additionally, information about the suspended buffer pool is not available when you use the **DISPLAY USAGE PSID** command.

System action

Any page sets that use the specified buffer pool are suspended and message CSQP059E: Page set *n* is suspended because it uses suspended buffer pool *n* is issued.

System programmer response

To use the buffer pool again, restart the queue manager with Version 8.0 new functions enabled with OPMODE.

To remove the buffer pool, set the OPMODE parameter to OPMODE=(NEWFUNC,800), or OPMODE=(NEWFUNC,900), change the **DEFINE PSID** command in the CSQINP1 initialization input data set to use a different buffer pool. Restart the queue manager, delete the buffer pool, then change the OPMODE parameter back to its previous value and restart the queue manager.

CSQP058E

Buffer pool *n* has had its LOCATION forced to BELOW because of the current value of OPMODE

Severity

4

Explanation

The queue manager was previously running with Version 8.0 new functions enabled with OPMODE when the buffer pool was created. The buffer pool had the LOCATION attribute set to ABOVE. Subsequently, the queue manager has been restarted with the OPMODE parameter changed to disable Version 8.0 new functions.

Buffer pools with a LOCATION value of ABOVE are not supported when Version 8.0 new functions are not enabled, because this support is available only from IBM MQ Version 8.0 onwards.

The specified buffer pool has had its LOCATION value set to BELOW. Additionally, the number of available buffers in the buffer pool is set to 1000 so that the storage limit below the bar is not exceeded. If the number of buffers in the buffer pool was originally less than 1000, then the number is not changed.

System action

Processing continues.

System programmer response

Consider increasing the number of buffers in the specified buffer pool by using the ALTER BUFFPOOL MQSC command.

CSQP059E

Page set *n* is suspended because it uses suspended buffer pool *n*

Severity

4

Explanation

The queue manager was previously running with Version 8.0 new functions enabled with OPMODE when the buffer pool was created. The buffer pool has an ID in the range of 16 to 99. Subsequently, the queue manager has been restarted with the OPMODE parameter changed to disable Version 8.0 new functions.

Buffer pools with an ID greater than 15 are not supported when Version 8.0 new functions are not enabled, because this support is available only from IBM MQ Version 8.0 onwards.

The specified buffer pool is marked as suspended. As a result, the specified page set that uses this buffer pool is suspended too.

Information about the suspended page set is kept in checkpoint records until the queue manager is restarted with Version 8.0 new functions enabled. The page set is then no longer suspended. Until the queue manager is restarted with Version 8.0 new functions enabled, any object definitions and messages on the page set are unavailable. An attempt to use a queue or topic that uses the page set results in an MQRC_PAGESET_ERROR message.

Note: You can delete a suspended page set only if the page set has no messages on it.

System action

Processing continues.

System programmer response

You can restart the queue manager with Version 8.0 new functions enabled with OPMODE, to take all buffer pools and associated page sets out of suspend mode.

Alternatively, bring the page set back online by associating the page set with a new buffer pool. You can do this without stopping the queue manager by using the FORMAT function of the utility program CSQUTIL, specifying TYPE(REPLACE). Issue a **DEFINE PSID** command to bring the page set back into use.

Note: All units of recovery that involved the suspended page set, except those units that are indoubt, were backed out by the queue manager when the page set was last used. Indoubt units of recovery can be resolved when the page set is again in use by the queue manager.

CSQP060E

Page set 0 must use one of buffer pools 0 to 15

Severity

12

Explanation

Page set 0 must be defined so that it uses buffer pool 0 to 15.

System action

Queue manager startup fails.

System programmer response

Define page set 0 so that it uses buffer pool 0 to 15. Generally, use buffer pool 0.

CSQP061I

ALTER BUFFPOOL *n* in progress, elapsed time *m* minutes

Severity

4

Explanation

The ALTER BUFFPOOL command has been issued for the specified buffer pool. If the command takes longer than approximately five minutes to process, this message is output approximately every five minutes until the command is complete.

Once the command is complete one or more of the following messages is output: CSQP023I, CSQP051I, CSQP052I, or CSQP053I.

This message might be output in the following scenarios:

- The specified buffer pool has had its LOCATION parameter changed from ABOVE to BELOW
- The specified buffer pool had its LOCATION parameter set to ABOVE and the number of buffers has been reduced by a large number

In most cases the ALTER BUFFPOOL command completes very quickly, and this message is not output. If this message is output, it should not be a cause for concern unless the value of the elapsed time becomes a large value - more than 30 minutes.

System action

Processing continues.

System programmer response

Monitor the job log for further output of this message, or a message indicating that the ALTER BUFFPOOL command has completed.

If this message is continually output and the elapsed time grows to a large value (more than 30 minutes) this might indicate a problem, so contact your IBM Service representative.

CSQP062I

Buffer pool *n* PAGECLAS changed, restart required to take effect

Severity

4

Explanation

The PAGECLAS attribute of the specified buffer pool has changed.

This change does not dynamically affect the type of pages used by the buffer pool, unless the LOCATION attribute is changed from BELOW to ABOVE at the same time. However the change is logged, and is applied when the queue manager is restarted.

System action

Processing continues. The buffer pool uses the previous value of the PAGECLAS attribute.

System programmer response

None, unless you require that the new PAGECLAS attribute of the specified buffer pool takes immediate effect.

In this case, either restart the queue manager or perform both of the following steps:

1. Buffer pool so that its LOCATION attribute is BELOW and its PAGECLAS is 4KB, and
2. Change the LOCATION attribute of the buffer pool to ABOVE, at the same time as changing the PAGECLAS attribute.

CSQP063E

PAGECLAS value must be 4KB if specified with LOCATION(BELOW)

Severity

8

Explanation

A buffer pool with a LOCATION value of ABOVE and PAGECLAS attribute that is not 4KB has been altered so that its LOCATION value is BELOW.

The only value of PAGECLAS that is valid with a LOCATION value of BELOW is 4KB.

System action

The command is ignored.

System programmer response

In addition to altering the LOCATION attribute to the value BELOW, alter the PAGECLAS attribute to the value 4KB.

CSQP064I

Buffer pool *n* definition in CSQINP1 data set used

Severity

4

Explanation

This message is issued at startup when the queue manager reads its log.

A buffer pool has been defined in the CSQINP1 data set, with the REPLACE attribute specified, so the definition for the buffer pool in the log of the queue manager is ignored.

Changes made to the buffer pool, using the ALTER BUFFPOOL command, when the queue manager was previously running have not occurred.

This message is only output if there is a difference between the definition for the buffer pool in the CSQINP1 data set and the log of the queue manager.

System action

The attribute values for the specified buffer pool are taken from the CSQINP1 data set rather than using the values stored in the log of the queue manager.

System programmer response

If the buffer pool definition in the CSQINP1 data set is the one you require, ignore the message.

Otherwise:

- Use the ALTER BUFFPOOL command to change the definition of the buffer pool, and also change its definition in CSQINP1 to match, or
- Remove the REPLACE attribute on the buffer pool definition in the CSQINP1 data set.

  Note, that instead of removing the REPLACE attribute you can specify the NOREPLACE attribute instead.

IMS adapter messages (CSQQ...): z/OS

CSQQ000I

IMS/TM *iiii* connected to queue manager *qqqq*

Severity
0

Explanation

This message is produced at the IMS master terminal when the IMS control region for IMS system *iiii* has successfully connected to queue manager *qqqq*.

CSQQ001I

IMS/TM *iiii* not connected to queue manager *qqqq*. Notify message accepted

Severity
0

Explanation

This message is produced at the IMS master terminal when the IMS control region for IMS system *iiii* has tried to connect to queue manager *qqqq* but the queue manager is not yet ready to make connections.

System action

The queue manager has accepted the notify message from IMS and when it is ready to make connections it will issue the z/OS command **MODIFY IMS** to cause IMS to attempt to make the connection again. IMS applications cannot access IBM MQ resources until the connection is made.

System programmer response

Resolve any other IBM MQ problems.

CSQQ002E

IMS/TM *iiii* failed to connect to queue manager *qqqq*, MQRC=*mqrc*

Severity
12

Explanation

This message is produced at the IMS master terminal when the IMS control region for IMS system *iiii* has failed to connect to queue manager *qqqq*. *mqrc* is the IBM MQ reason code for the failure.

System action

The IMS control region, and dependent regions are not connected to the queue manager. Any request from IMS applications for IBM MQ resources will fail.

System programmer response

Refer to API completion and reason codes for information about *mqrc* to determine the nature of the error.

CSQQ003E

IMS/TM *iiii* create thread failed while connecting to queue manager *qqqq*, MQRC=*mqrc*

Severity
12

Explanation

This message is produced at the IMS master terminal when the IMS control region for IMS system *iiii* has failed to connect to queue manager *qqqq*. *mqrc* is the IBM MQ reason code for the failure from theIBM MQ create thread function.

System action

The IMS control region, and dependent regions are not connected to the queue manager. Any request from IMS applications for IBM MQ resources will fail.

System programmer response

Refer to API completion and reason codes for information about *mqrc* to determine the cause of the problem.

CSQQ004E

IMS/TM *iiii* inquire indoubt failed while connecting to queue manager *qqqq*, MQRC=*mqrc*

Severity

12

Explanation

This message is produced at the IMS master terminal when the IMS control region for IMS system *iiii* has failed to connect to queue manager *qqqq*. *mqrc* is the IBM MQ reason code for the failure from the IBM MQ inquire indoubt function.

System action

The IMS control region, and dependent regions are not connected to the queue manager. Any request from IMS applications for IBM MQ resources will fail.

System programmer response

Refer to API completion and reason codes for information about *mqrc* to determine the nature of the error.

CSQQ005E

IMS/TM *iiii* establish exit failed while connecting to queue manager *qqqq*, MQRC=*mqrc*

Severity

12

Explanation

This message is produced at the IMS master terminal when the IMS control region for IMS system *iiii* has failed to connect to queue manager *qqqq*. *mqrc* is the IBM MQ reason code for the failure from IBM MQ establish exit function.

System action

The IMS control region, and dependent regions are not connected to the queue manager. Any request from IMS applications for IBM MQ resources will fail.

System programmer response

Refer to API completion and reason codes for information about *mqrc* to determine the cause of the error.

CSQQ007E

IMS/TM *iiii* resolve indoubt failed while connecting to queue manager *qqqq*, MQRC=*mqrc*

Severity

4

Explanation

This message is produced at the IMS master terminal when the queue manager has failed to resolve indoubt units of recovery during the connection process. *mqrc* is the IBM MQ reason code for the resolve in-doubt function failure.

System action

The IMS control region, and dependent regions are connected to the queue manager. IMS applications can access IBM MQ resources.

System programmer response

For information about resolving the IBM MQ unit of recovery associated with the in-doubt IMS unit of work, see Recovering IMS units of recovery manually.

CSQQ008I

nn units of recovery are still in doubt in queue manager *qqqq*

Severity

4

Explanation

This message is produced at the IMS master terminal when the queue manager has units of recovery still in doubt after all the IMS units of work have been resolved.

System action

The IMS control region, and dependent regions are connected to the queue manager. IMS applications can access IBM MQ resources.

System programmer response

See How in-doubt units of recovery are resolved from IMS for information about resolving the IBM MQ unit of recovery associated with the in-doubt IMS unit of work.

CSQQ010E

Error resolving unit of recovery *uuuu* (OASN *nnnn*) in queue manager *qqqq*, MQRC=*mqrc*

Severity

4

Explanation

This message is produced at the IMS master terminal when the queue manager is unable to resolve an indoubt unit of recovery. *uuuu* is the unit of work identifier in the same format as the reply from the DISPLAY THREAD command. *nnnn* is the IMS OASN (origin application sequence number), in decimal format.

System action

The IMS control region, and dependent regions are connected to the queue manager. IMS applications can access IBM MQ resources.

System programmer response

See the How in-doubt units of recovery are resolved from IMS for information about resolving the IBM MQ unit of recovery associated with the in-doubt IMS unit of work.

CSQQ011E

IMS/TM *iiii* terminate identify failed for connection to queue manager *qqqq*, MQRC=*mqrc*

Severity

12

Explanation

This message is produced at the IMS master terminal when the IMS control region for IMS system *iiii* has failed to disconnect from the queue manager *qqqq*. *mqrc* is the return code for the failure from the IBM MQ terminate identify function.

System action

The IMS control region, and dependent regions are not connected to the queue manager. Any request from IMS applications for IBM MQ resources will fail.

System programmer response

Refer to API completion and reason codes for information about *mqrc* to determine the cause of the error.

CSQQ013I

MQ commands cannot be issued using the /SSR command

Severity

4

Explanation

This message is produced at the IMS master terminal when the /SSR IMS command is used to issue an IBM MQ command; IBM MQ commands cannot be issued in this way.

System action

None

CSQQ014E

Unit of recovery *uuuu* (OASN *nnnn*) was not committed in queue manager *qqqq*

Severity

4

Explanation

This message is produced at the IMS master terminal when, following the abnormal termination of an application, the queue manager is unable to commit an indoubt unit of recovery as requested by IMS. *uuuu* is the unit of work identifier in the same format as the reply from the DISPLAY THREAD command. *nnnn* is the IMS OASN (origin application sequence number), in decimal format.

System action

The IMS control region, and dependent regions are connected to the queue manager. IMS applications can access IBM MQ resources.

System programmer response

See How in-doubt units of recovery are resolved from IMS for information about resolving the IBM MQ unit of recovery associated with the in-doubt IMS unit of work.

CSQQ015E

Unit of recovery *uuuu* (OASN *nnnn*) was not backed out in queue manager *qqqq*

Severity

4

Explanation

This message is produced at the IMS master terminal when, following the abnormal termination of an application, the queue manager is unable to back out an indoubt unit of recovery as

requested by IMS. *uuuu* is the unit of work identifier in the same format as the reply from the DISPLAY THREAD command. *nnnn* is the IMS OASN (origin application sequence number), in decimal format.

System action

The IMS control region, and dependent regions are connected to the queue manager. IMS applications can access IBM MQ resources.

System programmer response

See How in-doubt units of recovery are resolved from IMS for information about resolving the IBM MQ unit of recovery associated with the in-doubt IMS unit of work.

CSQQ100I

psb-name region-id Processing queue manager *name*

Severity

0

Explanation

This message identifies the queue manager that this instance of the IMS trigger monitor is connected to. *region-id* is the last four digits of the region identifier, or blank. This message is followed by message CSQQ110I, indicating the name of the initiation queue.

CSQQ101E

psb-name region-id Cannot open the initiation queue, MQCC=*mqcc* MQRC=*mqrc*

Severity

8

Explanation

CSQQTRMN has attempted to open an initiation queue, but the attempt was unsuccessful (for example, because the queue was not defined). *mqcc* and *mqrc* give the reason for the problem. *region-id* is the last four digits of the region identifier, or blank.

System action

CSQQTRMN ends.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc*, determine the cause of the problem, and restart CSQQTRMN.

CSQQ102E

psb-name region-id An IMS dl1-function call returned *pcb-status*

Severity

4

Explanation

A trigger message has been retrieved from the initiation queue which defines an IMS transaction to be started. However, the transaction cannot be started (for example, it cannot be found). *region-id* is the last four digits of the region identifier, or blank. *pcb-status* is the status code returned by IMS from the last *dl1-function* call.

System action

The trigger message is sent to the dead-letter queue. CSQQTRMN processes the next message.

System programmer response

See the *IMS/ESA® Application Programming: Data Communication* manual for information about *pcb-status*. Examine the trigger message on the dead-letter queue to find the IMS transaction name. Determine the reason for the problem, and restart the transaction.

CSQQ103E

psb-name region-id CSQQTRMN read a trigger message with an incorrect MQTM-StrucId of *struc-id*

Severity

4

Explanation

A trigger message has been retrieved, but the structure identifier of the message is not MQTM_STRUC_ID and so is not compatible with this version of CSQQTRMN. *region-id* is the last four digits of the region identifier, or blank.

System action

The trigger message is sent to the dead-letter queue. CSQQTRMN processes the next message.

System programmer response

Check the header of the message on the dead-letter queue. This will tell you where the trigger message came from. Correct the process that created the trigger message.

CSQQ104E

psb-name region-id CSQQTRMN does not support version *version*

Severity

4

Explanation

A trigger message has been retrieved, but the version identifier in MQTM is not version 1, and so is not compatible with this version of CSQQTRMN. *region-id* is the last four digits of the region identifier, or blank.

System action

The trigger message is sent to the dead-letter queue. CSQQTRMN processes the next message.

System programmer response

Check the header of the message on the dead-letter queue. This will tell you where the trigger message came from. Correct the process that created the trigger message.

CSQQ105E

psb-name region-id CSQQTRMN cannot start a process type of *type*

Severity

4

Explanation

A trigger message has been retrieved, but the process type in MQTM is not IMS, and so cannot be processed by this version of CSQQTRMN. *region-id* is the last four digits of the region identifier, or blank.

System action

The trigger message is sent to the dead-letter queue. CSQQTRMN processes the next message.

System programmer response

Check the header of the message on the dead-letter queue. This will tell you where the trigger message came from. Correct the process that created the trigger message.

CSQQ106E

psb-name region-id MQGET error, MQCC=*mqcc* MQRC=*mqrc*. CSQQTRMN will end

Severity

8

Explanation

An attempt to issue an `MQGET` call on the initiation queue has been unsuccessful. *region-id* is the last four digits of the region identifier, or blank. This message is followed by message CSQQ110I, indicating the name of the queue.

System action

CSQQTRMN ends.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* to determine the cause of the problem. Restart CSQQTRMN.

CSQQ107E

psb-name region-id Cannot connect to the queue manager, MQCC=*mqcc* MQRC=*mqrc*

Severity

8

Explanation

An attempt by the trigger monitor to connect to the queue manager identified in message CSQQ100I was unsuccessful. *region-id* is the last four digits of the region identifier, or blank.

System action

CSQQTRMN ends.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* to determine the cause of the problem.

CSQQ108I

psb-name region-id LTERM *lterm-name* not available. Switched to MASTER

Severity

4

Explanation

The LTERM specified to receive diagnostic messages cannot be used.

System action

Messages are sent to the master terminal.

System programmer response

Resolve why *lterm-name* was not available.

CSQQ109E

psb-name region-id MQCLOSE error, MQCC=*mqcc* MQRC=*mqrc*

Severity

8

Explanation

An attempt has been made to close a dead-letter queue, but the MQCLOSE call was unsuccessful. *region-id* is the last four digits of the region identifier, or blank. This message is followed by message CSQQ110I, indicating the name of the queue.

System action

CSQQTRMN ends.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* to determine the cause of the problem.

CSQQ110I

Queue name = *q-name*

Severity
0

Explanation

This message follows other messages and identifies the name of the queue in question. The accompanying messages indicate the event or problem associated with the queue.

CSQQ111E

psb-name region-id CSQQTRMN read a trigger message with an incorrect length of length

Severity
4

Explanation

This message is issued if the transaction CSQQTRMN receives a trigger message that does not match the MQTM control block. *region-id* is the last four digits of the region identifier, or blank.

System action

The message is sent to the dead-letter queue.

System programmer response

Look at the message on the dead-letter queue to establish why it did not match MQTM.

CSQQ112E

psb-name region-id MQOPEN error, MQCC=*mqcc* MQRC=*mqrc*

Severity
8

Explanation

An **MQOPEN** call has been unable to open a queue. *region-id* is the last four digits of the region identifier, or blank. This message is followed by message CSQQ110I indicating the name of the queue.

System action

CSQQTRMN ends.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* to determine the cause of the problem.

CSQQ113I

psb-name region-id This message cannot be processed

Severity

0

Explanation

When an attempt to process a message using an IBM MQ API call was unsuccessful, an attempt was made to put the message on the dead-letter queue. This was also unsuccessful and the *message-id* has been sent to the LTERM. *region-id* is the last four digits of the region identifier, or blank. This message is followed by message CSQ118I, indicating the message identifier.

System action

Processing continues.

System programmer response

Check for previous messages explaining why the dead-letter queue was not available (if a dead-letter queue has not been defined, no other messages relating to the problem will have been issued).

CSQQ114E

psb-name region-id MQINQ error, MQCC=*mqcc* MQRC=*mqrc*

Severity

8

Explanation

An attempt to use the `MQINQ` call to inquire about the attributes of a queue was unsuccessful. *region-id* is the last four digits of the region identifier, or blank. This message is followed by message CSQQ110I indicating the name of the queue.

System action

CSQQTRMN ends.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* to determine why an `MQINQ` call could not be made on the queue.

CSQQ115I

psb-name region-id Ending following termination of queue manager connection

Severity

0

Explanation

CSQQTRMN has terminated because the connection to the queue manager is no longer available.

CSQQ116E

psb-name region-id Cannot open the queue manager, MQCC=*mqcc* MQRC=*mqrc*

Severity

8

Explanation

An `MQOPEN` call to the queue manager was unsuccessful. *region-id* is the last four digits of the region identifier, or blank.

System action

CSQQTRMN ends.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* to determine the cause of the problem.

CSQQ117E

psb-name region-id Cannot query the queue manager, MQCC=*mqcc* MQRC=*mqrc*

Severity
8

Explanation

An MQINQ call to the queue manager was unsuccessful. *region-id* is the last four digits of the region identifier, or blank.

System action

CSQQTRMN ends.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* to determine the cause of the problem.

CSQQ118I

MsgID=*msg-id*

Severity
0

Explanation

This message follows message CSQQ113I, indicating the hexadecimal identifier of the message that could not be processed.

CSQQ119E

psb-name region-id Error *rc* from STORAGE OBTAIN

Severity
8

Explanation

CSQQTRMN tried to obtain storage, but received return code *rc* from z/OS.

System action

CSQQTRMN ends.

System programmer response

Determine the reason for the return code from the STORAGE OBTAIN request, and restart CSQQTRMN.

CSQQ120E

psb-name region-id MQPUT error, MQCC=*mqcc* MQRC=*mqrc*

Severity
8

Explanation

An attempt was made to put a message on a queue with an MQPUT call, but the attempt was unsuccessful. *region-id* is the last four digits of the region identifier, or blank. This message is followed by message CSQQ110I indicating the name of the queue.

System action

CSQQTRMN ends.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* to determine why an MQPUT call could not be made for the queue.

CSQQ121E

psb-name region-id Dead-letter queue is not defined for the queue manager

Severity
4

Explanation

A dead-letter queue has not been defined for the queue manager. *region-id* is the last four digits of the region identifier, or blank.

System action

The trigger message is discarded, and the process cannot be started.

System programmer response

Define a dead-letter queue if one is required.

CSQQ122E

psb-name region-id Cannot close the queue manager, MQCC=*mqcc* MQRC=*mqrc*

Severity
8

Explanation

CSQQTRMN was unable to close the queue manager after inquiring about the dead-letter queue. *region-id* is the last four digits of the region identifier, or blank.

System action

CSQQTRMN ends.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* to determine the cause of the problem.

CSQQ123E

psb-name region-id The dead-letter queue type is not QLOCAL

Severity
4

Explanation

The dead-letter queue defined was not of type local. *region-id* is the last four digits of the region identifier, or blank. This message is followed by message CSQQ110I, indicating the name of the queue.

System action

The message is not put to the dead-letter queue.

System programmer response

Define the dead-letter queue as a local queue.

CSQQ124E

psb-name region-id The dead-letter queue usage is not NORMAL

Severity
4

Explanation

The dead-letter queue defined is not of usage type normal. *region-id* is the last four digits of the region identifier, or blank. This message is followed by message CSQQ110I, indicating the name of the queue.

System action

The message is not put to the dead-letter queue.

System programmer response

Define the dead-letter queue to have usage type normal.

CSQQ125E

psb-name region-id No initiation queue identified

Severity
8

Explanation

CSQQTRMN did not find the initiation queue name in the input parameters.

System action

CSQQTRMN ends.

System programmer response

Examine the input parameters and look for other error messages to determine the reason for the failure. Restart CSQQTRMN.

CSQQ126E

psb-name region-id An IMS *call* call returned *pcb-status*

Severity
8

Explanation

A status code of *pcb-status* was returned from a DLI call.

System action

CSQQTRMN ends.

System programmer response

Determine the reason for the status code, and restart CSQQTRMN.

CSQQ150I

csect-name IBM MQ for z/OS V*n*

Severity
0

Explanation

This message is issued as part of the header to the report issued by the IMS trigger monitor program.

CSQQ151I

csect-name Trigger Monitor Input Report - *date time*

Severity
0

Explanation

This message is issued as part of the header to the report issued by the IMS trigger monitor program.

CSQQ152I

csect-name Unable to OPEN CSQQUT1 data set

Severity
8

Explanation

The IMS trigger monitor was unable to open the data set containing input control statements.

System action

Default values are used for the options.

System programmer response

Examine the error message that has been sent to the JES log to determine the reason for the error. Check that the data set has been correctly specified.

CSQQ153I

csect-name First token is not a valid keyword

Severity
8

Explanation

The input control statement does not start with a valid keyword.

System action

The statement is ignored.

System programmer response

Correct the syntax for the statement.

CSQQ159I

csect-name Trigger monitor options:

Severity
0

Explanation

The IMS trigger monitor has finished processing input control statements. The options that will be used follow.

Recovery manager messages (CSQR...): z/OS

CSQR001I
RESTART INITIATED

Explanation

This message delimits the beginning of the restart process within startup. The phases of restart are about to begin. These phases are necessary to restore the operational environment to that which existed at the time of the previous termination and to perform any recovery actions that might be necessary to return IBM MQ-managed resources to a consistent state.

CSQR002I
RESTART COMPLETED

Explanation

This message delimits the completion of the restart process within startup.

System action

Startup continues.

CSQR003I
RESTART - PRIOR CHECKPOINT RBA=*rba*

Explanation

The message indicates the first phase of the restart process is in progress and identifies the log positioning RBA of the checkpoint from which the restart process will obtain its initial recovery information.

System action

Restart processing continues.

CSQR004I
RESTART - UR COUNTS - IN COMMIT=*nnnn*, INDOUBT=*nnnn*, INFLIGHT=*nnnn*, IN BACKOUT=*nnnn*

Explanation

This message indicates the completion of the first phase of the restart process. The counts indicate the number of units of recovery with an execution state during a previous queue manager termination that indicates (to ensure MQ resource consistency) some recovery action must be performed during this restart process. The counts might provide an indication of the time required to perform the remaining two phases of restart (forward and backward recovery).

The IN COMMIT count specifies the number that had started, but not completed, phase-2 of the commit process. These must undergo forward recovery to complete the commit process.

The INDOUBT count specifies the number that were interrupted between phase-1 and phase-2 of the commit process. These must undergo forward recovery to ensure that resources modified by them are unavailable until their INDOUBT status is resolved.

The INFLIGHT count specifies the number that neither completed phase-1 of the commit process nor began the process of backing out. These must undergo backward recovery to restore resources modified by them to their previous consistent state.

The IN BACKOUT count specifies the number that were in the process of backing out. These must undergo backward recovery to restore resources modified by them to their previous consistent state.

System action

Restart processing continues.

CSQR005I
RESTART - FORWARD RECOVERY COMPLETE - IN COMMIT= *nnnn*, INDOUBT=*nnnn*

Explanation

The message indicates the completion of the forward recovery restart phase. The counts indicate the number of units of recovery with recovery actions that could not be completed during the phase. Typically, those in an IN COMMIT state remain because the recovery actions of some subcomponents have not been completed. Those units of recovery in an INDOUBT state will remain until connection is made with the subsystem that acts as their commit coordinator.

System action

Restart processing continues.

CSQR006I
RESTART - BACKWARD RECOVERY COMPLETE - INFLIGHT= *nnnn*, IN BACKOUT=*nnnn*

Explanation

The message indicates the completion of the backward recovery restart phase. The counts indicate the number of units of recovery with recovery actions that could not be completed during the phase. Typically, those in either state remain because the recovery actions of some subcomponents have not been completed.

System action

Restart processing continues.

CSQR007I
UR STATUS

Explanation

This message precedes a table showing the status of units of recovery (URs) after each restart phase. The message and the table will accompany the CSQR004I, CSQR005I, or CSQR006I message after each nested phase. At the end of the first phase, it shows the status of any URs that require processing. At the end of the second (forward recovery) and third (backout) phases, it shows the status of only those URs which needed processing but were not processed. The table helps to identify the URs that were active when the queue manager stopped, and to determine the log scope required to restart.

The format of the table is:

```
  T  CON-ID     THREAD-XREF     S   URID     TIME
```

The columns contain the following information:

T Connection type. The values can be:

- **B** Batch: From an application using a batch connection
- **R** RRS: From an RRS-coordinated application using a batch connection
- **C** CICS: From CICS
- **I** IMS: From IMS
- **S** System: From an internal function of the queue manager or from the channel initiator.

CON-ID
Connection identifier for related URs. Batch connections are not related to any other connection. Subsystem connections with the same identifier indicate URs that originated from the same subsystem.

THREAD-XREF
The recovery thread cross-reference identifier associated with the thread; see Connecting from the IMS control region for more information.

S Restart status of the UR. When the queue manager stopped, the UR was in one of these situations:

B INBACKOUT: the UR was in the must-complete phase of backout, and is yet to be completed

C INCOMMIT: the UR was in the must-complete phase of commit, and is yet to be completed

D INDOUBT: the UR had completed the first phase of commit, but IBM MQ had not received the second phase instruction (the UR must be remembered so that it can be resolved when the owning subsystem reattaches)

F INFLIGHT: the UR had not completed the first phase of commit, and will be backed out.

URID UR identifier, the log RBA of the beginning of this unit of recovery. It is the earliest RBA required to process the UR during restart.

TIME The time the UR was created, in the format *yyyymmdd hhmmss*. It is approximately the time of the first IBM MQ API call of the application or the first IBM MQ API call following a commit point.

CSQR009E
NO STORAGE FOR UR STATUS TABLE, SIZE REQUESTED= *xxxx*, REASON CODE=*yyyyyyyy*

Explanation

There was not enough storage available during the creation of the recoverable UR (unit of recovery) display table.

System action

Restart continues but the status table is not displayed.

System programmer response

Increase the region size of the xxxxMSTR region before restarting the queue manager.

CSQR010E
ERROR IN UR STATUS TABLE SORT/TRANSLATE, ERROR LOCATION CODE=*xxxx*

Explanation

An internal error has occurred.

System action

Restart continues but the status table is not displayed.

System programmer response

Note the error code in the message and contact your IBM support center.

CSQR011E
ERROR IN UR STATUS TABLE DISPLAY, ERROR LOCATION CODE=*xxxx*

Explanation

An internal error has occurred.

System action

Restart continues but the status table is not displayed.

System programmer response

Note the error code in the message and contact your IBM support center.

CSQR015E
CONDITIONAL RESTART CHECKPOINT RBA *rba* NOT FOUND

Explanation

The checkpoint RBA in the conditional restart control record, which is deduced from the end RBA or LRSN value that was specified, is not available. This is probably because the log data sets available for use at restart do not include that end RBA or LRSN.

System action

Restart ends abnormally with reason code X'00D99001' and the queue manager terminates.

System programmer response

Run the change log inventory utility (CSQJU003) specifying an ENDRBA or ENDLRSN value on the CRESTART control statement that is in the log data sets that are to be used for restarting the queue manager.

CSQR020I
OLD UOW FOUND

Explanation

During restart, a unit of work was found that predates the oldest active log. Information about the unit of work is displayed in a table in the same format as in message CSQR007I.

Old units of work can lead to extended restart times, as restart processing need to read archive logs to correctly process the unit of work. IBM MQ offers the opportunity to avoid this delay by allowing old units of work to be force committed.

Note: Force committing a unit of work can break the transactional integrity of updates between IBM MQ, and other resource managers involved in the original unit of work described in this message.

System action

Message CSQR021D is issued and the operator's reply is awaited.

CSQR021D
REPLY Y TO COMMIT OR N TO CONTINUE

Explanation

An old unit of work was found, as indicated in the preceding CSQR020I message.

System action

The queue manager waits for the operator's reply.

CSQR022I
OLD UOW COMMITTED, URID=*urid*

Explanation

This message is sent if the operator answers 'Y' to message CSQR021D.

System action

The indicated unit of work is committed.

CSQR023I
OLD UOW UNCHANGED, URID=*urid*

Explanation

This message is sent if the operator answers 'N' to message CSQR021D.

CSQR023I is also sent when an old unit of work which is already in the 'in-backout' state is identified. Units of work in the 'in-backout' state are ineligible for force commit processing as it can lead to a queue becoming unusable. For such units of work, the message CSQR021D is not issued, and no choice is possible.

System action

The indicated unit of work is left for handling by the normal restart recovery process.

CSQR026I
Long-running UOW shunted to RBA=*rba*, URID=*urid* connection name=*name*

Explanation

During checkpoint processing, an uncommitted unit of recovery was encountered that has been active for at least 3 checkpoints. The associated log records have been rewritten ('shunted') to a later point in the log, at RBA *rba*. The unit of recovery identifier *urid* together with the connection name *name* identify the associated thread.

System action

Processing continues.

System programmer response

Uncommitted units of recovery can lead to difficulties later, so consult with the application programmer to determine if there is a problem that is preventing the unit of recovery from being committed, and to ensure that the application commits work frequently enough.

CSQR027I
Long-running UOW shunting failed, URID=*urid* connection name=*name*

Explanation

During checkpoint processing, an uncommitted unit of recovery was encountered that has been acvtive for at least 3 checkpoints. However, the associated log records could not be rewritten ('shunted') to a later point in the log. The unit of recovery identifier *urid* together with the connection name *name* identify the associated thread.

System action

The unit of recovery is not shunted, and will not participate in any future log shunting.

System programmer response

The most likely cause is insufficient active log data sets being available, in which case you should add more log data sets for the queue manager to use. Use the DISPLAY LOG command or the print log map utility (CSQJU004) to determine how many log data sets there are and what their status is.

Uncommitted units of recovery can lead to difficulties later, so consult with the application programmer to determine if there is a problem that is preventing the unit of recovery from being committed, and to ensure that the application commits work frequently enough.

CSQR029I
INVALID RESPONSE - NOT Y OR N

Explanation

The operator did not respond correctly to the reply message CSQR021D. Either 'Y' or 'N' must be entered.

System action

The original message is repeated.

CSQR030I
Forward recovery log range from RBA=*from-rba* to RBA=*to-rba*

Explanation

This indicates the log range that must be read to perform forward recovery during restart.

System action

Restart processing continues.

CSQR031I
Reading log forwards, RBA=*rba*

Explanation

This is issued periodically during restart recovery processing to show the progress of the forward recovery phase. The log range that needs to be read is shown in the preceding CSQR030I message.

System action

Restart processing continues.

CSQR032I
Backward recovery log range from RBA=*from-rba* to RBA=*to-rba*

Explanation

This indicates the log range that must be read to perform backward recovery during restart.

System action

Restart processing continues.

CSQR033I
Reading log backwards, RBA=*rba*

Explanation

This is issued periodically during restart recovery processing to show the progress of the backward recovery phase. The log range that needs to be read is shown in the preceding CSQR032I message.

System action

Restart processing continues.

CSQR034I
Backward migration detected

Explanation

During queue manager restart it has been detected that one or more of the page sets that have been connected has been used at a higher version of queue manager code.

System action

The queue manager will automatically perform special processing during restart to alter any messages stored on those page sets so they can be read by the current version of the queue manager. This special processing is dependent on there being no unresolved units of work found at the end of restart, so you might be prompted by way of further messages during restart to force commit these.

Restart processing continues.

Topic manager messages (CSQT...): z/OS

CSQT806I

csect-name Queued Pub/Sub Daemon started

Severity
0

Explanation

Queued Pub/Sub Daemon started

System action

None

System programmer response

None

CSQT807I

csect-name Queued Pub/Sub Daemon ended

Severity
0

Explanation

The Queued Pub/Sub Daemon has ended.

System programmer response

None

CSQT809E

csect-name Unable to process publication, Queued Pub/Sub stream queue *queue-name* is GET(DISABLED)

Severity
8

Explanation

The stream queue, *queue-name*, has been GET(DISABLED) preventing the Queued Pub/Sub Daemon from processing publication messages.

System action

The Queued Pub/Sub Daemon will continue to process publication messages on other stream queues and subscriptions on all streams.

System programmer response

To resume processing publication messages alter the stream queue to be GET(ENABLED).

To quiesce the stream remove its name from SYSTEM.QPUBSUB.QUEUE.NAMELIST.

To quiesce the Queued Pub/Sub Daemon alter the queue manager to have PSMODE(COMPAT).

CSQT810E

csect-name Unable to process subscription requests, Queued Pub/Sub control queue is GET(DISABLED)

Severity
8

Explanation

The SYSTEM.BROKER.CONTROL.QUEUE has been GET(DISABLED) preventing the Queued Pub/Sub Daemon from processing subscription requests.

System action

The Queued Pub/Sub Daemon will continue to process publication messages on stream queues.

System programmer response

To resume processing subscription requests alter the SYSTEM.BROKER.CONTROL.QUEUE to be GET(ENABLED).

To quiesce the Queued Pub/Sub Daemon alter the queue manager to have PSMODE(COMPAT).

CSQT814E

csect-name Unable to resolve parent *queue_manager_name*

Severity

8

Explanation

In establishing a publish/subscribe hierarchy, the Queued Pub/Sub Daemon has been unable to resolve the parent *queue_manager_name*.

System action

The status of the publish/subscribe parent connection will be set to error.

System programmer response

Check that the parent queue manager is correctly specified.

Ensure that broker is able to resolve the queue manager name of the parent broker.

To resolve the queue manager name, at least one of the following resources must be configured:

- A transmission queue with the same name as the parent queue manager name.
- A queue manager alias definition with the same name as the parent queue manager name.
- A cluster with the parent queue manager a member of the same cluster as this queue manager.
- A cluster queue manager alias definition with the same name as the parent queue manager name.
- A default transmission queue, modify the parent queue manager name to blank, then set with the parent queue manager name.

CSQT816E

csect-name Unable to open Queued Pub/Sub control queue MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Severity

8

Explanation

The queue manager failed to open the Queued Publish/Subscribe control queue, SYSTEM.BROKER.CONTROL.QUEUE. The attempt to open the queue failed with completion code *mqcc* and reason *mqrc*. The most likely reasons for this error are that an application program has opened the control queue for exclusive access, or that the control queue has been defined incorrectly.

System action

The Queued Publish/Subscribe Daemon terminates.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form), then correct the problem and restart the Queued Publish/Subscribe interface.

CSQT817E

csect-name An invalid stream queue has been detected, queue *queue-name*

Severity

8

Explanation

The Pub/Sub Daemon attempted to use queue *queue-name* as a stream queue. The most likely reason for this error is that the queue is:

- Not a local queue.
- A shareable queue.
- A temporary dynamic queue.

System programmer response

Correct the problem with the queue *queue-name* or, if you do not intend to use it as a stream queue, remove it from the namelist SYSTEM.QPUBSUB.QUEUE.NAMELIST.

CSQT818E

csect-name Unable to open Queued Pub/Sub stream, queue *queue-name* MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity

8

Explanation

The queue manager has failed to open a stream queue *queue-name*. The attempt to open the queue failed with completion code *mqcc* and reason *mqrc*. The most likely reasons for this error are:

1. A new stream name has been added to SYSTEM.QPUBSUB.QUEUE.NAMELIST but the stream queue does not exist.
2. An application has the queue open for exclusive access.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQT819E

csect-name Queued Pub/Sub stream *stream-name* ended abnormally, reason=*mqrc*

Severity

8

Explanation

The Pub/Sub Daemon stream (*stream-name*) has ended abnormally for reason *mqrc*. The *mqrc* could be an internal return code. The queue manager will attempt to restart the stream. If the stream should repeatedly fail then the Pub/Sub Daemon will progressively increase the time between attempts to restart the stream.

System programmer response

Investigate why the problem occurred and take appropriate action to correct the problem. If the problem persists, save any generated output files and use the MQ Support site to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

CSQT820E

csect-name Queued Pub/Sub stream *stream-name* restarted

Severity
8

Explanation

The queue manager has restarted a stream that ended abnormally. This message will frequently be preceded by message CSQT819E indicating why the stream ended.

System programmer response

Correct the problem.

CSQT821E

csect-name Unable to contact parent *queue_manager_name*, reason=*mqrc*

Severity
8

Explanation

In establishing a publish/subscribe hierarchy, the Queued Pub/Sub Daemon is unable to send a message to the parent *queue_manager_name* for reason *mqrc*.

System action

The status of the publish/subscribe parent connection will be set to error.

System programmer response

Investigate why the problem occurred and determine a resolution.

To reattempt a parent queue manager connection:

- Set the parent queue manager name to blank.
- Take appropriate action to correct the problem.
- Re-specify the parent queue manager name

CSQT822E

csect-name Failed to register with parent *queue_manager_name*, reason *mqrc (mqrc-text)*

Severity
8

Explanation

The Queued Pub/Sub Daemon started and the PARENT queue manager was set to *queue_manager_name* in a queue manager attribute. The queue manager attempted to register as a child of the parent, but received an exception response indicating that it was not possible. The queue manager will retry to register periodically as a child. The child may not be able to process global publications or subscriptions correctly until this registration process has completed normally.

System programmer response

Investigate why the problem occurred and take appropriate action to correct the problem. The problem is likely to be caused by the parent queue manager not yet existing, or a problem with the transmission queue at the parent queue manager.

CSQT824I

csect-name Topic *topic-1* is dependent on PROXYSUB(FORCE) of topic *topic-2* from a different Pub/Sub hierarchy stream

Severity
4

Explanation

Topic object *topic-1* is a publish/subscribe hierarchy stream. Topic object *topic-2* is higher in the topic tree and has been configured with **PROXYSUB(FORCE)**, which results in a single wildcard proxy subscription being sent to the neighboring queue managers in the publish/subscribe hierarchy that support the *topic-2* stream. No further individual proxy subscriptions will be sent for any subscriptions made below *topic-2* in the topic tree, including below topic object *topic-1*. If a neighboring queue manager supports the *topic-1* stream, but not the *topic-2* stream, publications will not be sent to subscriptions to topic *topic-1* on this queue manager from that neighbor.

System programmer response

If the behavior described in the explanation is intended then no action is required. If not, alter the **PROXYSUB** attribute on topic *topic-1*, or *topic-2*, so both, or neither topics, are configured with the value **FORCE**.

CSQT826E

csect-name Failed to propagate subscription, stream *stream-name*, to queue manager *qm-name*, MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity

8

Explanation

The queue manager failed to propagate subscription to stream *stream-name* at queue manager *queue_manager_name* with reason code *mqrc*. An application has either registered or unregistered a subscription to stream *stream-name*. The queue manager has attempted to propagate the subscription change to the queue manager, but the request has not been successful. Messages published on the stream through the queue manager might not reach this queue manager.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

Investigate why the problem occurred and take appropriate action to correct the problem.

Use the following command to refresh proxy subscriptions:

```
REFRESH QMGR TYPE(PROXYSUB)
```

CSQT827E

csect-name Queued Pub/Sub internal subscription failed. Stream *stream-name* to queue manager *queue_manager_name* reason=*reason* MQRC= *mqrc*

Severity

8

Explanation

The queue manager failed to subscribe to stream *stream-name* at queue manager *queue_manager_name* with reason code *mqrc*. Related queue managers learn about each others configuration by subscribing to information published by each other. A queue manager discovered that one of these internal subscriptions has failed. The queue manager will reissue the subscription immediately. The queue manager cannot function correctly without knowing some information about neighboring queue managers. The information that this broker has about queue manager *queue_manager_name* is not complete and this could lead to subscriptions and publications not being propagated around the network correctly.

System programmer response

Investigate why the problem occurred and take appropriate action to correct the problem. The most likely cause of this failure is a problem with the transmission queue at the queue manager *queue_manager_name* or a problem with the definition of the route between this queue manager and queue manager *queue_manager_name*

CSQT831E

csect-name Unable to make subscription, reason=*mqrc* (*mqrc-text*), subscription name *sub-name*, topic *topic-string*

Severity

8

Explanation

A failure occurred while attempting to create a subscription to topic string *topic-string* using the subscription name *sub-name*. The associated reason code is *mqrc*. The *mqrc* could be an internal return code.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQT833E

csect_name Queue manager *queue_manager_name* introduced a loop into the Pub/Sub hierarchy

Severity

8

Explanation

The queue manager *queue_manager_name* introduced a loop in the Pub/Sub hierarchy. The Queued Publish/Subscribe Daemon on this queue manager will terminate immediately.

System programmer response

Remove queue manager *queue_manager_name* from the hierarchy, either by deleting the queue manager, or by removing knowledge of the queue manager's parent, using the ALTER QMGR PARENT(' ') command, or in exceptional circumstances, RESET QMGR TYPE(PUBSUB) PARENT(*queue_manager_name*).

CSQT834E

csect-name Conflicting queue manager names in the Pub/Sub hierarchy

Severity

8

Explanation

The names of the queue managers (*queue_manager_name*) and (*queue_manager_name*) in the Pub/Sub hierarchy both start with the same 12 characters. The first 12 characters of a queue manager name should be unique to ensure that no confusion arises within the hierarchy, and to guarantee unique message ID allocation.

CSQT835E

csect-name Unable to inform parent *parent-name* of new relation *queue_manager_name*, reason=*mqrc (mqrc-text)*

Severity

8

Explanation

The queue manager failed to notify its parent queue manager *parent-name* of the relation *queue_manager_name* in the Pub/Sub hierarchy. The notification message will be put to the parent's dead-letter queue. A failure to notify a queue manager of a new relation will mean that no loop detection can be performed for the new relation.

System programmer response

Diagnose and correct the problem on the parent queue manager. One possible reason for this is that the parent queue manager does not yet exist.

CSQT836E

csect-name Duplicate queue manager name *queue_manager_name* located in the Pub/Sub hierarchy

Severity
8

Explanation

Multiple instances of the queue manager name *queue_manager_name* have been located. This could either be the result of a previously resolved loop in the Pub/Sub hierarchy, or multiple queue managers in the Pub/Sub hierarchy having the same name.

System programmer response

If this queue manager introduced a loop in the hierarchy (typically identified by message CSQT833E), this message can be ignored. It is strongly recommended that every queue manager in a Pub/Sub hierarchy has a unique name. It is not recommended that multiple queue managers use the same name.

CSQT839E

csect-name Unexpected topology information received from queue manager *queue_manager_name*

Severity
8

Explanation

A queue manager has received a distributed publish/subscribe communication that it did not expect. The message was sent by queue manager *queue_manager_name*. The message will be processed according to the report options in that message. The most likely reason for this message is that the queue manager topology has been changed while distributed publish/subscribe communication messages were in transit (for example, on a transmission queue) and that a message relating to the previous queue manager topology has arrived at a queue manager in the new topology. This message may be accompanied by an informational FFST including details of the unexpected communication.

System programmer response

If the queue manager topology has changed and the queue manager named in the message is no longer related to the queue manager issuing this message, this message can be ignored. If the **RESET QMGR TYPE(PUBSUB)** command was issued to unilaterally remove knowledge of queue manager *queue_manager_name* from this queue manager, the **RESET QMGR TYPE(PUBSUB)** command should also be used to remove knowledge of this queue manager from queue manager *queue_manager_name*.

CSQT844E

csect-name The relation with *queue_manager_name* is unknown

Severity
8

Explanation

The RESET QMGR TYPE(PUBSUB) command has been issued in an attempt to remove a queue manager's knowledge of a relation of that queue manager. The relative *queue_manager_name* is unknown at queue manager *queue_manager_name*. If the parent KEYWORD was specified, the queue manager does not currently have a parent. If the CHILD keyword was specified, the queue manager does not recognize the named child.

System programmer response

Investigate why the queue manager is unknown.

CSQT848E

csect-name Failed to register proxy subscription for queue manager *qmgr-name*, stream *stream-name*, topic string *topic-string*, reason=*mqrc* (*mqrc-text*)

Severity

8

Explanation

The queue manager received a proxy subscription request for stream *stream-name* and topic *topic-string* from queue manager *qmgr-name*. The attempt to register the subscription was unsuccessful for reason *mqrc* (*mqrc-text* provides the MQRC in textual form). Messages published upon this topic will not be delivered to subscriptions on the relation queue manager.

System programmer response

Use the reason code to investigate why the failure occurred and take appropriate action to correct the problem. Use the command REFRESH QMGR TYPE(PROXYSUB) on the relation queue manager to refresh it's proxy subscriptions.

CSQT852E

csect-name Unable to propagate delete publication command, topic *topic-name*, stream *stream-name*, to queue manager *queue_manager_name*, reason=*mqrc* (*mqrc-text*)

Severity

8

Explanation

The queue manager failed to propagate delete publication command for stream *stream-name* to related queue manager *queue_manager_name* for reason *mqrc*. When an application issues a delete publication command to delete a global publication, the command has to be propagated to all queue managers in the sub-hierarchy supporting the stream. The queue manager reporting the error has failed to forward a delete publication command to a related queue manager *queue_manager_name* who supports stream *stream-name*. Delete publication commands are propagated without MQRO_DISCARD_MSG and the command message might have been written to a dead-letter queue. The topic for which the delete publication has failed is *topic-name*.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

If the delete publication has failed because the stream has been deleted at the related queue manager, this message can be ignored. Investigate why the delete publication has failed and take the appropriate action to recover the failed command.

CSQT853E

csect-name Unable to propagate delete publication command, topic *topic-name*, stream *stream-name*, relation *relation-name*, reason = *mqrc* (*mqrc-text*)

Severity

8

Explanation

The queue manager failed to propagate a delete publication command for stream *stream-name* to a previously related queue manager *relation-name*. In some cases the stream or the relation cannot be determined and so is shown as '????'.

When an application issues a delete publication command to delete a global publication, the command is propagated to all queue managers in the sub-hierarchy supporting the stream. The queue manager topology was changed after deleting the publication, but before a queue manager removed by the topology change processed the propagated delete publication message. The topic for which the delete publication has failed is *topic-name*. In some cases the topic cannot be determined and so is shown as '????'.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

It is the user's responsibility to quiesce queue manager activity before changing the queue manager topology using the RESET QMGR TYPE(PUBSUB) command. Investigate why this delete publication activity was not quiesced. The delete publication command will have been written to the dead-letter queue at the queue manager that was removed from the topology. In this case, further action might be necessary to propagate the delete publication command that was not quiesced before the RESET QMGR TYPE(PUBSUB) command was issued.

CSQT854E

csect-name Unable to propagate delete publication command, topic *topic-name*, stream *stream-name* to queue manager *queue_manager_name*

Severity

8

Explanation

When an application issues a delete publication command, the command has to be propagated to all queue managers in the sub-hierarchy supporting the stream. At the time the delete publication was propagated, queue manager *queue_manager_name* was a known relation of this message queue manager supporting stream *stream-name*. Before the delete publication command arrived at the related queue manager, the queue manager topology was changed so that queue manager *queue_manager_name* no longer supported stream *stream-name*. The topic for which the delete publication has failed is *topic-name*.

System programmer response

It is the user's responsibility to quiesce queue manager activity before changing the stream topology of the queue manager. Investigate why this delete publication activity was not quiesced. The delete publication command will have been written to the dead-letter queue at queue manager *queue_manager_name*.

CSQT855E

csect-name Queued Pub/Sub Daemon failed, reason=*mqrc*

Severity

8

Explanation

An attempt has been made to run the queued publish/subscribe interface (Queued Pub/Sub Daemon) but the interface has ended for reason *mqrc*.

If *mqrc* is a number in the range of 2000 - 3000, it is an API reason code. If it is of the form 5*nnn*, it is a queued publish/subscribe message code associated with the message CSQT *nnn*E, which is normally issued previously.

System programmer response

If *mqrc* is an API reason code, see API completion and reason codes for more information about the *mqrc*. If *mqrc* is a queued publish/subscribe message code, see the corresponding message explanation for more information. Where no such message exists, see "Queued Publish/Subscribe message codes" on page 5226 for the corresponding message number.

Determine why the queued publish/subscribe daemon ended. The message logs for the Channel Initiator might contain more detailed information about why the queued publish/subscribe daemon cannot be started. Resolve the problem that is preventing the daemon from completing and restart the Channel Initiator.

CSQT856E

csect-name Unable to process publish command message for stream *stream-name*, reason=*mqrc (mqrc-text)*

Severity

8

Explanation

The Queued Pub/Sub Daemon failed to process a publish message for stream *stream-name*. The queue manager was unable to write the publication to the dead-letter queue and was not permitted to discard the publication. The queue manager will temporarily stop the stream and will restart the stream and consequently retry the publication after a short interval.

System programmer response

Investigate why the error has occurred and why the publication cannot be written to the dead-letter queue. Either manually remove the publication from the stream queue, or correct the problem that is preventing the queue manager from writing the publication to the dead-letter queue.

CSQT857E

csect-name Unable to process control command message, reason=*mqrc (mqrc-text)*

Severity

8

Explanation

The Queued Pub/Sub Daemon failed to process a command message on the SYSTEM.BROKER.CONTROL.QUEUE. The queue manager was unable to write the command message to the dead-letter queue and was not permitted to discard the command message. The queue manager will temporarily stop the stream and will restart the stream and consequently retry the command message after a short interval. Other queue manager control commands cannot be processed until this command message has been processed successfully or removed from the control queue.

System programmer response

Investigate why the error has occurred and why the command message cannot be written to the dead-letter queue. Either, manually remove the command message from the stream queue, or correct the problem that is preventing the broker from writing the command message to the dead-letter queue.

CSQT858E

csect-name Unable to send publication to subscriber queue, queue *queue-name*, to queue manager *queue_manager_name*, reason=*mqrc* (*mqrc-text*)

Severity

8

Explanation

A failure has occurred sending a publication to subscriber queue *queue-name* at queue manager *queue_manager_name* for reason *mqrc*. The broker configuration options prevent it from recovering from this failure by discarding the publication or by sending it to the dead-letter queue. Instead the queue manager will back out the unit of work under which the publication is being sent and retry the failing command message a fixed number of times. If the problem still persists, the queue manager will then attempt to recover by failing the command message with a negative reply message. If the issuer of the command did not request negative replies, the queue manager will either discard or send to the dead-letter queue the failing command message. If the queue manager configuration options prevent this, the queue manager will restart the affected stream, which will reprocess the failing command message again. This behavior will be repeated until such time as the failure is resolved. During this time the stream will be unable to process further publications or subscriptions.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

Usually the failure will be due to a transient resource problem, for example, the subscriber queue, or an intermediate transmission queue, becoming full. Use reason code *mqrc* to determine what remedial action is required. If the problem persists for a long time, you will notice the stream being continually restarted by the queue manager. Evidence of this occurring will be a large number of CSQT820E messages, indicating stream restart, being written to the Channel Initiator log. In such circumstances, manual intervention will be required to allow the queue manager to dispose of the failing publication. To do this, you will need to end the Queued Pub/Sub Daemon using the ALTER QMGR PSMODE(COMPAT), change the appropriate queue manager attributes; PSNPMSG, PSNPRES, PSSYNCPT, and restart it using ALTER QMGR PSMODE(ENABLED). This will allow the publication to be sent to the rest of the subscribers, while allowing the Queued Pub/Sub Daemon to discard or send to the dead-letter queue the publication that could not be sent.

CSQT859E

csect-name Queued Pub/Sub stream *stream-name* terminating, reason=*mqrc (mqrc-text)*

Severity

8

Explanation

The stream *stream-name* has run out of internal resources and will terminate with reason code *mqrc* (*mqrc-text* provides the MQRC in textual form). If the command in progress was being processed under syncpoint control, it will be backed out and retried when the stream is restarted by the queue manager. If the command was being processed out of syncpoint control, it will not be able to be retried when the stream is restarted.

System programmer response

This message should only be issued in very unusual circumstances. If this message is issued repeatedly for the same stream, and the stream is not especially large in terms of subscriptions, topics, and retained publications, save all generated diagnostic information and use either the IBM MQ Support site, or IBM Support Assistant (ISA) to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

CSQT864E

csect-name Unable to put a reply message, queue *queue-name* queue manager(*qm-name*) MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity

8

Explanation

While processing a publish/subscribe command, the queue manager could not send a reply message to the queue *queue-name* at the queue manager *qm-name* for MQRC=*mqrc*. The queue manager was also unable to write the message to the dead-letter queue. Since the command is being processed under syncpoint control, the queue manager will attempt to retry the command in the hope that the problem is only of a transient nature. If, after a set number of retries, the reply message still could not be sent, the command message will be discarded if the report options allow it. If the command message cannot be discarded, the stream will be restarted, and processing of the command message recommenced.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

Use reason code *mqrc* to determine what remedial action is required. If the failure is due to a resource problem (for example, a queue being full), you might find that the problem has already cleared itself. If not, this message will be issued repeatedly each time the command is retried. In this case you are strongly advised to define a dead-letter queue to receive the reply message so that the Queued Pub/Sub Daemon can process other commands while the problem is being investigated. Check the application from which the command originated and ensure that it is specifying its reply-to queue correctly.

CSQT866E

csect-name Queued Pub/Sub command message discarded. Reason=*mqrc (mqrc-text)*

Severity

8

Explanation

The queue manager failed to process a publish/subscribe command message, which has now been discarded. The queue manager will begin to process new command messages again.

System programmer response

Look for previous error messages to indicate the problem with the command message. Correct the problem to prevent the failure from happening again.

CSQT875E

csect-name Unable to put message to the dead-letter-queue, reason=*mqrc (mqrc-text)* (DLH reason=*mqrc2 (mqrc2-text)*)

Severity

8

Explanation

The queue manager attempted to put a message to the dead-letter queue *queue-name* but the message could not be written to the dead-letter queue for reason *mqrc (mqrc-text* provides the MQRC in textual form). The message was being written to the dead-letter-queue with a reason of *mqrc2* (*mqrc2-text* provides the MQRC in textual form).

System programmer response

Determine why the message cannot be written to the dead-letter-queue. Also, if the message was not deliberately written to the dead-letter-queue, for example by a channel exit, determine why the message was written to the dead-letter-queue and resolve the problem that is preventing the message from being sent to its destination.

CSQT876E

csect-name Parent conflict detected in Pub/Sub hierarchy with queue manager *queue_manager_name*

Severity

8

Explanation

The queue manager *queue_manager_name* has been started, naming this queue manager as its parent. This queue manager has already named queue manager *queue_manager_name* as its parent. The queue manager will send an exception message to the queue manager *queue_manager_name* indicating that a conflict has been detected. The most likely reason for this message is that the queue manager topology has been changed while distributed publish/subscribe communication messages were in transit (for example, on a transmission queue) and that a message relating to the previous queue manager topology has arrived at a queue manager in the new topology. This message might be accompanied by an informational FFST including details of the unexpected communication.

System programmer response

If the queue manager topology has changed and the queue manager named in the message no longer identifies this queue manager as its parent, this message can be ignored - for example, if the command ALTER QMGR PARENT(' ') was issued. If queue manager *queue_manager_name* has been defined as this queue manager's parent, and this queue manager has been defined as queue manager *queue_manager_name*'s parent, the ALTER QMGR command should be used to resolve the conflict by specifying the correct PARENT.

CSQT882E

csect-name Message written to the dead-letter queue, for reason=*mqrc (mqrc-text)*

Severity

8

Explanation

The queue manager has written a message to the dead-letter queue for reason *mqrc* (*mqrc-text* provides the MQRC in textual form). Note. After the first occurrence of this message for a stream, it will only be written periodically.

System programmer response

Determine why the message was written to the dead-letter queue, and resolve the problem that is preventing the message from being sent to its destination.

CSQT883E

csect-name Queued Pub/Sub state not recorded

Severity

0

Explanation

The Queued Pub/Sub state on stream *stream-name* not recorded while processing a publication outside of syncpoint. A nonpersistent publication has requested a change to either a retained message or a publisher registration. This publication is being processed outside of syncpoint because the queue manager has been configured with the queue manager attribute PSSYNCPT set to IFPER. A failure has occurred hardening either the publisher registration or the retained publication to the queue manager's local queue. All state changes attempted as a result of this publication will be backed-out. Processing of the publication will continue and the queue manager will attempt to deliver it to all subscribers.

System programmer response

Investigate why the failure occurred. It is probably due to a resource problem occurring on the queue manager. The most likely cause is 'queue full' on a queue. If your publications also carry state changes, you are advised to send them either as persistent publications or set the queue manager attribute PSSYNCPT to YES. In this way, they will be carried out under syncpoint and the queue manager can retry them in the event of a failure such as this.

CSQT884E

csect-name Queued Pub/Sub control queue is not a local queue

Severity

8

Explanation

The queue manager has detected that the queue SYSTEM.BROKER.CONTROL.QUEUE exists and is not a local queue. This makes the queue unsuitable for use as the control queue. The Pub/Sub Daemon task will terminate immediately.

System programmer response

Delete the definition of the existing queue and, if required, re-create the queue to be of type MQQT_LOCAL.

CSQT895I

csect-name Queued Pub/Sub Daemon detected missing retained messages

Severity

4

Explanation

The Queued Pub/Sub Daemon uses retained messages to communicate with other members of publish subscribe hierarchies.

The retained message was missing and has been republished.

System action

Retained messages seem to have been removed from the SYSTEM.RETAINED.PUB.QUEUE. The Queued Pub/Sub Daemon has attempted to recover by republishing retained messages.

System programmer response

If you are unaware of a reason why retained messages have been removed this might be a symptom of a more serious problem that requires further investigation.

CSQT899E

csect-name Unable to establish parent relationship to child queue manager *qmname*

Severity

8

Explanation

The queue manager is unable to establish the requested parent relationship to queue manager *qmname* because that queue manager is already a child.

System action

The existing child relationship to queue manager *qmname* remains unaffected.

System programmer response

To prevent this message being issued, the parent definition on the queue manager must be removed by issuing the **ALTER QMGR PARENT(' ')** MQSC command. To ensure that the required topology is established, review the existing parent definitions and update appropriately.

CSQT960I

csect-name Distributed Pub/Sub command processor stopped

Severity

0

Explanation

The distributed Pub/Sub command processor stopped. This may be for one of three reasons:

- The channel initiator is stopping.
- The channel initiator is starting and the queues used by the distributed Pub/Sub command processor have not been defined because distributed Pub/Sub command processor is not required.
- An error has occurred

System action

Processing continues, but distributed Pub/Sub is not available.

System programmer response

If an error has occurred, investigate the problem reported in the preceding messages.

CSQT961I

csect-name Distributed Pub/Sub publication processor stopped

Severity

0

Explanation

The distributed Pub/Sub publication processor stopped. This can be for one of three reasons:

- The channel initiator is stopping.
- The channel initiator is starting and the queues used by the distributed Pub/Sub command processor have not been defined because distributed Pub/Sub publication processor is not required.
- An error has occurred

System action

Processing continues, but distributed Pub/Sub is not available.

System programmer response

If an error has occurred, investigate the problem reported in the preceding messages.

CSQT962I

csect-name Distributed Pub/Sub proxy-subscription fan out processor stopped

Severity

0

Explanation

The distributed Pub/Sub proxy-subscription stopped. This can be for one of three reasons:

- The channel initiator is stopping.
- The channel initiator is starting and the queues used by the distributed pub/sub proxy-subscription fan out processor have not been defined because distributed pub/sub proxy-subscription fan out processor is not required.
- An error has occurred

System action

Processing continues, but distributed Pub/Sub is not available.

System programmer response

If an error has occurred, investigate the problem reported in the preceding messages.

CSQT963E

csect-name Queued pub/sub daemon unavailable

Severity
8

Explanation

The Distributed publish/subscribe process was unable to contact the Queued Pub/Sub Daemon. The problem will be reported in preceding messages.

System action

Hierarchical connections cannot be processed until the problem is rectified.

System programmer response

Investigate the problem reported in the preceding messages. When the Daemon becomes available, it might be necessary to issue the REFRESH QMGR TYPE(PROXYSUB) command to resynchronize subscriptions.

CSQT964I

csect-name Pub/Sub hierarchy relation connected, (queue manager *qmgr-name*)

Severity
0

Explanation

A publish/subscribe hierarchy connection has been established with child or parent queue manager *qmgr-name*.

CSQT965I

csect-name Pub/Sub hierarchy relation disconnected, (queue manager *qmgr-name*)

Severity
0

Explanation

A publish/subscribe hierarchy connection has ended with child or parent queue manager *qmgr-name*.

CSQT966E

csect-name A previous publication is being incorrectly processed again

Severity
8

Explanation

A publication, previously processed by this queue manager, has been received.

This is caused by an invalid configuration of a hierarchy and a pub/sub cluster.

System action

This message will not be re-published and will be processed according to the message's report options. Additional messages might be written if this publication is sent to the dead-letter queue.

System programmer response

Correct the configuration to remove the loop. Check the message properties in the dead-letter queue to determine the route taken.

CSQT967E

csect-name Unable to deliver proxy subscription to queue manager *queue_manager_name*, reason=*mqrc (mqrc-text)*

Severity

8

Explanation

Unable to deliver proxy subscription to queue manager *queue_manager_name*. Reason code: *mqrc* (*mqrc-text* provides the MQRC in textual form).

This might result in subscriptions not receiving publications from *queue_manager_name*.

System programmer response

Correct the configuration to allow proxy subscriptions to be delivered to *queue_manager_name*. When the problem has been resolved, it will be necessary to perform a **REFRESH QMGR TYPE(PROXYSUB)** to resynchronize subscriptions.

CSQT968I

csect-name Topic *topic-1* in cluster *cluster_name* is dependent on PROXYSUB(FORCE) of topic *topic-2*

Severity

4

Explanation

Topic object *topic-1* is defined in cluster *cluster_name*, and is below topic object *topic-2* in the topic tree. Topic object *topic-2* has been configured with **PROXYSUB(FORCE)** to generate a wildcard proxy subscription, so no further individual proxy subscriptions are sent for any subscriptions made below *topic-2* in the topic tree. However, *topic-2* is not in the same cluster as *topic-1*, and the wildcard proxy subscription is not sent to neighboring queue managers in the cluster in which *topic-1* is defined. Therefore, publications from cluster *cluster_name* might not be sent to subscriptions to *topic-1* on this queue manager.

System programmer response

If the behavior described in the explanation is intended no action is required. If not, alter the **PROXYSUB** attribute on topic *topic-1*, or *topic-2*, so both, or neither topics, are configured with the value **FORCE**.

CSQT971E

csect-name task failed to quiesce

Severity

8

Explanation

The indicated Distributed Publish/Subscribe task was requested to quiesce but failed to do so within the timeout interval.

There are four classes of task:

Distributed Pub/Sub Publish Task

Receives publications from remote queue managers in a Publish/Subscribe cluster and republishes into the local queue manager

Distributed Pub/Sub Command Task
Receives command messages from remote queue managers in a Publish/Subscribe cluster to create or cancel proxy subscriptions on behalf of remote queue managers.

Distributed Pub/Sub Fan Out Task
Sends command messages to remote queue managers in Publish/Subscribe clusters and Publish/Subscribe hierarchies in response to changes in the local queue manager state.

Distributed Pub/Sub Controller
Controls the starting and stopping of the Distributed Publish/Subscribe tasks during channel initiator startup and shutdown and also when enabling and disabling Publish/Subscribe.

System action

The Queued Pub/Sub Daemon will be forcibly closed.

System programmer response

Check the job log for additional messages, or an FFST™, that might explain why the task has failed to quiesce.

CSQT972E

csect-name Unable to put Distributed Pub/Sub fan-out request to *q-name*, reason=*mqrc (mqrc-text)*

Severity
8

Explanation

An attempt to put a subscription fan-out request on the distributed publish/subscribe fan-out request queue *q-name* failed with reason code *mqrc (mqrc-text* provides the MQRC in textual form).

CSQT973E

csect-name Distributed Pub/Sub subscribing inhibited, topic string *topic-string*, (queue manager *qm-name*)

Severity
8

Explanation

Topic *topic-string* has been disabled for subscribe. This prevents distributed publish/subscribe from creating a subscription on behalf of another queue manager *qm-name* within the topology.

CSQT974E

csect-name Distributed Pub/Sub publication inhibited, topic string *topic-string*

Severity
8

Explanation

Topic *topic-string* has been disabled for publish. This prevents distributed publish/subscribe from publishing a message received from another queue manager within the topology. This message will not be re-published and will be processed according to the report options in the message. Additional messages will be written if this publication is sent to the dead-letter queue.

CSQT975I

csect-name task has started

Severity
0

Explanation

The indicated Distributed Publish/Subscribe task has started. This message typically occurs during channel initiator startup, or when enabling Publish/Subscribe.

There are four classes of task:

Distributed Pub/Sub Publish Task
Receives publications from remote queue managers in a Publish/Subscribe cluster and republishes into the local queue manager

Distributed Pub/Sub Command Task
Receives command messages from remote queue managers in a Publish/Subscribe cluster to create or cancel proxy subscriptions on behalf of remote queue managers.

Distributed Pub/Sub Fan Out Task
Sends command messages to remote queue managers in Publish/Subscribe clusters and Publish/Subscribe hierarchies in response to changes in the local queue manager state.

Distributed Pub/Sub Controller
Controls the starting and stopping of the Distributed Publish/Subscribe tasks during channel initiator startup and shutdown, and also when enabling and disabling Publish/Subscribe.

System action

None.

System programmer response

None.

CSQT976I

csect-name task has stopped

Severity
0

Explanation

The indicated Distributed Publish/Subscribe task has stopped. This message typically occurs during channel initiator shutdown, or when disabling Publish/Subscribe.

There are four classes of task:

Distributed Pub/Sub Publish Task
Receives publications from remote queue managers in a Publish/Subscribe cluster and republishes into the local queue manager

Distributed Pub/Sub Command Task
Receives command messages from remote queue managers in a Publish/Subscribe cluster to create or cancel proxy subscriptions on behalf of remote queue managers.

Distributed Pub/Sub Fan Out Task
Sends command messages to remote queue managers in Publish/Subscribe clusters and Publish/Subscribe hierarchies in response to changes in the local queue manager state.

Distributed Pub/Sub Controller
Controls the starting and stopping of the Distributed Publish/Subscribe tasks during channel initiator startup and shutdown and also when enabling and disabling Publish/Subscribe.

System action

None.

System programmer response

None.

CSQT977I

csect-name Establishing Pub/Sub hierarchy relation, (queue manager *qmgr-name*)

Severity
0

Explanation

The queue manager is establishing a Publish/Subscribe hierarchy connection with a child or parent queue manager *qmgr-name*.

System action

None.

System programmer response

None.

CSQT978E

csect-name Unable to create/cancel proxy subscription, for queue manager *queue_manager_name*, topic string *topic-string*, reason=*mqrc* (*mqrc-text*)

Severity
8

Explanation

The Distributed Pub/Sub Command Task is unable to create or cancel a proxy subscription for queue manager *queue_manager_name* on topic *topic-string* for reason code *mqrc* (*mqrc-text* provides the MQRC in textual form).

A failure to create or cancel a proxy subscription will result in this queue manager not having a correct knowledge of subscriptions on other queue managers in the Publish/Subscribe topology. This may result in this queue manager not delivering publications to other queue managers.

System programmer response

Correct the cause of the indicated reason code.

Once the problem has been resolved it may be necessary to perform a REFRESH QMGR TYPE(PROXYSUB) command to resynchronise any subscriptions.

CSQT979E
csect-name Distributed Pub/Sub proxy subscription from qmgr-name rejected due to PSCLUS(DISABLED)

Explanation

A cluster subscription has been sent to this queue manager over a channel from qmgr-name but the queue manager attribute PSCLUS has been set to DISABLED, indicating that Publish/Subscribe activity is not expected between queue managers in this cluster.

System action

The proxy subscription request is ignored and no subscription is locally registered.

System programmer response

To enable publish/subscribe clustering, alter the PSCLUS attribute on all queue managers in the cluster to ENABLED. You may also need to issue **REFRESH CLUSTER** and **REFRESH QMGR** commands

as detailed in the documentation for the PSCLUS attribute. If you are not using publish/subscribe clusters you should delete the clustered topic object, and ensure PSCLUS is DISABLED on all queue managers.

CSQT980I

csect-name Distributed Pub/Sub proxy subscription re-synchronization completed

Severity

0

Explanation

During restart processing the Distributed Pub/Sub process was unable to determine that the proxy subscription state was consistent so a re-synchronization with remote queue managers has been performed.

This is usually seen when a queue manager was not quiesced cleanly during its previous shutdown, or when the system was particularly busy at that time.

System action

Processing continues.

System programmer response

None.

CSQT981E

csect-name Distributed Pub/Sub disabled whilst in a Pub/Sub cluster

Severity

4

Explanation

This queue manager is a member of a Publish/Subscribe cluster but Publish/Subscribe has been disabled.

System action

Other queue managers within the Publish/Subscribe Cluster will continue to send publications and proxy subscriptions to this queue manager. They will accumulate on the Publish/Subscribe Cluster system queues and will not be processed until Publish/Subscribe is enabled. If these queues become full channel failure may occur, which will affect the operation of Publish/Subscribe on other queue managers in the cluster. This will also affect the delivery of other messages, unrelated to Publish/Subscribe, that are sent to this queue manager from other queue managers within the cluster.

System programmer response

Enable Publish/Subscribe by setting **PSMODE** to **ENABLED** or **COMPAT** with the **ALTER QMGR** command then the **REFRESH QMGR TYPE(PROXYSUB)** command should be issued to resynchronise subscriptions.

CSQT982E

csect-name Queued Pub/Sub disabled whilst in a Pub/Sub hierarchy

Severity

4

Explanation

This queue manager is a member of a Publish/Subscribe hierarchy but Queued Publish/Subscribe has been disabled.

System action

Any parent-child relations within the Publish/Subscribe hierarchy will continue to send publications and proxy subscriptions to this queue manager. They will accumulate on the Queued Publish/Subscribe system queues and will not be processed until Queued Publish/Subscribe is enabled. If the Queued Publish/Subscribe system queues become full channel failure may occur, which will affect the operation of Publish/Subscribe on parent-child relations sending messages to this queue manager. This will also affect the delivery of other messages, unrelated to Publish/Subscribe, that are to be delivered using the same channels.

System programmer response

Enable Queued Publish/Subscribe by setting **PSMODE** to **ENABLED** with the **ALTER QMGR** command. Once Queued Publish/Subscribe has been restarted, use the **DISPLAY PUBSUB ALL** command to confirm this has completed, the **REFRESH QMGR TYPE(PROXYSUB)** command must be issued to resynchronize

CSQT983E

csect-name task failed, reason *mqrc (mqrc-text)*, retry in *n* minutes

Severity

4

Explanation

The *task* encountered a problem and will retry the command in *n* minutes. Earlier messages might have been issued in the queue manager or system error logs providing additional detail.

This message might be issued by a number of tasks:

Distributed Pub/Sub Publish Task

Other queue managers within the cluster will continue to send publications to this queue manager. The publications will accumulate on the Publish/Subscribe Cluster system queue (SYSTEM.INTER.QMGR.PUBS) and will not be processed until the problem is resolved. If these queues become full channel failure might occur, which will affect the operation of Publish/Subscribe on other queue managers in the cluster. This will also affect the delivery of other messages, unrelated to Publish/Subscribe, that are sent to this queue manager from other queue managers in the cluster.

Distributed Pub/Sub Command Task

Other queue managers within the cluster will continue to send proxy subscriptions to this queue manager. Subscriptions will accumulate on the Publish/Subscribe Cluster system queue (SYSTEM.INTER.QMGR.CONTROL) and will not be processed until the problem is resolved. Other queue managers will not receive publications from this queue manager on topics for which proxy subscriptions have yet to be processed. If the Publish/Subscribe Cluster system queue becomes full channel failure might occur, which will affect the operation of Publish/Subscribe on other queue managers in the cluster. This will also affect the delivery of other messages, unrelated to Publish/Subscribe, that are sent to this queue manager from other queue managers in the cluster.

Distributed Pub/Sub Fan Out Task

This task will not send proxy subscription messages to other queue managers in a Publish/Subscribe Cluster or parent-child relations within a Publish/Subscribe hierarchy until the problem is rectified. On topics for which proxy subscriptions have yet to be sent, this queue manager will not receive publications from other queue managers in a Publish/Subscribe Cluster, or parent-child relations in a Publish/Subscribe hierarchy.

System action

None

System programmer response

If possible, rectify the identified problem, or contact your IBM support center.

When the problem has been rectified wait for *task* to retry the command.

CSQT984E

csect-name task has encountered *n* occurrences of reason *mqrc (mqrc-text)* while attempting to process a message.

Severity

4

Explanation

The *task* is currently unable to process a message due to reason *mqrc* (*mqrc-text* provides the MQRC in textual form). The task has encountered this *n* times; it will continue to retry the command until the problem has been rectified.

This message might be issued by a number of tasks:

Distributed Pub/Sub Publish Task

Other queue managers within the cluster will continue to send publications to this queue manager. Publications will accumulate on the Publish/Subscribe Cluster system queue (SYSTEM.INTER.QMGR.PUBS) and will not be processed until the problem is resolved. If these queues become full channel failure might occur, which will affect the operation of Publish/Subscribe on other queue managers in the cluster. This will also affect the delivery of other messages, unrelated to Publish/Subscribe, that are sent to this queue manager from other queue managers within the cluster.

Distributed Pub/Sub Command Task

Other queue managers within the cluster will continue to send proxy subscriptions to this queue manager. Subscriptions will accumulate on the Publish/Subscribe Cluster system queue (SYSTEM.INTER.QMGR.CONTROL) and will not be processed until the problem is resolved. Other queue managers will not receive publications from this queue manager on topics for which proxy subscriptions have yet to be processed. If the Publish/subscriber Cluster system queue becomes full channel failure might occur, which will affect the operation of Publish/Subscribe on other queue managers in the cluster. This will also affect the delivery of other messages, unrelated to Publish/Subscribe, that are sent to this queue manager from other queue managers within the cluster.

Distributed Pub/Sub Fan Out Task

This task will not send proxy subscription messages to other queue managers in a Publish/Subscribe Cluster or parent-child relations within a Publish/Subscribe hierarchy until the problem is rectified. On topics for which proxy subscriptions have yet to be sent this queue manager will not receive publications from other queue managers in a Publish/Subscribe Cluster or parent-child relations within a Publish/Subscribe hierarchy.

System action

None

System programmer response

If possible, rectify the identified problem, or contact your IBM support center.

When the problem has been rectified wait for *task* to retry the command.

CSQT987E

csect-name task failed due to reason *mqrc (mqrc-text)* Retry in *n* minutes

Severity

4

Explanation

The *task* encountered a problem. Earlier messages might have been issued in the queue manager or system error logs providing additional detail. The task will retry the command in *n* minutes.

Other queue managers within the cluster will continue to send proxy subscriptions to this queue manager. They will accumulate on the Publish/Subscribe cluster system queue and will not be processed until the problem is resolved.

Other queue managers will not receive publications from this queue manager on topics for which proxy subscriptions have yet to be processed.

If the Publish/subscriber cluster system queue becomes full, channel failure might occur, which will affect the operation of Publish/Subscribe on other queue managers in the cluster. This will also affect the delivery of other messages, unrelated to Publish/Subscribe, that are sent to this queue manager from other queue managers within the cluster.

System action

None

System programmer response

If possible, rectify the identified problem, or contact your IBM support center.

When the problem has been rectified wait for *task* to retry the command.

CSQT988E

csect-name task failed due to reason *mqrc (mqrc-text)* Retry in *n* minutes

Severity

4

Explanation

The *task* encountered a problem. Earlier messages might have been issued in the queue manager or system error logs providing additional detail. The task will retry the command in *n* minutes.

This task will not send proxy subscription messages to other queue managers in a Publish/Subscribe cluster or parent-child relations within a Publish/Subscribe hierarchy until the problem is rectified.

On topics for which proxy subscriptions have yet to be sent, this queue manager will not receive publications from other queue managers in a Publish/Subscribe cluster or parent-child relations within a Publish/Subscribe hierarchy.

System action

None

System programmer response

If possible, rectify the identified problem, or contact your IBM support center.

When the problem has been rectified wait for *task* to retry the command.

CSQT989E

csect-name task has encountered *n* occurrences of reason *mqrc (mqrc-text)* while attempting to process a message.

Severity

4

Explanation

The *task* is currently unable to process a message due to reason *mqrc (mqrc-text)*. Note that (*(mqrc-text)* provides the MQRC in textual form).

The task has encountered this *n* times. The task continues to retry the command until the problem has been rectified.

Other queue managers within the cluster continue to send proxy subscriptions to this queue manager. The subscriptions will accumulate on the Publish/Subscribe cluster system queue and will not be processed until the problem is resolved.

Other queue managers will not receive publications from this queue manager on topics for which proxy subscriptions have yet to be processed.

If the Publish/subscriber cluster system queue becomes full, channel failure might occur, which will affect the operation of Publish/Subscribe on other queue managers in the cluster.

This will also affect the delivery of other messages, unrelated to Publish/Subscribe, that are sent to this queue manager from other queue managers within the cluster.

System action

None

System programmer response

If possible, rectify the identified problem, or contact your IBM support center.

When the problem has been rectified wait for *task* to retry the command.

CSQT990E

csect-name task has encountered *n* occurrences of reason *mqrc (mqrc-text)* while attempting to process a message.

Severity

4

Explanation

The *task* is currently unable to process a message due to reason *mqrc (mqrc-text)*. Note that (*(mqrc-text)* provides the MQRC in textual form).

The task has encountered this *n* times. The task continues to retry the command until the problem has been rectified.

This task will not send proxy subscription messages to other queue managers in a Publish/Subsccribe Cluster or parent-child relations within a Publish/Subscribe hierarchy until the problem is rectified.

On topics for which proxy subscriptions have yet to be sent this queue manager will not receive publications from other queue managers in a Publish/Subscribe cluster or parent-child relations within a Publish/Subscribe hierarchy.

System action

None

System programmer response

If possible, rectify the identified problem, or contact your IBM support center.

When the problem has been rectified wait for *task* to retry the command.

CSQT991I

csect-name task has recovered from previous error condition

Severity

0

Explanation

The *task* has recovered from the previously reported error condition.

System action

Processing continues.

System programmer response

None.

CSQT992E

csect-name task has written a message to the dead-letter queue, reason *mqrc (mqrc-text)*

Severity

8

Explanation

The *task* has written a message to the dead-letter queue due to reason*mqrc* (*mqrc-text* provides the MQRC in textual form).

If *task* is the Distributed Pub/Sub Command Task, other queue managers will not receive publications from this queue manager on any topics for which this message is requesting proxy subscriptions be created.

If *task* is the Distributed Pub/Sub Fan Out Task, this queue manager will not receive publications from other queue managers on any topics for which this is requesting a proxy subscription be created.

System programmer response

Determine why the message was written to the dead-letter queue, and resolve the problem that is preventing the message from being sent to its destination.

If *task* is the Distributed Pub/Sub Command Task, or the Distributed Pub/Sub Fan Out Task, it may be necessary to issue the **REFRESH QMGR TYPE(PROXYSUB)** command when the problem has been resolved to resynchronize the subscription state with other queue managers.

CSQT996E

csect-name Creation of proxy subscription failed on queue manager *qmgr-name*, cluster *cluster_name*, topic string *topic-string*, reason=*mqrc (mqrc-text)*

Severity

8

Explanation

The proxy subscription in publish/subscribe cluster *cluster_name* on topic *topic-string* could not be created on queue manager *qmgr-name* due to reason *mqrc* (*mqrc-text* provides the MQRC in textual form). The failure to create the proxy subscription will prevent publications made on queue manager *qmgr-name* to topic *topic-string* being delivered to subscriptions on this queue manager. If this queue manager is also participating as a member of a publish/subscribe hierarchy any subscriptions to topic *topic-string* on other members of the publish/subscribe hierarchy will not receive publications from queue manager *qmgr-name*.

System programmer response

Correct the cause of the indicated reason code on queue manager *qmgr-name*. When the problem has been resolved issue the **REFRESH QMGR TYPE(PROXYSUB)** command on the remote queue manager to resynchronize the subscription state with other queue managers.

CSQT997E

csect-name Cancellation of proxy subscription failed on queue manager *qmgr-name*, cluster *cluster_name*, topic string *topic-string*, reason=*mqrc (mqrc-text)*

Severity
8

Explanation

The proxy subscription in publish/subscribe cluster *cluster_name* on topic *topic-string* could not be canceled on queue manager *qmgr-name* due to reason *mqrc* (*mqrc-text* provides the MQRC in textual form). The failure to cancel the proxy subscription will result in publications made on queue manager *qmgr-name* to topic *topic-string* to continue being delivered to this queue manager.

System programmer response

Correct the cause of the indicated reason code on queue manager *qmgr-name*. When the problem has been resolved issue the **REFRESH QMGR TYPE(PROXYSUB)** command on the remote queue manager to resynchronize the subscription state with other queue managers.

CSQT998E

csect-name Proxy subscription re-synchronization failed on queue manager *qmgr-name*, cluster *cluster_name*, reason=*mqrc (mqrc-text)*

Severity
8

Explanation

The request to resynchronize the subscription state with other queue managers in publish/subscribe cluster *cluster_name* failed on queue manager *qmgr-name* due to reason *mqrc* (*mqrc-text* provides the MQRC in textual form). There might be topic strings for which proxy subscriptions have not been created. Publications made on queue manager *qmgr-name* to those topics will not be delivered to subscriptions on this queue manager. If this queue manager is also participating as a member of a publish/subscribe hierarchy any subscriptions to those topics on other members of the publish/subscribe hierarchy will not receive publications from queue manager *qmgr-name*. There might also be topic strings for which proxy subscriptions have not been canceled on queue manager *qmgr-name*. Any publications made on that queue manager will continue to be delivered to this queue manager.

System programmer response

Correct the cause of the indicated reason code on queue manager *qmgr-name*. When the problem has been resolved issue the **REFRESH QMGR TYPE(PROXYSUB)** command on the remote queue manager to resynchronize the subscription state with other queue managers.

CSQT999E

csect-name task has encountered a message that is not valid on queue *queue*

Severity
4

Explanation

The queue *queue* is for exclusive use by the internal queue manager task *task*, and is used to maintain a distributed publish/subscribe topology. The task has encountered a message on the queue that is not valid

System action

The message is processed according to its report options. Additional console messages might be output if the message is put to the dead-letter queue.

An informational FFST, including details of the message that is not valid, might also be generated.

System programmer response

Ensure no applications put messages directly to the named queue, and ensure message exits do not alter system messages put to the queue. If the problem persists contact your IBM support center.

Utilities messages (CSQU...): z/OS

CSQU000I
csect-name IBM MQ for z/OS V*n*

Explanation

This is part of the header to the report issued by the utility program.

CSQU001I
csect-name Queue Manager Utility - *date time*

Explanation

This is part of the header to the report issued by the utility program.

System action

The message is followed by a copy of the function statements from the SYSIN data set.

CSQU002E
Unable to get storage of size *n* bytes, return code=*ret-code*

Explanation

An attempt to obtain some storage failed.

System action

The function is terminated, and any queue updates are backed out.

System programmer response

If you encounter this error when submitting JCL to run CSQUTIL functions, make sure that you have defined an adequate value for the **REGION** size parameter or set the **REGION** size to `0M` in the JCL. For example:

```
//SCOPY EXEC PGM=CSQUTIL,REGION=0M //STEPLIB DD DISP=SHR,DSN=thlqual.SCSQANLE
 //DD DISP=SHR,DSN=thlqual.SCSQAUTH ...
```

For more details about setting the **REGION** parameter, see Copying queues into a data set while the queue manager is running (COPY).

See the *MVS Programming: Assembler Services Reference* manual for information about the return code from the STORAGE or GETMAIN request.

CSQU003E
Unable to free storage at *address*, return code=*ret-code*

Explanation

An attempt to release storage at address *address* back to the system failed.

System action

The program usually ignores the error and continues with its function.

System programmer response

See the *MVS Programming: Assembler Services Reference* manual for information about the return code from the STORAGE or FREEMAIN request.

CSQU005I
COMMIT successfully completed

Explanation

An MQCMIT call returned a completion code of MQCC_OK.

CSQU006I
BACKOUT successfully completed

Explanation

An MQBACK call returned a completion code of MQCC_OK.

System action

The function is terminated.

System programmer response

Investigate the error that caused the backout to be done.

CSQU007E
MQCMIT failed. MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

The utility program was unable to commit the last set of changes.

System action

The updates are backed out, and the function is terminated.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form). Resubmit the job if required.

CSQU008E
MQBACK failed. MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

The utility program was unable to back out the last set of changes.

System action

None, the function is already being terminated because of the error that led to attempting the backout.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form). Resubmit the job if required.

CSQU009E
MQCONN failed for *conn-id*. MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

An attempt to connect to a queue manager or queue-sharing group named *conn-id* was unsuccessful.

System action

The requested function is not performed.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form). Resubmit the job if required.

CSQU010E
MQDISC failed for *conn-id*. MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

An attempt to disconnect from a queue manager or queue-sharing group named *conn-id* was unsuccessful.

System action

The utility program terminates. (This is not an error, because the disconnection request is the last function that the utility program processes.)

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQU011I
Commands from CSQINPX - *date time*

Explanation

This follows message CSQU000I as part of the header to the messages that indicate the progress of the utility program.

It is produced when the utility is invoked by distributed queuing to handle the CSQINPX data set.

CSQU012I
Initialization command handling completed

Explanation

The initialization command handler, which processes the CSQINPX command data set, completed successfully.

CSQU013E
Initialization command handling failed, RC=*return-code*

Explanation

The initialization command handler, which processes the CSQINPX command data set, did not complete successfully. *return-code* shows the type of error:

00000008
Some or all of the commands were not processed.

0000000C
Severe error; this is most likely because the CSQINPX or CSQOUTX data sets are defined erroneously.

System action

The initialization command handler ends, but the channel initiator continues.

System programmer response

Refer to the CSQOUTX data set and to the preceding messages for more information about the error.

For information about the initialization command handler and the CSQINPX or CSQOUTX data sets, see Initialization and configuration files. For information about the COMMAND statement, see Issuing commands to IBM MQ (COMMAND).

CSQU020E
Unable to OPEN *ddname* data set

Explanation

The program was unable to open data set *ddname*.

System action

If the SYSPRINT or SYSIN data sets cannot be opened, the utility program terminates. For other data sets, the function requesting them is not performed.

System programmer response

Examine the error message that was sent to the job log to determine the reason for the error. Check that the data set was correctly specified.

CSQU021E
Data set *ddname* does not have a record format of VBS

Explanation

The program opened the data set *ddname*, but the data set did not have a record format of VBS.

System action

If the LOAD input data set cannot be opened, the utility program terminates.

System programmer response

Examine the error message that was sent to the job log to determine the reason for the error. Check that the data set was correctly specified and is of the correct record format.

CSQU023E
Unable to CLOSE *ddname* data set

Explanation

The input data set *ddname* is still open after a request was made to close it.

System action

The program continues with its termination procedures.

System programmer response

Examine the error message that was sent to the job log to determine the reason for the error. Check that the data set was correctly specified.

CSQU030E
Page *nn* in data set *ddname* is invalid

Explanation

The utility program encountered a page that is invalid in the page set data set *ddname*. If the page number is 0, it might be that the data set is not the page set that is implied by *ddname*.

System action

The function is terminated.

System programmer response

Check that the page set has not been corrupted, and that the page set number corresponds to the DDname.

CSQU031E
Queue *q-name* with disposition QMGR or COPY does not exist

Explanation

The specified queue does not exist with disposition QMGR or COPY. (There might be such a queue with disposition SHARED, but the SCOPY function does not operate on shared queues.)

System action

The function is terminated.

System programmer response

Check the queue name that was specified.

CSQU032E
Page set *psid* is invalid

Explanation

The utility program encountered a page set that is invalid. The page set is in an inconsistent state and so the stand-alone utility functions cannot process it.

System action

The function is terminated.

System programmer response

This might be the result of performing a fuzzy backup (as described in How to back up and recover pagesets) or because the queue manager terminated abnormally. Restart the queue manager and then terminate it normally.

CSQU036E
Utility not available - restricted functionality

Explanation

The utility cannot operate because the installation and customization options chosen for IBM MQ do not allow all functions to be used.

System action

The utility is terminated.

CSQU037I
function has been stabilized with *version* function

Explanation

The utility function identified by *function* has been stabilized with the functional capabilities of version *version*.

System action

Processing continues and additional messages might be output providing further information.

System programmer response

Review the use of the utility function.

CSQU038I
Use runmqsc -n on your client machine for client channel definitions from version 8.0.

Explanation

The MAKECLNT utility function of CSQUTIL, that generates a client channel definition table (CCDT), has been stabilized.

From version 8.0, use the runmqsc utility on the client machine to generate the CCDT instead.

CSQU040E
Unable to GET from *ddname* data set

Explanation

The program was unable to read a record from the *ddname* data set.

System action

The function is terminated, and any queue updates are backed out.

System programmer response

Examine the error message that was sent to the job log to determine the reason for the error. Check that the data set was correctly specified.

CSQU043E
Unable to PUT to *ddname* data set

Explanation

The program was unable to write the next record to the *ddname* data set. Either the data set was not opened, or there was a QSAM error.

System action

The function is terminated, and any queue updates are backed out.

System programmer response

Examine the error message that was sent to the job log to determine the reason for the error. Check that the data set was correctly specified.

CSQU044I
Commands cannot be made for queue managers other than the target, *qmgr-name*

Explanation

Some of the DISPLAY object commands for the COMMAND function with MAKEDEF, MAKEREP, MAKEALT, or MAKEDEL used the CMDSCOPE option, and so information about objects for queue managers other than the target queue manager *qmgr-name* was received. Commands are not generated for such objects.

System programmer response

Avoid using CMDSCOPE in conjunction with the MAKEDEF, MAKEREP, MAKEALT, or MAKEDEL options. Use a separate COMMAND function for each target queue manager, with separate data sets for each set of generated commands.

CSQU045I
n data records read

Explanation

This indicates how many data records were read from the input data set specified by the DATA keyword for the current function.

CSQU046I
Making client channel definitions in *ddname* data set using CCSID *ccsid*

Explanation

This indicates that the COMMAND function will build client channel definitions in data set *ddname*, and that the data will have a coded character set identifier of *ccsid*.

CSQU047E
Unable to convert data for client channel definitions. MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

When building a client channel definition file, data for a channel or authentication information object could not be converted from the character set used by the queue manager to that requested by the CCSID keyword.

System action

The channel or authentication information definition is not built.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form). Resubmit the job if required.

CSQU048I
n authentication objects included, *m* excluded

Explanation

This indicates, for the current function, how many sets of authentication information were included in the client channel definition file, and how many were excluded. Authentication information may be excluded because:

- the LDAPUSER and LDAPPWD attributes are not blank
- there are too many sets of information
- there was a data conversion error.

System programmer response

If some information was excluded, check that the authentication information objects were selected correctly.

CSQU049I
n client channel definitions made

Explanation

This indicates how many client channel definitions were made by the current function.

CSQU050E
Command of length *length* is too long. Command rejected

Explanation

In the COMMAND function, the assembled command had more than 32 762 characters.

System action

The command is ignored, and no more commands are processed.

System programmer response

Check that the command is correctly formed according to the concatenation rules

CSQU051E
Command responses not received after *n* seconds

Explanation

In the COMMAND function, get processing for a response was timed out whilst more responses were expected.

System action

The next command will be processed, unless there have been too many timeouts.

System programmer response

Increase the value of RESPTIME, especially if the command is being sent to a remote queue manager, and check the remote queue definitions.

If the problem persists, check the definitions of the system-command input queue and the system-command reply queue; ensure that they are enabled for MQGET and MQPUT. If the definitions are correct, stop and restart the command server.

CSQU052E
Too many timeouts

Explanation

In the COMMAND function, get processing for a response timed out four times.

System action

No more commands are processed.

System programmer response

See message CSQU051E.

CSQU053E
DISPLAY command response not recognized

Explanation

In the COMMAND function, the responses to a DISPLAY command were not as expected.

System action

The DISPLAY command response is shown as is, rather than being formatted. The next command is processed.

System programmer response

Check the load libraries used are consistent with the queue manager being used.

Contact your IBM support center to report the problem.

CSQU054I
Executing function for object type *objtyp*

Explanation

The utility program is executing function *function* to process objects of the type indicated.

CSQU055I
Target queue manager is *qmgr-name*

Explanation

This indicates which queue manager your commands are directed to.

CSQU056I
Making commands in *ddname* data set

Explanation

This indicates that commands for the COMMAND function with MAKEDEF, MAKEREP, MAKEALT, or MAKEDEL, or for the SDEFS function will be built in data set *ddname*.

CSQU057I
n commands read

Explanation

This indicates how many commands were read from the command input data set by the current function.

CSQU058I
n commands issued and responses received, *m* failed

Explanation

This indicates, for the current function, how many commands were sent and produced responses, and how many of these did not execute successfully.

CSQU059I
n cmd commands made

Explanation

This indicates how many commands (called *cmd*) were made for the current function.

CSQU060E

Incorrect data length for message *msg-no*. *act-length* bytes found, *exp-length* bytes expected

Severity

8

Explanation

In the LOAD or SLOAD function, when attempting to read the record for message number *msg-no* for the queue being processed, the actual record length was found to be different to the expected record length.

System action

Processing for the command is terminated.

System programmer response

Check that the data set was created by the COPY function.

CSQU061E

An error occurred accessing the *in-ddname* data set for message *msg-no*. Reason=*reason-code*

Explanation

When executing the LOAD, SLOAD or ANALYZE function and attempting to read message *msg-no* for queue being processed, an error was detected. The reason code specifies the specific error, as follows:

4 First record in the data set does not identify a queue

8 Unexpected end-of-file

12 Unknown record type

System action

Processing for the command is terminated.

System programmer response

Check that the data set was created by the COPY function, and is not corrupted.

CSQU062E

Incorrect format data record

Explanation

In the LOAD function, the utility program encountered a record that it does not recognize while reading from the input data set.

System action

The function is terminated, and any queue updates are backed out.

System programmer response

Check that the data set was created by the COPY function, and is not corrupted.

CSQU063E

The *in-ddname* data set is empty

Severity

8

Explanation

When executing the LOAD, SLOAD or ANALYZE function, the input data set (DDname *in-ddname*) was empty.

System action

Processing for the command is terminated.

System programmer response

Check that the data set was successfully created by the COPY function.

CSQU070I
Command processing stopped

Explanation

In the COMMAND function, with FAILURE(STOP) specified, a command did not execute successfully.

System action

No more commands are processed.

CSQU071E
Incomplete command

Explanation

In the COMMAND function, end of data on the input data set was reached before the building of a command was complete.

System action

The command is ignored. There are no more commands to process.

System programmer response

Check that the command is correctly formed according to the concatenation rules.

CSQU080E
MQCLOSE failed for queue *q-name*. MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

The MQCLOSE call for *q-name* was unsuccessful. If this is for the system-command input queue when using the COMMAND function, message CSQU055I follows showing the target queue manager that was being used.

System action

The function is terminated.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form). Resubmit the job if required.

CSQU082E
MQGET failed for queue *q-name*. MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

The MQGET call for *q-name* was unsuccessful.

System action

The function is terminated, and any queue updates are backed out.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form). Resubmit the job if required.

CSQU083E
MQOPEN failed for queue *q-name*. MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

The MQOPEN call for *q-name* was unsuccessful. If the queue is a model queue, the requested dynamic queue name is appended in parentheses. If this is for the system-command input queue when using the COMMAND function, message CSQU055I follows showing the target queue manager that was being used.

System action

The function is terminated, and all queue updates are backed out.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form). Resubmit the job if required.

CSQU085E
MQPUT failed for queue *q-name*. MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

The MQPUT call for *q-name* was unsuccessful. If this is for the system-command input queue when using the COMMAND function, message CSQU055I follows showing the target queue manager that was being used.

System action

The function is terminated, and all queue updates are backed out.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form). Resubmit the job if required.

CSQU087I
MAXUMSGS reached. A syncpoint has been forced

Explanation

Because MAXUMSGS was reached, a syncpoint was taken which commits the queue changes made so far.

System action

The function continues, but no further functions will be processed.

System programmer response

None, unless the function fails for some reason after this message. In that case, note that some queue changes will have been committed, and you should make appropriate adjustments before rerunning the job.

CSQU090E
OPEN failed for *ddname* data set. VSAM return code=*rc* reason code=*reason*

Explanation

The utility program received a VSAM OPEN error for the page set it was attempting to process (pointed to by *ddname*).

System action

The page set is not processed.

System programmer response

See the *DFSMS/MVS Macro Instructions for Data Sets* for information about the return and reason codes from VSAM. If necessary, resubmit the job.

CSQU091E
ddname data set is non-empty. Page set not formatted

Explanation

Data set *ddname* was opened, but it is not empty.

System action

The page set is not formatted.

System programmer response

Ensure that the data sets specified are empty, and resubmit the job if necessary.

CSQU092I
function completed for *ddname* data set

Explanation

Processing of *ddname* data set for function *function* has completed.

System action

Processing continues with the next page set.

CSQU093E
PUT failed for *ddname* data set. VSAM return code=*rc* reason code=*code*

Explanation

The utility program received a VSAM PUT error for the page set it was attempting to process (pointed to by *ddname*).

System action

Processing for the page set is terminated, and the function continues with the next page set.

System programmer response

See the *DFSMS/MVS Macro Instructions for Data Sets* for information about the return and reason codes from VSAM. If necessary, resubmit the job.

CSQU094E
CLOSE failed for *ddname* data set. VSAM return code=*rc* reason code=*reason*

Explanation

The utility program received a VSAM CLOSE error for the page set it was attempting to process (pointed to by *ddname*).

System action

Processing for the page set is terminated, and the function continues with the next page set.

System programmer response

See the *DFSMS/MVS Macro Instructions for Data Sets* for information about the return and reason codes from VSAM. If necessary, resubmit the job.

CSQU095E
No page sets identified. *function* terminated

Explanation

A request to format or reset a page set was unsuccessful because there were no page set data sets with DD names in the range CSQP0000 through CSQP0099.

System action

Processing is terminated.

System programmer response

Add DD statements for the required page set data sets, and resubmit the job.

CSQU100E
ddname DD statement missing

Explanation

Data set *ddname* does not have a DD statement in the JCL.

System action

The utility is terminated.

System programmer response

Add the required statement to the JCL, and resubmit the job.

CSQU101E
DD statement missing for page set *psid*

Explanation

A page set is referenced, but there is no DD statement for it in the JCL. The DD name required is CSQP00*nn*, where *nn* is the page set number.

System action

The utility is terminated.

System programmer response

Add the required statement to the JCL, and resubmit the job.

CSQU102E
No functions requested

Explanation

There are no function statements in the SYSIN data set.

System action

The utility is terminated.

CSQU103E
Either keyword *keyword1* or *keyword2* must be specified

Explanation

The statement syntax is incorrect because it requires that one of the keywords *keyword1* or *keyword2* be specified, but not both.

System action

The utility is terminated.

System programmer response

See MQSC commands for information about the correct syntax required for the statement, then resubmit the job.

CSQU104E
Invalid value *value* for keyword *keyword*

Explanation

The statement syntax is incorrect because the value given for keyword *keyword* is not valid.

System action

The utility is terminated.

System programmer response

See MQSC commands for information about the correct syntax required for the statement, and resubmit the job.

CSQU105E
Incompatible keywords or values for function *function*

Explanation

The statement syntax is incorrect because a keyword or its value that is specified conflicts with another keyword or its value.

System action

The utility is terminated.

System programmer response

See MQSC commands for information about the correct syntax required for the statement, and resubmit the job.

CSQU106E
Invalid function *function*

Explanation

The statement syntax is incorrect because the function *function* is not recognized.

System action

The utility is terminated.

System programmer response

See MQSC commands for a list of valid functions, and resubmit the job.

CSQU107E
Invalid *function* statement syntax

Explanation

The syntax of the *function* statement is incorrect:

- there are too many keywords or values
- required keywords are missing
- it cannot be parsed.

System action

The utility is terminated.

System programmer response

See MQSC commands for information about the correct syntax required for the statement, and resubmit the job.

CSQU108E
Value missing for keyword *keyword*

Explanation

Keyword *keyword* should be followed by a value, but the value is missing.

System action

The utility is terminated.

System programmer response

See MQSC commands for information about the correct syntax required for the statement, and resubmit the job.

CSQU109E
Value not allowed for keyword *keyword*

Explanation

Keyword *keyword* should not be followed by a value, but a value is specified.

System action

The utility is terminated.

System programmer response

See Configuring z/OS for information about the correct syntax required for the statement, and resubmit the job.

CSQU110E
Required keyword missing for keyword *keyword*

Explanation

The statement syntax is incorrect because keyword *keyword* can be specified only if some other keyword is also specified, but that other keyword is missing.

System action

The utility is terminated.

System programmer response

See MQSC commands for information about the correct syntax required for the statement, then resubmit the job.

CSQU111E
Invalid keyword *keyword* for function *function*

Explanation

The statement syntax is incorrect because the keyword *keyword* is not valid for function *function*.

System action

The utility is terminated.

System programmer response

See MQSC commands for information about the correct syntax required for the statement, and resubmit the job.

CSQU112E
Incomplete statement

Explanation

End of data on the input data set was reached before the building of a statement was complete.

System action

The utility is terminated.

System programmer response

Check that the statement is correctly formed according to the concatenation rules.

CSQU113E
Too many statement continuations

Explanation

The statement has more than 10 continuations.

System action

The utility is terminated.

System programmer response

Check that the statement is correctly formed according to the concatenation rules.

CSQU114E
Keyword *keyword* repeated

Explanation

The statement syntax is incorrect because a keyword is repeated.

System action

The utility program is terminated.

System programmer response

Check the syntax in the input data set. See CSQUTIL for further information about the utility program.

CSQU115E
Unable to find queues for page set *psid* - command responses not received

Explanation

In the COPY or EMPTY function, the queue manager could not determine which queues are in page set *psid* because the response to a command was not received in time.

System action

The function is terminated.

System programmer response

Check the definitions of the system-command input queue and the system-command reply queue; ensure that they are enabled for MQGET and MQPUT. If the definitions are correct, stop and restart the command server.

CSQU116I
No storage classes found for page set *psid*

Explanation

The page set specified has no storage classes associated with it.

System action

The function is terminated.

System programmer response

Define a storage class for the page set, and rerun the job if required.

CSQU117I
No queues found for page set *psid*

Explanation

The page set specified has no queues associated with it that are eligible for the requested function. For the COPY and EMPTY functions, there are no local queues; for the SCOPY function, there are no local queues with messages.

System action

The function is terminated.

System programmer response

If required, correct the page set specified, and rerun the job.

CSQU120I
Connecting to *conn-id*

Explanation

The utility program is connecting to the named queue manager or queue-sharing group.

CSQU121I
Connected to queue manager *qmgr-name*

Explanation

The utility program connected successfully to queue manager *qmgr-name*.

CSQU122I
Executing *function-name*

Explanation

The utility program is executing function *function-name*.

CSQU123I
Processing *ddname* data set, mode FORCE

Explanation

The current function of the utility program is handling data set *ddname* using the FORCE option.

CSQU124I
Processing *ddname* data set

Explanation

The current function of the utility program is handling data set *ddname*.

CSQU125I
n page sets attempted

Explanation

This indicates how many page sets the current function attempted to process.

CSQU126I
n page sets processed successfully

Explanation

This indicates how many page sets were processed successfully by the current function.

CSQU127I
Executing *function* using input from *ddname* data set

Explanation

The utility program is executing function *function* using input from *ddname*.

CSQU128I
Executing *function* outputting to *ddname* data set

Explanation

The utility program is executing function *function*, and is writing the output to *ddname*.

CSQU129I
Copying page set *psid*

Explanation

The utility program is copying page set *psid*.

CSQU130I
Copying queue *q-name*

Explanation

The utility program is copying queue *q-name*.

CSQU131I
n messages copied successfully

Explanation

This indicates how many messages were copied successfully when copying a queue.

CSQU133I
n queues attempted

Explanation

This indicates how many queues the program attempted to copy while copying a page set.

CSQU134I
n queues copied successfully

Explanation

This indicates how many queues were copied successfully while copying a page set.

CSQU135I

Loading queue *sourceq* to *targetq*

Severity
0

Explanation

When executing the LOAD or SLOAD function, identifies the name of the target queue being loaded, and the name of the queue on the input data set from which messages are being copied.

CSQU136I

msg-count messages (*msg-from-msg-to*) have been loaded (total size *text-length*)

Severity
0

Explanation

When executing the LOAD or SLOAD function, this error code indicates that a number of messages have been successfully loaded on to the target queue from the input data set.

- *msg-count* is the number of messages loaded

- *msg-from-msg-to* is the message number range in the messages for the queue on the input data set.
- *text-length* is the total length of the message texts loaded (in MB or KB)

CSQU137I
Skipping queue *q-name*

Explanation

This indicates that queue *q-name* is being bypassed, because of the SKIPQS or FROMQ option used with the LOAD function.

CSQU138I
n queues loaded successfully

Explanation

This indicates how many queues were loaded successfully.

CSQU139I
Emptying page set *psid*

Explanation

The utility program is emptying page set *psid*.

CSQU140I
Emptying queue *q-name*

Explanation

The utility program is emptying queue *q-name*.

CSQU141I
n messages deleted successfully

Explanation

This indicates how many messages were deleted while emptying a queue.

CSQU142I
n queues emptied successfully

Explanation

This indicates how many queues were emptied.

CSQU143I
n function statements attempted

Explanation

This indicates the number of *function* statements attempted by the utility program.

CSQU144I
n function statements executed successfully

Explanation

This indicates the number of *function* statements executed successfully by the utility program.

CSQU145I
function statement failed

Explanation

The utility program experienced an error while executing function *function*.

System action

The utility program terminates.

System programmer response

Check the other messages issued to determine where the error occurred, and what caused it.

CSQU146I

msg-count messages (*msg-from-msg-to*) skipped (total size *text-length*). Reason=*reason-code*

Severity

0

Explanation

When executing the LOAD or SLOAD function, indicates that a number of messages have been ignored from the input data set.

- *msg-count* is the number of messages ignored
- *msg-from-msg-to* is the message number range in the messages for the queue on the input data set.
- *text-length* is the total length of the message texts ignored (in MB or KB)

The reason code indicates why the messages were ignored:

4 messages skipped due to *skipmsgs* parameter in LOAD or SLOAD command

8 messages skipped due to an MQPUT error

12 messages skipped due to an error on MQOPEN

16 messages skipped due to an MQPUT error immediately following a sync point

20 messages skipped due to an error on MQCLOSE

24 messages skipped due to an error when taking a sync point

28 messages skipped due to *MSGCOUNT* limit (in the LOAD or SLOAD command) being reached

CSQU147I

csect-name Utility terminated, return code=*ret-code*

Explanation

The utility has terminated because a severe error or forced syncpoint occurred meaning that no further functions should be run. *ret-code* is the return code from the utility.

System action

The utility ends.

System programmer response

See IBM MQ for z/OS codes for information about the return code from the utility.

CSQU148I

csect-name Utility completed, return code=*ret-code*

Explanation

The utility completed, all required functions having been attempted. *ret-code* is the return code from the utility.

System action

The utility ends.

System programmer response

Check any functions that failed.

CSQU150I
function completed for data set *ddname1* to data set *ddname2*

Explanation

Processing for data set *ddname1* has completed, with output to *ddname2*.

System action

Processing continues with the next page set.

CSQU151I
No matching CSQSnnnn and CSQTnnnn DD statements. *function* terminated

Explanation

A COPYPAGE or RESETPAGE function was unsuccessful because there were no matching pairs of page set data sets with names CSQS0000 through CSQS0099 and CSQT0000 through CSQT0099.

System action

The function is terminated.

System programmer response

Add DD statements for the required page set data sets, and resubmit the job.

CSQU152I
ddname1 DD statement missing. No action taken for *ddname2* data set

Explanation

Only one of the source-target pair of page set data sets (CSQSnnnn and CSQTnnnn) was specified.

System action

The function continues.

System programmer response

Add DD statements for the required page set data sets, and resubmit the job.

CSQU154E
Target data set *ddname* is smaller than source data set. Function terminated

Explanation

A COPYPAGE or RESETPAGE function could not process a page set data set because the target data set *ddname* was too small.

System action

Processing continues with the next page set.

CSQU155I

Processing queue *queue-name*

Severity
0

Explanation

When executing the ANALYZE function, indicates the start of processing queue *queue-name* from the input data set.

CSQU156E
GET failed for *ddname* data set. VSAM return code=*rc* reason code=*code*

Explanation

The utility program received a VSAM GET error for the page set it was attempting to process (pointed to by *ddname*).

System action

Processing for the page set is terminated, and the function continues with the next page set.

System programmer response

See the *DFSMS/MVS Macro Instructions for Data Sets* manual for information about the return and reason codes from VSAM. If necessary, resubmit the job.

CSQU157I
Processing data set *ddname1* to *ddname2*

Explanation

The current function is handling data set *ddname1*, with output to *ddname2*.

CSQU158E
Target data set *ddname2* is not newly formatted

Explanation

The COPYPAGE and RESETPAGE functions can only be used with a newly formatted target page set.

System action

Processing continues with the next page set.

System programmer response

Specify a valid target page set, and resubmit the job.

CSQU159E
Source data set *ddname1* is not a page set

Explanation

CSQUTIL COPYPAGE or RESETPAGE functions were unable to recognize the data set as an IBM MQ Page set. This could be due to an invalid data set, or a back level version of IBM MQ libraries being used.

System action

Processing continues with the next page set.

System programmer response

Check the data set is a valid IBM MQ page set.

Check the IBM MQ libraries being used are the same as the libraries used by the queue manager.

CSQU160E
Data set *ddname* is not suitable for use with the function

Explanation

The function should only be used with page sets for a queue manager that terminated normally.

System action

Processing continues with the next page set.

System programmer response

Specify a valid page set, and resubmit the job.

CSQU161I
ddname contains *pp* pages and was formatted as page set *nn*

Explanation

This is part of the response to the PAGEINFO function for data set *ddname*.

It shows the size of the page set, and the page set number that was assumed when it was formatted. The number is derived from the DD name used when formatting, which was CSQP00*nn*.

CSQU162I
ddname is used as page set *psid* for queue manager *qmgr-name*

Explanation

This is part of the response to the PAGEINFO function for data set *ddname*.

The page set has been used by the queue manager shown. The page set number is not necessarily the same as that with which it was formatted, as shown in message CSQU161I.

CSQU163I
ddname has page set recovery RBA = *rba*

Explanation

This is part of the response to the PAGEINFO function for data set *ddname*.

CSQU164I
ddname System recovery RBA for all page sets successfully processed = *rba*

Explanation

This is part of the response to the PAGEINFO function. Note that this RBA relates only to those page sets processed; it does not relate to the whole queue manager unless all the page sets for the queue manager are included.

CSQU165I
Processing *ddname* data set, TYPE(*type*)

Explanation

This current function of the utility program is handling data set *ddname* with the options shown.

CSQU166I
Processing *ddname* data set, TYPE(*type*), mode FORCE

Explanation

This current function of the utility program is handling data set *ddname* with the options shown.

CSQU167I
ddname has never been initialized by a queue manager

Explanation

This is part of the response to the PAGEINFO function for data set *ddname*.

CSQU168E
Requested page sets are for more than one queue manager

Explanation

The page sets for which information was requested are associated with more than one queue manager. No system recovery RBA can therefore be determined.

System action

Processing continues.

System programmer response

Specify a set of page sets for a single queue manager, and resubmit the job.

CSQU169E

MQPUT of message *msg-no* failed. MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Severity
8

Explanation

When executing the LOAD or SLOAD function, an MQPUT failed for message number *msg-no* in the queue currently being processed on the input data. The *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form) indicate the reason for failure.

System action

Processing for the command is terminated.

System programmer response

Using the MQ completion code and reason code in the message, determine the cause of error and correct the problem. Then rerun the LOAD or SLOAD, starting with the queue being processed at the time of the error. If any messages had been successfully loaded from the input queue before the failure, use the *SKIPMSGS* parameter on the LOAD or SLOAD command to bypass those messages.

CSQU170I

msg-count messages (*msg-from-msg-to*) found (total size *text-length*)

Severity
0

Explanation

When executing the ANALYZE function, this message is displayed for the queue being processed from the input data set. The number of messages and the total length of the message text are shown.

CSQU171E

Queue *queue-name* was not found in the input data set

Severity
8

Explanation

The LOAD or SLOAD function being executed specified a source queue name of *queue-name* which was not found on the input data set.

System action

Processing for the command is terminated.

System programmer response

Specify the correct input file, correct queue name in the command, and try again.

CSQU172I
Processing *function-name* for data set *ddname*, *current-page* of *total-pages* pages processed, *percentage*% complete

Explanation

If a CSQUTIL function to process a page set is long-running, this message is issued periodically to indicate how many pages have been processed so far.

CSQU179E

The transmission queue cannot be switched because the channel initiator is not active

Severity

8

Explanation

The utility program is unable to initiate a switch of transmission queue for the channel identified in the preceding CSQU183I message because the channel initiator is not active.

System action

Processing continues, however, the transmission queue for the affected channel is not switched.

System programmer response

Start the channel initiator, then either restart the channel or rerun the command to initiate the switch of transmission queue.

CSQU180E

csect-name Unable to load module *module-name*, reason=*ssssrrrr*

Explanation

The utility program was unable to load the requested channel initiator parameter module. *ssss* is the completion code and *rrrr* is the reason code (both in hexadecimal) from the z/OS LOAD service.

System action

The function is terminated.

System programmer response

Check the member name specified on the XPARM fumction, and ensure that the module is in the library specified by the DDNAME keyword.

CSQU181E

csect-name module-name is not a valid channel initiator parameter module

Severity

8

Explanation

The module specified for channel initiator parameters is not in the correct format.

System action

The function is terminated.

System programmer response

Check the member name specified on the XPARM function.

CSQU182E

An error occurred obtaining the list of channels, reason *mqrc (mqrc-text)*

Severity

8

Explanation

The utility program was unable to identify the list of channels to process.

System action

Processing for the command is terminated.

System programmer response

Use the reason code to identify and resolve the error, then rerun the command if required.

See API completion and reason codes for information about *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQU183I

Channel *channel-name*

Severity
0

Explanation

The utility program is processing the requested function for the channel named *channel-name*.

System action

Processing continues.

System programmer response

Check subsequent messages to determine whether the requested function was processed successfully.

CSQU184I

n channels processed

Severity
0

Explanation

This message identifies the number of channels that were processed by the requested function.

System action

Processing continues.

System programmer response

None required.

CSQU185I

A switch of transmission queue is not required for this channel

Severity
0

Explanation

This message identifies that a switch of transmission queue is not required for the channel identified in the preceding CSQU183I message. This indicates the channel uses the currently configured transmission queue.

System action

Processing continues.

System programmer response

None required.

CSQU186I

A switch of transmission queue is pending for this channel

Severity

0

Explanation

This message identifies that a switch of transmission queue is pending for the channel identified in the preceding CSQU183I message. This indicates the configured transmission queue for the channel has been changed, either by updating the **DEFCLXQ** queue manager attribute, or by altering the value of the **CLCHNAME** attribute of a transmission queue, since the channel last started.

The switch of transmission queue will occur the next time the channel starts or if the switch is initiated using the CSQUTIL function **SWITCH CHANNEL**.

A switch operation is also reported as pending if the operation was previously initiated, but the queue manager was stopped while messages were being moved from the old transmission queue to the new transmission queue. To resume the switch operation either start the channel or use CSQUTIL to initiate the switch.

System action

Processing continues.

System programmer response

None required.

CSQU187I

A switch of transmission queue is in progress for this channel

Severity

0

Explanation

This message identifies that a switch of transmission queue isin progress for the channel identified in the preceding CSQU183I message.

System action

Processing continues.

System programmer response

Use console messages issued by the queue manager to determine the status of the switch operation, if required.

CSQU188I

From transmission queue *xmit-qname*

Severity

0

Explanation

This message is issued with other messages, such as CSQU186I, CSQU187I, and CSQU195I. It identifies the name of the transmission queue a channel is switching, or will switch, from.

System action

Processing continues.

System programmer response

None required.

CSQU189I

To transmission queue *xmit-qname*

Severity
0

Explanation

This message is issued with other messages, such as CSQU186I, CSQU187I, and CSQU195I. It identifies the name of the transmission queue a channel is switching, or will switch, to.

System action

Processing continues.

System programmer response

None required.

CSQU190I

There are *num-msgs* messages queued for this channel on *xmitq-name*

Severity
0

Explanation

This message is issued with CSQU186I and identifies there are currently *num-msgs* messages queued for the channel on the transmission queue *xmitq-name*, that need to be moved when the transmission queue is switched.

System action

Processing continues.

System programmer response

None required.

CSQU191E

Unable to access transmission queue *xmitq-name*, reason *mqrc (mqrc-text)*

Severity
8

Explanation

This message is issued with CSQU186I if the transmission queue, *xmitq-name*, that is currently used by the channel, cannot be accessed for the reason *mqrc* (*mqrc-text* provides the MQRC in textual form). This transmission queue must be accessible to move messages for the channel to the new transmission queue.

System action

Processing for the command is terminated.

System programmer response

Use the reason code to identify and resolve the error, then rerun the command if required.

See API completion and reason codes for information about *mqrc* (*mqrc-text* provides the MQRC in textual form).

Alternatively, use the CSQUTIL function **SWITCH CHANNEL** with the **MOVEMSGS(NO)** option to switch transmission queue without moving messages. If this option is selected it is the responsibility of the system programmer to resolve any messages for the channel on the transmission queue, *xmitq-name*, after the switch has completed.

CSQU192E

The status of this channel is unavailable, reason *mqrc (mqrc-text)*

Severity

8

Explanation

The utility program was unable to identify the current status of the channel identified in the preceding CSQU183I message to determine if a switch of transmission queue is pending or in progress.

System action

Processing continues, but the transmission queue for the affected channel is not switched if this was requested.

System programmer response

Use the reason code to identify and resolve the error, then rerun the command if required.

See API completion and reason codes for information about *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQU193E

The transmission queue cannot be switched because the channel is active

Severity

8

Explanation

The utility program was unable to initiate a switch of transmission queue for the channel identified in the preceding CSQU183I message because the channel status was neither **STOPPED** nor **INACTIVE**.

If the cluster sender channel is showing STOPPED status, but message CSQU193E is still reporting, the channel cannot stop immediately. Message reallocation is taking place while a request to STOP CHANNEL *channel-name* is made.

The channel continues to reallocate messages, and stops once this process is complete. This process can take some length of time if there are a large number of messages on the queue assigned to this channel.

You should wait for sufficient time to ensure that message reallocation completes, then switch the transmission queue.

System action

Processing continues, but the transmission queue for the affected channel is not switched.

System programmer response

Stop the channel, then either restart the channel or rerun the command to initiate the switch of transmission queue.

CSQU194E

The switch of transmission queue failed, reason *mqrc (mqrc-text)*

Severity

8

Explanation

The utility program was unable to switch the transmission queue for the channel identified in the preceding CSQU183I message.

System action

Processing continues, but the transmission queue for the affected channel is not switched.

System programmer response

Use the reason code to identify and resolve the error, then rerun the command if required.

See API completion and reason codes for information about *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQU195I

Switching the transmission queue for this channel

Severity

0

Explanation

A switch of transmission queue has been initiated for the channel identified in the preceding CSQU183I message.

System action

Processing continues.

System programmer response

Use subsequent messages to determine if the switch of transmission queue completes successfully

CSQU196I

Moving messages for this channel - *num-msgs* messages moved

Severity

0

Explanation

A switch of transmission queue requires that messages for a channel are moved from the old transmission queue to the new transmission queue. This message is periodically issued during a switch of transmission queue to report the progress of this operation for the channel identified in the preceding CSQU183I message.

System action

Processing continues.

System programmer response

If this message is repeatedly issued it might indicate the old transmission queue cannot be drained of messages for the channel, which means the switching process can not complete. Applications continue to put messages to the old transmission queue during the switching process to preserve ordering.

If the switching process cannot complete, this might indicate that messages are being put to the old transmission queue faster than they can be moved by the switching process, or uncommitted messages remain on the old transmission queue for the channel.

Use console messages issued by the queue manager, such as CSQM554I, and commands such as **DISPLAY QSTATUS**, to determine why the switch operation is unable to complete.

CSQU197I

Moving of messages complete - *num-msgs* messages moved

Severity

0

Explanation

A switch of transmission queue requires that messages for a channel are moved from the old transmission queue to the new transmission queue. This message indicates the process of moving messages for the channel identified in the preceding CSQU183I message has completed. The number of messages that were moved to the new transmission queue is identified by *num-msgs*.

System action

Processing continues.

System programmer response

None required.

CSQU198I

The transmission queue has been switched successfully

Severity
0

Explanation

A switch of transmission queue for the channel identified in the preceding CSQU183I message has completed successfully.

System action

Processing continues.

System programmer response

None required.

CSQU199E

Function requires command level *required-cmdlevel*, the queue manager's command level is *qmgr-cmdlevel*

Severity
8

Explanation

The utility program was unable to perform the requested function, identified by *function*, because this is not supported by the queue manager to which it is connected. To perform the requested function the queue manager's command level must be *required-cmdlevel* or greater, but the queue manager's command level is *qmgr-cmdlevel*.

System action

Processing for the command is terminated.

System programmer response

Ensure the utility program connected to the required queue manager. If this was correct, the queue manager must be upgraded before the requested function can be used.

CSQU200I
csect-name Dead-letter Queue Handler Utility - *date time*

Explanation

This is part of the header to the report issued by the utility program.

CSQU201I
Processing queue *q-name*

Explanation

The dead-letter queue handler has parsed the rules table without detecting any errors and is about to start processing the queue identified in the message.

CSQU202I
Dead-letter queue handler ending. Successful actions: *n1* retries, *n2* forwards, *n3* discards

Explanation

The dead-letter queue handler is ending because there are no more messages on the dead-letter queue, or because the queue manager is shutting down, or because the dead-letter queue handler detected an error. The message indicates how many dead-letter queue messages were successfully handled.

System action

The utility terminates.

System programmer response

If the utility ended because of an error, investigate the problem reported in the preceding messages.

CSQU203I
n messages remain on the dead-letter queue

Explanation

The message indicates how many messages are left on the dead-letter queue when the dead-letter queue handler ends.

CSQU210I
Message does not have a valid MQDLH

Explanation

The dead-letter queue handler retrieved a message from the dead-letter queue, but the message was not prefixed by a valid dead-letter queue header (MQDLH). This typically occurs because an application is writing directly to the dead-letter queue but is not prefixing messages with a valid MQDLH.

System action

The message is left on the dead-letter queue and the dead-letter queue handler continues to process the dead-letter queue.

This message is issued only once the first time such a message is encountered.

System programmer response

Remove all the invalid messages from the dead-letter queue. Do not write messages to the dead-letter queue unless they are prefixed by a valid MQDLH.

CSQU211I
Unable to put message, line *n* MQRC=*mqrc (mqrc-text)*

Explanation

The dead-letter queue handler tried to redirect a message to another queue as requested, but the MQPUT call was unsuccessful.

System action

The retry count for the message is incremented; processing continues.

System programmer response

Refer to API completion and reason codes for information about *mqrc* (*mqrc-text*provides the MQRC in textual form). The line number *n* of the rules table used to determine the action for the message will help identify the queue to which the message was being put.

CSQU212I
Unable to inquire dead-letter queue, MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

An MQINQ call for the dead-letter queue was unsuccessful.

System action

Processing continues.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQU213I
Unable to convert message, MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

An MQGET call encountered a data conversion problem.

System action

The message is rolled back and remains on the queue. Processing of the remaining messages on the queue continues. Use an alternative means to remove this message from the dead-letter queue.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQU220E
Unable to connect to queue manager qmgr-name, MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

The dead-letter queue handler could not connect to the requested queue manager.

System action

The utility is terminated.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQU221E
Unable to open queue manager, MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

An MQOPEN call for the queue manager was unsuccessful.

System action

The utility is terminated.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQU222E
Unable to inquire queue manager, MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

An MQINQ call for the queue manager was unsuccessful.

System action

The utility is terminated.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQU223E
Unable to close queue manager, MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

An MQCLOSE call for the queue manager was unsuccessful.

System action

The utility is terminated.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQU224E
Unable to browse dead-letter queue *q-name*, MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

An MQOPEN call for browsing the dead-letter queue was unsuccessful. This is typically because of one of the following reasons:

- Another process has opened the queue for exclusive access.
- An invalid queue name was specified.
- The alias name for one of the following modules has been lost:
  - CSQBSRV
  - CSQAPEPL
  - CSQBCRMH
  - CSQBAPPL

System action

The utility is terminated.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQU225E
Unable to close dead-letter queue, MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

An MQCLOSE call for the dead-letter queue was unsuccessful.

System action

The utility is terminated.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQU226E
Line *n*: *keyword*(*value*) invalid or outside permitted range

Explanation

The value supplied for the specified keyword in line *n* of the rules table was outside the valid range of values or otherwise invalid.

System action

The utility is terminated.

System programmer response

Correct the rules table and restart the dead-letter queue handler.

CSQU227E
Unable to get message from dead-letter queue, MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

An MQGET call for the dead-letter queue was unsuccessful.

System action

The utility is terminated.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQU228E
Unable to commit or backout dead-letter queue action, MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

An MQCMIT or MQBACK call for the dead-letter queue was unsuccessful.

System action

The utility is terminated.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQU229E
Rules table is invalid or missing

Explanation

The rules table contained no valid message templates or was not supplied at all.

System action

The utility is terminated.

System programmer response

Correct the rules table as indicated in the preceding messages and restart the dead-letter queue handler.

CSQU230E
Unable to obtain storage

Explanation

The dead-letter queue handler was unable to obtain storage.

This problem would typically arise as a result of some wider problem. For example, if there is a persistent problem that is causing messages to be written to the dead-letter queue and the same problem (for example, queue full) is preventing the dead-letter queue handler from taking the requested action with the message, ever-increasing amounts of storage would be required.

System action

The utility is terminated.

System programmer response

Increase the storage available to the utility. Investigate whether some wider problem exists, and if the dead-letter queue contains a large number of messages.

CSQU231E
Line *n*: parameter *keyword* exceeds maximum length

Explanation

The value for the specified parameter in line *n* of the rules table is too long.

System action

The utility is terminated.

System programmer response

Correct the rules table and restart the dead-letter queue handler.

CSQU232E
Line *n*: parameter *keyword* is duplicated

Explanation

Two or more parameters of the same type were supplied in line *n* of the rules table.

System action

The utility is terminated.

System programmer response

Correct the rules table and restart the dead-letter queue handler.

CSQU233E
Line *n*: syntax error

Explanation

There is a syntax error in line *n* of the rules table.

System action

The utility is terminated.

System programmer response

Correct the rules table and restart the dead-letter queue handler.

CSQU234E
Unable to release storage

Explanation

The dead-letter queue handler was unable to release storage.

System action

The utility is terminated.

System programmer response

Investigate the problem reported in the preceding messages.

CSQU235E
Line *n*: *keyword* value invalid or outside permitted range

Explanation

The value supplied for the specified parameter in line *n* of the rules table was outside the valid range of values or otherwise invalid.

System action

The utility is terminated.

System programmer response

Correct the rules table and restart the dead-letter queue handler.

CSQU236E
n error(s) in rules table

Explanation

Errors were detected in the rules table.

System action

The utility is terminated.

System programmer response

Correct the rules table as indicated in the preceding messages and restart the dead-letter queue handler.

CSQU237E
Line *n*: invalid keyword combination

Explanation

There is an invalid combination of parameters in line *n* of the rules table. For example: no ACTION specified, ACTION(FWD) specified without FWDQ, HEADER specified without ACTION(FWD).

System action

The utility is terminated.

System programmer response

Correct the rules table and restart the dead-letter queue handler.

CSQU249E
Unable to disconnect from queue manager, MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Explanation

An MQDISC call for the queue manager was unsuccessful.

System action

The utility is terminated.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

V 9.0.1

CSQU300I

Incorrect parameters provided. Usage information follows:

CSQUDSPM (-m QMgrName) (-o status | -s) (-o all) (-su) (-a)

-m QMgrName: Display information for this queue manager only

-o status : Display operational status of the queue manager, or queue managers

-s : Display operational status of the queue manager, or queue managers

-o all : Display all details of the queue manager, or queue managers

-su : Do not show information for queue managers that have an unknown version

-a : Display information for running queue managers only

Explanation

Incorrect parameters were passed to CSUDSPM.

System action

The utility stops processing.

System programmer response

Correct the parameters and resubmit the utility.

V 9.0.1

CSQU307I

No queue manager with the specified name exists

Explanation

The CSQUDSPM utility was run, specifying a queue manager name that could not be located.

System action

The utility stops processing.

System programmer response

Either correct the queue manager name, or run the utility without specifying a particular queue manager.

CSQU500I

csect-name Queue-sharing Group Utility - *date time*

Explanation

This is part of the header to the report issued by the utility program.

CSQU501I

function function requested

Explanation

This identifies the utility function requested.

CSQU502I

Queue manager=*qmgr-name*

Explanation

This identifies the queue manager name for which the function is requested.

CSQU503I
QSG=*qsg-name*, Db2 DSG=*dsg-name*, Db2 ssid=*db2-name*

Explanation

This identifies the queue-sharing group, Db2 data-sharing group, and Db2 subsystem names for which the function is requested.

CSQU504E
Unable to LOAD *module-name*, reason=*ssssrrrr*

Explanation

The utility was unable to load a required module. *ssss* is the completion code and *rrrr* is the reason code (both in hexadecimal) from the z/OS LOAD service.

System action

The utility terminates.

System programmer response

Check the console for messages indicating why the module was not loaded. See the *MVS Programming: Assembler Services Reference* manual for information about the codes from the LOAD request.

Ensure that the module is in the required library, and that it is referenced correctly. The utility attempts to load this module from the library data sets under the STEPLIB DD statement.

CSQU505E
No EXEC PARM parameters

Explanation

No parameters for the utility were specified in EXEC PARM field.

System action

The utility program is terminated.

System programmer response

Specify the required parameters and rerun the job.

CSQU506E
Invalid EXEC PARM function parameter

Explanation

The function requested for the utility, as the first parameter in EXEC PARM field, was invalid.

System action

The utility program is terminated.

System programmer response

Correct the parameter and rerun the job.

CSQU507E
Wrong number of EXEC PARM parameters for function

Explanation

The number of parameters for the utility specified in EXEC PARM field was incorrect for the function requested.

System action

The utility program is terminated.

System programmer response

Correct the parameters and rerun the job.

CSQU508E
Invalid EXEC PARM parameter *n*

Explanation

The *n*th parameter for the utility specified in EXEC PARM field was invalid for the function requested, or omitted but required by the function requested.

System action

The utility program is terminated.

System programmer response

Correct the parameter and rerun the job.

CSQU509E
Too many EXEC PARM parameters

Explanation

The number of parameters for the utility specified in EXEC PARM field was too many for the function requested.

System action

The utility program is terminated.

System programmer response

Correct the parameters and rerun the job.

CSQU510I
SQL error information

Explanation

An SQL error has occurred. Diagnostic information follows in message CSQU511I.

System action

See the preceding utility error message.

System programmer response

Look at the information in message CSQU511I to determine the reason for the SQL error.

CSQU511I
DSNT408I SQLCODE = `-sql-code`, explanation

Explanation

This message provides additional diagnostic information direct from Db2. It is followed by further CSQU511I messages, similar to the following example block:

```
CSQU511I   DSNT408I SQLCODE = -805, ERROR:  DBRM OR PACKAGE NAME DSNV11P1..CSQ5B9-
CSQU511I            00.1A47B13F08B31B99 NOT FOUND IN PLAN CSQ5B900. REASON 03
CSQU511I   DSNT418I SQLSTATE   = 51002 SQLSTATE RETURN CODE
CSQU511I   DSNT415I SQLERRP    = DSNXEPM SQL PROCEDURE DETECTING ERROR
CSQU511I   DSNT416I SQLERRD    = -251  0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
CSQU511I   DSNT416I SQLERRD    = X'FFFFFF05'  X'00000000'  X'00000000'
CSQU511I            X'FFFFFFFF'  X'00000000'  X'00000000' SQL DIAGNOSTIC
CSQU511I            INFORMATION
```

System action

The utility program is terminated.

System programmer response

Use the diagnostic information to determine the reason for the SQL error and correct the problem.

CSQU512E
Utility terminated, Db2 tables in use

Explanation

The queue-sharing group utility cannot run because the Db2 tables it uses are reserved by another job. The most likely reason is that another instance of the utility is running, or that a queue manager in the queue-sharing group is in the process of starting.

System action

The utility program is terminated.

System programmer response

Rerun the job later.

CSQU513E
Utility terminated, not APF authorized

Explanation

The queue-sharing group utility is not APF authorized.

System action

The utility program is terminated.

System programmer response

Ensure that the library data sets under the STEPLIB DD statement comply with the rules for APF authorization, and rerun the job.

CSQU514E
RRSAF function *call-name* failed, RC=*rc*

Explanation

The RRS function specified by *call-name* returned an unexpected reason code specified by *rc*.

System action

The utility program is terminated.

System programmer response

Consult the *Db2 messages* and *DB2 codes* sections within the DB2 for z/OS product documentation for an explanation of the RRSAF reason code.

Take corrective action if necessary and resubmit the job.

CSQU515E
Unable to access Db2 tables, RC=*rc* reason=*reason*

Explanation

The call to CSQ5ARO2 module failed with a return code specified by *rc* and reason code specified by *reason*.

System action

The utility program is terminated.

System programmer response

Resubmit the job. If the problem persists, note the error codes in the message and contact your IBM support center.

CSQU517I
XCF group *xcf-name* already defined

Explanation

Informational message indicating that the XCF group name specified by *xcf-name* already exists.

CSQU518E
XCF IXCQUERY member error, RC=*rc* reason=*reason*

Explanation

An unexpected return code specified by *rc* with reason code specified by *reason* was returned from an IXCQUERY request.

System action

The utility program is terminated.

System programmer response

See the *z/OS MVS Sysplex Services Reference* manual for an explanation of the IXCQUERY return and reason codes.

Take corrective action if necessary and resubmit the job.

CSQU520I
Summary information for XCF group *xcf-name*

Explanation

Informational message indicating that summary data for the XCF group specified by *xcf-name* follows.

CSQU521I
Group contains *n* members:

Explanation

Informational message indicating that the group specified by message CSQU517I contains *n* members.

CSQU522I
Member=*xcf-name*, state=*sss*, system=*sys-name*

Explanation

Informational message indicating that the XCF group member specified by *xcf-name* has a state of *sss* and last executed on system *sys-name*.

CSQU523I
User data=*xxx*

Explanation

Informational message containing the 32 bytes of XCF user data to accompany informational message CSQU522I.

CSQU524I
QMGR number=nn

Explanation

Informational message containing the number of the QMGR in the queue-sharing group to accompany informational message CSQU522I. The QMGR number is stored in the Db2 tables, the

XCF group member and the connections to the CF structures. It is generated when a QMGR is added to a queue-sharing group using CSQ5PQSG.

CSQU525E
Db2 *db2-name* is not a member of data-sharing group *dsg-name*

Explanation

There was an inconsistency between the Db2 ssid and data-sharing group name provided in the EXEC PARM field. Db2 ssid specified by *db2-name* is not a member of the DB2 data-sharing group specified by *dsg-name*.

System action

The utility program is terminated.

System programmer response

Ensure that the Db2 ssid specified is a member of the DB2 data-sharing group specified.

CSQU526I
Connected to Db2 *db2-name*

Explanation

The utility program connected successfully to Db2 subsystem *db2-name*.

CSQU527E
No eligible Db2 currently active

Explanation

If a Db2 ssid was specified in the EXEC PARM field this indicates that the Db2 subsystem is not currently active on the z/OS system on which the utility job executed.

If a Db2 data-sharing group name was specified in the EXEC PARM field then no eligible Db2 subsystem was active on the z/OS system on which the utility job executed.

System action

The utility program is terminated.

System programmer response

If a Db2 ssid was specified in the EXEC PARM field then ensure that it is active on the z/OS system on which the utility job will execute.

If a Db2 data-sharing group name was specified in the EXEC PARM field then ensure that at least one eligible Db2 subsystem is active on the z/OS system on which the utility job will execute.

CSQU528I
Disconnected from Db2 *db2-name*

Explanation

The utility program disconnected successfully from Db2 subsystem *db2-name*.

CSQU529E
QSG *qsg-name* entry cannot be removed, *n* members are still defined

Explanation

A request to remove the queue-sharing group name in *qsg-name* failed because *n* members are still defined to it.

System action

The utility program is terminated.

System programmer response

All members of the queue-sharing group must be removed from it before the queue-sharing group itself can be deleted. Use the preceding CSQU522I message to identify which queue-sharing group members are still defined to the queue-sharing group.

Note: Members in a state of ACTIVE or FAILED cannot be removed from a queue-sharing group.

CSQU530E
QMGR *qmgr-name* entry cannot be removed from queue-sharing group *qsg-name*, status is *sss*

Explanation

The queue manager named by *qmgr-name* cannot be removed from the queue-sharing group named by *qsg-name* because it is in an incorrect XCF member state as specified by *sss*.

System action

The utility program is terminated.

System programmer response

To remove a queue manager from the queue-sharing group it must have XCF member state CREATED or QUIESCED.

If the XCF member state is ACTIVE then stop the queue manager with a STOP QMGR command and resubmit the job.

If the XCF member state is FAILED then start the queue manager and stop it normally using the STOP QMGR command and resubmit the job.

CSQU531E
QSG *qsg-name* entry cannot be removed, not found in DB2 table *table-name*

Explanation

An attempt to remove the queue-sharing group *qsg-name* failed because no entry for it was found in the Db2 table *table-name*.

System action

The utility program is terminated.

System programmer response

Ensure that the queue-sharing group *qsg-name* was originally defined in the table *table-name*.

Check that the utility job connected to the correct Db2 data-sharing group. If necessary resubmit the job.

CSQU532E
QSG *qsg-name* entry cannot be deleted, DB2 entries still exist for it

Explanation

An attempt to remove the queue-sharing group *qsg-name* was returned a DB2 constraint failure because queue manager entries still exist in the CSQ.ADMIN_B_QMGR table.

System action

The utility program is terminated.

System programmer response

Examine the CSQ.ADMIN_B_QMGR table to determine which queue managers are still defined to the queue-sharing group *qsg-name*.

Use the REMOVE QMGR function of the CSQ5PQSG utility to remove the entries and then resubmit the job.

CSQU533E
SQL error. Db2 table=*table-name*, code=*sqlcode*, state=*sss*, data=*sqlerrcd*

Explanation

An unexpected SQL error was returned from Db2. An operation on the table named by *table-name* was returned an SQLCODE specified by *sqlcode* with STATE specified by *sss* and SQLERRCD values specified by *sqlerrcd*.

System action

The utility program is terminated.

System programmer response

See the *Db2 for z/OS Messages and Codes* manual for an explanation of the SQL codes.

Resubmit the job if required.

CSQU534E
SQL services error, Db2 table=*table-name* RC=rc

Explanation

An SQL error occurred during an operation on the table specified by *table-name*, as reported in the preceding CSQU533E message. A return code of *rc* was returned from the internal service routine.

System action

The utility program is terminated.

System programmer response

See message CSQU533E.

CSQU535I
QSG *qsg-name* entry successfully removed from DB2 table *table-name*

Explanation

Informational message indicating that the queue-sharing group named by *qsg-name* was successfully removed.

CSQU536E
Unable to add queue-sharing group *qsg-name* entry, entry already exists in DB2 table *table-name*

Explanation

An attempt to add the queue-sharing group *qsg-name* failed because an entry already exists in the Db2 table *table-name*.

System action

The utility program is terminated.

CSQU537I
csect-name queue-sharing group *qsg-name* entry successfully added to DB2 table *table-name*

Explanation

The request to add the queue-sharing group *qsg-name* to the DB2 table *table-name* completed successfully.

CSQU538E
Member record found for QMGR *qmgr-name* XCF group *xcf-name*

Explanation

Informational message indicating that a member record for the queue manager named in *qmgr-name* already exists in the XCF group named by *xcf-name*.

CSQU539E
No QMGR *qmgr-name* entry found in queue-sharing group *qsg-name*

Explanation

An attempt to remove the queue manager named by *qmgr-name* from the queue-sharing group named by *qsg-name* failed because no entry was found in the Db2 tables.

System action

The utility program is terminated.

CSQU540E
Unable to remove QMGR *qmgr-name* - not terminated normally, or needed for recovery

Explanation

The queue manager named by *qmgr-name* cannot be removed from the queue-sharing group because it is currently active, or because it terminated abnormally during its last execution, or because it is needed for backup and recovery purposes.

System action

The utility program is terminated.

System programmer response

If the queue manager is active then stop the queue manager with a STOP QMGR command and resubmit the job.

If the queue manager terminated abnormally during its last execution then start the queue manager and stop it normally using the STOP QMGR command and resubmit the job.

If neither of these cases applies, or if it still cannot be removed, it must be needed for backup and recovery purposes. See Managing queue-sharing groups for information about removing such a queue manager from a queue-sharing group.

CSQU541E
QSG array manipulation error, RC=rc

Explanation

An internal error occurred during manipulation of the queue-sharing group array data.

An internal routine returned a completion code specified by *rc*.

System action

The utility program is terminated.

System programmer response

Resubmit the job. If the problem persists, note the error codes in the message and contact your IBM support center.

CSQU542E
Update unsuccessful for queue-sharing group *qsg-name*, RC=*rc*

Explanation

An attempt to update the Db2 row for the queue-sharing group named by *qsg-name* failed with return code *rc*.

rc shows the type of failure:

00F5000C
Queue-sharing group row no longer exists

00F50010
Internal error

00F50018
Referential constraint failure

00F50028
Internal error

System action

The utility program is terminated.

System programmer response

Resubmit the job. If the problem persists contact your IBM support center.

CSQU543E
Delete unsuccessful for QMGR *qmgr-name*, RC=*rc*

Explanation

The attempt to delete the queue manager *qmgr-name* failed with return code *rc*.

rc shows the type of failure: 00F5000C, queue manager row no longer exists.

System action

Processing continues.

System programmer response

This might be an indication that the request was made against the wrong DB2 data-sharing group or that a previous attempt terminated prematurely. For the former, the utility should be executed against the correct DB2 data-sharing group. For the latter, no further action need be taken.

CSQU544E
IXCDELET request for QMGR *qmgr-name* unsuccessful, RC=*rc* reason=*reason*

Explanation

During an attempt to delete queue manager *qmgr-name*, an IXCDELET request was returned an IXC return code of *rc* and reason code of *reason*.

System action

The utility program is terminated.

System programmer response

See the *z/OS MVS Sysplex Services Reference* manual for an explanation of the IXCDELET return and reason codes.

Take corrective action if necessary and resubmit the job.

CSQU545E
IXCCREAT request for QMGR *qmgr-name* unsuccessful, RC=*rc* reason=*reason*

Explanation

During an attempt to add queue manager *qmgr-name*, an IXCCREAT request was returned an IXC return code of *rc* and reason code of *reason*.

System action

The utility program is terminated.

System programmer response

See the *z/OS MVS Sysplex Services Reference* manual for an explanation of the IXCCREAT return and reason codes.

Take corrective action if necessary and resubmit the job.

CSQU546E
Unable to add QMGR *qmgr-name* entry, already exists in Db2 table *table-name*

Explanation

The attempt to add an entry for queue manager *qmgr-name* to the Db2 table *table-name* failed because a row already exists for the queue manager.

System action

The utility program is terminated.

System programmer response

Examine the Db2 table specified by *table-name* and determine whether the entry for the queue manager specified by *qmgr-name* is for the correct queue-sharing group. If it is, no further action is required.

CSQU547E
Unable to add QMGR *qmgr-name* entry, no queue-sharing group *qsg-name* entry exists in Db2 table *table-name*

Explanation

The attempt to add queue manager *qmgr-name* failed because there is no queue-sharing group entry for the queue-sharing group *qsg-name* in the Db2 table *table-name*.

System action

The utility program is terminated.

System programmer response

To add a queue manager to a queue-sharing group the Db2 CSQ.ADMIN_B_QSG table must contain a queue-sharing group record for the queue-sharing group named by *qsg-name*.

Examine the Db2 tables and if necessary run the CSQ5PQSG utility ADD QSG function prior to resubmitting this job.

CSQU548E
QMGR *qmgr-name* cannot be added to queue-sharing group *qsg-name*, no unassigned QMGR number

Explanation

The attempt to add queue manager *qmgr-name* to the queue-sharing group *qsg-name* failed because all queue manager numbers are in use.

System action

The utility program is terminated.

System programmer response

A maximum of 32 queue managers can be defined to a queue-sharing group at any one time. If the queue-sharing group named by *qsg-name* already contains 32 queue managers then the only course of action is to create a new queue-sharing group or remove an existing queue manager.

CSQU549I
QMGR *qmgr-name* entry successfully added to QSG *qsg-name*

Explanation

The request to add queue manager *qmgr-name* to the queue-sharing group *qsg-name* completed successfully.

CSQU550I
QMGR *qmgr-name* entry successfully removed from QSG *qsg-name*

Explanation

The request to remove queue manager *qmgr-name* from the queue-sharing group *qsg-name* completed successfully.

CSQU551I
QSG *qsg-name* entry successfully added

Explanation

The request to add queue-sharing group *qsg-name* completed successfully.

CSQU552I
QSG *qsg-name* entry successfully removed

Explanation

The request to remove queue-sharing group *qsg-name* completed successfully.

CSQU553E
QMGR *qmgr-name* exists in Db2 table *table-name* as a member of a different queue-sharing group *qsg-name*

Explanation

An attempt to add the queue manager specified by *qmgr-name* into a queue-sharing group failed because the Db2 table specified by *table-name* indicates that the queue manager is already a member of a different queue-sharing group identified by *qsg-name*.

System action

The utility program is terminated.

System programmer response

A queue manager can be a member of only one queue-sharing group at any one time.

Either remove the queue manager from the queue-sharing group it is in and resubmit the job or take no further action.

CSQU554E
QMGR *qmgr-name* entry cannot be removed from queue-sharing group *qsg-name*, needed for structure *struc-name* backup

Explanation

The queue manager named by *qmgr-name* cannot be removed from the queue-sharing group named by *qsg-name* because it has information about backups for structure *struc-name*. (The value shown for *struc-name* is the 12-character name as used by IBM MQ not the external name used by z/OS which includes the queue-sharing group name.)

If the queue manager is needed for more than one structure, this message will be issued for each one.

System action

The utility program is terminated.

System programmer response

Using another queue manager in the queue-sharing group, take a backup of the structure. Ensure that the EXCLINT time value used in the BACKUP CFSTRUCT command is less than the time since the queue manager that you are trying to remove was last stopped. Then resubmit the job.

When removing the last queue manager in a queue-sharing group, you must use the FORCE option, rather than REMOVE. This removes the queue manager from the queue-sharing group, whilst not performing the consistency checks of the queue manager logs being required for recovery. You should only perform this action if you are going to be deleting the queue-sharing group; see Removing a queue manager from a queue-sharing group for more information on managing queue-sharing groups.

CSQU555E
QMGR *qmgr-name release* level is incompatible with queue-sharing group *qsg-name* in Db2 table *table-name*

Explanation

An attempt to add the queue manager specified by *qmgr-name* into a queue-sharing group failed because the Db2 table specified by *table-name* indicates that another queue manager in the queue-sharing group is at an incompatible release level.

System action

The utility program is terminated.

System programmer response

Only queue managers with compatible release levels can be members of the same queue-sharing group. For information about migration and compatibility between releases, see Maintaining and migrating.

CSQU556I
QSG *qsg-name* may contain unexpected characters

Explanation

The queue-sharing group *qsg-name* being added specifies a queue-sharing group name that either contains the '@' character, or is shorter than four characters and therefore has '@' characters appended to the short name to make the name four characters in length.

System action

Processing to add the queue-sharing group continues. The utility will complete with return code 4.

System programmer response

Verify that the queue-sharing group name specified by *qsg-name* is the intended name to be used for the queue-sharing group. If not, use the utility to remove the queue-sharing group, correct the queue-sharing group name, and resubmit the request to add the queue-sharing group.

The '@' character, although allowed in the *qsg-name*, is inadvisable as it is not supported as a character in an IBM MQ object name. Any definition such as queue manager alias definitions or other objects that need to refer to the *qsg-name*, will be unable to refer to the *qsg-name*. If possible, avoid using these characters.

CSQU557E
The QMGR and queue-sharing group names must be different

Explanation

The attempt to add a queue manager to a queue-sharing group failed because queue managers cannot have the same name as the queue-sharing group to which they belong.

System action

The utility program is terminated.

CSQU558E
QMGR *qmgr-name* entry cannot be removed from queue-sharing group *qsg-name*, SMDS for structure *struc-name* is not empty

Explanation

The queue manager named by *qmgr-name* cannot be removed from the queue-sharing group named by *qsg-name* because it owns a shared message data set for structure *struc-name* which is not marked as empty, so it may still contain current message data. (The value shown for *struc-name* is the 12-character name as used by IBM MQ, not the external name used by z/OS which includes the queue-sharing group name.)

System action

The utility program is terminated.

System programmer response

The queue manager cannot be removed until the owned shared message data set has been marked as empty, indicating that it has been closed normally by the owning queue manager at a time when it does not contain any message data. All shared messages with message data in the data set must have been read or marked as deleted first, and the owning queue manager must be connected to the structure in order to remove the deleted messages and free the data set space.

The current status of each shared message data set for the structure can be displayed using the command **DISPLAY CFSTATUS(struc-name) TYPE(SMDS)**.

CSQU560I
Full name of admin structure is *admin-strname*

Explanation

This shows the full external name of the administration structure as used by z/OS, which includes the queue-sharing group name.

CSQU561E
Unable to get attributes for admin structure, IXLMG RC=*rc* reason code=*reason*

Explanation

An attempt to add a queue manager to a queue-sharing group failed; it was not possible to check the attributes of the administration structure because there was an XES IXLMG service error. The full name of the administration structure is given in the following CSQ570I message.

System action

The utility program terminates. The queue manager is not added to the queue-sharing group.

System programmer response

Investigate the return and reason codes from the IXLMG service (both shown in hexadecimal), which are described in the *z/OS MVS Programming: Services Reference* manual. If you are unable to resolve the problem, contact your IBM support center.

CSQU562E
Admin structure attributes temporarily unavailable

Explanation

An attempt to add a queue manager to a queue-sharing group failed; it was not possible to check the attributes of the administration structure because they were currently unavailable. The full name of the administration structure is given in the following CSQ570I message.

System action

The utility program terminates. The queue manager is not added to the queue-sharing group.

System programmer response

Rerun the job later.

CSQU563I
Admin structure is defined in CF *cf-name*, allocated size *mm* KB, maximum entries *nn*

Explanation

This shows the current attributes of the administration structure for the queue-sharing group. It is defined in the coupling facility named *cf-name*.

CSQU564E
Queue managers cannot be added to queue-sharing group *qsg-name*, admin structure too small

Explanation

An attempt to add a queue manager to a queue-sharing group failed; the current administration structure allocation is too small for a queue-sharing group with the requested number of queue managers. The full name of the administration structure is given in the following CSQ570I message.

System action

The utility program terminates. The queue manager is not added to the queue-sharing group.

System programmer response

See Specifying offload options for shared messages for information about coupling facility structure sizes for use with queue-sharing groups.

The administration structure allocation must be increased before a new queue manager can be added to the queue-sharing group. This may involve one or more of the following steps:

- Update the administration structure definition using the IXLMIAPU utility.
- Refresh the currently active CFRM policy.
- Dynamically alter the current allocation of the administration structure using the z/OS SETXCF START,ALTER command.

Rerun the job when the administration structure allocation has been increased.

CSQU565E
Unable to get attributes for admin structure, CF in failed state

Explanation

An attempt to add a queue manager to a queue-sharing group failed; it was not possible to check the attributes of the administration structure because it is in a failed state. The full name of the administration structure is given in the following CSQ570I message.

System action

The utility program terminates. The queue manager is not added to the queue-sharing group.

System programmer response

Use the z/OS DISPLAY XCF,STRUCTURE command to display the status of all structures in the currently active CFRM policy.

If the administration structure has failed, starting a queue manager in the queue-sharing group will cause the structure to be allocated according to the current CFRM policy.

CSQU566I
Unable to get attributes for admin structure, CF not found or not allocated

Explanation

In attempting to add a queue manager to a queue-sharing group, it was not possible to check the attributes of the administration structure because it has not yet been defined to the CFRM policy, or is not currently allocated in a coupling facility. The full name of the administration structure is given in the following CSQ570I message. If the structure is not allocated, then the structure will be allocated when the first queue manager starts.

System action

Processing continues.

System programmer response

Use the z/OS command DISPLAY XCF,STRUCTURE,STRNAME=<CFSTRNAME> to display the status (including size) of all structures in the currently active CFRM policy.

Ensure a structure definition exists in the CFRM policy. It will be needed before the queue manager can be started.

CSQU567E
QMGR *qmgr-name* not added to Db2 table due to a number mismatch.

Explanation

The QMGR qmgr-name could not be added to Db2 tables due to a mismatch in the QMGR numbers as indicated by message CSQU568E issued earlier.

System action

The utility terminates.

System programmer response

Add the QMGRs in the order corresponding to their QMGR number values in the XCF group, as can be seen by message CSQU524I when running CSQ5PQSG queue-sharing group utility with the "VERIFY QSG" parameter.

If the issue is linked to a persistent failing connection to the CSQ_ADMIN structure, the problem can be resolved by clearing the CF structure using the SETXCF FORCE command.

CSQU568E
QMGR number mismatch for QMGR *qmgr-name* in queue-sharing group *qsg-name*: Db2 value=*nn*, XCF member value=*nn*, CSQ_ADMIN connection value=*nn*

Explanation

The QMGR number is stored in the Db2 tables, the XCF group member, and the connections to the CF structures. The QMGR number is created when a QMGR is added to a queue-sharing group by using the queue-sharing group utility (CSQ5PQSG).

This message indicates that there is a mismatch in the stored values for QMGR *qmgr-name* in queue-sharing group *qsg-name* which will prevent the QMGR from starting.

System action

The utility terminates after all members in the XCF group have been processed.

System programmer response

If the QMGR number value is -1, the entry does not exist. Use the CSQ5PQSG utility with "ADD QMGR" parameter to add the missing entry.

If the QMGR number value is 0, the value has not been initialised (XCF group member and CSQ_ADMIN connection values only). Start the QMGR to initialize the value.

If QMGR number value is greater than 0, collect the items listed in the Coupling Facility problem determination guide and contact your IBM support center.

CSQU569E
Unexpected CSQ_ADMIN connection found for QMGR *qmgr-name*

Explanation

For each QMGR in the queue-sharing group there should only be one connection to the CSQ_ADMIN structure. This message is issued for each additional connection found.

System action

The utility terminates after all members in the XCF group have been processed.

System programmer response

This situation should not occur. The connections can be displayed using the display XCF command for the CSQ_ADMIN structure.

Collect the items listed in the Coupling Facility problem determination guide and contact your IBM support center.

CSQU570I
QSG *qsg-name* successfully verified

Explanation

The request to verify information for queue-sharing group *qsg-name* completed successfully. All the information is consistent.

CSQU571E
QSG *qsg-name* entry cannot be verified, not found in Db2 table *table-name*

Explanation

An attempt to verify the queue-sharing group *qsg-name* failed because no entry for it was found in the Db2 table *table-name*.

System action

The utility program is terminated.

System programmer response

Ensure that the queue-sharing group *qsg-name* was originally defined in the table *table-name*. Check that the utility job connected to the correct DB2 data-sharing group.

If necessary resubmit the job.

CSQU572E
Usage map *map-name* and Db2 table *table-name* inconsistent

Explanation

While verifying a queue-sharing group, an inconsistency was found between the information in the usage map *map=name* and the Db2 table *table-name*. The following messages give more details about the inconsistency.

System action

Processing continues.

System programmer response

Check that the utility job connected to the correct Db2 data-sharing group. If necessary resubmit the job.

Contact your IBM support center for assistance.

CSQU573E
QMGR *qmgr-name* in table entry *entry-number* not set in usage map

Explanation

While verifying a queue-sharing group, an inconsistency was found between the information in a usage map and the corresponding Db2 table. The inconsistency is described in the message; preceding message CSQU572E identifies the usage map and table.

System action

Processing continues.

System programmer response

See message CSQU572E.

CSQU574E
QMGR *qmgr-name* in usage map has no entry in table

Explanation

While verifying a queue-sharing group, an inconsistency was found between the information in a usage map and the corresponding Db2 table. The inconsistency is described in the message; preceding message CSQU572E identifies the usage map and table.

System action

Processing continues.

System programmer response

See message CSQU572E.

CSQU575E
Structure *struc-name* in table entry *entry-number* not set in usage map

Explanation

While verifying a queue-sharing group, an inconsistency was found between the information in a usage map and the corresponding Db2 table. The inconsistency is described in the message; preceding message CSQU572E identifies the usage map and table. (The value shown for *struc-name* is the 12-character name as used by IBM MQ, not the external name used by z/OS which includes the queue-sharing group name.)

System action

Processing continues.

System programmer response

See message CSQU572E.

CSQU576E
Structure *struc-name* in usage map has no entry in table

Explanation

While verifying a queue-sharing group, an inconsistency was found between the information in a usage map and the corresponding Db2 table. The inconsistency is described in the message; preceding message CSQU572E identifies the usage map and table. (The value shown for *struc-name* is the 12-character name as used by IBM MQ, not the external name used by z/OS which includes the queue-sharing group name.)

System action

Processing continues.

System programmer response

See message CSQU572E.

CSQU577E
Queue *q-name* in table entry *entry-number* not set in usage map for structure *struc-name*

Explanation

While verifying a queue-sharing group, an inconsistency was found between the information in a usage map and the corresponding Db2 table. The inconsistency is described in the message; preceding message CSQU572E identifies the usage map and table. (The value shown for *struc-name* is the 12-character name as used by IBM MQ, not the external name used by z/OS which includes the queue-sharing group name.)

System action

Processing continues.

System programmer response

See message CSQU572E.

CSQU578E
Queue *q-name* in usage map for structure *struc-name* has no entry in table

Explanation

While verifying a queue-sharing group, an inconsistency was found between the information in a usage map and the corresponding Db2 table. The inconsistency is described in the message; preceding message CSQU572E identifies the usage map and table. (The value shown for *struc-name* is the 12-character name as used by IBM MQ, not the external name used by z/OS which includes the queue-sharing group name.)

System action

Processing continues.

System programmer response

See message CSQU572E.

CSQU580I
DSG *dsg-name* is ready for migration

Explanation

The request to migrate the data-sharing group *dsg-name* to use new Db2 tables successfully verified that the data-sharing group is ready to be migrated.

System programmer response

Perform the migration.

CSQU581E
DSG *dsg-name* has incompatible QMGR levels in QSG *qsg-name*

Explanation

The data-sharing group *dsg-name* cannot be migrated to use new Db2 tables because the levels of the queue managers in queue-sharing group *qsg-name*, which uses the data-sharing group, are incompatible.

System action

The utility program is terminated.

System programmer response

To perform the migration, all the queue managers in all the queue-sharing groups that use the data-sharing group must have installed a PTF and been started, to bring them to the necessary level. Examine the CSQ.ADMIN_B_QMGR Db2 table to determine the levels of the queue

managers and those which need to be upgraded. Look at the fields QMGRNAME, MVERSIONL, MVERSIONH and investigate the queue managers with the lower values in MVERSIONH.

For information about migration and compatibility between releases, see Maintaining and migrating.

CSQU582E
DSG *dsg-name* has already been migrated

Explanation

The data-sharing group *dsg-name* cannot be migrated to use new Db2 tables because it has already been migrated.

System action

The utility program is terminated.

System programmer response

As part of the migration, the CSQ.OBJ_B_CHANNEL Db2 table will have its row size increased above 4 KB. The utility found that such a row size slready exists. Examine the CSQ.OBJ_B_CHANNEL Db2 table to verify that the migration has already occurred.

For information about migration and compatibility between releases, see Maintaining and migrating.

CSQU583I
QSG *qsg-name* in DSG *dsg-name* is ready for migration

Explanation

The request to migrate the queue-sharing group *qsg-name* in data-sharing group *dsg-name* to use new Db2 tables successfully verified that the queue-sharing group is ready to be migrated.

System programmer response

Perform the migration. You should do this as a conditional step in the same job as the utility migration request, as shown in the sample jobs CSQ4570T and CSQ4571T in the SCSQPROC library.

CSQU584E
QSG *qsg-name* in DSG *dsg-name* has incompatible QMGR levels

Explanation

The queue-sharing group *qsg-name* in data-sharing group *dsg-name* cannot be migrated to use new Db2 tables because the levels of the queue managers using the data-sharing group are incompatible.

System action

The utility program is terminated.

System programmer response

To perform the migration, all the queue managers in all the queue-sharing groups that use the data-sharing group must have installed a PTF and been started, to bring them to the necessary level. Examine the CSQ.ADMIN_B_QMGR Db2 table to determine the levels of the queue managers and those which need to be upgraded.

For information about migration and compatibility between releases, see Maintaining and migrating.

CSQU585E
QSG *qsg-name* entry cannot be migrated, not found in Db2 table *table-name*

Explanation

The queue-sharing group, *qsg-name*, cannot be migrated because no entry was found for it in the Db2 table, *table-name*.

System action

The utility program is terminated.

System programmer response

Ensure that the queue-sharing group *qsg-name* was originally defined in the table *table-name*.

Check that the utility job is connected to the correct Db2 data-sharing group. If necessary resubmit the job.

CSQU586I

QMGR *qmgr-name* entry being removed from queue-sharing group *qsg-name*, needed for structure *struc-name* backup

Explanation

The queue manager named by *qmgr-name* is being force removed from the queue-sharing group named by *qsg-name* and it has information about backups for structure *struc-name*. (The value shown for *struc-name* is the 12-character name as used by IBM MQ, not the external name used by Db2 which includes the queue-sharing group name.)

If the queue manager *qmgr-name* is added back to the queue-sharing group it will cause an inconsistent state that could prevent structure *struct-name* from being recovered should it fail before a structure backup is taken.

If the queue manager is needed for more than one structure, this message is issued for each one.

System action

The queue manager is removed from the queue-sharing group, and the utility program ends with return code 4.

System programmer response

If CF structure *struc-name* is usable, take a backup of CF structure *struc-name* as soon as possible using another queue manager in the queue-sharing group. Otherwise, if the queue manager *qmgr-name* is added back to the queue-sharing group it should be restarted before recovering structure *struc-name*.

CSQU587I

QMGR *qmgr-name* entry being removed from queue-sharing group *qsg-name*, SMDS for structure *struc-name* is not empty

Explanation

The queue manager named by *qmgr-name* is being removed from the queue-sharing group named by *qsg-name* while it owns a shared message data set for structure *struc-name* that is not marked as empty, so it may still contain current message data. (The value shown for *struc-name* is the 12-character name as used by IBM MQ, not the external name used by z/OS which includes the queue-sharing group name.)

System action

The queue manager is removed from the queue-sharing group, and the utility program ends with return code 4.

Messages on the SMDS for queue manager *qmgr-name* will remain accessible as long as the SMDS is retained.

CSQU680E

Db2 and CF structure out of sync for list header *list-header-number* in structure *struc-name*

Severity
8

Explanation

The row for the shared queue in Db2 represents a different queue than the one found in the CF structure for list *header list-header-number* in structure *struc-name*. This inconsistency causes the queue manager to abend with 5C6-00C51053 and issue message CSQE137E. Messages CSQU681I and CSQU682I are also issued, providing further details.

System action

The mismatch is reported and the utility continues processing.

System programmer response

Collect the items listed in Coupling facility problem determination and in Db2 manager problem determination and contact your Db2 support center.

CSQU681I

Db2 entry for list header *list-header-number* in structure *struc-name: queue-name*

Severity
0

Explanation

This message is issued with message CSQU680E. *Queue-name* is the name of the queue found in Db2 for list header *list-header-number* in structure *struc-name*.

System action

The mismatch is reported and the utility continues processing.

System programmer response

Collect the items listed in Coupling facility problem determination and in Db2 manager problem determination and contact your Db2 support center.

CSQU682I

CF entry for list header *list-header-number* in structure *struc-name: queue-name*

Severity
0

Explanation

This message is issued with message CSQU680E. *Queue-name* is the name of the queue found in the CF for list header *list-header-number* in structure *struc-name*.

System action

The mismatch is reported and the utility continues processing.

System programmer response

Collect the items listed in Coupling facility problem determination and in Db2 manager problem determination and contact your Db2 support center.

CSQU683E

Missing CF entry for list header *list-header-number* in structure *struc-name*

Severity
8

Explanation

The Db2 entry for list header *list-header-number* in structure *struc-name* indicates that a current copy is available in the CF, however, the copy is not found. This inconsistency causes return code 2085 for applications trying to use this queue.

System action

The mismatch is reported and the utility continues processing.

System programmer response

Starting or restarting one of the queue managers in the queue-sharing group will resolve the problem. If the problem persists, collect the items listed in Coupling facility problem determination and in Db2 manager problem determination and contact your IBM support center.

CSQU684I

Structure *struc-name* has not yet been allocated by a queue manager

Severity

0

Explanation

The CF structure *struc-name* is not allocated. This happens when the first **IXLCONN** to the structure is issued, and should only be issued by a queue manager in the QSG.

System action

The utility continues processing.

System programmer response

None.

CSQU685I

Structure *struc-name* connected

Severity

0

Explanation

The utility has successfully connected to CF structure *struc-name*.

System action

The utility continues processing.

System programmer response

None.

CSQU686E

Structure *struc-name* connection failed, **IXLCONN** RC=*return-code* reason=*reason*

Severity

8

Explanation

The utility failed to connect to CF structure *struc-name*.

System action

The utility skips any further queues for this structure and continues processing.

System programmer response

Examine the return and reason codes to determine why the **IXLCONN** connect command failed.

CSQU687I

Structure *struc-name* disconnected

Severity

0

Explanation

The utility has disconnected from CF structure *struc-name*.

System action

The utility continues processing.

System programmer response

None.

CSQU688E

Missing Db2 entry for list header *list-header-number* in structure *struc-name*

Severity

0

Explanation

The CF entry for list header *list-header-number* in structure*struc-name* indicates that a current copy is available in Db2, however, the copy is not found. This inconsistency causes a problem if a new queue is defined for the same list header.

System action

The mismatch is reported and the utility continues processing.

System programmer response

Collect the items listed in Coupling facility problem determination and in Db2 manager problem determination and contact your IBM support center.

CSQU689E

Unexpected return code for structure *struc-name*, **IXLLSTE** RC=*return-code* reason=*reason*

Severity

8

Explanation

The utility failed to read a list entry from the CF structure *struc-name*.

System action

The utility skips any further queues for this structure and continues processing.

System programmer response

Examine the return and reason codes to determine why the **IXLLSTE** read failed.

CSQU950I

csect-name IBM MQ for z/OS V*n*

Explanation

This is part of the header to the report issued by the utility program.

CSQU951I

csect-name Data Conversion Exit Utility - *date time*

Explanation

This is part of the header to the report issued by the utility program.

CSQU952I
csect-name Utility completed, return code=*ret-code*

Explanation

The utility completed. The return code is 0 if all the input was processed successfully, or 8 if any errors were found.

System action

The utility ends.

System programmer response

If the return code is non-zero, investigate the errors that were reported.

CSQU954I
n structures processed

Explanation

This indicates how many data structures were processed by the utility program.

CSQU956E
Line *line-number*: structure array field has incorrect dimension

Explanation

The dimension specified for a structure array field was not correct.

System action

Processing stops.

System programmer response

Correct the field specification and resubmit the job.

CSQU957E
Line *line-number*: structure has field following a variable length field

Explanation

There was an error in the indicated line. A variable length field must be the last field of a structure.

System action

Processing continues.

System programmer response

Correct the field specification and resubmit the job.

CSQU958E
Line *line-number*: structure field name has unsupported type 'float'

Explanation

There was an error in the indicated line. A field had a type of 'float', which is not supported.

System action

Processing continues.

System programmer response

Correct the field specification and resubmit the job, or provide your own routine for converting such fields.

CSQU959E
Line *line-number*: structure field name has unsupported type 'double'

Explanation

There was an error in the indicated line. A field had a type of 'double', which is not supported.

System action

Processing continues.

System programmer response

Correct the field specification and resubmit the job, or provide your own routine for converting such fields.

CSQU960E
Line *line-number*: structure field name has unsupported type 'pointer'

Explanation

There was an error in the indicated line. A field had a type of 'pointer', which is not supported.

System action

Processing continues.

System programmer response

Correct the field specification and resubmit the job, or provide your own routine for converting such fields.

CSQU961E
Line *line-number*: structure field name has unsupported type 'bit'

Explanation

There was an error in the indicated line. A field had a type of 'bit', which is not supported.

System action

Processing continues.

System programmer response

Correct the field specification and resubmit the job, or provide your own routine for converting such fields.

CSQU965E
Invalid EXEC PARM

Explanation

The EXEC PARM field was not blank.

System action

The utility is terminated.

System programmer response

Change the JCL, and resubmit the job.

CSQU968E
Unable to OPEN *ddname* data set

Explanation

The program was unable to open data set *ddname*.

System action

The utility is terminated.

System programmer response

Examine the error message that was sent to the job log to determine the reason for the error. Check that the data set was correctly specified.

CSQU970E
Line line-number: syntax error

Explanation

There was a syntax error in the indicated line.

System action

Processing stops.

System programmer response

Correct the error and resubmit the job.

CSQU971E
Unable to GET from *ddname* data set

Explanation

The program was unable to read a record from the *ddname* data set.

System action

The utility is terminated.

System programmer response

Examine the error message that was sent to the job log to determine the reason for the error. Check that the data set was correctly specified.

CSQU972E
Unable to PUT to *ddname* data set

Explanation

The program was unable to write the next record to the *ddname* data set.

System action

The utility is terminated.

System programmer response

Examine the error message that was sent to the job log to determine the reason for the error. Check that the data set was correctly specified.

CSQU999E
Unrecognized message code *ccc*

Explanation

An unexpected error message code was issued by the utility.

System action

Processing continues.

System programmer response

Note the code *ccc* (which is shown in hexadecimal) and contact your IBM support center to report the problem.

Agent services messages (CSQV...): z/OS

CSQV086E
QUEUE MANAGER ABNORMAL TERMINATION REASON= *reason-code*

Explanation

The queue manager is ending abnormally, because an error that cannot be corrected has occurred. This message, which is not automatically deleted from the operator console, is issued during abnormal termination. *reason-code* is the termination reason code. If this abnormal termination is invoked multiple times, the termination reason code that accompanies this message is the reason associated with the first invocation.

System action

Abnormal termination processing continues.

System programmer response

For additional information, look up the reason code in "IBM MQ for z/OS codes" on page 5019.

This message is accompanied by one or more dumps. Obtain a copy of SYS1.LOGREC after the queue manager completely terminates, and the dumps. If you suspect an error in IBM MQ, see Troubleshooting and support for information about identifying and reporting the problem.

CSQV400I
ARCHIVE LOG QUIESCE CURRENTLY ACTIVE

Explanation

An ARCHIVE LOG MODE(QUIESCE) command is currently active. This message is part of the DISPLAY LOG or DISPLAY THREAD command report.

System action

This message is issued as information only. It indicates that the ARCHIVE LOG MODE(QUIESCE) command has not completed and, consequently, updates against IBM MQ resources have been temporarily suspended. This might result in active threads being suspended awaiting termination of the quiesce period. Processing otherwise continues normally.

CSQV401I
DISPLAY THREAD REPORT FOLLOWS -

Explanation

This message is issued as the title for the DISPLAY THREAD command report output. It precedes the other messages generated by this command:

- Message CSQV402I provides the formatted report when the detailed status of active threads is requested using TYPE(ACTIVE).
- Message CSQV432I provides the formatted report when the summary status of active threads is requested using TYPE(REGIONS).
- Message CSQV406I provides the formatted report when the status of in-doubt threads is requested using TYPE(INDOUBT).
- Message CSQV436I provides the formatted report when the status of in-doubt threads on another queue manager is requested using TYPE(INDOUBT) with QMNAME.

System action

Processing continues normally.

CSQV402I
ACTIVE THREADS -

Explanation

This message is the response to the DISPLAY THREAD TYPE(ACTIVE) command. It provides the status information for each active thread, as follows:

```
  NAME S T REQ THREAD-XREF USERID ASID URID  name s t req thread-xref userid asid urid
⋮
  DISPLAY ACTIVE REPORT COMPLETE
```

where:

name The connection name, which is one of the following:

- z/OS batch job name
- TSO user ID
- CICS APPLID
- IMS region name
- Channel initiator job name

s Connection status code:

N The thread is in IDENTIFY status.

T The thread has issued CREATE THREAD.

Q The CREATE THREAD request has been queued. The associated allied task is placed in a wait state.

C The thread is queued for termination as a result of the termination of the associated allied task. If this thread is also the last (or only) IBM MQ thread for the address space, the associated allied task is placed in a wait state.

D The thread is in the process of termination as a result of the termination of the associated allied task. If this thread is also the last (or only) IBM MQ thread for the address space, the associated allied task is placed in a wait state.

An asterisk is appended if the thread is active within IBM MQ.

t Connection type code:

B Batch: From an application using a batch connection

R RRS: From an RRS-coordinated application using a batch connection

C CICS: From CICS

I IMS: From IMS

S System: From an internal function of the queue manager or from the channel initiator.

req A wraparound counter to show the number of IBM MQ requests.

thread-xref
The recovery thread cross-reference identifier associated with the thread.

userid The user ID associated with a connection. If not signed-on, this field is blank.

asid A hexadecimal number representing the ASID of the home address space.

urid Unit of recovery identifier. This is the log RBA of the current unit of recovery associated with the thread. If there is no current unit of recovery, it is shown as 0000000000000000.

Exceptionally, the last line might be:

```
  DISPLAY ACTIVE TERMINATED WITH MAX LINES
```

if the report was generated in response to a command from a z/OS console and more than 252 response messages were generated. Only 252 response messages are returned.

System action

Processing continues normally.

CSQV406I
INDOUBT THREADS -

Explanation

This message is the response to the DISPLAY THREAD TYPE(INDOUBT) command. It provides the status information for each in-doubt thread, as follows:

```
NAME THREAD-XREF URID NID  name thread-xref urid origin-id
⋮
DISPLAY INDOUBT REPORT COMPLETE
```

where:

name The connection name, which is one of the following:

- z/OS batch job name
- TSO user ID
- CICS APPLID
- IMS region name
- Channel initiator job name

thread-xref
The recovery thread cross-reference identifier associated with the thread. See Connecting from the IMS control region for more information.

urid Unit of recovery identifier. This is the log RBA of the current unit of recovery associated with the thread. (This is omitted if the command was issued from a z/OS console with a non-specific connection name.)

origin-id
The origin identifier, a unique token identifying the unit of recovery within the queue manager. This has the form *origin-node.origin-urid*, where:

origin-node
A name that identifies the originator of the thread. (This is omitted for batch RRS connections.)

origin-urid
The hexadecimal number assigned to the unit of recovery for this thread by the originating system.

Exceptionally, the last line might be:

```
DISPLAY INDOUBT TERMINATED WITH MAX LINES
```

if the report was generated in response to a command from a z/OS console and more than 252 in-doubt threads were eligible for display.

System action

Processing continues normally.

CSQV410I
NO ACTIVE CONNECTION FOUND FOR NAME=*connection-name*

Explanation

The DISPLAY THREAD command was unable to find any active connection associated with *connection-name*.

System action

Command processing continues.

CSQV411I
NO ACTIVE THREADS FOUND FOR NAME=*connection-name*

Explanation

The DISPLAY THREAD command was unable to locate any active threads associated with *connection-name*.

System action

Command processing continues.

CSQV412I
csect-name NO INDOUBT THREADS FOUND FOR NAME=connection name

Explanation

The DISPLAY THREAD command was unable to locate any in-doubt threads associated with *connection name*.

System action

Command processing continues.

CSQV413E
csect-name CONNECTION NAME MISSING

Explanation

A connection name was not supplied with the command, and a default connection name cannot be determined.

System action

Command processing terminates.

CSQV414I
THREAD NID=*origin-id* COMMIT SCHEDULED

Explanation

The thread specified by the recovery origin identifier *origin-id* is scheduled for COMMIT recovery action.

System action

Processing continues.

CSQV415I
THREAD NID=*origin-id* BACKOUT SCHEDULED

Explanation

The thread specified by the recovery origin identifier *origin-id* is scheduled for BACKOUT recovery action.

System action

Processing continues.

CSQV416E
THREAD NID=*origin-id* IS INVALID

Explanation

The RESOLVE INDOUBT command determined that the input format for the specified thread *origin-id* is invalid.

System action

Command processing continues.

CSQV417I
THREAD NID=*origin-id* NOT FOUND

Explanation

The RESOLVE INDOUBT command was unable to locate the thread specified by the recovery origin identifier *origin-id* to be scheduled for recovery. Either the thread identifier is incorrect, or the thread is no longer in an in-doubt state.

System action

Command processing continues.

CSQV419I
NO ACTIVE CONNECTIONS FOUND

Explanation

A DISPLAY THREAD(*) TYPE(ACTIVE) or TYPE(REGIONS) command was issued for all threads, but no active connections were found.

System action

Command processing continues.

CSQV420I
NO INDOUBT THREADS FOUND

Explanation

A DISPLAY THREAD(*) TYPE(INDOUBT) command was issued for all threads, but no in-doubt threads were found.

System action

Command processing continues.

CSQV423I
cmd MESSAGE POOL SIZE EXCEEDED

Explanation

The storage requirement needed to generate responses for the command *cmd* exceeded the maximum size of the message buffer pool.

System action

Processing is terminated.

CSQV424I
THREAD ID=*thread-xref* COMMIT SCHEDULED

Explanation

The thread specified by the recovery thread cross-reference identifier *thread-xref* is scheduled for COMMIT recovery action.

System action

Processing continues.

CSQV425I
THREAD ID=*thread-xref* BACKOUT SCHEDULED

Explanation

The thread specified by the recovery thread cross-reference identifier *thread-xref* is scheduled for BACKOUT recovery action.

System action

Processing continues.

CSQV427I
THREAD ID=*thread-xref* NOT FOUND

Explanation

The RESOLVE INDOUBT command was unable to locate the thread specified by the recovery thread cross-reference identifier *thread-xref* to be scheduled for recovery. Either the thread identifier is incorrect, or the thread is no longer in an in-doubt state.

System action

Command processing continues.

CSQV428I
CURRENT THREAD LIMIT OF *nnn* EXCEEDED. CREATE THREAD FOR JOB *jobname* DEFERRED

Explanation

A job requested a connection to the queue manager, but the current number of connections is the maximum allowed.

System action

The request for a connection is suspended, and waits until another connection ends.

System programmer response

If this situation occurs frequently, contact your IBM support center for assistance.

CSQV432I
ACTIVE THREADS -

Explanation

This message is the response to the DISPLAY THREAD TYPE(REGIONS) command. It provides the status information for each active connection, as follows:

```
  NAME TYPE USERID ASID THREADS  name type userid asid threads
⋮
  DISPLAY ACTIVE REPORT COMPLETE
```

where:

name The connection name, which is one of the following:

- z/OS batch job name
- TSO user ID
- CICS APPLID
- IMS region name
- Channel initiator job name

type The connection type:

CICS From CICS.

IMS From IMS.

BATCH
From an application using a batch connection.

RRSBATCH
From an RRS-coordinated application using a batch connection.

CHINIT
From the channel initiator.

userid The user ID associated with a connection. If not signed-on, this field is blank.

asid A hexadecimal number representing the ASID of the home address space.

threads
The number of active threads associated with the connection. This excludes fixed internal threads, such as those for the CICS adapter tasks, or the channel initiator listeners.

Exceptionally, the last line might be:

```
DISPLAY ACTIVE TERMINATED WITH MAX LINES
```

if the report was generated in response to a command from a z/OS console and more than 252 response messages were generated. Only 252 response messages are returned.

System action

Processing continues normally.

CSQV433I
'QMNAME' NOT ALLOWED, NOT IN QUEUE-SHARING GROUP

Explanation

A DISPLAY THREAD TYPE(INDOUBT) or RESOLVE INDOUBT command specifying the QMNAME keyword was issued, but the requesting queue manager *qmgr-name* is not in a queue-sharing group or the requested queue manager *qmgr-name* is not a member of the queue-sharing group.

System action

Processing for the command is terminated.

CSQV434E
'QMNAME' ALLOWED ONLY WITH TYPE(INDOUBT)

Explanation

A DISPLAY THREAD command specifying the QMNAME keyword was issued, but TYPE(INDOUBT) was not specified.

System action

Processing for the command is terminated.

CSQV435I
QMNAME(qmgr-name) IS ACTIVE, COMMAND IGNORED

Explanation

A DISPLAY THREAD TYPE(INDOUBT) or RESOLVE INDOUBT command specifying the QMNAME keyword was issued, but the requested queue manager *qmgr-name* is active.

System action

Processing for the command is terminated.

CSQV436I
INDOUBT THREADS FOR *qmgr-name* -

Explanation

This message comprises the response to the DISPLAY THREAD TYPE(INDOUBT) command when the QMNAME keyword was specified. It provides the status information for each in-doubt unit-of-work on the requested queue manager; the information is displayed in the same format as in message CSQV406I.

System action

Processing continues normally.

CSQV437I
CANNOT RESOLVE THREAD NID=*origin-id*, SOME RESOURCES UNAVAILABLE

Explanation

The RESOLVE INDOUBT command was unable to schedule the thread specified by the recovery origin identifier *origin-id* for recovery, because not all the resources necessary for recovery were available.

System action

The identified thread will remain in-doubt.

CSQV450I
csect-name Unable to open *ddname* data set

Explanation

The *ddname* data set could not be opened, as reported in the preceding messages.

System action

Processing continues, but functions that require the data set will be inhibited. For example, if the exit library data set CSQXLIB cannot be opened, cluster workload user exits will not be available.

System programmer response

Investigate the problem reported in the preceding messages.

CSQV451I
csect-name Unable to get storage for exits, RC=*return-code*

Explanation

An attempt to obtain some storage for use by exits failed. *return-code* is the return code (in hexadecimal) from the z/OS STORAGE service.

System action

Processing continues, but cluster workload user exits will not be available.

System programmer response

See the *MVS Programming: Assembler Services Reference* manual for information about the return code from the STORAGE request.

CSQV452I
csect-name Cluster workload exits not available

Explanation

Cluster workload user exit functions will not be available, because:

- There is no CSQXLIB DD statement in the started task JCL procedure for the queue manager, xxxxMSTR
- The EXITTCB system parameter is zero.

System action

Processing continues, but cluster workload user exits will not be available.

System programmer response

If you want to use cluster workload exits, add the required statement to the queue manager started task JCL procedure and specify a non-zero value for the EXITTCB system parameter. For more information about cluster workload exits, see Cluster workload exit programming.

CSQV453I

csect-name Unable to load *module-name*, reason=*ssssrrrr*

Explanation

The queue manager was unable to load a module required for exits. *ssss* is the completion code and *rrrr* is the reason code (both in hexadecimal) from the z/OS LOAD service.

System action

Processing continues, but cluster workload user exits will not be available.

System programmer response

Check the console for messages indicating why the module was not loaded. See the *MVS Programming: Assembler Services Reference* manual for information about the codes from the LOAD request.

Ensure that the module is in the required library, and that it is referenced correctly. The queue manager attempts to load this module from the library data sets under the STEPLIB DD statement of its started task JCL procedure xxxxMSTR.

CSQV455E

csect-name Cluster workload exit exit-name timed out

Explanation

A cluster workload user exit did not return to the queue manager within the allowed time, as specified by the EXITLIM system parameter.

System action

The exit is disabled until its load module in the CSQXLIB data set is refreshed.

System programmer response

Investigate why your exit is not returning in time.

CSQV456E

csect-name Cluster workload exit error, TCB=*tcb-name* reason=*sssuuu-reason*

Explanation

The exit subtask using TCB *tcb-name* is ending abnormally because an error that cannot be corrected has occurred in a cluster workload user exit. *sss* is the system completion code, *uuu* is the user completion code, and *reason* is the associated reason code (all in hexadecimal).

System action

The subtask ends abnormally, and a dump is normally issued. The exit is disabled until its load module in the CSQXLIB data set is refreshed.

System programmer response

User completion codes are generally the result of errors detected by the exit itself. If a system completion code is shown, see the *MVS System Codes* manual for information about the problem in your exit.

CSQV457E

csect-name Unable to establish ESTAE, RC=*return-code*

Severity
8

Explanation

During startup processing, the recovery environment for a cluster workload user exit task could not be set up. *return-code* is the return code (in hexadecimal) from the z/OS ESTAE service.

System action

The task does not start. Cluster workload user exits will be available providing at least one task starts.

System programmer response

See the *MVS Programming: Assembler Services Reference* manual for information about the return code from the ESTAE request. If you are unable to solve the problem, contact your IBM support center for assistance.

CSQV459I
csect-name Unable to free storage for exits, RC=*return-code*

Explanation

An attempt to release some storage that was used by exits failed. *return-code* is the return code (in hexadecimal) from the z/OS STORAGE service.

System action

Processing continues.

System programmer response

See the *MVS Programming: Assembler Services Reference* manual for information about the return code from the STORAGE request.

CSQV460I
csect-name Cluster workload exits are disabled but **CLWLEXIT** is set.

Explanation

A **CLWLEXIT** value is on the queue manager when it is started, however cluster workload exits are not enabled, and are prevented from operating.

System action

Message CSQV461D is issued, but **CLWLEXITs** are not enabled.

CSQV461D
csect-name

Reply `Y` to continue startup with CLWLEXIT not enabled, or `N` to shutdown.

Explanation

Issued after message CSQV460I. Due to a **CLWLEXIT** being set in the queue manager with cluster workload exits not enabled, a reply is required to continue startup with cluster workload exits not enabled.

System action

Queue manager startup waits for the reply from the operator. Replying `Y` allows the queue manager to continue startup with cluster workload exists not enabled. Replying `N` shuts down the queue manager with abend reason 00D40039.

While cluster workload exits are not enabled, the **CLWLEXIT** value can only be changed to a blank value, and the exit will not function. You should use the IBM MQ supplied workload balancing

algorithm and attributes, to alter how objects are selected, and remove the **CLWLEXIT** value. See Workload balancing in clusters for more information.

For further information contact IBM Support.

Instrumentation facilities messages (CSQW...): z/OS

CSQW001I
ASYNCHRONOUSLY GATHERED DATA IS BEING FORMATTED

Explanation

The dump formatting exit is not using summary dump records for formatting. The formatted control blocks might not contain the same values as they did at the time of the error.

System action

Dump formatting continues.

System programmer response

If you want summary dump records to be used, do not specify the 'SUMDUMP=NO' operand on the MQ DUMP DISPLAY MAIN MENU (if you are using the dump display panels), or in the CSQWDMP verbexit (if you are using line mode IPCS).

CSQW002I
SUMMARY DUMP RECORDS ARE BEING FORMATTED

Explanation

The dump formatting exit is using MQ summary dump record information to format its control blocks.

System action

Dump formatting continues.

System programmer response

If you do not want IBM MQ summary dump records to be used in formatting, specify the 'SUMDUMP=NO' and 'SUBSYS=subsystem name' on the MQ DUMP DISPLAY MAIN MENU (if you are using the dump display panels), or in the CSQWDMP verbexit (if you are using line mode IPCS). Both operands are required.

CSQW004E
ONE OR MORE OPERANDS ARE NOT VALID. FORMATTING TERMINATED

Explanation

An invalid operand was specified on the MQ DUMP DISPLAY MAIN MENU (if you are using the dump display panels), or in the CSQWDMP verbexit (if you are using line mode IPCS).

System action

The dump formatting exit terminates.

System programmer response

Correct the operand specified by message CSQW007E.

CSQW006E
THE ERLY BLOCK CANNOT BE ACCESSED OR IT IS INVALID

Explanation

The dump formatting exit could not locate its anchor block.

System action

The dump formatting exit terminates.

System programmer response

Specify 'SUBSYS=subsystem name', and 'SUMDUMP=NO' on the MQ DUMP DISPLAY MAIN MENU (if you are using the dump display panels), or in the CSQWDMP verbexit if you are using line mode IPCS.

CSQW007E
OPERAND IS NOT VALID: *xxxx*

Explanation

The specified operand was not a valid dump formatting operand.

System action

The dump formatting exit terminates.

System programmer response

Check the dump formatting operands.

CSQW008E
THE SCOM CANNOT BE ACCESSED OR IT IS INVALID

Explanation

An error was encountered while trying to retrieve the SCOM.

System action

The dump formatting exit terminates.

System programmer response

If 'SUMDUMP=NO' was specified on the MQ DUMP DISPLAY MAIN MENU (if you are using the dump display panels), or in the CSQWDMP verbexit (if you are using line mode IPCS) omit it and resubmit the request. Otherwise, specify this operand, and resubmit the request.

CSQW009E
THE ADDRESS SPACE REQUESTED IS NOT AVAILABLE

Explanation

The MQ control blocks for the address space specified could not be located.

System action

Formatting continues of any other requested dump segment.

System programmer response

Check the ASID specified. The ASID must be specified in hexadecimal.

CSQW010E
THE TRACE RMFT CANNOT BE ACCESSED OR IT IS INVALID

Explanation

The MQ trace table could not be located.

System action

Formatting of the MQ trace table is bypassed, and formatting continues of any other requested dump segment.

System programmer response

If 'SUMDUMP=NO' was specified try formatting the dump again using the summary dump because it could contain the information required to access this data.

If 'SUMDUMP=NO' was not specified, and the summary dump was used, try formatting the dump again specifying this option because the summary dump data could have been corrupted.

CSQW011I
A LARGER REGION SIZE IS REQUIRED FOR THIS JOB

Explanation

The dump formatting exit could not obtain a large enough work buffer to process the summary dump records.

System action

The dump formatting exit terminates.

System programmer response

Rerun the job, specifying a larger TSO region size (or a larger region size if running in batch).

CSQW013I
DMPW NOT FOUND IN SUMMARY DUMP

Explanation

The dump formatting exit was unable to locate the DMPW control block in the summary record portion of the dump data set. Because the DMPW provides the main anchor block for the dump formatter, processing is terminated.

System action

The dump formatting exit terminates.

System programmer response

Specify 'SUBSYS=xxxx' to identify which address space to format information for.

CSQW014I
REQUIRED SUMMARY DUMP RECORDS ARE NOT IN THIS DUMP. WILL ATTEMPT TO FORMAT FROM NON-SUMMARY DUMP

Explanation

Expected data could not be found in the summary dump. This message is issued for information only. Dump formatting continues.

System action

Formatting is attempted using information found from the full dump instead of the summary dump.

CSQW015I
SSCVT NOT LOCATED, CHECK THE SUBSYSTEM NAME SPECIFIED

Explanation

In a search through the SSCVT chain, a match of the subsystem name in the SSCVTs and the subsystem name specified was not found.

System action

Formatting for the named subsystem is not done.

System programmer response

Specify the subsystem name correctly.

CSQW016I
THE RMVT CANNOT BE ACCESSED OR IT IS INVALID

Explanation

The dump formatting exit could not locate the RMVT. The RMVT is required for formatting the MQ trace table and a number of other MQ control blocks.

System action

Formatting of the MQ trace table is bypassed, and formatting of other requested dump segments continues.

System programmer response

If 'SUMDUMP=NO' was specified try formatting the dump again using the summary dump because it could contain the information required to access this data.

If 'SUMDUMP=NO' was not specified, and the summary dump was used, try formatting the dump again specifying this option because the summary dump data could have been corrupted.

CSQW017E
MAXIMUM STACK LEVEL EXCEEDED

Explanation

This condition is usually caused by the MQ control block formatter looping. The stack array is depleted and can no longer accommodate control blocks.

System action

Dump formatting is terminated.

System programmer response

Contact your IBM support center.

CSQW018I
SUBSYS= SPECIFIED INCORRECTLY OR MISSING. REQUIRED IF SUMDUMP=NO SPECIFIED

Explanation

The 'SUMDUMP=NO' option was specified, but either the 'SUBSYS=' operand is missing, or it was incorrectly specified.

System action

Dump formatting is terminated.

System programmer response

Specify the name of the subsystem in the 'SUBSYS=' operand, and resubmit the request.

CSQW020I
UNSUCCESSFUL SEARCH FOR THE ERLY CONTROL BLOCK

Explanation

A key control block could not be located in the dump.

System action

Dump formatting is terminated.

System programmer response

Check that the 'SUBSYS=' operand was correctly specified, and resubmit the request.

CSQW022I
THE RESIDENT TRACE WAS NOT ACTIVE AT THE TIME OF DUMP

Explanation

Trace table formatting has been attempted, but no trace table existed at the time of the dump.

System action

Dump formatting continues with any other control blocks that were to be formatted.

CSQW023I
THE TRACE TABLE ENTRY IS OUT OF SEQUENCE OR OVERLAID

Explanation

A trace entry is overlaid by another trace entry of a different time stamp. This message is issued to flag an unrecognized trace entry. This error can occur if the dump is initiated by operator command, because the MQ address space continues to run while the dump is being taken.

System action

Formatting of the trace table continues.

CSQW024I
TRACE TABLE

Explanation

This identifies the start of the formatted trace table.

System action

Trace table formatting follows.

CSQW025I
ERROR ACCESSING THE TRACE TABLE

Explanation

A nonzero return code was returned from the storage access routine when accessing the trace table.

System action

Trace table formatting is bypassed.

CSQW026I
CONTROL BLOCK SUMMARY (ALL ADDRESS SPACES)

Explanation

This messages provides descriptive information about the type of formatting being produced.

System action

Dump formatting continues.

CSQW027I
CONTROL BLOCK SUMMARY (SINGLE ADDRESS SPACE)

Explanation

This messages provides descriptive information about the type of formatting being produced.

System action

Dump formatting continues.

CSQW028I
CONTROL BLOCK SUMMARY (LONG FORM GLOBAL)

Explanation

This messages provides descriptive information about the type of formatting being produced.

System action

Dump formatting continues.

CSQW029I
CONTROL BLOCK SUMMARY (SHORT FORM GLOBAL)

Explanation

This messages provides descriptive information about the type of formatting being produced.

System action

Dump formatting continues.

CSQW030E
DUMP ACCESS ERROR ACCESSING THE CONTROL BLOCK STRUCTURE TABLE IN THE DUMP

Explanation

A control block identifying the structure of MQ control blocks could not be found.

System action

Control block formatting is terminated.

System programmer response

Check the z/OS console to see if any messages were produced to indicate that there was a problem when the dump was taken. If you suspect an error in IBM MQ, see Troubleshooting and support for information about reporting the problem.

CSQW032E
ERROR ACCESSING ANCHOR CONTROL BLOCK

Explanation

A control block cannot be accessed from the dump.

System action

Control block formatting is terminated.

System programmer response

Check the z/OS console to see if any messages were produced to indicate that there was a problem when the dump was taken. If you suspect an error in IBM MQ, see Troubleshooting and support for information about reporting the problem.

CSQW033I
BEGINNING FORMATTING

Explanation

Formatting of MQ control blocks is beginning.

CSQW034I
TRACE TABLE AND GLOBAL BLOCKS ALREADY FORMATTED

Explanation

An indicative dump is being requested. The MQ trace table and the global blocks have already been formatted with first dump (full dump) for this abend dump (SNAP) invocation. These are, therefore, not formatted for this task.

CSQW035I
WARNING - NO TASK RELATED CONTROL BLOCKS FOR THIS TASK

Explanation

The task for which the dump is being requested is not identified to MQ. Task-related control blocks are not dumped. The MQ trace table and global blocks are dumped only if the SYSABEND DD statement is present and only if this is the first of the dumps (full dump) for this abend dump (SNAP) invocation.

System action

No MQ formatting is done for the specified task.

CSQW036I
CONTROL BLOCKS FOR TASKS ASSOCIATED WITH THE ABOVE RECOVERY COORDINATOR TASK

Explanation

The formatted blocks following this message are associated with tasks that have been identified to MQ with the 'recovery coordinator = no' option. These tasks might not have invoked SNAP, but they are associated with the task that did.

System action

The appropriate control blocks are formatted.

System programmer response

Examine the control blocks for relevant information.

CSQW037I
TASK RELATED CONTROL BLOCKS FOR THIS TASK

Explanation

The formatted blocks following this message are associated with the current task.

System action

The appropriate control blocks are formatted.

System programmer response

Examine the control blocks for relevant information.

CSQW038I
END OF FORMATTING

Explanation

Formatting of MQ control blocks is completed.

CSQW039I
FORMATTING COMPLETE FOR THIS DUMP

Explanation

The dump formatting exit has completed its processing for this dump data set.

CSQW041E
THE TAB CANNOT BE ACCESSED OR IT IS INVALID

Explanation

The MQ trace table anchor block could not be located.

System action

Formatting of the MQ trace table is bypassed, and formatting of any other requested dump segment continues.

System programmer response

If 'SUMDUMP=NO' was specified try formatting the dump again using the summary dump because it could contain the information required to access this data.

If 'SUMDUMP=NO' was not specified, and the summary dump was used, try formatting the dump again specifying this option because the summary dump data could have been corrupted.

Check the z/OS console to see if any messages were produced to indicate that there was a problem when the dump was taken. If you suspect an error in IBM MQ, see Troubleshooting and support for information about reporting the problem.

CSQW042E
REQUIRED SUMMARY DUMP RECORDS ARE NOT IN THIS DUMP. RERUN SPECIFYING SUBSYS= PARAMETER

Explanation

The summary dump records were not found in the dump. When this occurs the dump formatter needs the subsystem name to be able to identify which address space is to be formatted.

System action

Dump formatting is terminated.

System programmer response

Rerun the formatting specifying the parameter the subsystem name (using 'SUBSYS=').

CSQW049I
OLDEST SLOT ADDRESS INVALID, FORMATTING TRACE TABLE FROM FIRST ENTRY

Explanation

There are several pointers in the control block that defines the trace. One points to the start of the storage that contains the trace data, one to the end, and one to the next free record. The formatter has detected that the pointer to the next free record is outside the range indicated by the pointers to the start and end of the storage.

System action

Dump formatting continues, but from the physical start of the trace table, not the oldest record.

System programmer response

If the time of day values are meaningful, and in sequence, scan down the formatted trace to find the latest trace record written.

CSQW050I
ssnm NO SDWA/LOGREC, ABN=*comp-reason*, U=*userid*, M=*module*, C=*compid.vrm.comp-function*

Explanation

This message provides the default SVC dump title (SDUMP) associated with the SYS1.DUMP data set, when an SDWA was unavailable during recovery processing. The individual variable fields contain:

Field	Contents
ssnm	MQ subsystem name
ABN	The abend completion code, followed by the abend reason code
U	The user ID for the individual subsystem user
M	The function recovery routine responsible for the dump
C	The component-ID
vrm	The MQ version, release number, and modification level

comp-function
The component-ID function

System action

Dump processing continues.

System programmer response

Since the SDWA provides important diagnostic information to assist in problem determination, the recovery environment at time of error should be examined to determine why an SDWA was not provided for this ABEND.

In a non-recovery environment, there might be valid reasons for the lack of an SDWA (for example, the operator could have initiated the dump).

CSQW051E
ERROR DURING DUMP PROCESSING

Explanation

This message is generated by the recovery routine of the SDUMP dump data gathering service when an error is encountered during dump processing.

System action

Processing of the SUMLSTA user storage areas is terminated, an SVC dump is requested, and control is returned to RTM.

System programmer response

This error is documented in a SYS1.LOGREC record. This message can be issued because of an error in the invocation of SDUMP, or because of an error in SDUMP itself, or during control block examination and access.

CSQW053I
VRA DIAGNOSTIC INFORMATION REPORT

Explanation

The variable recording area (VRA) is part of the system diagnostic work area (SDWA) and contains MQ diagnostic information. The VRA is extracted and displayed in this report.

For information about this report, see Troubleshooting and support .

System action

Dump formatting continues.

CSQW054I
NO VRA DATA RECORDED IN SDWA

Explanation

The SDWA obtained from the SYS1.DUMP data set contained no diagnostic information in the VRA.

System action

VRA report generation is bypassed, dump format processing continues.

CSQW055I
UNABLE TO LOCATE SDWA

Explanation

The z/OS summary dump data access service routine (IEAVTFRD) was unable to locate the SDWA in the summary data portion of the SYS1.DUMP data set. SVC dumps only contain an SDWA if they are initiated by MQ. If the dump was initiated by any other means (such as the operator) the SDWA will not be present.

System action

No VRA is produced, and dump formatting continues.

CSQW056I
VRA DIAGNOSTIC REPORT COMPLETE

Explanation

The dump formatter has completed processing of the VRA diagnostic report.

System action

Dump formatting continues.

CSQW059I
SUMMARY OF CONNECTED JOBS

Explanation

A summary of information about connected jobs follows.

System action

Job summary information follows.

CSQW060I
BEGIN SAVE AREA TRACE

Explanation

This message identifies the start of the MQ register save area trace report which appears in the formatted section of an MQ SVC dump. This report is useful for problem determination because it contains the save areas for the agent execution block (EB) in error, and all associated agent EBs, traced from the point of error and displayed in order of invocation.

System action

Save area trace format processing continues for the agent EB in error, and all associated agent EBs.

CSQW061I
SAVE AREA TRACE COMPLETE

Explanation

This message indicates that the MQ formatted save area trace report (CSQW060I) is complete.

System action

Dump formatting continues.

CSQW062I
R6 (*R6-contents*) DOES NOT CONTAIN A VALID EB ADDRESS

Explanation

During dump format processing of the MQ formatted save area trace report (CSQW060I), register 6 (R6) did not contain the address of a valid agent execution block (EB).

System action

Save area trace format processing is terminated for the current agent EB, and all prior EBs.

CSQW063E
name (*address*) ASID (*asid*) NOT FOUND IN DUMP

Explanation

During processing of the save area trace report (CSQW060I), a control block or save area was not found in the dump data set.

Because the dump formatter uses the MQ and z/OS control blocks defined under the *name* field of this message to locate individual register save areas, subsequent save areas located using the *named* control block or save area will not be displayed in the report.

name Identifies the name of the control block or save area that was not found in the dump data set:

SA Indicates a save area

ASCE MQ address space control element

EB MQ execution block

TCB z/OS task control block

RB z/OS request block

XSB z/OS extended status block

PSA z/OS prefix save area

SDWA
z/OS system diagnostic work area

STSV z/OS SRB status save area

STKE z/OS cross memory stack element

address
The address of the named control block or save area.

asid The address space identifier associated with the control block or save area.

Due to the execution structures and environmental restrictions of selected MQ and z/OS control structures, some control blocks and save areas associated with these execution environments will not be included in the dump data set.

System action

Register save area trace format processing for the current save area chains is terminated. Subsequent save area processing will vary depending on the specific control block or save area that was available, and the MQ agent execution environments at the time of the error.

CSQW064I
ERROR BLOCK NOT FOUND IN DUMP

Explanation

The dump formatter was unable to format a control block because the storage could not be found.

System action

Dump formatting continues.

CSQW065I
ERROR BLOCK LENGTH INCORRECT

Explanation

During the formatting of a control block, a mismatch was found between the expected length and the value determined from the dump.

System programmer response

You might find this message helpful when solving a more serious problem because it might indicate that a control block has been corrupted.

CSQW066I
ERROR BLOCK ID INCORRECT

Explanation

Each control block type has a unique identifier for verification. During the formatting of the control block, a mismatch occurred between the value expected and the value found in the control block in the dump.

System programmer response

This message could indicate that storage has been overlaid, and you might find it helpful when solving a more serious problem because it might indicate that a control block has been corrupted.

CSQW067I
ERROR BLOCK CHAINED FROM THIS BLOCK NOT FOUND IN DUMP

Explanation

Control blocks can contain pointers to other control blocks. A control block pointed to by the current control block could not be found in the dump.

System programmer response

This message could indicate that storage has been overlaid, and you might find it helpful when solving a more serious problem. The control block pointed to will have error message CSQW064I associated with it.

CSQW068I
ERROR BLOCK CHAINED FROM THIS BLOCK HAS INCORRECT ID

Explanation

Each control block type has a unique identifier for verification. During the formatting of a control block pointed to by the current control block, a mismatch occurred between the value expected and the value found in the control block in the dump.

System programmer response

This message could indicate that storage has been overlaid, and you might find it helpful when solving a more serious problem because it might indicate that a control block has been corrupted. The control block in error has error message CSQW066I associated with it.

CSQW069I
ERROR BLOCK EYECATCHER INCORRECT

Explanation

Each control block type has a unique eyecatcher for verification. During the formatting of the control block, a mismatch occurred between the value expected and the value found in the control block in the dump.

System programmer response

This message could indicate that storage has been overlaid, and you might find it helpful when solving a more serious problem because it might indicate that a control block has been corrupted.

CSQW070I
DUMP TITLE *dump-title*

Explanation

This shows the title of the dump.

CSQW072I
ENTRY: MQ user parameter trace

Explanation

This message is inserted into the formatted MQ trace to indicate that the control block was traced on entry to MQ.

CSQW073I
EXIT: MQ user parameter trace

Explanation

This message is inserted into the formatted MQ trace to indicate that the control block was traced on exit from MQ.

CSQW074I
ERROR: MQ user parameter trace

Explanation

This message is inserted into the formatted MQ trace to indicate that the control block was traced because it was determined to be in error.

CSQW075I
WARNING - data was truncated at 256 bytes

Explanation

This message is inserted into the formatted MQ trace when a control block has exceeded a 256 byte length limit.

CSQW076I
Return code was *mqrc*

Explanation

This message is inserted into the formatted MQ trace when an error has been detected. *mqrc* is the return code. Refer to API completion and reason codes for information about this code.

CSQW105E
ERROR DURING LOAD OR VALIDATION OF A CONTROL BLOCK STRUCTURE TABLE MODULE

Explanation

The MQ dump formatting facility cannot be used to format control blocks. An error occurred during the startup process while attempting to LOAD one of the Control Block Structures Table modules (CSQWDST1, CSQWDST2, CSQWDST3, and CSQWDST4) from the MQ program library.

System action

Queue manager startup processing continues.

System programmer response

If you expect to experience problems, stop your queue manager, resolve the problem, and restart. If you do not anticipate that this error will cause problems, you can stop and restart the queue manager at a convenient time.

CSQW108E
UNABLE TO AUTOMATICALLY START '*type*' TRACE

Explanation

System parameters indicated that an MQ trace should be started automatically during queue manager initialization, but the queue manager was unable to start the trace.

System action

Queue manager initialization continues.

System programmer response

Start the trace with the START TRACE command after queue manager initialization is complete.

CSQW109E
TRACE INITIALIZATION PARAMETERS UNAVAILABLE, DEFAULTS USED FOR '*type*' TRACE

Explanation

The trace function was unable to access the trace initialization parameters defined by the CSQ6SYSP macro. Default values as defined by that macro are assumed for trace parameters.

System action

Queue manager initialization continues.

System programmer response

Determine if the system parameter load module (the default version is called CSQZPARM) is missing or inaccessible. Trace can be started with the START TRACE command.

CSQW120E
DEST VALUE IS INVALID FOR '*type*' TRACE

Explanation

A trace command has been entered, but the specified destination value is not valid for the trace type requested.

System action

Processing for the TRACE command is terminated.

System programmer response

If a START TRACE command was entered, specify a valid destination for the trace. Otherwise, a DISPLAY TRACE command can be issued to determine what traces are currently active. See MQSC commands for information about valid destinations.

CSQW121E
CLASS VALUE IS INVALID FOR '*type*' TRACE

Explanation

A trace command has been entered, but the specified class value is not valid for the trace type requested.

System action

Processing for the TRACE command is terminated.

System programmer response

If a START TRACE command was entered, specify a valid class for the trace. Otherwise, a DISPLAY TRACE command can be issued to determine what options are currently active. See MQSC commands for information about valid classes.

CSQW122E
'*keyword*' IS NOT VALID FOR '*type*' TRACE

Explanation

A trace command has been entered, but *keyword* is not valid for the trace type specified.

System action

Processing for the TRACE command is terminated.

System programmer response

Either the named keyword must be omitted from the command, or a different type of trace must be specified. See MQSC commands for information about valid combinations of keywords and trace types.

CSQW123I
csect-name TRACE RECORDING HAS BEEN RESUMED ON *dest*

Explanation

dest destination has resumed acceptance of trace data after an error.

System action

Data recording is resumed.

CSQW124E
csect-name '*type*' TRACE TERMINATED RC=*code* RMID=*nn*

Explanation

During processing *type* trace, processing ended due to an error. A trace type of blank indicates all tracing has stopped. RMID, displayed in decimal, identifies the resource manager. For information on IBM MQ RMIDs, see the TRACE commands in MQSC commands.

code, displayed in hexadecimal, specifies the return, reason, or abend code associated with the action. Refer to "IBM MQ for z/OS codes" on page 5019 for information about these codes.

Further collection of the named trace is stopped. If it is necessary to resume collection of the trace, a START TRACE command can be issued. However if another error is experienced, the problem should be resolved before starting the trace collection again.

System action

Processing for the named trace type is stopped. The message is not externalized by the functional recovery routine, but is output whenever an IFC event is driven at a later time. A trace type of blank indicates all tracing has stopped.

System programmer response

Investigate the reasons for the error. If necessary to collect the named trace, issue a START TRACE command to resume processing.

CSQW125E
MULTIPLE VALUES NOT ALLOWED FOR *keyword* AND *keyword*

Explanation

Multiple values were specified for both of the named keywords. At most one of these keywords is allowed multiple values on a single command.

System action

Processing for the command is terminated.

System programmer response

Reenter a valid command. See MQSC commands for additional information.

CSQW126E
'*type*' TRACE NOT ALLOWED, ACTIVE TRACE TABLE FULL

Explanation

The *type* trace cannot be started because the active trace table has reached the maximum number of active traces allowed.

System action

Processing for the command is terminated.

System programmer response

Use the DISPLAY TRACE command to see if an active trace could be stopped. An active trace must be stopped before any other start trace command will be processed.

CSQW127I
CURRENT TRACE ACTIVITY IS -

Explanation

This message is issued in response to the DISPLAY TRACE command. For each trace that is active, the message indicates the trace number, the type of trace, the class(es) within type, the destination(s) for the trace entries, the user ID, and the RMID(s), as follows:

```
TNO TYPE CLASS DEST USERID RMID  tno type class dest userid rmid
⋮
END OF TRACE REPORT
```

The trace number *tno* can be:

01-03 A trace started automatically when the queue manager started, or a trace started by a START TRACE command.

04-32 A trace started by a START TRACE command.

00 The global trace started automatically when the channel initiator started.

Notes:

1. For TRACE(S) CLASS(4) (channel initiator statistics), the traces will only be gathered when the channel initiator is active and message CSQX128I has been output.
2. For TRACE(A) CLASS(4) (channel accounting), the traces will only be gathered when the channel initiator is active and message CSQX126I has been output.

CSQW130I
'*type*' TRACE STARTED, ASSIGNED TRACE NUMBER *tno*

Explanation

In response to a command, or automatically during queue manager initialization, a *type* trace has been started and assigned the trace number *tno*. Multiple messages are possible when the start command specifies multiple user identifiers.

System action

Processing for the request continues. If the specified trace applies to the channel initiator, a request will be queued: see message CSQW152I.

CSQW131I
STOP TRACE SUCCESSFUL FOR TRACE NUMBER(S) *tno,...*

Explanation

In response to a command, the trace number(s), *tno,...*, have been stopped. Up to five trace numbers can be listed. If more than five traces have been stopped, another CSQW131I message is sent.

System action

Processing for the request continues. If the specified trace applies to the channel initiator, a request will be queued: see message CSQW152I.

CSQW132I
ALTER TRACE SUCCESSFUL FOR TRACE NUMBER *tno*

Explanation

The trace number *tno* has been altered.

System action

Processing for the request continues.

CSQW133E
csect-name TRACE DATA LOST, *dest* NOT ACCESSIBLE RC=*code*

Explanation

The destination specified stopped accepting trace data during a trace. Some external condition caused the data rejection. The reason for the error is defined by the return code (RC). The value of *code* can be:

- The hexadecimal return code from SMF. See the *MVS System Management Facilities (SMF)* manual for the specific value.
- The hexadecimal return code from the GTF request

  04 GTF trace and/or USR tracing is not active
- The hexadecimal return code from the SRV request

  10 The serviceability routine is absent

  xx The serviceability routine return code

System action

Trace processing continues, although data is lost.

System programmer response

Investigate the GTF or SMF facility to determine why data is not being accepted. You can issue a START TRACE command to record the data at another destination. The DISPLAY TRACE command shows what types of data were recorded at the specified destination.

See the *MVS System Management Facilities (SMF)* manual for an explanation of the return code value.

CSQW135I
'*type*' TRACE ALREADY ACTIVE, TRACE NUMBER *tno*

Explanation

type trace was already active with trace number *tno*.

System action

Processing for the trace already in progress will continue.

CSQW137I
SPECIFIED TRACE NOT ACTIVE

Explanation

Either:

- A command requested action for a specific trace, but that trace could not be found in the active trace table.
- A command requested action for all traces, but there are no traces active.

System action

Processing for the command continues.

System programmer response

Issue an unqualified DISPLAY TRACE command (that is, DISPLAY TRACE(*) without any other keywords) to determine all the active trace entries.

CSQW138E
IFCID *ifcid-number* IS INVALID

Explanation

The specified IFCID number is outside the range of valid IFCID numbers or is an IFCID number which is not allowed on a trace command.

System action

Processing of the trace command is terminated before any trace functions are performed.

System programmer response

See the TRACE commands in MQSC commands and Line trace for more information.

CSQW144E
CHANNEL INITIATOR NOT ACTIVE

Explanation

TRACE(CHINIT) was specified, but the channel initiator is not active.

System action

The command is not actioned.

System programmer response

Issue the START CHINIT command to start the channel initiator, and reissue the command.

CSQW149E
RMID 231 IS OBSOLETE - USE TRACE(CHINIT)

Explanation

The command specifies RMID 231, which was formerly used for channel initiator traces, but is now obsolete. For channel initiator traces, specify TRACE(CHINIT).

System action

The command is not actioned.

System programmer response

Issue the command correctly. If both queue manager and channel initiator tracing is required, issue two separate commands.

CSQW152I
TRACE REQUEST FOR CHANNEL INITIATOR QUEUED

Explanation

Initial processing for a trace command has completed successfully. The command requires further action by the channel initiator, for which a request has been queued.

System action

A request has been queued for the channel initiator. Further messages will be produced when the command has been completed.

CSQW153E
csect-name STORAGE NOT AVAILABLE FOR NEW TRACE TABLE

Explanation

There is insufficient storage in ECSA for a new global trace table as requested by a previous SET SYSTEM TRACTBL command.

System action

Processing continues using the existing global trace table.

System programmer response

Investigate how ECSA storage is being used. Issue a further SET SYSTEM TRACTBL command to set the trace table size to an acceptable value.

CSQW200E
Error during STORAGE OBTAIN macro. Return code= *rc*

Explanation

The z/OS STORAGE macro was issued to obtain storage for the trace formatter. The request failed with return code *rc*.

System action

Formatting of control blocks stops, and a hexadecimal dump of the record is produced. (This might be only part of the logical record.)

System programmer response

See the *MVS Assembler Services Reference* manual for information about *rc*. You can usually resolve this problem by increasing the size of your TSO or batch region. When the problem has been solved, retry the operation.

CSQW201E
Error during STORAGE RELEASE macro. Return code= *rc*

Explanation

The z/OS STORAGE macro was issued to release some storage. The request failed with return code *rc*.

System action

Formatting of control blocks stops, and a hexadecimal dump of the record is produced. (This might be only part of the logical record.)

System programmer response

Try processing the dump again. If the problem persists, note the value of *rc*, and contact your IBM support center.

CSQW202E
Incomplete trace record detected

Explanation

A long trace record has been segmented, and the start record for the record currently being processed has not been processed.

This usually occurs when records within a time range have been selected for processing. The record with the start of segment flag is probably before the start of the selected time interval. This can also occur if the Generalized Trace Facility (GTF) is unable to write all records to the GTF data set.

System action

A hexadecimal dump of the record is produced, and formatting continues with the next record. (You will receive this message for each subsequent part of this logical record.)

System programmer response

Select a slightly earlier start time for your time interval (one tenth of a second for example) and retry the operation. If this is not successful, it is possible that your trace table has wrapped, and the start record has been overwritten.

CSQW204E
Internal error

Explanation

An internal error has occurred.

System action

A hexadecimal dump of the record is produced, and formatting continues with the next record. This message might be followed by message CSQW202E.

System programmer response

Try processing the dump again. If the problem persists, contact your IBM support center.

CSQW205E
Internal error

Explanation

An internal error has occurred.

System action

This, and all subsequent records are displayed in hexadecimal. IBM MQ trace formatting is suppressed.

System programmer response

Try processing the dump again. If the problem persists, contact your IBM support center.

CSQW206I
Accounting record

Explanation

This message identifies this record as an accounting record.

System action

A hexadecimal dump of the record is produced, and formatting continues with the next record.

CSQW207I
A Null Self Defining section was detected

Explanation

The MQ trace formatter has detected a self-defining section of zero length.

System action

Formatting continues with the next self-defining section.

CSQW208E
Invalid address detected

Explanation

The MQ trace formatter has been passed an invalid address. The address is in low storage.

System action

Formatting of the record is suppressed. Formatting continues with the next record.

CSQW209I
A null length data item was detected

Explanation

The MQ trace formatter detected a data item of zero length.

System action

Formatting continues with the next data item.

CSQW210E
Invalid record detected

Explanation

The format of a record was different from the format expected by the IBM MQ trace formatter.

System action

A hexadecimal dump is produced, and formatting continues with the next record.

System programmer response

Try processing the dump again. If the problem persists, contact your z/OS support center.

CSQW701E
csect-name ENFREQ request failed, RC=*rc*

Explanation

A z/OS ENFREQ request failed. *rc* is the return code (in hexadecimal) from the request.

System action

Processing continues.

System programmer response

See the *MVS Authorized Assembler Services Reference* manual for information about the return code the ENFREQ request.

Distributed queuing messages (CSQX...): z/OS

CSQX000I

IBM MQ for z/OS V*n*

Severity
0

Explanation

This message is issued when the channel initiator starts, and shows the release level.

CSQX001I

csect-name Channel initiator starting

Severity
0

Explanation

The channel initiator address space is starting, in response to a START CHINIT command.

System action

Channel initiator startup processing begins. Message CSQX022I is sent when the startup process has completed.

CSQX002I

csect-name Queue-sharing group is *qsg-name*

Severity
0

Explanation

This is issued during channel initiator startup processing or in response to the DISPLAY CHINIT command if the queue manager that the channel initiator uses is in a queue-sharing group.

System action

Processing continues.

CSQX003I

csect-name Obsolete parameter module ignored

Severity
0

Explanation

The START CHINIT command specified a parameter module name using the PARM keyword. The use of a channel initiator parameter module is obsolete, so the name is ignored.

System action

Processing continues.

System programmer response

Channel initiator parameters are specified by queue manager attributes. Use the ALTER QMGR command to set the values you want.

CSQX004I
Channel initiator is using *mm* MB of local storage, *nn* MB are free

Explanation

Displays the amount of virtual storage currently used and available in the extended private region. Both values are displayed in megabytes (1048576 bytes), and are approximations.

This message is logged at channel initiator start and then either every hour if the usage does not change or when the memory usage changes (up or down) by more than 2%.

System action

Processing continues.

System programmer response

No action is required at this time. However, a frequent occurrence of this message might be an indication that the system is operating beyond the optimum region for the current configuration.

CSQX005E

csect-name Channel initiator failed to start

Severity
8

Explanation

A severe error, as reported in the preceding messages, occurred during channel initiator startup processing.

System action

The channel initiator started task terminates.

System programmer response

Investigate the problem reported in the preceding messages.

CSQX006E

csect-name Channel initiator failed while stopping

Severity

8

Explanation

A severe error, as reported in the preceding messages, occurred during channel initiator termination processing.

System action

The channel initiator started task terminates.

System programmer response

Investigate the problem reported in the preceding messages.

CSQX007E

csect-name Unable to connect to queue manager *qmgr-name*, MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity

8

Explanation

An attempt by the channel initiator to connect to the queue manager was unsuccessful.

System action

If the error occurred during the channel initiator startup procedure, the channel initiator does not start. In other cases, the component where the error occurred (message channel agent, dispatcher, adapter subtask, SSL server subtask, repository manager, or listener) does not start and the function it provides is unavailable; in most cases, the end result is that the channel initiator terminates.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

If you are unable to solve the problem, contact your IBM support center.

CSQX008E

csect-name Unable to disconnect from queue manager *qmgr-name*, MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity

4

Explanation

An attempt by the channel initiator to disconnect from the queue manager was unsuccessful.

System action

Processing continues.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

If you are unable to solve the problem, contact your IBM support center.

CSQX009I

csect-name Channel initiator stopping

Severity
8

Explanation

A severe error, as reported in the preceding messages, occurred during channel initiator processing; the channel initiator is unable to continue.

System action

The channel initiator terminates.

System programmer response

Investigate the problem reported in the preceding messages.

CSQX010I

csect-name Channel initiator stopped

Severity
0

Explanation

The channel initiator terminated following an error, as reported in the preceding messages.

System action

None.

CSQX011I

csect-name Client attachment available

Severity
0

Explanation

Clients can be attached to and MQI channels can be used with the channel initiator.

System action

The channel initiator startup processing continues.

CSQX012E

csect-name Unable to open *ddname* data set

Severity
4

Explanation

The *ddname* data set could not be opened, as reported in the preceding messages.

System action

Processing continues, but functions that require the data set will be inhibited. For example, if the exit library data set CSQXLIB cannot be opened, user channel and channel auto-definition exits will not be available, and channels that use them will not start. If the error information data set CSQSNAP cannot be opened, the error information will be lost.

System programmer response

Investigate the problem reported in the preceding messages.

CSQX013I

csect-name Address conflict for listener, port *port* address *ip-address*, TRPTYPE=TCP INDISP=*disposition*

Severity

4

Explanation

A STOP LISTENER or START LISTENER command was issued specifying TRPTYPE(*trptype*) and INDISP(*disposition*), but that listener was already active for a port and IP address combination that conflicted with the requested port and IP address. If *ip-address* is '*', all IP addresses were requested.

The port and IP address combination specified must match a combination for which the listener is active. It cannot be a superset or a subset of that combination.

System action

None.

System programmer response

Reissue the command correctly if necessary.

CSQX014E

csect-name Listener exceeded channel limit, TRPTYPE=*trptype* INDISP=*disposition*

Severity

8

Explanation

The number of current channels using the indicated communications system *trptype* is the maximum allowed. The listener cannot accept an incoming request to start another channel; if the maximum is 0, the listener itself cannot start. (The name of the channel requested cannot be determined because the listener could not accept the request.) Current channels include stopped and retrying channels as well as active channels.

disposition shows which type of incoming requests the listener was handling:

QMGR
those directed to the target queue manager

GROUP
those directed to the queue-sharing group.

The maximum allowed is specified in the TCPCHL or LU62CHL queue manager attribute, but may be reduced if a dispatcher fails, or if TCP/IP resources are restricted (as reported by message CSQX118I).

System action

The channel or listener does not start.

System programmer response

If the maximum allowed is zero, communications using the indicated system *trptype* are not allowed, and no such channels can be started. The listener also cannot be started. If the maximum allowed is non-zero, wait for some of the operating channels to terminate before restarting the remote channel, or use the ALTER QMGR command to increase TCPCHL or LU62CHL.

CSQX015I

csect-name started dispatchers started, *failed* failed

Severity
0

Explanation

The channel initiator startup procedure has started the requested number of dispatchers; *started* dispatchers started successfully and *failed* dispatchers did not start.

System action

The channel initiator startup processing continues. The number of current TCP/IP and LU 6.2 channels allowed will be reduced proportionately if some dispatchers did not start.

System programmer response

If the message indicates that some dispatchers failed, investigate the problem reported in the preceding messages.

CSQX016I

csect-name Listener already started, TRPTYPE=*trptype* INDISP=*disposition*

Severity
0

Explanation

A START LISTENER command was issued specifying TRPTYPE(*trptype*) and INDISP(*disposition*), but that listener was already active.

System action

None.

CSQX017I

csect-name Listener already started, port *port* address *ip-address*, TRPTYPE=TCP TRPTYPE=TCP INDISP=*disposition*

Severity
0

Explanation

A START LISTENER command was issued specifying TRPTYPE(TCP) and INDISP(*disposition*), but that listener was already active for the requested port and IP address. If *ip-address* is '*', all IP addresses were requested.

System action

None.

CSQX018I

csect-name Listener already stopped or stopping, TRPTYPE=*trptype* INDISP=*disposition*

Severity
0

Explanation

A STOP LISTENER or START LISTENER command was issued specifying TRPTYPE(*trptype*) and INDISP(*disposition*), but that listener was already stopped or in the process of stopping.

System action

None.

CSQX019I

csect-name Listener already stopped or stopping, port *port* address *ip-address*, TRPTYPE=TCP INDISP=*disposition*

Severity

0

Explanation

A STOP LISTENER or START LISTENER command was issued specifying TRPTYPE(*trptype*) and INDISP(*disposition*), but that listener was already stopped or in the process of stopping for the requested port and IP address. If *ip-address* is '*', all IP addresses were requested.

System action

None.

CSQX020I

csect-name Shared channel recovery completed

Severity

0

Explanation

The channel initiator startup procedure has successfully completed the shared channel recovery process, for channels that were owned by itself and for channels that were owned by other queue managers.

System action

Processing continues.

System programmer response

See message CSQM052I issued by the queue manager for more details.

CSQX021E

csect-name Shared channel recovery error

Severity

0

Explanation

The channel initiator startup procedure did not complete the shared channel recovery process, because an error occurred.

System action

The recovery process is terminated; some channels might have been recovered, while others have not.

System programmer response

See the error messages (such as CSQM053E) issued by the queue manager for more details. When the problem has been resolved, either start any unrecovered channels manually, or restart the channel initiator.

CSQX022I

csect-name Channel initiator initialization complete

Severity

0

Explanation

Initialization of the channel initiator completed normally, and the channel initiator is ready for use. Note, however, that processing of the CSQINPX command data set might still be in progress; its completion is shown by message CSQU012I.

System action

None.

CSQX023I

csect-name Listener started, port *port* address *ip-address* TRPTYPE=*trptype* INDISP=*disposition*

Severity

0

Explanation

A listener has been started specifying TRPTYPE(*trptype*) and INDISP(*disposition*). This could either be because a START LISTENER command was issued, or because the listener was retrying. That listener is now active for the requested port and IP address. If *ip-address* is *, all IP addresses were requested.

System action

None.

CSQX024I

csect-name Listener stopped, port *port* address *ip-address* TRPTYPE=*trptype* INDISP=*disposition*

Severity

0

Explanation

A STOP LISTENER command was issued specifying TRPTYPE(*trptype*) and INDISP(*disposition*), or IBM MQ has tried to stop a listener because of a failure. That listener is no longer active for the requested port and IP address. If *ip-address* is *, all IP addresses were requested.

System action

None.

CSQX026E

csect-name Unable to locate the trace header, RC=12

Severity

8

Explanation

The trace formatting routine was unable to locate the trace control information in the trace data space in a dump of the channel initiator address space.

System action

Formatting terminates.

System programmer response

The most likely cause is that the dump has not been produced correctly. Re-create the dump, and try again.

CSQX027E

csect-name Unable to get storage, RC=*return-code*

Severity

8

Explanation

An attempt to obtain some storage failed. *return-code* is the return code (in hexadecimal) from the z/OS STORAGE service.

System action

The component where the error occurred (message channel agent, dispatcher, adapter subtask, SSL server subtask, listener, repository manager, supervisor, or trace formatter) usually terminates; in many cases, the end result will be that the channel initiator terminates.

System programmer response

For information about the return code from the STORAGE request, see STORAGE - Obtain and release storage.

CSQX028E

csect-name Unable to free storage, RC=*return-code*

Severity

8

Explanation

An attempt to release some storage failed. *return-code* is the return code (in hexadecimal) from the z/OS STORAGE service.

System action

The component where the error occurred (message channel agent, dispatcher, adapter subtask, SSL server subtask, repository manager, or listener) usually ignores the error and continues processing.

System programmer response

See the *MVS Programming: Assembler Services Reference* manual for information about the return code from the STORAGE request.

CSQX029I

csect-name Queue manager *qmgr-name* stopping, MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity

0

Explanation

In response to an MQ API call, the queue manager notified the channel initiator that it is stopping.

System action

The channel initiator terminates.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQX030I

csect-name '*type*' trace started, assigned trace number *tno*

Explanation

During channel initiator initialization, a *type* trace has been started automatically and assigned the trace number *tno*.

System action

Processing continues.

CSQX031E

csect-name Initialization command handler ended abnormally, reason=*00sssuuu*

Severity

8

Explanation

The initialization command handler, which processes the CSQINPX command data set, is ending abnormally. *sss* is the system completion code, and *uuu* is the user completion code (both in hexadecimal).

System action

The initialization command handler ends abnormally, but the channel initiator continues.

System programmer response

If a system completion code is shown, see the *MVS System Codes* manual for information about the problem; the message will normally be preceded by other messages giving additional information.

The most likely cause is erroneous definition of the CSQINPX and CSQOUTX data sets. For information about the initialization command handler and these data sets, see Initialization commands. If you are unable to solve the problem, contact your IBM support center.

CSQX032I

csect-name Initialization command handler terminated

Severity

4

Explanation

The initialization command handler, which processes the CSQINPX command data set, was terminated before completing all the commands because the channel initiator is stopping, and so cannot process any more commands.

System action

The initialization command handler ends.

System programmer response

Refer to the CSQOUTX data set for information about the commands that were processed. If the channel initiator is not stopping because of a STOP command, refer to the preceding messages for information about the problem causing it to stop.

For information about the initialization command handler, see Initialization commands.

CSQX033E

csect-name Channel initiator stopping because of errors

Severity

8

Explanation

A severe error, as reported in the preceding messages, occurred during channel initiator processing; the channel initiator is unable to continue.

System action

The channel initiator terminates.

System programmer response

Investigate the problem reported in the preceding messages.

CSQX034I

csect-name Channel initiator stopping because queue manager is stopping

Severity

0

Explanation

The queue manager notified the channel initiator that it is stopping.

System action

The channel initiator terminates.

CSQX035I

csect-name Connection to queue manager *qmgr-name* stopping or broken, MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity

0

Explanation

In response to an MQ API call, the channel initiator found that its connection to the queue manager was no longer available.

System action

The channel initiator terminates.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQX036E

csect-name Unable to open *object-type*(*name*), MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity

8

Explanation

An MQOPEN call for *name* was unsuccessful; *object-type* indicates whether *name* is a queue name, queue manager name, namelist name, channel name, topic name, or authentication information name. (The channel initiator can access channel definitions and authentication information as objects using the MQ API.)

System action

The component where the error occurred (message channel agent, dispatcher, adapter subtask, SSL server subtask, repository manager, listener, or supervisor) terminates. In the case of a message channel agent, the associated channel will be stopped.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

The most common cause of the problem will be that the channel and queue definitions are incorrect.

CSQX037E

csect-name Unable to get message from *name*, MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity
8

Explanation

An MQGET call for queue *name* was unsuccessful.

System action

The component where the error occurred (message channel agent, dispatcher, adapter subtask, SSL server subtask, repository manager, listener, or supervisor) terminates. In the case of a message channel agent, the associated channel will be stopped.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQX038E

csect-name Unable to put message to *name*, MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity
8

Explanation

An MQPUT call for queue *name* was unsuccessful.

System action

The component where the error occurred (message channel agent, dispatcher, adapter subtask, SSL server subtask, repository manager, listener, or supervisor) terminates. In the case of a message channel agent, the associated channel will be stopped.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQX039E

csect-name Unable to close *name*, MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity
4

Explanation

An MQCLOSE call for *name* was unsuccessful; *name* can be a queue name, queue manager name, namelist name, channel name, or authentication information name. (The channel initiator can access channel definitions and authentication information as objects using the IBM MQ API.)

System action

Processing continues.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQX040E

csect-name Unable to inquire attributes for *name*, MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity
8

Explanation

An MQINQ call for *name* was unsuccessful; *name* may be a queue name, queue manager name, namelist name, channel name, or authentication information name. (The channel initiator can access channel definitions and authentication information as objects using the MQ API.)

System action

The component where the error occurred (message channel agent, dispatcher, adapter subtask, SSL server subtask, repository manager, listener, or supervisor) terminates. In the case of a message channel agent, the associated channel will be stopped.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQX041E

csect-name Unable to set attributes for *name*, MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity

8

Explanation

An MQSET call for queue *name* was unsuccessful.

System action

The component where the error occurred (message channel agent, dispatcher, adapter subtask, SSL server subtask, listener, or supervisor) terminates. In the case of a message channel agent, the associated channel will be stopped.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQX042E

csect-name Unable to define *comp* to CTRACE, RC=*rc* reason=*reason*

Severity

8

Explanation

The CTRACE component definitions (for component *comp*) required by the channel initiator could not be defined. *rc* is the return code and *reason* is the reason code (both in hexadecimal) from the z/OS CTRACE service.

System action

The channel initiator does not start.

System programmer response

See the *MVS Authorized Assembler Services Reference* manual for information about the return and reason codes from the CTRACE request. If you are unable to solve the problem, contact your IBM support center.

CSQX043E

csect-name Unable to delete *comp* from CTRACE, RC=*rc* reason=*reason*

Severity

4

Explanation

The CTRACE component definitions (for component *comp*) used by the channel initiator could not be deleted. *rc* is the return code and *reason* is the reason code (both in hexadecimal) from the z/OS CTRACE service.

System action

Channel initiator termination processing continues.

System programmer response

See the *MVS Authorized Assembler Services Reference* manual for information about the return and reason codes from the CTRACE request. If you are unable to solve the problem, contact your IBM support center.

CSQX044E

csect-name Unable to initialize PC routines, RC=*rc* reason=*reason*

Severity

8

Explanation

The PC routines required by the channel initiator could not be defined. The reason code *reason* shows which z/OS service failed:

00E74007
LXRES failed

00E74008
ETCRE failed

00E74009
ETCON failed

rc is the return code (in hexadecimal) from the indicated z/OS service.

System action

The channel initiator does not start.

System programmer response

See the *MVS Authorized Assembler Services Reference* manual for information about the return codes from the z/OS services. If you are unable to solve the problem, contact your IBM support center.

CSQX045E

csect-name Unable to load *module-name*, reason=*ssssrrrr*

Explanation

The channel initiator was unable to load a required module. *ssss* is the completion code and *rrrr* is the reason code (both in hexadecimal) from the z/OS LOAD service.

System action

The component where the error occurred (message channel agent, dispatcher, adapter subtask, SSL server subtask, repository manager, or listener) does not start and the function it provides is unavailable; in many cases, the end result is that the channel initiator terminates.

System programmer response

Check the console for messages indicating why the module was not loaded. See the *MVS Programming: Assembler Services Reference* manual for information about the codes from the LOAD request.

Ensure that the module is in the required library, and that it is referenced correctly. The channel initiator attempts to load this module from the library data sets under the STEPLIB DD statement of its started task JCL procedure xxxxCHIN.

CSQX046E

csect-name Unable to initialize data conversion services, reason=*reason*

Severity

8

Explanation

The data conversion services required by the channel initiator could not be initialized. The reason code *reason* shows why:

00C10002
Unable to load modules

00C10003
Insufficient storage

other Internal error

System action

The channel initiator does not start.

System programmer response

Check the console for messages indicating that a module was not loaded. Ensure that the module is in the required library, and that it is referenced correctly. The channel initiator attempts to load this module from the library data sets under the STEPLIB DD statement of its started task JCL procedure xxxxCHIN.

If you are unable to solve the problem, contact your IBM support center.

CSQX047E

csect-name Unable to commit messages for *name*, MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity

8

Explanation

An MQCMIT call involving messages for queue *name* was unsuccessful.

System action

The component where the error occurred (supervisor) terminates.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQX048I

csect-name Unable to convert message for *name*, MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity

0

Explanation

A message being put to an IMS bridge queue *name* required data conversion, but the conversion was not successful.

System action

The message is put without conversion, and processing continues.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQX049E

csect-name Unable to retrieve token for name *name*, RC=*rc*

Severity

8

Explanation

A token in a name/token pair required by the channel initiator could not be retrieved. *rc* is the return code (in hexadecimal) from the z/OS IEANTRT service.

System action

The channel initiator does not start.

System programmer response

See the *MVS Authorized Assembler Services Reference* manual for information about the return code from the IEANTRT request. If you are unable to solve the problem, contact your IBM support center.

CSQX050E

csect-name Unable to create access list for queue manager, RC=*rc*

Severity

8

Explanation

The channel initiator could not create the necessary storage access list for the queue manager to use. *rc* is the return code (in hexadecimal) from the z/OS ALESERV service.

System action

The channel initiator does not start.

System programmer response

See the *MVS Authorized Assembler Services Reference* manual for information about the return code from the ALESERV request. If you are unable to solve the problem, contact your IBM support center.

CSQX051E

csect-name Unable to share storage with the queue manager, RC=*rc*

Severity

8

Explanation

A request by the channel initiator to allow the queue manager to share some storage failed. *rc* is the return code (in hexadecimal) from the z/OS IARVSERV service.

System action

The channel initiator does not start.

System programmer response

See the *MVS Authorized Assembler Services Reference* manual for information about the return code from the IARVSERV request. If you are unable to solve the problem, contact your IBM support center.

CSQX052E

csect-name Timer task attach failed, RC=*return-code*

Severity
8

Explanation

The repository manager task could not be attached. *return-code* is the return code (in hexadecimal) from the z/OS ATTACH service.

System action

The channel initiator terminates.

System programmer response

See the *MVS Programming: Assembler Services Reference* manual for information about the return code from the ATTACH request. If you are unable to solve the problem, contact your IBM support center.

CSQX053E

csect-name Error information recorded in CSQSNAP data set

Severity
8

Explanation

An internal error has occurred. Information about the error is written to the data set identified by the CSQSNAP DD statement of the channel initiator started task JCL procedure, xxxxCHIN.

System action

Processing continues.

System programmer response

Collect the items listed in the Problem Determination section and contact your IBM support center.

CSQX054E

csect-name Repository manager ended abnormally, reason=*sssuuu-reason*

Severity
8

Explanation

The repository manager is ending abnormally because an error that cannot be corrected has occurred. *sss* is the system completion code, *uuu* is the user completion code, and *reason* is the associated reason code (all in hexadecimal).

System action

The repository manager ends abnormally, and a dump is normally issued. The channel initiator will attempt to restart it.

System programmer response

User completion codes are generally the result of errors detected by the Language Environment; see the *Language Environment for z/OS Debugging Guide and Runtime Messages* for information about these codes. Otherwise, contact your IBM support center to report the problem.

CSQX055E

csect-name Repository manager attach failed, RC=*return-code*

Severity

8

Explanation

The repository manager task could not be attached. *return-code* is the return code (in hexadecimal) from the z/OS ATTACH service.

System action

The channel initiator terminates.

System programmer response

See the *MVS Programming: Assembler Services Reference* manual for information about the return code from the ATTACH request. If you are unable to solve the problem, contact your IBM support center.

CSQX056E

csect-name Preinitialization services request failed, function code=*func*, RC=*rc*

Severity

8

Explanation

A preinitialization services (CEEPIPI) call failed. *func* is the function code used (in decimal) and *rc* is the return code (in hexadecimal) from the call.

System action

The component where the error occurred (message channel agent or SSL server subtask) terminates. In the case of a message channel agent, the associated channel will be stopped.

System programmer response

See the *Language Environment for z/OS & VM Programming Guide* for information about the return code from the CEEPIPI call. If you are unable to solve the problem, contact your IBM support center.

CSQX057E

csect-name Cluster cache task attach failed, RC=*return-code*

Severity

8

Explanation

The channel initiator cluster cache task could not be attached. *return-code* is the return code (in hexadecimal) from the z/OS ATTACH service.

System action

The channel initiator terminates.

System programmer response

See the *MVS Programming: Assembler Services Reference* manual for information about the return code from the ATTACH request. If you are unable to solve the problem, contact your IBM support center.

CSQX058E

csect-name Pause service *service-name* failed, RC=*return-code*

Severity
8

Explanation

An error occurred processing a pause element. *return-code* is the return code (in hexadecimal) from the z/OS pause service *service-name*.

System action

The component where the error occurred (message channel agent, repository manager, cluster cache extension task,) usually terminates; in many cases, the end result will be that the channel initiator terminates.

System programmer response

See the *MVS Programming: Assembler Services Reference* manual for information about the return code from the request. If you are unable to solve the problem, contact your IBM support center.

CSQX059E

csect-name Unable to increase cluster cache

Severity
8

Explanation

The dynamic cluster cache cannot be increased because the channel initiator cluster cache task encountered an error.

System action

The channel initiator terminates.

System programmer response

Investigate the problem reported in any preceding messages.

CSQX060E

csect-name Queued Pub/Sub task attach failed, RC=*reason-code*

Severity
8

Explanation

The queued Publish/Subscribe task could not be attached. The *return-code* is the return code (in hexadecimal) from the z/OS ATTACH service.

System action

The channel initiator terminates.

System programmer response

See the *MVS Programming: Assembler Services Reference* manual for information about the return code from the ATTACH request. If you are unable to solve the problem, contact your IBM support center.

CSQX061E

csect-name Distributed Pub/Sub Offloader task attach failed, RC=*return-code*

Severity
8

Explanation

The Distributed Pub/Sub Offloader task could not be attached. *Return-code* is the return code (in hexadecimal) from the z/OS ATTACH service.

System action

The channel initiator terminates.

System programmer response

See the *MVS Programming: Assembler Services Reference* manual for information about the return code from the ATTACH request. If you are unable to solve this problem, contact your IBM support center.

CSQX062E

csect-name Distributed Pub/Sub tasks have insufficient command authority

Severity
8

Explanation

The PSMODE queue manager attribute has a value other than DISABLED but the channel initiator has insufficient authority to issue the DISPLAY PUBSUB command. Until such authority is granted, distributed publish/subscribe is unavailable.

System action

The channel initiator attempts to restart the distributed Pub/Sub tasks at 1 minute intervals. This message is issued on each subsequent attempt until the required authority has been granted or publish/subscribe is disabled.

System programmer response

Grant the channel initiator the required authority to access the command server queues and issue the DISPLAY PUBSUB command. For the required security definitions, see Security considerations for the channel initiator on z/OS. Alternatively, if no publish subscribe operation is required, setting the PSMODE queue manager attribute to DISABLED prevents this message from being issued.

CSQX063I

csect-name Distributed Pub/Sub Offloader started

Severity
0

Explanation

The Distributed Pub/Sub Offloader task has started successfully.

System programmer response

None

CSQX064I

csect-name Distributed Pub/Sub Offloader stopped

Severity
0

Explanation

The Distributed Pub/Sub command Offloader task has stopped. This can be for one of three reasons:

- The channel initiator is stopping.
- The channel initiator is starting and the queues used by the distributed pub/sub offloader have not been defined because distributed pub/sub command processing is not required.
- An error has occurred.

System action

Processing continues, but distributed pub/sub is not available.

System programmer response

If an error has occurred, investigate the problem reported in the preceding messages.

CSQX065E

csect-name Unexpected error in distributed pub/sub Offloader

Severity

8

Explanation

The Distributed Pub/Sub command Offloader encountered an unexpected error

System action

Distributed publish/subscribe might no longer be available.

System programmer response

Investigate the problem reported in the preceding messages. If there are none or this does not resolve the problem contact IBM support.

CSQX066E

csect-name Refresh proxy subscriptions failed

Severity

8

Explanation

A REFRESH QMGR TYPE(PROXYSUB) was issued, but could not complete. This could be because the Channel Initiator is shutting down, or as a result of an error.

System action

Processing continues, but remote subscriptions are not resynchronized.

System programmer response

If an error has occurred, investigate the problem reported in the preceding messages.

CSQX067E

csect-name Error removing non durable remote subscriptions

Severity

8

Explanation

The Pub/Sub Offloader task is ending but was unable to remove one or more remote proxy subscriptions. If no previous error has occurred, this is likely to have been triggered by Queue Manager shut down.

System action

Processing continues, but remote subscriptions might continue to exist which are no longer valid. This could cause a build-up of publications for this Queue Manager on remote transmission queues.

System programmer response

If the Queue Manager is to be restarted immediately, these subscriptions will be cleaned up when initial resynchronization with the cluster occurs. If this is not the case, proxy subscriptions might need to be manually removed using DELETE SUB on other Queue Managers in the cluster. Investigate the problem reported in the preceding messages to see why resynchronization failed.

CSQX068I

csect-name Channel initiator has scavenged *mm* MB of transmission buffers

Explanation

Displays the amount of virtual storage that has been freed by the channel initiator transmission buffer scavenger task. This virtual storage value is displayed in megabytes (1048576 bytes), and is an approximation.

This message is logged when the amount of virtual storage used by the channel initiator is more than 75%. If storage has been freed the CSQX004I message is issued.

System action

Processing continues.

System programmer response

No action is required at this time. However, a frequent occurrence of this message might indicate the system is operating beyond the optimum region for the current configuration.

CSQX069E

csect-name Distributed Pub/Sub Offloader ended abnormally, reason=*sssuuu-reason*

Severity

8

Explanation

The Distributed Pub/Sub Offloader task is ending abnormally because an error that cannot be corrected has occurred. *sss* is the system completion code, *uuu* is the user completion code, and *reason* is the associated reason code (all in hexadecimal).

System action

The Distributed Pub/Sub Offloader task ends abnormally, and a dump is normally issued. Distributed publish/subscribe is no longer available.

System programmer response

User completion codes are generally the result of errors detected by the Language Environment; see the *Language Environment for z/OS Debugging Guide and Runtime Messages* for information about these codes. Otherwise, contact your IBM support center to report the problem.

CSQX070I

csect-name CHINIT parameters ...

Severity

0

Explanation

The channel initiator is being started with the parameter values shown in the following messages: CSQX071I, CSQX072I, CSQX073I, CSQX074I, CSQX075I, CSQX076I, CSQX078I, CSQX079I, CSQX080I, CSQX081I, CSQX082I, CSQX085I, CSQX090I, CSQX091I, CSQX092I, CSQX094I, CSQX099I.

System action

The channel initiator startup processing continues.

System programmer response

Channel initiator parameters are specified by queue manager attributes. Use the ALTER QMGR command to set the values you want.

CSQX093I

csect-name WLM/DNS is no longer supported

Severity
4

Explanation

The QMGR attribute DNSWLM is set to YES. This feature is no longer supported by z/OS Communications Server.

System action

Processing continues, but registration to the WLM/DNS server will not be attempted.

System programmer response

Issue the command

```
ALTER QMGR DNSWLM(NO)
```

and consider using Sysplex Distributor instead. See Establishing a TCP connection Using Sysplex Distributor.

CSQX100E

csect-name Dispatcher failed to start, TCB=*tcb-name*

Severity
8

Explanation

A severe error, as reported in the preceding messages, occurred during dispatcher startup processing.

System action

The channel initiator will attempt to restart the dispatcher. The number of current TCP/IP and LU 6.2 channels allowed will be reduced proportionately.

System programmer response

Investigate the problem reported in the preceding messages.

CSQX101E

csect-name Dispatcher unable to schedule essential process *process*

Severity
8

Explanation

During dispatcher startup processing, one of the essential dispatcher processes (named *process*) could not be scheduled.

System action

The dispatcher does not start.

System programmer response

The most likely cause is insufficient storage. If increasing the available storage does not solve the problem, contact your IBM support center.

CSQX102E

csect-name Dispatcher linkage stack error, TCB=*tcb-name*

Severity
8

Explanation

The dispatcher using TCB *tcb-name* found an inconsistency in the linkage stack.

System action

The dispatcher ends abnormally with completion code X'5C6' and reason code X'00E7010E', and a dump is issued. The channel initiator will attempt to restart it.

System programmer response

The most likely cause is incorrect use of the linkage stack by a user channel exit; exits must issue any MQ API calls and return to the caller at the same linkage stack level as they were entered. If exits are not being used, or if they do not use the linkage stack, contact your IBM support center to report the problem.

CSQX103E

csect-name Dispatcher unexpected error, TCB=*tcb-name* RC=*return-code*

Severity
8

Explanation

The dispatcher using TCB *tcb-name* had an internal error.

System action

The dispatcher ends abnormally with completion code X'5C6' and reason code X'00E7010F', and a dump is issued. The channel initiator will attempt to restart it.

System programmer response

Contact your IBM support center to report the problem.

CSQX104E

csect-name Unable to establish ESTAE, RC=*return-code*

Severity
8

Explanation

During startup processing, the recovery environment could not be set up. *return-code* is the return code (in hexadecimal) from the z/OS ESTAE service.

System action

The component that was starting (dispatcher, adapter subtask, SSL server subtask, supervisor, repository manager, or channel initiator itself) does not start.

System programmer response

See the *MVS Programming: Assembler Services Reference* manual for information about the return code from the ESTAE request. If you are unable to solve the problem, contact your IBM support center.

CSQX106E

csect-name Unable to connect to TCP/IP using USS, service '*serv*' RC=*return-code* reason=*reason*

Severity

4

Explanation

Use of TCP/IP with the UNIX System Services (USS) sockets interface was requested, but an error occurred. *return-code* and *reason* are the return and reason codes (both in hexadecimal) from the USS service *serv* that gave the error.

The most likely causes are:

- The user ID that the channel initiator uses is not set up correctly for use with USS. For example, it may not have a valid OMVS segment defined or its security profile may be incomplete.
- The TCPNAME queue manager attribute does not specify a valid TCP/IP stack name. These stack names are defined in the SUBFILESYSTYPE NAME parameter in member BPXPRMxx for SYS1.PARMLIB.
- The MAXFILEPROC or MAXPROCUSER parameter in member BPXPRMxx for SYS1.PARMLIB is too small.

System action

Processing continues, but communications using TCP/IP with the USS sockets interface will not be available.

System programmer response

See the *z/OS UNIX System Services Messages and Codes* manual for information about the codes from the service request.

CSQX110E

csect-name User data conversion exit error, TCB=*tcb-name* reason=*sssuuu-reason*

Severity

8

Explanation

A process for the dispatcher using TCB *tcb-name* is ending abnormally because an error that cannot be corrected has occurred in a user data conversion exit. *sss* is the system completion code, *uuu* is the user completion code, and *reason* is the associated reason code (all in hexadecimal).

System action

The process ends abnormally, and a dump is normally issued. The channel is stopped, and must be restarted manually.

System programmer response

User completion codes are generally the result of errors detected by the Language Environment; see the *Language Environment for z/OS Debugging Guide and Runtime Messages* for information about these codes. If a system completion code is shown, see the *MVS System Codes* manual for information about the problem in your exit.

CSQX111E

csect-name User channel exit error, TCB=*tcb-name* reason=*sssuuu-reason*

Severity

8

Explanation

A process for the dispatcher using TCB *tcb-name* is ending abnormally because an error that cannot be corrected has occurred in a user channel exit. *sss* is the system completion code, *uuu* is the user completion code, and *reason* is the associated reason code (all in hexadecimal).

System action

The process ends abnormally, and a dump is normally issued. The channel is stopped, and must be restarted manually. For auto-defined channels, the channel does not start.

System programmer response

User completion codes are generally the result of errors detected by the Language Environment; see the *Language Environment for z/OS Debugging Guide and Runtime Messages* for information about these codes. If a system completion code is shown, see the *MVS System Codes* manual for information about the problem in your exit.

CSQX112E

csect-name Dispatcher process error, TCB=*tcb-name* reason=*sssuuu-reason*

Severity

8

Explanation

A process run by the dispatcher using TCB *tcb-name* is ending abnormally because an error that cannot be corrected has occurred. *sss* is the system completion code, *uuu* is the user completion code, and *reason* is the associated reason code (all in hexadecimal).

System action

The process ends abnormally, and a dump is normally issued. If the process is a message channel agent, the channel is stopped, and will need to be restarted manually.

System programmer response

User completion codes are generally the result of errors detected by the Language Environment; see the *Language Environment for z/OS Debugging Guide and Runtime Messages* for information about these codes. If a system completion code is shown, and you are using user channel exits, check that your exit is setting its parameter lists correctly; otherwise, contact your IBM support center.

CSQX113E

csect-name Dispatcher ended abnormally, TCB=*tcb-name* reason=*sssuuu-reason*

Severity

8

Explanation

The dispatcher using TCB *tcb-name* is ending abnormally because an error that cannot be corrected has occurred. *sss* is the system completion code, *uuu* is the user completion code, and *reason* is the associated reason code (all in hexadecimal).

System action

The dispatcher ends abnormally, and a dump is normally issued. The channel initiator will attempt to restart it.

System programmer response

User completion codes are generally the result of errors detected by the Language Environment; see the *Language Environment for z/OS Debugging Guide and Runtime Messages* for information about these codes. Otherwise, contact your IBM support center.

CSQX114E

csect-name Dispatcher failed, reason=*reason*

Severity

8

Explanation

A dispatcher ended abnormally, as reported in the preceding messages, and could not be restarted. *reason* shows the type of failure:

0000000A
Startup error

0000000B
Linkage stack error

0000000D
Uncorrectable error

other Completion code in the form 00*sssuuu*, where *sss* is the system completion code and *uuu* is the user completion code (both in hexadecimal).

System action

The channel initiator will attempt to restart the dispatcher. The number of current TCP/IP and LU 6.2 channels allowed will be reduced proportionately.

System programmer response

Investigate the problem reported in the preceding messages.

CSQX115E

csect-name Dispatcher not restarted - too many failures

Severity

8

Explanation

A dispatcher failed; because it had already failed too many times, the channel initiator did not attempt to restart it.

System action

The dispatcher is not restarted. The number of current TCP/IP and LU 6.2 channels allowed is reduced proportionately, and other processing capacity might be reduced.

System programmer response

Investigate the problems causing the dispatcher failures.

CSQX116I

csect-name Dispatcher restarted, *number* dispatchers active

Severity

0

Explanation

A dispatcher failed, but was successfully restarted by the channel initiator. *number* dispatchers are now active.

System action

Processing continues. The number of current TCP/IP and LU 6.2 channels allowed will be increased proportionately.

CSQX117I

csect-name Outgoing shared channels are restricted from starting for TCP communication

Severity

0

Explanation

A CHISERVP() service parm flag has been set which restricts the ability for this queue manager from being able to start an outgoing shared TCP channel. For more details on this flag contact IBM support. .

System action

Processing continues. This queue manager is unable to start outgoing shared TCP channels, and will not be selected during IBM MQ workload balanced start of a shared channel. This restriction persists until the flag is disabled and the channel initiator is restarted.

CSQX118I

csect-name TCP/IP channel limit reduced to *nn*

Severity

0

Explanation

This is issued during channel initiator startup processing and in response to the DISPLAY CHINIT command if the maximum number of current TCP/IP channels allowed is less than is specified in the TCPCHL queue manager attribute. This error can occur because:

- TCP/IP resources are restricted. The UNIX Systems Services MAXFILEPROC parameter (specified in the BPXPRMxx member of SYS1.PARMLIB) controls how many sockets each task is allowed: that is, how many channels each dispatcher is allowed
- Some dispatchers have failed and not been restarted; the number of current TCP/IP channels allowed is reduced proportionately

System programmer response

If TCP/IP resources are restricted, consider increasing either the UNIX Systems Services MAXFILEPROC parameter or the number of dispatchers if you need more current TCP/IP channels.

CSQX119I

csect-name LU 6.2 channel limit reduced to *nn*

Severity

0

Explanation

This is issued during channel initiator startup processing and in response to the DISPLAY CHINIT command if the maximum number of current LU 6.2 channels allowed is less than is specified in the LU62CHL queue manager attribute. This can occur because some dispatchers have failed and not been restarted; the number of current LU 6.2 channels allowed will be reduced proportionately.

CSQX120I

csect-name Shared channel recovery started for channels owned by this queue manager

Severity

0

Explanation

The channel initiator startup procedure is starting the shared channel recovery process, for channels that are owned by itself.

System action

Processing continues

System programmer response

See message CSQM052I issued by the queue manager for more details.

CSQX121I

csect-name Shared channel recovery started for channels owned by other queue managers in the same QSG

Severity

0

Explanation

The channel initiator startup procedure is starting the shared channel recovery process, for channels that are owned by other queue managers.

System action

Processing continues

System programmer response

See message CSQM052I issued by the queue manager for more details.

CSQX122E

csect-name Failed to process channel accounting, RC=*retcode*

Severity

8

Explanation

The channel initiator SMF task encountered an error processing channel accounting data. *retcode* contains the hexadecimal return code.

System action

Processing continues.

System programmer response

Contact your IBM support center.

CSQX123E

csect-name Failed to process channel initiator statistics, RC=*retcode*

Severity
8

Explanation

The channel initiator SMF task encountered an error processing channel initiator statistics data. *retcode* contains the hexadecimal return code.

System action

Processing continues.

System programmer response

Contact your IBM support center.

CSQX124E

csect-name SMF task ended abnormally, RC=*retcode*, reason=*reason*

Severity
8

Explanation

The channel initiator SMF task ended abnormally. Possible values for *reason* are:

C59592
The channel initiator failed to notify the SMF task to shutdown. *retcode* is the return code from the z/OS IEAVRLS service.

C59593
The SMF task encountered an error entering, or resuming from, the paused state. *retcode* is the return code from the z/OS IEAVPSE service.

C59594
During initialization of the SMF task an error occurred obtaining a pause element token (PET). *retcode* is the return code from the z/OS IEAVAPE service.

C59595
During initialization of the SMF task an error occurred obtaining storage.

System action

The channel initiator attempts to reattach the SMF task, unless the error occurred during:

- Channel initiator shutdown
- Obtaining storage (reason C59595)

System programmer response

For reason C59595, check MEMLIMIT for the channel initiator, or refer to the 256MB recommended limit.

For the other reasons, contact your IBM support center.

CSQX126I

csect-name Channel accounting collection started

Severity
0

Explanation

The channel initiator has started collecting channel accounting data.

System action

Channel accounting data for channels with STATCHL (HIGH | MED | LOW) is collected and written to the System Management Facility (SMF).

CSQX127I

csect-name Channel accounting collection stopped

Severity

0

Explanation

The channel initiator has stopped collecting channel accounting data.

System action

Channel accounting data that has been collected for channels with STATCHL (HIGH | MED | LOW) is written to the System Management Facility (SMF).

CSQX128I

csect-name Channel initiator statistics collection started

Severity

0

Explanation

The channel initiator has started collecting channel initiator statistics data.

System action

Channel initiator statistics data is collected and written to the System Management Facility (SMF).

CSQX129I

csect-name Channel initiator statistics collection stopped

Severity

0

Explanation

The channel initiator has stopped collecting channel initiator statistics data.

System action

Channel initiator statistics data that has been collected is written to the System Management Facility (SMF).

CSQX130E

csect-name queue-name is defined on a non-recoverable CF structure

Severity

8

Explanation

The shared channel synchronization queue *queue-name* is defined on a Coupling Facility (CF) structure that does not support recovery. This means that if the structure fails, shared channels might report message sequence errors, and might also lose messages.

System action

Processing continues.

System programmer response

Alter the CFSTRUCT object for the CF structure, where the shared channel synchronization queue is defined to RECOVER(YES), or plan to move the shared channel synchronization queue to the CSQSYSAPPL structure, which should be defined with RECOVER(YES).

CSQX140E

csect-name Adapter failed to start

Severity

8

Explanation

A severe error, as reported in the preceding messages, occurred during adapter subtask startup processing.

System action

The channel initiator will attempt to restart the adapter subtask.

System programmer response

Investigate the problem reported in the preceding messages.

CSQX141I

csect-name started adapter subtasks started, *failed* failed

Severity

0

Explanation

The channel initiator startup procedure has started the requested number of adapter subtasks; *started* adapter subtasks started successfully and *failed* adapter subtasks did not start.

System action

The channel initiator startup processing continues.

System programmer response

If the message indicates that some adapter subtasks failed, investigate the problem reported in the preceding messages.

CSQX142E

csect-name Adapter subtask failed to start, TCB=*tcb-name*

Severity

8

Explanation

A severe error, as reported in the preceding messages, occurred during adapter subtask startup processing.

System action

The channel initiator will attempt to restart the adapter subtask.

System programmer response

Investigate the problem reported in the preceding messages.

CSQX143E

csect-name Adapter subtask ended abnormally, TCB=*tcb-name* reason=*sssuuu-reason*

Severity

8

Explanation

The adapter subtask using TCB *tcb-name* is ending abnormally because an error that cannot be corrected has occurred. *sss* is the system completion code, *uuu* is the user completion code, and *reason* is the associated reason code (all in hexadecimal).

System action

The adapter subtask ends abnormally, and a dump is normally issued. The channel initiator will attempt to restart it.

System programmer response

If you are using user channel exits, check that your exit is setting its parameter lists correctly. User completion codes are generally the result of errors detected by the Language Environment; see the *Language Environment for z/OS Debugging Guide and Runtime Messages* for information about these codes. Otherwise, contact your IBM support center.

CSQX144E

csect-name Adapter subtask attach failed, RC=*return-code*

Severity

8

Explanation

An adapter subtask could not be attached. *return-code* is the return code (in hexadecimal) from the z/OS ATTACH service.

System action

The adapter subtask is not restarted.

System programmer response

See the *MVS Programming: Assembler Services Reference* manual for information about the return code from the ATTACH request. If you are unable to solve the problem, contact your IBM support center.

CSQX145E

csect-name Adapter subtask not restarted - too many failures

Severity

8

Explanation

An adapter subtask failed; because it had already failed too many times, the channel initiator did not attempt to restart it.

System action

The adapter subtask is not restarted; processing capacity might therefore be reduced.

System programmer response

Investigate the problems causing the adapter subtask failures.

CSQX146I

csect-name Adapter subtask restarted, *active* subtasks active

Severity

0

Explanation

A adapter subtask failed, but was successfully restarted by the channel initiator. *active* adapter subtasks are now active.

System action

Processing continues.

CSQX150E

csect-name SSL server failed to start

Severity
8

Explanation

A severe error, as reported in the preceding messages, occurred during SSL server subtask startup processing.

System action

The channel initiator will attempt to restart the SSL server subtask.

System programmer response

Investigate the problem reported in the preceding messages.

CSQX151I

csect-name started SSL server subtasks started, *failed* failed

Severity
0

Explanation

The channel initiator startup procedure has started the requested number of SSL server subtasks; *started* SSL server subtasks started successfully and *failed* SSL server subtasks did not start.

System action

The channel initiator startup processing continues.

System programmer response

If the message indicates that some SSL server subtasks failed, investigate the problem reported in the preceding messages.

CSQX152E

csect-name SSL server subtask failed to start, TCB=*tcb-name*

Severity
8

Explanation

A severe error, as reported in the preceding messages, occurred during SSL server subtask startup processing.

System action

The channel initiator will attempt to restart the SSL server subtask.

System programmer response

Investigate the problem reported in the preceding messages.

CSQX153E

csect-name SSL server subtask ended abnormally, TCB=*tcb-name* reason=*sssuuu-reason*

Severity
8

Explanation

The SSL server subtask using TCB *tcb-name* is ending abnormally because an error that cannot be corrected has occurred. *sss* is the system completion code, *uuu* is the user completion code, and *reason* is the associated reason code (all in hexadecimal).

System action

The SSL server subtask ends abnormally, and a dump is normally issued. The channel initiator will attempt to restart it.

System programmer response

If you are using user channel exits, check that your exit is setting its parameter lists correctly. User completion codes are generally the result of errors detected by the Language Environment; see the *Language Environment for z/OS Debugging Guide and Runtime Messages* for information about these codes. Otherwise, contact your IBM support center.

CSQX154E

csect-name SSL server subtask attach failed, RC=*return-code*

Severity
8

Explanation

An SSL server subtask could not be attached. *return-code* is the return code (in hexadecimal) from the z/OS ATTACH service.

System action

The SSL server subtask is not restarted.

System programmer response

See the *MVS Programming: Assembler Services Reference* manual for information about the return code from the ATTACH request. If you are unable to solve the problem, contact your IBM support center.

CSQX155E

csect-name SSL server subtask not restarted - too many failures

Severity
8

Explanation

An SSL server subtask failed; because it had already failed too many times, the channel initiator did not attempt to restart it.

System action

The SSL server subtask is not restarted; processing capacity might therefore be reduced.

System programmer response

Investigate the problems causing the SSL server subtask failures.

CSQX156I

csect-name SSL server subtask restarted, *active* subtasks active

Severity
0

Explanation

A SSL server subtask failed, but was successfully restarted by the channel initiator. *active* SSL server subtasks are now active.

System action

Processing continues.

CSQX160E

csect-name SSL communications unavailable

Severity
4

Explanation

V 9.0.3 SSLKEYR is required when communicating with the service.

SSL communications are requested but an error, as reported in the preceding messages, occurred during channel initiator startup processing.

System action

Processing continues.

System programmer response

Investigate the problem reported in the preceding messages. If you do not want to use SSL communications, set the SSLTASKS queue manager attribute to 0.

CSQX161E

csect-name SSL key repository name not specified

Severity
4

Explanation

V 9.0.3 SSLKEYR is required when communicating with the service.

SSL communications are requested but no SSL key repository name (SSLKEYR) is specified; that is, the SSLTASKS queue manager attribute is non-zero, but the SSLKEYR queue manager attribute is blank.

System action

Processing continues, but communications using SSL will not be available.

System programmer response

Use the ALTER QMGR command to specify a name for the SSL key repository with the SSLKEYR attribute, and restart the channel initiator. If you do not want to use SSL communications, set the SSLTASKS queue manager attribute to 0.

CSQX162E

csect-name SSL CRL namelist is empty or wrong type

Severity
4

Explanation

SSL communications are requested but the SSL authentication namelist specified by the SSLCRLNL queue manager attribute is empty or not of type AUTHINFO.

System action

If this message is displayed during CHINIT startup, then MQ communications using SSL are not available.

If the message is displayed after a change to the existing MQ SSL configuration and issuing the REFRESH SECURITY TYPE(SSL) command, then the changed MQ SSL configuration is rejected and the current MQ SSL configuration remains in force. This is to prevent a set of valid and working MQ SSL definitions being inadvertently deactivated by an incorrect change.

Processing continues.

System programmer response

Correct the definitions of the namelist, and start the channel initiator again. If you do not want to use SSL communications, set the SSLTASKS queue manager attribute to 0.

CSQX163I

csect-name SSL CRL namelist has too many names - first *n* used

Severity

4

Explanation

The SSL authentication namelist specified by the SSLCRLNL queue manager attribute has more names than are supported. The number supported is *n*.

System action

Processing continues; the excess names are ignored.

System programmer response

Correct the definitions of the namelist.

CSQX164E

csect-name Unable to access SSL key repository

Severity

4

Explanation

The SSL key respository, with a name that is specified by the SSLKEYR queue manager attribute, could not be accessed.

The most likely causes are:

- The specified key repository does not exist.
- The channel initiator does not have permission to read the specified key repository.
- The channel initiator was unable to connect to the LDAP server specified in an authentication information object listed in the SSL CRL namelist.
- When using shared key rings, the name is not prefixed with 'userid/'.

System action

Processing continues, but communications using SSL will not be available. Channels using SSL communications will not start.

System programmer response

Check that:

- the SSL key repository name is specified correctly; if using a shared key ring, it is prefixed with 'userid/'
- the key ring specified as the SSL key repository exists, and the channel initiator has permission to read it

- the LDAP name is specified correctly and that it is available.

For more information, refer to System SSL RC 202.

CSQX165I

csect-name SSL key repository refresh already in progress

Severity
0

Explanation

A REFRESH SECURITY TYPE(SSL) command was issued, but an SSL key repository refresh was already in progress.

System action

The command is ignored. The refresh currently in progress continues.

CSQX166E

csect-name AuthInfo *auth-info-name* has wrong type

Severity
4

Explanation

The SSL authentication namelist specified by the SSLCRLNL queue manager attribute contains the name of an authentication information object that has an AUTHTYPE of OCSP.

System action

Processing continues, but communications using SSL will not be available.

System programmer response

Correct the definitions supplied in the namelist so that only authentication information objects with AUTHTYPE of CRLLDAP are named, and restart the channel initiator. If you do not want to use SSL communications, set the SSLTASKS queue manager attribute to 0.

CSQX181E

csect-name Invalid response *response* set by exit *exit-name*

Severity
8

Explanation

The user exit *exit-name* returned an invalid response code (*response*, shown in hexadecimal) in the *ExitResponse* field of the channel exit parameters (MQCXP).

System action

Message CSQX190E is issued giving more details, and the channel stops. For auto-defined channels, the channel does not start.

System programmer response

Investigate why the user exit program set an invalid response code.

CSQX182E

csect-name Invalid secondary response *response* set by exit *exit-name*

Severity
8

Explanation

The user exit *exit-name* returned an invalid secondary response code (*response*, shown in hexadecimal) in the *ExitResponse2* field of the channel exit parameters (MQCXP).

System action

Message CSQX190E is issued giving more details, and the channel stops. For auto-defined channels, the channel does not start.

System programmer response

Investigate why the user exit program set an invalid secondary response code.

CSQX184E

csect-name Invalid exit buffer address *address* set by exit *exit-name*

Severity
8

Explanation

The user exit *exit-name* returned an invalid address for the exit buffer when the secondary response code in the *ExitResponse2* field of the channel exit parameters (MQCXP) is set to MQXR2_USE_EXIT_BUFFER.

System action

Message CSQX190E is issued giving more details, and the channel stops. For auto-defined channels, the channel does not start.

System programmer response

Investigate why the user exit program set an invalid exit buffer address. The most likely cause is failing to set a value, so that it is 0.

CSQX187E

csect-name Invalid header compression value set by exit *exit-name*

Severity
8

Explanation

The user exit *exit-name* returned a header compression value that was not one of those which were negotiated as acceptable when the channel started.

System action

Message CSQX190E is issued giving more details, and the channel stops. For auto-defined channels, the channel does not start.

System programmer response

Investigate why the user exit program set an invalid value. If necessary, alter the channel definitions so that the required compression value is acceptable.

CSQX188E

csect-name Invalid message compression value set by exit *exit-name*

Severity
8

Explanation

The user exit *exit-name* returned a message compression value that was not one of those which were negotiated as acceptable when the channel started.

System action

Message CSQX190E is issued giving more details, and the channel stops. For auto-defined channels, the channel does not start.

System programmer response

Investigate why the user exit program set an invalid value. If necessary, alter the channel definitions so that the required compression value is acceptable.

CSQX189E

csect-name Invalid data length *length* set by exit *exit-name*

Severity

8

Explanation

The user exit *exit-name* returned a data length value that was not greater than zero.

System action

Message CSQX190E is issued giving more details, and the channel stops. For auto-defined channels, the channel does not start.

System programmer response

Investigate why the user exit program set an invalid data length.

CSQX190E

csect-name Channel *channel-name* stopping because of error in exit *exit-name*, Id=*ExitId* reason=*ExitReason*

Severity

8

Explanation

The user exit *exit-name* invoked for channel *channel-name* returned invalid values, as reported in the preceding messages. *ExitId* shows the type of exit:

11 MQXT_CHANNEL_SEC_EXIT, security exit

12 MQXT_CHANNEL_MSG_EXIT, message exit

13 MQXT_CHANNEL_SEND_EXIT, send exit

14 MQXT_CHANNEL_RCV_EXIT, receive exit

15 MQXT_CHANNEL_MSG_RETRY_EXIT, message retry exit

16 MQXT_CHANNEL_AUTO_DEF_EXIT, auto-definition exit

and *ExitReason* shows the reason for invoking it:

11 MQXR_INIT, initialization

12 MQXR_TERM, termination

13 MQXR_MSG, process a message

14 MQXR_XMIT, process a transmission

15 MQXR_SEC_MSG, security message received

16 MQXR_INIT_SEC, initiate security exchange

17 MQXR_RETRY, retry a message

18 MQXR_AUTO_CLUSSDR, auto-definition of cluster-sender channel

28 MQXR_AUTO_CLUSRCVR, auto-definition of cluster-receiver channel

System action

The channel stops. The associated transmission queue may be set to GET(DISABLED) and triggering turned off. For auto-defined channels, the channel does not start.

System programmer response

Investigate why the user exit program set invalid values.

CSQX191I

csect-name Channel *channel-name* beginning message reallocation

Severity

0

Explanation

The channel *channel-name* is entering message reallocation because it cannot currently deliver messages to the destination queue manager.

System action

Messages that are not bound to a particular queue manager will be workload balanced. This may take some time if there are a large number of messages assigned to this channel. Check how many using the **DISPLAY CHSTATUS(*channel-name*) XQMSGSA** command.

System programmer response

If reallocation is not required, for example because the destination queue manager is now available, reallocation can be interrupted using **STOP CHANNEL MODE(FORCE)**.

CSQX192E

csect-name Channel *channel-name* unable to stop, message reallocation in progress

Severity

8

Explanation

A request to stop channel *channel-name* was made, but the channel cannot stop immediately because message reallocation is taking place.

System action

The channel continues to reallocate messages. This process can take some time to complete if there are a large number of messages assigned to the channel on its transmission queue. An increase in CPU utilization might be observed during this time. Upon completion of the reallocation process the channel ends.

System programmer response

The number of messages to be reallocated can be determined using the **DISPLAY CHSTATUS(*channel-name*) XQMSGSA** command.

Turn on the **MONCHL** attribute of the channel and check how many users are using the **DISPLAY CHSTATUS(*channel-name*) XQMSGSA** command. The value of **MONCHL** should be LOW,MEDIUM or HIGH. See MONCHL for further information.

If reallocation is not required, for example because the destination queue manager is not available, reallocation can be interrupted using the **STOP CHANNEL MODE(FORCE)** command.

V 9.0.3

CSQX193I

csect-name Successfully registered with an instance of IBM Cloud Product Insights in IBM Cloud (formerly Bluemix®).

Severity
0

Explanation

The queue manager has been successfully registered with an IBM Cloud Product Insights service. Log in to IBM Cloud (formerly Bluemix) to view registered queue managers.

System action

Processing continues.

System programmer response

None.

V 9.0.3

CSQX194E

csect-name Registration with IBM Cloud Product Insights in IBM Cloud at *url* failed with status codes *http_code* (*toolkit_code explanation*).

Severity
8

Explanation

The queue manager was unable to register with the IBM Cloud Product Insights service.

System action

Another upload attempt will be made at the next upload interval.

System programmer response

Use the error codes and explanation to identify the issue. The explanation can come from the z/OS Client Web Enablement Toolkit or from the IBM Cloud Product Insights service.

Check the following:

- The `APIKey` and `ServiceURL` are specified in the `ReportingService` stanza in the CSQMQIN DD card of the queue manager.
- The channel initiator has network access to the IBM Cloud service.
- The channel initiator has a SSL key ring (SSLKEYR), and the IBM Cloud certificates are connected to the key ring.

CSQX196E

csect-name Data length *data-length* set by exit *exit-name* is larger than agent buffer length *ab-length*

Severity
8

Explanation

The user exit *exit-name* returned data in the supplied agent buffer, but the length specified is greater than the length of the buffer.

System action

Message CSQX190E is issued giving more details, and the channel stops. For auto-defined channels, the channel does not start.

System programmer response

Investigate why the user exit program set an invalid data length.

CSQX197E

csect-name Data length *data-length* set by exit *exit-name* is larger than exit buffer length *eb-length*

Severity
8

Explanation

The user exit *exit-name* returned data in the supplied exit buffer, but the length specified is greater than the length of the buffer.

System action

Message CSQX190E is issued giving more details, and the channel stops. For auto-defined channels, the channel does not start.

System programmer response

Investigate why the user exit program set an invalid data length.

V 9.0.3

CSQX198E

csect-name Upload of usage to IBM Cloud Product Insights in IBM Cloud at *url* failed with status codes *http_code* (*toolkit_code explanation*).

Severity
8

Explanation

The queue manager was unable to upload statistics with the IBM Cloud Product Insights service.

System action

Processing continues.

System programmer response

Use the error codes and explanation to identify the issue. Check the following:

- The `APIKey` and `ServiceURL` are specified in the `ReportingService` stanza in the CSQMQINI DD card of the queue manager.
- The channel initiator has network access to the IBM Cloud service.
- The channel initiator has a SSL key ring (SSLKEYR), and the IBM Cloud certificates are connected to the key ring.

CSQX199E

csect-name Unrecognized message code *ccc*

Severity
8

Explanation

An unexpected error message code has been issued by the channel initiator.

System action

Another upload attempt will be made at the next upload interval.

System programmer response

Use the error codes and explanation to identify the issue. Check the following:

- The `APIKey` and `ServiceURL` are specified in the V 9.0.5 `ReportingService` stanza in the CSQMQINI DD card of the queue manager.
- The channel initiator has network access to the IBM Cloud (formerly Bluemix) service.
- The channel initiator has a SSL key ring (SSLKEYR), and the IBM Cloud certificates are connected to the key ring.

CSQX201E

csect-name Unable to allocate conversation, channel *channel-name* connection *conn-id* TRPTYPE=*trptype* RC=*return-code* (*return-text*) reason=*reason*

Severity

8

Explanation

An attempt to allocate a conversation on connection *conn-id* was not successful. The associated channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'. *trptype* shows the communications system used:

TCP TCP/IP

LU62 APPC/MVS

The return code from it was: (in hexadecimal) *return-code*, (in text) *return-text*. For some errors, there may also be an associated reason code *reason* (in hexadecimal) giving more information.

System action

The channel is not started.

System programmer response

The error may be due to an incorrect entry in the channel definition or some problems in the APPC setup. Correct the error and try again

It could also be that the listening program at the remote end is not running. If so, perform the necessary operations to start the listener for *trptype*, and try again.

See "Communications protocol return codes" on page 5211 for information about the cause of the return code from the communications system. If using TCP/IP, see the *z/OS UNIX System Services Messages and Codes* manual for information about the reason code.

CSQX202E

csect-name Connection or remote listener unavailable, channel channel-name connection conn-id TRPTYPE=*trptype* RC=*return-code* (*return-text*) reason=*reason*

Severity

8

Explanation

An attempt to allocate a conversation was not successful because the connection *conn-id* was unavailable. The associated channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'. *trptype* shows the communications system used:

TCP TCP/IP

LU62 APPC/MVS

The return code from it was: (in hexadecimal) *return-code*, (in text) *return-text*. For some errors, there might also be an associated reason code *reason* (in hexadecimal) giving more information.

System action

The attempt to start the channel is retried.

System programmer response

Try again later.

A likely cause is that the listener at the remote end was not running or has been started using the wrong port or LU name. If this is the case, perform the necessary operations to start the appropriate listener, and try again.

See "Communications protocol return codes" on page 5211 for information about the cause of the return code from the communications system.

If using TCP/IP, see the z/OS UNIX System Services Messages and Codes manual for information about the reason code.

If you receive reason code 468:

- You are not using the correct IP address.
- The listener for the port might not be active.
- A firewall does not allow the connection.

  When there are multiple links defined on a z/OS image, the image can have multiple host names depending on the link. You need to ensure that the correct host name is used as the sender end. Use the NETSTAT HOSTs command to display the host names on the image.

CSQX203E

csect-name Error in communications configuration, channel channel-name connection conn-id TRPTYPE=*trptype* RC=*return-code* (*return-text*) reason=*reason*

Severity

8

Explanation

An attempt to allocate a conversation on connection *conn-id* was not successful because of a communications configuration error. The associated channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'. *trptype* shows the communications system used:

TCP TCP/IP

LU62 APPC/MVS

The return code from it was: (in hexadecimal) *return-code*, (in text) *return-text*. For some errors, there might also be an associated reason code *reason* (in hexadecimal) giving more information.

System action

The channel is not started.

System programmer response

See "Communications protocol return codes" on page 5211 for information about the cause of the return code from the communications system.

Probable causes are:

- If the communications protocol is TCP/IP:
  - The connection name specified is incorrect, or that it cannot be resolved to a network address, or the name may not be in the name server. Correct the error and try again.
  - If the return code is zero, there is a name server problem. The OMVS command OPING usually fails in the same way. Resolve this failure and restart the channel. Check the `/etc/resolv.conf` file and check that the correct name server address is specified in the NSINTERADDR statement.

- If the communications protocol is LU 6.2:
  - One of the transmission parameters (MODENAME or TPNAME or PARTNER_LU) in the side information is incorrect, or that there is no side information for the symbolic destination name specified as the connection name. Correct the error and try again.
  - An LU 6.2 session has not been established, perhaps because the LU has not been enabled. Issue the z/OS command VARY ACTIVE if this is the case.

See the *z/OS UNIX System Services Messages and Codes* manual for information about the reason code.

CSQX204E

csect-name Connection attempt rejected, channel channel-name connection conn-id TRPTYPE=*trptype* RC=*return-code* (*return-text*) reason=*reason*

Severity

8

Explanation

An attempt to connect on connection *conn-id* was rejected. The associated channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'. *trptype* shows the communications system used:

TCP TCP/IP

LU62 APPC/MVS

The return code from it was: (in hexadecimal) *return-code*, (in text) *return-text*. For some errors, there might also be an associated reason code *reason* (in hexadecimal) giving more information.

System action

The channel is not started.

System programmer response

Check the appropriate listener has been started on the remote end.

See “Communications protocol return codes” on page 5211 for information about the cause of the return code from the communications system.

If the communications protocol is LU 6.2, it is possible that either the user ID or password supplied at the remote LU is incorrect. The remote host or LU may not be configured to allow connections from the local host or LU.

If the communications protocol is TCP/IP, it is possible that the remote host does not recognize the local host. See the *z/OS UNIX System Services Messages and Codes* manual for information about the reason code.

CSQX205E

csect-name Unable to resolve network address, channel channel-name connection conn-id TRPTYPE=TCP RC=*return-code* (*return-text*) reason=*reason*

Severity

8

Explanation

The supplied connection name *conn-id* could not be resolved into a TCP/IP network address. The associated channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'. *trptype* shows the communications system used:

TCP TCP/IP

LU62 APPC/MVS

The return code from it was: (in hexadecimal) *return-code*, (in text) *return-text*. For some errors, there might also be an associated reason code *reason* (in hexadecimal) giving more information.

System action

The channel is not started.

System programmer response

Check the local TCP/IP configuration. Either the name server does not contain the host or LU name, or the name server was not available.

See "Communications protocol return codes" on page 5211 for information about the cause of the return code from TCP/IP. See the *z/OS UNIX System Services Messages and Codes* manual for information about the reason code.

CSQX206E

csect-name Error sending data, channel *channel-name* connection *conn-id* (queue manager *qmgr-name*) TRPTYPE=*trptype* RC=*return-code* (*return-text*) reason=*reason*

Severity

8

Explanation

An error occurred sending data to *conn-id*, which might be due to a communications failure. The associated channel is *channel-name* and the associated remote queue manager is *qmgr-name*; in some cases the names cannot be determined and so are shown as '????'. *trptype* shows the communications system used:

TCP TCP/IP

LU62 APPC/MVS

The return code from it was: (in hexadecimal) *return-code*, (in text) *return-text*. For some errors, there might also be an associated reason code *reason* (in hexadecimal) giving more information.

System action

The channel is stopped. The associated transmission queue might be set to GET(DISABLED) and triggering turned off.

System programmer response

See "Communications protocol return codes" on page 5211 for information about the cause of the return code from the communications system. If using TCP/IP, see the *z/OS UNIX System Services Messages and Codes* manual for information about the reason code.

Note that the error might have occurred because the channel at the other end has stopped for some reason, for example an error in a receive user exit.

CSQX207E

csect-name Invalid data received, connection *conn-id* (queue manager *qmgr-name*) TRPTYPE=*trptype*

Severity

8

Explanation

Data received from connection *conn-id* was not in the required format. The associated remote queue manager is *qmgr-name*; in some cases its name cannot be determined and so is shown as '????'. The data that has been sent may come from something other than a queue manager or client. *trptype* shows the communications system used:

TCP TCP/IP

LU62 APPC/MVS

System action

The data is ignored.

System programmer response

A likely cause is that an unknown host or LU is attempting to send data.

CSQX208E

csect-name Error receiving data, channel *channel-name* connection *conn-id* (queue manager *qmgr-name*) TRPTYPE=*trptype* RC=*return-code* (*return-text*) reason=*reason*

Severity

8

Explanation

An error occurred receiving data from connection *conn-id*, which may be due to a communications failure. The associated channel is *channel-name* and the associated remote queue manager is *qmgr-name*; in some cases the names cannot be determined and so are shown as '????'. *trptype* shows the communications system used:

TCP TCP/IP

LU62 APPC/MVS

The return code from it was: (in hexadecimal) *return-code*, (in text) *return-text*. For some errors, there might also be an associated reason code *reason* (in hexadecimal) giving more information.

System action

The channel is stopped. The associated transmission queue may be set to GET(DISABLED) and triggering turned off.

System programmer response

See “Communications protocol return codes” on page 5211 for information about the cause of the return code from the communications system. If using TCP/IP, see Return code 00000461 for more information about the reason code.

CSQX209E

csect-name Connection unexpectedly terminated, channel *channel-name* connection *conn-id* (queue manager *qmgr-name*) TRPTYPE=*trptype* RC=*return-code* (*return-text*)

Severity

8

Explanation

An error occurred receiving data from connection *conn-id*. The connection to the remote host or LU has unexpectedly terminated. The associated channel is *channel-name* and the associated remote queue manager is *qmgr-name*; in some cases the names cannot be determined and so are shown as '????'. *trptype* shows the communications system used:

TCP TCP/IP

LU62 APPC/MVS

However, this message can also occur in cases where there is no error; for example, if the TCP/IP command TELNET is issued that is directed at the port which the channel initiator is using.

The return code from it was: (in hexadecimal) *return-code*, (in text) *return-text*. For some errors, there might also be an associated reason code *reason* (in hexadecimal) giving more information.

System action

If a channel is involved, it is stopped. The associated transmission queue may be set to GET(DISABLED) and triggering turned off.

System programmer response

Review the local and remote console logs for reports of network errors.

See"Communications protocol return codes" on page 5211 for information about the cause of the return code from the communications system. If using TCP/IP, see the *z/OS UNIX System Services Messages and Codes* manual for information about the reason code.

CSQX210E

csect-name Unable to complete bind, channel *channel-name* connection *conn-id* TRPTYPE=LU62 RC=*return-code* (*return-text*) reason=*reason*

Severity

8

Explanation

An incoming attach request arrived on connection *conn-id*, but the local host or LU was unable to complete the bind. The associated channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'.

The return code from APPC/MVS allocate services was: (in hexadecimal) *return-code*, (in text) *return-text*. For some errors, there might also be an associated reason code *reason* (in hexadecimal) giving more information.

System action

The channel is not started.

System programmer response

Check the APPC/MVS configuration.

See "APPC/MVS return codes" on page 5214 for the cause of the return code from APPC/MVS allocate services, and the *Writing Servers for APPC/MVS* manual for more information.

CSQX212E

csect-name Unable to allocate socket, channel *channel-name* TRPTYPE=TCP RC=*return-code* (*return-text*) reason=*reason*

Severity

8

Explanation

A TCP/IP socket could not be created, possibly because of a storage problem. The associated channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'.

The return code from TCP/IP was: (in hexadecimal) *return-code*, (in text) *return-text*. For some errors, there might also be an associated reason code *reason* (in hexadecimal) giving more information.

System action

The channel is not started.

System programmer response

See “Communications protocol return codes” on page 5211 for information about the cause of the return code from TCP/IP. See the *z/OS UNIX System Services Messages and Codes* manual for information about the reason code.

CSQX213E

csect-name Communications error, channel *channel-name* TRPTYPE=*trptype* function *func* RC=*return-code* (*return-text*) reason=*reason*

Severity

8

Explanation

An unexpected communications error occurred for a listener or a channel. If it was for a listener, the *csect-name* is CSQXCLMA, and the channel name is shown as '????'. If it was for a channel, the channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'.

trptype shows the communications system used:

TCP TCP/IP

LU62 APPC/MVS

func is the name of the TCP/IP or APPC/MVS function that gave the error. In some cases the function name is not known and so is shown as '????'.

return-code is

- normally, the return code (in hexadecimal) from the communications system function
- for an LU 6.2 listener, it might be the reason code (in hexadecimal) from APPC/MVS allocate services
- if it is of the form 10009*nnn* or 20009*nnn*, it is a distributed queuing message code.

return-text is the text form of the return code.

For some errors, there might also be an associated reason code *reason* (in hexadecimal) giving more information.

System action

If the error occurred for a channel, the channel is stopped. For a listener, the channel is not started or, in some cases, the listener terminates.

System programmer response

See “Communications protocol return codes” on page 5211 for information about the cause of the return code from the communications system.

A distributed queuing message code *nnn* is generally associated with message CSQX*nnn*E, which will normally be issued previously. See that message explanation for more information. Where no such message is described, see “Distributed queuing message codes” on page 5224 for the corresponding message number.

Check for error messages on the partner system that might indicate the cause of the problem.

CSQX215E

csect-name Communications network not available, TRPTYPE=*trptype*

Severity

8

Explanation

An attempt was made to use the communications system, but it has not been started or has stopped. *trptype* shows the communications system used:

TCP TCP/IP

LU62 APPC/MVS

System action

The channel or listener is not started.

System programmer response

Start the communications system, and try again.

CSQX218E

csect-name Listener not started - unable to bind, port *port* address *ip-address* TRPTYPE=TCP INDISP=*disposition* RC=*return-code*

Severity

8

Explanation

An attempt to bind the TCP/IP socket to the indicated listener port was not successful. *ip-address* is the IP address used, or '*' if the listener is using all IP addresses. The return code (in hexadecimal) from TCP/IP was *return-code*.

disposition shows which type of incoming requests the listener was handling:

QMGR
those directed to the target queue manager

GROUP
those directed to the queue-sharing group.

System action

The listener is not started.

System programmer response

The failure could be due to another program using the same port number.

See “Communications protocol return codes” on page 5211 for information about the return code from TCP/IP.

CSQX219E

csect-name Listener stopped - error creating new connection, TRPTYPE=TCP INDISP=*disposition*

Severity

8

Explanation

An attempt was made to create a new TCP/IP socket because an attach request was received, but an error occurred.

disposition shows which type of incoming requests the listener was handling:

QMGR
those directed to the target queue manager

GROUP
those directed to the queue-sharing group.

System action

The listener stops. The channel initiator will attempt to restart it, at the intervals specified by the LSTRTMR queue manager attribute.

System programmer response

The failure might be transitory, try again later. If the problem persists, it might be necessary to stop some other jobs that use TCP/IP, or to restart TCP/IP.

CSQX220E

csect-name Communications network not available, channel *channel-name* TRPTYPE=*trptype*

Severity

8

Explanation

An attempt was made to use the communications system by a channel or a listener, but it has not been started or has stopped. If it was for a channel, the channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'. If it was for a listener, the channel name is again shown as '????'. *trptype* shows the communications system used:

TCP TCP/IP

LU62 APPC/MVS

System action

The channel or listener is not started.

System programmer response

Start the communications system, and try again.

CSQX228E

csect-name Listener unable to start channel, channel *channel-name* TRPTYPE=*trptype* INDISP=*disposition* connection=*conn-id*

Severity

8

Explanation

An incoming attach request arrived from *conn-id*, but the listener for *trptype* could not start an instance of a channel to respond to it. The associated channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'.

disposition shows which type of incoming requests the listener was handling:

QMGR
those directed to the target queue manager

GROUP
those directed to the queue-sharing group.

However, this message can also occur in cases where there is no error; for example, if the TCP/IP command TELNET is issued that is directed at the port which the channel initiator is using.

System action

If a channel is involved, it is not started.

System programmer response

The failure could be because the channel initiator is currently too busy; try again when there are fewer channels running. If the problem persists, increase the number of dispatchers used by the channel initiator.

CSQX234I

csect-name Listener stopped, TRPTYPE=*trptype* INDISP=*disposition*

Severity

0

Explanation

The specified listener terminated. This could be for a number of reasons including, but not limited to, those in the following list:

- a STOP command was issued
- the listener was retrying
- an error occurred in the communications system

trptype is the transport type.

disposition shows which type of incoming requests the listener was handling:

QMGR
those directed to the target queue manager

GROUP
those directed to the queue-sharing group.

System action

Processing continues. If the listener was not deliberately stopped, the channel initiator will attempt to restart the listener, at the intervals specified by the LSTRTMR queue manager attribute.

System programmer response

If the listener was not deliberately stopped, look at any preceding messages relating to the channel initiator or to the TCP/IP, OMVS, or APPC address spaces to determine the cause.

CSQX235E

csect-name Invalid local address *local-addr*, channel *channel-name* TRPTYPE=*trptype* RC=*return-code* (*return-text*) reason=*reason*

Severity

8

Explanation

The supplied local address *local-addr* could not be resolved to a TCP/IP network address. The associated channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'. *trptype* shows the communications system used:

TCP TCP/IP

LU62 APPC/MVS

The return code from it was: (in hexadecimal) *return-code*, (in text) *return-text*. For some errors, there might also be an associated reason code *reason* (in hexadecimal) giving more information.

System action

The channel is not started.

System programmer response

Check the local TCP/IP configuration. Either the name server does not contain the host name, or the name server was not available.

See “Communications protocol return codes” on page 5211 for information about the cause of the return code from TCP/IP.

CSQX239E

csect-name Unable to determine local host name, channel *channel-name* TRPTYPE=TCP RC=*return-code* (*return-text*) reason=*reason*

Severity

8

Explanation

An attempt was made to start a channel or listener using TCP/IP, but the TCP/IP gethostname call failed. If it was for a channel, the channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'. If it was for a listener, the channel name is again shown as '????'.

The return code from it was: (in hexadecimal) *return-code*, (in text) *return-text*. For some errors, there might also be an associated reason code *reason* (in hexadecimal) giving more information.

System action

The channel or listener is not started.

System programmer response

See “Communications protocol return codes” on page 5211 for information about the cause of the return code from TCP/IP.

CSQX250E

csect-name Listener ended abnormally, TRPTYPE=*trptype* INDISP=disposition, reason=*sssuuu-reason*

Severity

8

Explanation

The specified listener is ending abnormally because an error that cannot be corrected has occurred. *sss* is the system completion code, *uuu* is the user completion code, and *reason* is the associated reason code (all in hexadecimal).

disposition shows which type of incoming requests the listener was handling:

QMGR
those directed to the target queue manager

GROUP
those directed to the queue-sharing group.

System action

The listener ends abnormally, and a dump is normally issued. The channel initiator will attempt to restart the listener, at the intervals specified by the LSTRTMR queue manager attribute.

System programmer response

User completion codes are generally the result of errors detected by the Language Environment; see the *Language Environment for z/OS Debugging Guide and Runtime Messages* for information about these codes. Otherwise, contact your IBM support center.

CSQX251I

csect-name Listener started, TRPTYPE=*trptype* INDISP=*disposition*

Severity

0

Explanation

The specified listener started successfully. This may be as a result of a START LISTENER command, or because the listener restarted automatically following an error.

disposition shows which type of incoming requests the listener was handling:

QMGR
those directed to the target queue manager

GROUP
those directed to the queue-sharing group.

System action

Processing continues.

CSQX256E

csect-name Listener stopped - error selecting new connection, TRPTYPE=TCP INDISP=*disposition*

Severity
8

Explanation

An error occurred in the listener select processing. The listener was notified by TCP/IP, but no attach request was received.

disposition shows which type of incoming requests the listener was handling:

QMGR
those directed to the target queue manager

GROUP
those directed to the queue-sharing group.

System action

The listener stops. The channel initiator will attempt to restart it, at the intervals specified by the LSTRTMR queue manager attribute.

System programmer response

The failure might be transitory, try again later. If the problem persists, it might be necessary to stop some other jobs that use TCP/IP, or to restart TCP/IP.

CSQX257I

csect-name Listener unable to create new connection, TRPTYPE=TCP INDISP=*disposition*

Severity
4

Explanation

An attempt was made to create a new TCP/IP socket because an attach request was received, but an error occurred.

disposition shows which type of incoming requests the listener was handling:

QMGR
those directed to the target queue manager

GROUP
those directed to the queue-sharing group.

System action

The listener continues to run, but the connection is not created.

System programmer response

The failure might be transitory, try again later. If the problem persists, it might be necessary to stop some other jobs that use TCP/IP, or to restart TCP/IP.

CSQX258E

csect-name Listener stopped - error accepting new connection, TRPTYPE=TCP INDISP=*disposition*

Severity

8

Explanation

An error occurred in the listener accept processing. The listener was notified by TCP/IP, but no attach request was received.

disposition shows which type of incoming requests the listener was handling:

QMGR
those directed to the target queue manager

GROUP
those directed to the queue-sharing group.

System action

The listener stops. The channel initiator will attempt to restart it, at the intervals specified by the LSTRTMR queue manager attribute.

System programmer response

The failure might be transitory, try again later. If the problem persists, it might be necessary to stop some other jobs that use TCP/IP, or to restart TCP/IP.

CSQX259E

csect-name Connection timed out, channel *channel-name* connection *conn-id* (queue manager *qmgr-name*) TRPTYPE=*trptype*

Severity

8

Explanation

The connection *conn-id* timed out. The associated channel is *channel-name* and the associated remote queue manager is *qmgr-name*; in some cases the names cannot be determined and so are shown as '????'. *trptype* shows the communications system used:

TCP TCP/IP

LU62 APPC/MVS

Probable causes are:

- A communications failure.
- For a message channel, if the Receive Timeout function is being used (as set by the RCVTIME, RCVTTYPE, and RCVTMIN queue manager attributes) and no response was received from the partner within this time.
- For an MQI channel, if the Client Idle function is being used (as set by the DISCINT server-connection channel attribute) and the client application did not issue an MQI call within this time.

System action

The channel stops.

System programmer response

For a message channel, check the remote end to see why the time out occurred. Note that, if retry values are set, the remote end will restart automatically. If necessary, set the receive wait time for the queue manager to be higher.

For an MQI channel, check that the client application behavior is correct. If so, set the disconnect interval for the channel to be higher.

CSQX261E

csect-name No suitable IP stack available, channel *channel-name*, connection *conn-id*

Severity

8

Explanation

An attempt to allocate a conversation on connection *conn-id* for channel *channel-name* using TCP/IP communications was not successful because the IP stack used did not support the IP address family required for the connection.

System action

The channel is not started.

System programmer response

If the channel's CONNAME attribute resolves to an IPv6 address, then ensure the stack being used by the combination of the TCPNAME queue manager attribute and the channel's LOCLADDR attribute supports IPv6. If the channel's CONNAME attribute resolves to an IPv4 address, then ensure the stack being used by the combination of the TCPNAME queue manager attribute and the channel's LOCLADDR attribute supports IPv4.

CSQX262E

csect-name Communications canceled, channel *channel-name* TRPTYPE=*trptype*

Severity

8

Explanation

An unexpected communications error occurred for a listener or a channel. This error occurs if the channel was stopped with mode FORCE and the communications session was canceled.

The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'. *trptype* shows the communications system used:

TCP TCP/IP

LU62 APPC/MVS

System action

The channel is stopped.

System programmer response

Restart the channel if appropriate.

CSQX293I

csect-name Channel *channel-name* has initiated a switch of transmission queue from *old-xmitq* to *new-xmitq*

Severity

0

Explanation

A switch of transmission queue for the channel identified by *channel-name* is required due to a change to the default cluster transmission queue configuration of the queue manager, or to the cluster channel name attribute of a cluster transmission queue. This message is issued by the channel initiator when the process of switching the transmission queue from *old-xmitq* to *new-xmitq* is started.

System action

The queue manager is notified to start the switching process for the channel.

The channel continues to run after closing the old transmission queue and switching to use the new transmission queue instead.

System programmer response

None.

CSQX294E

csect-name Transmission queue status unavailable, channel *channel-name*

Severity
8

Explanation

The transmission queue for the cluster-sender channel identified by *channel-name* cannot be determined because when the queue manager started it was unable to load the persisted transmission queue state from the queue SYSTEM.CHANNEL.SYNCQ.

System action

The channel ends abnormally.

System programmer response

If the queue manager is unable to load the persisted transmission queue state during startup it issues message CSQM561E.

CSQX295E

csect-name Cluster transmission queue restricted, channel *channel-name*, transmission queue *xmitq-name*

Severity
8

Explanation

The cluster-sender channel identified by *channel-name* has been configured to use the transmission queue *xmitq-name*, however, the use of transmission queues other than SYSTEM.CLUSTER.TRANSMIT.QUEUE has subsequently been disabled by the queue manager mode of operation (OPMODE).

System action

The channel ends abnormally.

System programmer response

Ensure the queue manager mode of operation is correctly configured. If the configuration is correct, perform the following actions:

1. Change the default cluster transmission queue configuration of the queue manager so that all cluster-sender channels default to use the transmission queue SYSTEM.CLUSTER.TRANSMIT.QUEUE. You can do this by changing the value of the **DEFCLXQ** queue manager attribute to **SCTQ**.

2. Identify any manually defined transmission queues that have a non-blank cluster channel name attribute value by using the **DISPLAY QLOCAL(*) WHERE(CLCHNAME NE ' ')** command. Change the cluster channel name attribute value of these queues to blank.
3. Restart the channel.

CSQX296E

csect-name Password protection negotiation failed for channel *channel-name*, connection *conn-id*

Severity

8

Explanation

The channel *channel-name* could not be established because it failed to agree a password protection algorithm with the remote machine *conn-id*.

System action

The channel does not start.

System programmer response

Check whether password protection settings prevent interoperability with the remote machine.

Alternatively, consider using SSL or TLS to protect passwords instead. You must use a non-null CipherSpec to protect passwords.

CSQX403I

csect-name Auto-definition of channel *channel-name* suppressed by exit *exit-name*

Severity

0

Explanation

In response to a request to start a channel that was not defined, an attempt was made to define it automatically. The channel auto-definition exit *exit-name* prevented it being defined.

System action

The channel is not started.

CSQX404I

csect-name Phase one of REFRESH CLUSTER REPOS(YES) has completed, cluster *cluster_namen* objects changed

Severity

0

Explanation

Phase one of REFRESH CLUSTER has completed.

Applications attempting to access cluster resources may see failures to resolve cluster resources until phase two of REFRESH CLUSTER is complete.

Phase two is complete once all new information has been received from other members of the cluster.

Monitor your SYSTEM.CLUSTER.COMMAND.QUEUE to determine when it has reached a consistently empty state to indicate that the refresh process has completed.

System action

None.

CSQX405I

csect-name FORCEREMOVE QUEUES(YES) command processed, cluster *cluster_name* target *target*

Severity
0

Explanation

The repository manager successfully processed a RESET CLUSTER ACTION(FORCEREMOVE) command with the QUEUES(YES) option for the indicated cluster and target queue manager.

System action

None.

CSQX406E

csect-name REFRESH CLUSTER REPOS(YES) command failed, cluster *cluster_name* - *qmgr-name* is a full repository

Severity
8

Explanation

The repository manager could not process a REFRESH CLUSTER command with the REPOS(YES) option for the indicated cluster, because the local queue manager provides full repository management service for the cluster.

System action

The command is ignored.

System programmer response

Reissue the command with the correct values or on the correct queue manager. It might be necessary to change the queue manager so that it is not a full repository for the cluster.

CSQX407I

csect-name Cluster queue *q-name* definitions inconsistent

Severity
4

Explanation

The definition of a cluster queue has different values for the DEFPRTY, DEFPSIST, DEFPRESP, and DEFBIND attributes on the various queue managers in the cluster.

All definitions of the same cluster queue must be identical. Problems might arise if your applications rely on one of these attributes to determine messaging behavior. For example, if an application opens a cluster queue with the option MQOO_BIND_AS_Q_DEF, and the different instances of the queue have different DEFBIND values, the behavior of the message transfer depends on which instance of the queue happens to be selected when it is opened.

System action

None.

System programmer response

Alter the definitions of the queue on the various queue managers so that they have identical values for these attributes.

CSQX410I

csect-name Repository manager started

Severity
0

Explanation

The repository manager started successfully.

System action

None.

CSQX411I

csect-name Repository manager stopped

Severity
0

Explanation

The repository manager stopped. This may be for one of three reasons:

- The channel initiator is stopping.
- The channel initiator is starting and the queues used by the repository manager have not been defined because clustering is not required.
- An error has occurred.

System action

Processing continues, but clustering is not available.

System programmer response

If an error has occurred, investigate the problem reported in the preceding messages.

CSQX412E

csect-name Misdirected repository command, target *target-id* sender *sender-id*

Severity
8

Explanation

The repository manager received a command intended for some other queue manager, with an identifier that is *target-id*. The command was sent by the queue manager with identifier *sender-id*.

System action

The command is ignored, and the error is reported to the sender.

System programmer response

Check the channel and cluster definitions of the sending queue manager.

CSQX413E

csect-name Repository command format error, command code *command*

Severity
8

Explanation

An internal error has occurred.

System action

The command is ignored, and the error is reported to the sender; the repository manager continues processing. Information about the error is written to the data set identified by the CSQSNAP DD statement of the channel initiator started task JCL procedure, xxxxCHIN.

System programmer response

Collect the items listed in the Problem Determination section and contact your IBM support center.

CSQX415E

csect-name Repository command state error, command code *command* cluster object *object-name* sender *sender-id*

Severity
8

Explanation

An internal error has occurred.

System action

The command is ignored; the repository manager continues processing. Information about the error is written to the data set identified by the CSQSNAP DD statement of the channel initiator started task JCL procedure, xxxxCHIN.

System programmer response

Collect the items listed in the Problem Determination section and contact your IBM support center.

CSQX416E

csect-name Repository command processing error, RC=*return-code* command code *command* cluster object *object-name* sender *sender-id*

Severity
8

Explanation

An internal error has occurred.

System action

The command is ignored; the repository manager continues processing. Information about the error is written to the data set identified by the CSQSNAP DD statement of the channel initiator started task JCL procedure, xxxxCHIN.

System programmer response

Collect the items listed in the Problem Determination section and contact your IBM support center.

CSQX417I

csect-name Cluster-senders remain for removed queue manager *qmgr-name*

Severity
0

Explanation

The indicated queue manager has been deleted or forcibly removed from a cluster, but there are manually-defined cluster-sender channels that refer to it. This means that the repository manager will continue to send cluster information to the removed queue manager.

System programmer response

Delete the manually-defined cluster-sender channels that refer to *qmgr-name*.

CSQX418I

csect-name Only one repository for cluster *cluster_name*

Severity
0

Explanation

The repository manager has received information about a cluster for which it is the only full repository.

System action

None.

System programmer response

If you require a second full repository, alter the REPOS or REPOSNL attribute of the second queue manager that is to have a full repository for the cluster to specify the cluster name.

CSQX419I

csect-name No cluster-receivers for cluster *cluster_name*

Severity
0

Explanation

The repository manager has received information about a cluster for which no cluster-receiver channels are known.

System action

None.

System programmer response

Define cluster-receiver channels for the cluster on the local queue manager.

CSQX420I

csect-name No repositories for cluster *cluster_name*

Severity
0

Explanation

The repository manager has received information about a cluster for which no full repositories are known.

System action

None.

System programmer response

Define a cluster-sender channel for connecting to the queue manager that is the full repository for the cluster, or alter the REPOS or REPOSNL attribute of the queue manager that is to have a full repository for the cluster to specify the cluster name.

CSQX422E

csect-name Repository manager error, RC=*return-code*

Severity
8

Explanation

An internal error has occurred.

System action

The repository manager attempts to continue processing. Information about the error is written to the data set identified by the CSQSNAP DD statement of the channel initiator started task JCL procedure, xxxxCHIN.

System programmer response

Collect the items listed in the Problem Determination section and contact your IBM support center.

CSQX425E

csect-name Repository command merge error, command code *command* cluster object *object-name* sender *sender-id*

Severity

8

Explanation

An internal error has occurred.

System action

The command is ignored; the repository manager continues processing. Information about the error is written to the data set identified by the CSQSNAP DD statement of the channel initiator started task JCL procedure, xxxxCHIN.

System programmer response

Collect the items listed in the Problem Determination section and contact your IBM support center.

CSQX426E

csect-name Undeliverable repository command, channel *channel-name* target *target-id* command code *command*

Severity

8

Explanation

The repository manager tried to send a command to another queue manager using channel *channel-name*. The other queue manager, with identifier *target-id*, could not be found.

System action

The command is ignored.

System programmer response

Check the channel and cluster definitions of the sending and receiving queue managers.

CSQX427E

csect-name Cluster-sender not connected to repository, cluster *cluster_name* channel *channel-name* target *target-id*

Severity

8

Explanation

A cluster-sender channel must be connected to a queue manager that is a full repository for all the clusters for the channel, and the corresponding cluster-receiver channel must be in the same clusters. Channel *channel-name* in cluster *cluster_name* does not satisfy this. *target-id* is the identifier of the target queue manager for the channel.

System action

The command is ignored.

System programmer response

Check the definition of the channel on both queue managers to ensure that it is connected to a full repository for the clusters, and that it is in the same clusters on both queue managers.

CSQX428E

csect-name Unexpected publication of a cluster queue, cluster *cluster_name* cluster queue *q-name* sender *sender-id*

Severity

8

Explanation

The repository manager received a publication for cluster queue *q-name* from another queue manager, with an identifier *sender-id*, relating to cluster *cluster_name*. The local queue manager cannot accept the command because it is not a full repository for the cluster and thus it does not have an interest in the cluster queue.

This can also occur because a command destined for the local repository manager is delayed in the network and is out of date when it arrives, for example because a REFRESH CLUSTER command has been issued on the local repository manager and caused its view of the cluster to change.

System action

The command is ignored.

System programmer response

If the local partial repository queue manager is supposed to be a full repository for the cluster, use the ALTER QMGR command to specify a repository or repository namelist which contains the cluster. If the local queue manager is correctly a partial repository for the cluster, ensure that the remote queue manager does not have a manually defined cluster sender directed at the local partial repository.

If the message occurs because a command is out of date, the message can be ignored.

CSQX429E

csect-name Unexpected deletion of a cluster queue, cluster *cluster_name* cluster queue *q-name*

Severity

8

Explanation

The repository manager received a deletion for cluster queue *q-name* from another queue manager, with an identifier *sender-id*, relating to cluster *cluster_name*. The local queue manager cannot accept the command because it is not a full repository for the cluster and thus it does not have an interest in the cluster queue.

This can also occur because a command destined for the local repository manager is delayed in the network and is out of date when it arrives, for example because a REFRESH CLUSTER command has been issued on the local repository manager and caused its view of the cluster to change.

System action

The command is ignored.

System programmer response

If the local partial repository queue manager is supposed to be a full repository for the cluster, use the ALTER QMGR command to specify a repository or repository namelist which contains the cluster. If the local queue manager is correctly a partial repository for the cluster, ensure that the remote queue manager does not have a manually defined cluster sender directed at the local partial repository.

If the message occurs because a command is out of date, the message can be ignored.

CSQX430E

csect-name Unexpected queue manager repository command, cluster *cluster_name* channel *channel-name* sender *sender-id*

Severity

8

Explanation

The repository manager received a command from another queue manager, with an identifier that is *sender-id*, relating to cluster *cluster_name*. The local queue manager cannot accept the command because it is not a full repository for the cluster, it does not have an interest in the cluster channel, and it does not have any matching cluster-sender channels. The cluster-sender channel used by the other queue manager was *channel-name*.

This message might appear on a queue manager that has defined a cluster-sender channel to another queue manager that does not host a full repository, if the other queue manager is later modified to host a full repository.

System action

The command is ignored.

System programmer response

Check the definition of the channel on the sending queue manager to ensure that it is connected to a full repository for the cluster.

Ensure the `CLUSTER` and `CLUSNL` values are consistent, and that you have not specified a *cluster_name* when you meant a *cluster-namelist*.

CSQX431I

csect-name Repository unavailable, cluster *cluster_name* channel *channel-name* sender *sender-id*

Severity

0

Explanation

The repository manager received a command from another queue manager, with identifier *sender-id*, reporting that it is no longer a full repository for cluster *cluster_name*.

System action

The cluster-sender channel *channel-name* is changed so that it can no longer be used to access the other queue manager in relation to the cluster.

CSQX432I

csect-name Unexpected cluster query received, cluster *cluster_name* cluster object *object-name* sender *sender-id*

Severity

8

Explanation

The repository manager received a query for cluster object *object-name* from another queue manager, with an identifier *sender-id*, relating to cluster *cluster_name*. The local queue manager cannot accept the command because it is not a full repository for the cluster.

This can also occur because a command destined for the local repository manager is delayed in the network and is out of date when it arrives, for example because a REFRESH CLUSTER command has been issued on the local repository manager and caused its view of the cluster to change.

System action

The command is ignored.

System programmer response

If the local partial repository queue manager is supposed to be a full repository for the cluster, use the ALTER QMGR command to specify a repository or repository namelist which contains the cluster. If the local queue manager is correctly a partial repository for the cluster, ensure that the remote queue manager does not have a manually defined cluster sender directed at the local partial repository.

If the message occurs because a command is out of date, the message can be ignored.

CSQX433E

csect-name Cluster-receiver and cluster-sender differ, cluster *cluster_name* channel *channel-name* sender *sender-id*

Severity

8

Explanation

The repository manager received a command from another queue manager, with identifier *sender-id*. The cluster-sender channel *channel-name* on that queue manager is in cluster *cluster_name*, but the corresponding cluster-receiver channel on the local queue manager is not.

System action

The command is ignored.

System programmer response

Change the definition of the channel so that it is in the same clusters on both queue managers.

CSQX434E

csect-name Unrecognized message on *name*

Severity

8

Explanation

The channel initiator found a message on one of its queues that either had a format that could not be recognized or did not come from a queue manager or channel initiator.

System action

The message is put on the dead-letter queue.

System programmer response

Examine the message on the dead-letter queue to determine the originator of the message.

CSQX435E

csect-name Unable to put repository manager message, target *target-id* MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity
4

Explanation

The repository manager tried to send a message to SYSTEM.CLUSTER.COMMAND.QUEUE on another queue manager with an identifier that is *target-id*, but the MQPUT call was unsuccessful.

System action

Processing continues, but repository information may be out of date.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

Check the channel and cluster definitions on the local and target queue managers, and ensure that the channels between them are running.

When the problem is corrected, the repository information will normally be updated automatically. The REFRESH CLUSTER command can be used to be sure that the repository information is up to date.

This error may occur if the REFRESH CLUSTER REPOS(YES) command is issued against a full repository, as the full repository will then be temporarily unable to fulfil requests from other repositories until it has rebuilt the cluster. If there is more than one full repository for the cluster, the problem will resolve itself. If there is only a single full repository for the cluster, the REFRESH CLUSTER command will need to be run against all the other queue managers in the cluster to make them contact the full repository again.

CSQX436E

csect-name Unable to put repository manager message, cluster *cluster_name* MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity
4

Explanation

The repository manager tried to send a message to SYSTEM.CLUSTER.COMMAND.QUEUE on a queue manager that has the full repository for the specified cluster, but the MQPUT was unsuccessful.

System action

Processing continues, but repository information may be out of date.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

Check the channel and cluster definitions on the local and target queue managers, and ensure that the channels between them are running.

When the problem is corrected, the repository information will normally be updated automatically. The REFRESH CLUSTER command can be used to be sure that the repository information is up to date.

CSQX437E

csect-name Unable to commit repository changes

Severity
4

Explanation

The repository manager tried to commit some updates to the repository but was unsuccessful.

System action

Processing continues, but local repository information might be out of date.

System programmer response

If this occurs when the channel initiator is stopping, it can be ignored because the local repository information will normally be updated automatically when the channel initiator is restarted. If there is an isolated occurrence at other times, use the REFRESH CLUSTER command to bring the local repository information up to date.

If the problem persists, contact your IBM support center.

CSQX438E

csect-name Unable to reallocate messages, channel *channel-name* MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity

8

Explanation

The repository manager was unable to reallocate messages for the specified channel to another destination.

System action

The messages remain on the transmission queue.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

Use this information in conjunction with any preceding error messages to determine the cause of the problem. When the problem is corrected, restart the channel.

CSQX439E

csect-name Repository error for channel *channel-name*

Severity

8

Explanation

An internal error has occurred.

System action

The repository manager attempts to continue processing. Information about the error is written to the data set identified by the CSQSNAP DD statement of the channel initiator started task JCL procedure, xxxxCHIN.

System programmer response

Collect the items listed in the Problem Determination section and contact your IBM support center.

CSQX440E

csect-name FORCEREMOVE command failed, cluster *cluster_name* target *target* - repository is not on *qmgr-name*

Severity
8

Explanation

The repository manager could not process a RESET CLUSTER ACTION(FORCEREMOVE) command for the indicated cluster and target queue manager, because the local queue manager does not provide a full repository management service for the cluster.

System action

The command is ignored.

System programmer response

Reissue the command with the correct values or on the correct queue manager.

CSQX441I

csect-name FORCEREMOVE command processed, cluster *cluster_name* target *target*

Severity
0

Explanation

The repository manager successfully processed a RESET CLUSTER ACTION(FORCEREMOVE) command for the indicated cluster and target queue manager.

System action

None.

CSQX442I

csect-name Phase one of REFRESH CLUSTER has completed, cluster *cluster_namen* objects changed

Severity
0

Explanation

Phase one of REFRESH CLUSTER has completed.

Applications attempting to access cluster resources may see failures to resolve cluster resources until phase two of **REFRESH CLUSTER** is complete.

Phase two is complete once all new information has been received from other members of the cluster.

Monitor your SYSTEM.CLUSTER.COMMAND.QUEUE to determine when it has reached a consistently empty state to indicate that the refresh process has completed.

System action

None.

CSQX443I

csect-name SUSPEND QMGR command processed, cluster *cluster_namen* objects changed

Severity
0

Explanation

The repository manager successfully processed a SUSPEND QMGR command for the indicated cluster. (Where the command specified a namelist of clusters, the message is issued only for the first cluster in the namelist.)

System action

None.

CSQX444I

csect-name RESUME QMGR command processed, cluster *cluster_namen* objects changed

Severity

0

Explanation

The repository manager successfully processed a RESUME QMGR command for the indicated cluster. (Where the command specified a namelist of clusters, the message is issued only for the first cluster in the namelist.)

System action

None.

CSQX447E

csect-name Unable to backout repository changes

Severity

8

Explanation

Following an error, the repository manager tried to backout some updates to the local repository but was unsuccessful.

System action

The repository manager terminates.

System programmer response

If the repository manager subsequently restarts successfully, or if on restarting the channel initiator the repository manager subsequently starts successfully, this can be ignored.

If not, contact your IBM support center.

CSQX448E

csect-name Repository manager stopping because of errors. Restart in *n* seconds

Severity

8

Explanation

A severe error, as reported in the preceding messages, occurred during repository manager processing; the repository manager is unable to continue.

System action

The repository manager terminates. The channel initiator will try to restart it after the specified interval.

System programmer response

Correct the problem reported in the preceding messages.

CSQX449I

csect-name Repository manager restarted

Severity

0

Explanation

The repository manager restarted successfully following an error.

System action

None.

CSQX453E

csect-name FORCEREMOVE command failed, cluster *cluster_name* target *target* is not unique

Severity
8

Explanation

The repository manager could not process a RESET CLUSTER ACTION(FORCEREMOVE) command for the indicated cluster and target queue manager, because there is more than one queue manager with the specified name in the cluster.

System action

The command is ignored.

System programmer response

Reissue the command specifying the identifier (QMID) of the queue manager to be removed, rather than its name.

CSQX455E

csect-name FORCEREMOVE command failed, cluster *cluster_name* target *target* not found

Severity
8

Explanation

The repository manager could not process a RESET CLUSTER ACTION(FORCEREMOVE) command for the indicated cluster and target queue manager, because no information about that queue manager was found in the local repository.

System action

The command is ignored.

System programmer response

Reissue the command specifying the correct queue manager name or identifier.

CSQX456I

csect-name Full repository update not received, cluster *cluster_name* queue *q-name* (queue manager *qmgr-name*)

Severity
0

Explanation

The repository manager found a cluster queue that had been used sometime in the last 30 days, and for which updated information should have been received. However, no such information has been received. The queue is *q-name* in *cluster_name*, and its queue manager is *qmgr-name*.

If the queue manager is a partial repository for the queue, the updated information should have been sent from a full repository. If the queue manager is a full repository, the updated information should have been sent from the queue manager on which the queue is defined.

System action

The repository manager keeps information about this queue for a further 60 days from when the error first occurred. If information has not been sent to a full repository then this queue is not used to satisfy any new requests for cluster resources made to this full repository.

System programmer response

If the queue is still required, check that:

- The cluster channels to and from the queue manager that is the full repository for the cluster, and between there and the queue manager where the queue is located, are able to run.
- The repository managers on those queue managers have not ended abnormally.

CSQX457I

csect-name Repository available, cluster *cluster_name* channel *channel-name* sender *sender-id*

Severity

0

Explanation

The repository manager received a command from another queue manager, with identifier *sender-id*, reporting that it is once again a full repository for cluster *cluster_name*.

System action

The cluster-sender channel *channel-name* is changed so that it can be used to access the other queue manager in relation to the cluster.

CSQX458E

csect-name Unable to access repository cache exclusively, TCB= *tcb-name* has *num-registrations* outstanding registrations

Explanation

During an operation that requires exclusive access to the cache, another task was found to be registered. If the queue manager finds registrations still exist after waiting for the task to remove its registrations, the queue manager issues this message. The task preventing exclusive access to the repository cache has *num-registrations* outstanding registrations.

System action

Processing continues.

System programmer response

Determine if this task is still running or terminated. If the task is not running or if the problem persists collect the items listed in the Problem determination on z/OS section and contact your IBM support center.

CSQX459E

csect-name Cluster topic topic-name from qmgr-name rejected due to PSCLUS(DISABLED)

Explanation

Information regarding cluster topic topic-name has been sent to this queue manager over a channel from qmgr-name but the queue manager attribute PSCLUS has been set to DISABLED, indicating that Publish/Subscribe activity is not expected between queue managers in this cluster.

System action

The cluster topic definition is ignored and will not be visible from this queue manager.

System programmer response

To enable publish/subscribe clustering, alter the PSCLUS attribute on all queue managers in the cluster to ENABLED. You may also need to issue REFRESH CLUSTER and REFRESH QMGR

commands as detailed in the documentation for the PSCLUS attribute. If you are not using publish/subscribe clusters you should delete the clustered topic object, and ensure PSCLUS is DISABLED on all queue managers.

CSQX460E

csect-name Cluster cache is full

Severity

8

Explanation

No more space is available in the cluster cache area.

System action

The repository manager terminates. The channel initiator will try to restart it after the specified interval.

System programmer response

The problem may be temporary. If it persists, the queue manager must be restarted; this will cause more space to be allocated for the cluster cache area.

Consider changing the cluster cache type system parameter CLCACHE to dynamic, so that more space for the cache will be obtained automatically as required. (If you are using a cluster workload exit, ensure that it supports a dynamic cluster cache.) For information about the system parameters for the CSQ6SYSP macro, see Using CSQ6SYSP.

CSQX461I

csect-name Cluster cache entry corrected, cluster queue manager *clusqmgr-name* channel *channel-name* connection *conn-id*

Severity

4

Explanation

At channel initiator restart, the repository manager found a corrupted entry in the cluster cache. The entry has been corrected.

System action

Processing continues. The cluster channel to which the entry refers, *channel-name* using connection *conn-id*, will be available for use.

System programmer response

None. You can verify that the entry was successfully corrected by issuing the command DISPLAY CLUSQMGR(*clusqmgr-name*) on the queue manager where this message was issued.

CSQX462E

csect-name Cluster cache entry is unusable, cluster queue manager *clusqmgr-name* channel *channel-name* connection *conn-id*

Severity

8

Explanation

At channel initiator restart, the repository manager found a corrupted entry in the cluster cache which could not be corrected.

System action

The corrupted entry is ignored. The cluster channel to which it refers, *channel-name* using connection *conn-id*, will not be usable.

System programmer response

The corrupted entry must be corrected and reintroduced by issuing the command

```
ALTER CHANNEL(channel-name) CHLTYPE(CLUSRCVR)
```

on the cluster queue manager *clusqmgr-name*. You can verify that the entry was successfully reintroduced by issuing the command DISPLAY CLUSQMGR(*clusqmgr-name*) on the queue manager where this message was issued.

CSQX463E

csect-name Error accessing cluster cache entry

Severity

8

Explanation

There was an internal error when accessing a cluster cache entry.

System action

Information about the error is written to the data set identified by the CSQSNAP DD statement of the channel initiator started task JCL procedure, xxxxCHIN. The component where the error occurred (message channel agent, repository manager) usually terminates; in some cases, the end result will be that the channel initiator terminates.

System programmer response

Collect the items listed in the Problem Determination section and contact your IBM support center.

CSQX465I

csect-name New cluster topic definition inconsistent, topic *topic-name*, queue manager identifier *qmid*, attribute *attr*

Severity

4

Explanation

The definition of the cluster topic *topic-name*, defined on queue manager identifier *qmid* has different *attr* attribute values than one or more cluster topics that already exist in the cluster cache. The existing topic objects are reported by message CSQX466I.

All definitions of the same cluster topic should be identical; otherwise, problems may arise if your applications rely on one of these attributes to determine messaging behavior. For example, if an application opens a cluster topic and the different instances of the topic have different TOPICSTR values, the behavior of the message transfer depends on which instance of the topic happens to be selected when it is opened.

System action

None.

System programmer response

Alter the definitions of the topic on the various queue managers so that they have identical values for all attributes.

CSQX466I

csect-name Cluster topic definitions inconsistent, topic *topic-name*, queue manager identifier *qmid* attribute *attr*

Severity

4

Explanation

The definition of the cluster topic *topic-name*, defined on queue manager identifier *qmid* has different *attr* attribute value than a cluster topic being added to the cluster cache. The topic object being added is reported by message CSQX465I.

All definitions of the same cluster topic should be identical; otherwise, problems may arise if your applications rely on one of these attributes to determine messaging behavior. For example, if an application opens a cluster topic and the different instances of the topic have different TOPICSTR values, the behavior of the message transfer depends on which instance of the topic happens to be selected when it is opened.

System action

None.

System programmer response

Alter the definitions of the topic on the various queue managers so that they have identical values for all attributes.

CSQX467E

Repository error for topic *topic-name*, MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity

8

Explanation

The cluster repository was unable to insert or delete topic *topic-name* due to an unexpected error in the queue manager.

System action

The repository manager terminates. The channel initiator tries to restart the repository manager after an interval. See message CSQX448E for more information.

System programmer response

For more information about *mqcc* and *mqrc* completion codes (*mqrc-text* provides the MQRC in textual form), see API completion and reason codes.

Contact your IBM support center with the reason code provided for this failure.

CSQX468I

csect-name Queue manager *qmgr-uuid1* has replaced queue manager *qmgr-uuid2* in a cluster due to reuse of channel *channel-name*

Severity

0

Explanation

Queue manager *qmgr-uuid1* has joined a cluster using a cluster receiver channel with the same name as one that has already been defined by queue manager *qmgr-uuid2*. All cluster receiver channels used within a cluster must be uniquely named.

System action

Queue manager *qmgr-uuid1* uses channel *channel-name*. Queue manager *qmgr-uuid2* cannot successfully participate in the cluster while queue manager *qmgr-uuid1* is a member.

System programmer response

The use of a channel name currently associated with a different queue manager in the cluster can be intentional, for example it is possible the original queue manager has been deleted and re-created as a new queue manager. However, accidental duplication of a channel name across multiple queue managers would also result in this behavior. If this action was not intended review the configuration of the queue managers.

CSQX469E

csect-name Update not received for CLUSRCVR channel *channel-name* hosted on queue manager *qmid* in cluster *cluster_name*, expected *n* days ago, *m* days remaining

Severity

8

Explanation

The repository manager detected that the CLUSRCVR channel has not been republished by its owning queue manager. This republish action should have happened automatically *n* days ago, or in the time between then and now.

System action

The repository manager will check for this condition approximately every hour, continuing for a period of approximately *m* days from now. If an update for the CLUSRCVR channel is received during this period, these messages will stop. If no update is received, these messages will continue to be written. However, after this period has elapsed, if no update has been received, the local queue manager will discard its knowledge of this channel, and these messages will stop. You should be aware that Partial Repository queue managers in this cluster will cease to be able to use the channel at about that time.

System programmer response

There are several possible responses:

1. If the channel had been removed intentionally, and is no longer required, you should consider removing it fully via the RESET CLUSTER command.
2. There is a long-running problem with the local queue manager's CLUSRCVR in cluster *cluster_name*. If this is true, then correct the problem urgently, to ensure that updates for the cluster are received.
3. There is a long-running problem on the remote queue manager's CLUSSDR in cluster *cluster_name*. If this is true, then correct the problem urgently, to ensure that updates for the cluster are sent.
4. Check that the repository manager on the remote queue manager has not ended abnormally.
5. The remote queue manager is out of step with this queue manager, potentially due to a restore of the queue manager from a backup. The remote queue manager must issue REFRESH CLUSTER to synchronize with other queue managers in the cluster.

If the above items have been checked, and this problem persists over several days, causing repeats of this error message in the local queue manager's error logs, contact your IBM support center.

CSQX470E

csect-name Channel *channel-name* has the wrong disposition *disposition*

Severity

8

Explanation

The action you requested cannot be performed on channel *channel-name* because it has the wrong disposition. For example, the action asked for a shared channel, but its disposition is private.

System action

The requested action is not performed.

System programmer response

Check whether the channel name is specified correctly. If it is, check that:

- The channel has been defined correctly
- The transmission queue name identifies the correct queue, and that queue has the required disposition.

The disposition of an instance of a channel is **not** related to that specified by QSGDISP in the channel definition:

- A sending channel is *shared* if its transmission queue is shared, and *private* if it is not.
- A receiving channel is *shared* if it was started in response to an inbound transmission directed to the queue-sharing group, and *private* if it was started in response to an inbound transmission directed to the queue manager.

CSQX471I

csect-name nn shared channels to restart, *nn* requests issued

Severity

0

Explanation

The channel initiator is shutting down; it owns some active shared sending channels, and they have not been requested to stop. Requests to restart these channels on another queue manager have been issued as shown.

System action

The channel initiator shutdown processing continues.

System programmer response

If the numbers in the message differ, the channel initiator was not able to issue restart requests for all the channels. In this case, use the DISPLAY CHSTATUS command to determine which channels are still owned by the queue manager for the channel initiator that is shutting down, and which therefore have not been restarted, and restart them manually as required.

CSQX475I

csect-name Channel *channel-name* adopted, connection *conn-id*

Severity

0

Explanation

Channel *channel-name*, which was orphaned because of a communications error, has been adopted by a new instance of the channel, from connection *conn-id*.

System action

Processing continues.

CSQX476E

csect-name Channel *channel-name* is active on *qmgr-name*, shared status entry found

Severity

8

Explanation

An operation was requested on a channel that is active. Because the channel is shared, it might be active on another queue manager. If the channel is a receiver, a previous instance of it might have been orphaned and therefore still be active.

System action

The request fails.

System programmer response

For operations other than starting the channel, either stop the channel manually, or wait for it to terminate, and try the operation again. It might be necessary to use MODE(FORCE) to stop the channel manually if the Adopt MCA function is not being used. Using the Adopt MCA function avoids the need for manual intervention to handle orphaned receiver channels.

If the channel is not running on the named queue manager, then there is an orphaned shared status entry, which mighty be because a loss of connectivity to Db2 occurred. If the problem persists, contact your IBM support center.

CSQX477E

csect-name Channel *channel-name* is active, transmission queue *queue-name* in use on *qmgr-name*

Severity
8

Explanation

An operation was requested on a channel that is active. The queue *queue-name* named as a transmission queue in the channel definition for *channel-name* is in use on another member of the queue-sharing group, *qmgr-name*.

System action

The request fails.

System programmer response

Do the following, as appropriate:

- Check if the channel is already running
- Check if another channel is using the queue by using the DISPLAY QSTATUS command
- Ensure the queue name is specified correctly in the channel definition
- Alter the queue usage attribute of the queue to that of a transmission queue.

If the channel is already running, for operations other than starting the channel, either stop the channel manually, or wait for it to terminate, and retry the operation. It may be necessary to use MODE(FORCE) to stop the channel manually if the Adopt MCA function is not being used. Using the Adopt MCA function will avoid the need for manual intervention to handle orphaned receiver channels.

CSQX478E

csect-name Channel *channel-name* is active on *qmgr-name*, connection tag in use

Severity
8

Explanation

An operation was requested on a channel that is active. The connection tag used to serialize the channel within the queue-sharing group is currently in use. Because the channel is shared, it might be active on another queue manager. If the channel is a receiver, a previous instance of it might have been orphaned and therefore still be active.

In addition to the CSQX478E for a shared channel, another possible symptom is CSQX514E: *csect-name* Channel channel-name is active on qmgr-name. The new instance of the channel is starting with a different IP address from the running instance. If the sender's IP address changed or might translate into more than one address, set ADOPTCHK to QMNAME using the ALTER QMGR command. For example, /cpf ALTER QMGR ADOPTCHK(QMNAME) where "cpf" is the command prefix for the queue manager subsystem.

System action

The request fails.

System programmer response

For operations other than starting the channel, either stop the channel manually, or wait for it to terminate, and try the operation again. It might be necessary to use MODE(FORCE) to stop the channel manually if the Adopt MCA function is not being used. Using the Adopt MCA function avoids the need for manual intervention to handle orphaned receiver channels.

CSQX479E

csect-name Channel *channel-name* is active on *qmgr-name*, shared channel adoption failed

Severity

8

Explanation

An attempt was made to adopt channel *channel-name*, which was orphaned because of a communications error. It failed, either because the channel could not be stopped or because a response was not received from the queue manager *qmgr-name*.

System action

The request fails, and the orphaned channel might remain active.

System programmer response

Investigate any preceding error messages to discover why the adopt failed. Either stop the channel manually, or wait for it to terminate, and try the operation again. It might be necessary to use MODE(FORCE) to stop the channel manually.

CSQX482E

csect-name Shared channel function not available

Severity

8

Explanation

During the execution of a channel command, or during shared channel processing, an internal function required by the channel initiator was found to be unavailable.

System action

The channel command fails or the channel stops.

System programmer response

Check that the Db2 tables required by IBM MQ are correctly defined, and restart the queue manager and Db2 if necessary. If these appear to be running correctly, display the information in the shared channel status (CSQ.ADMIN_B_SCST) and the shared synchronization key (CSQ.ADMIN_B_SSKT) Db2 tables, and contact your IBM support center for further assistance. For further information, and for details of a sample job (CSQ45STB) which shows the information in the Db2 tables, see Problem determination on z/OS.

CSQX483E

csect-name Db2 not available

Severity
8

Explanation

Because Db2 is not available, or is no longer available, the channel initiator cannot do processing for a shared channel.

System action

The channel command fails or the channel stops.

System programmer response

Use the preceding messages on the z/OS console to investigate why Db2 is not available, and restart it if necessary.

CSQX484E

csect-name Error accessing Db2

Severity
8

Explanation

Because there was an error in accessing Db2, the channel initiator cannot do processing for a shared channel.

System action

The channel command fails or the channel stops.

System programmer response

Resolve the error reported in the preceding messages.

CSQX485E

csect-name Shared channel status error

Severity
8

Explanation

During the execution of a channel command, or during shared channel processing, shared channel status or shared synchronization key information, held in Db2, was found to be corrupted.

System action

The channel command fails or the channel stops.

System programmer response

Check that the Db2 tables required by IBM MQ are correctly defined, and restart Db2 if necessary. If Db2 appears to be running correctly, display the information in the shared channel status (CSQ.ADMIN_B_SCST) and the shared synchronization key (CSQ.ADMIN_B_SSKT) Db2 tables, and contact your IBM support center for further assistance. For further information, and for details of a sample job (CSQ45STB) which shows the information in the Db2 tables, see Problem determination on z/OS.

CSQX486E

csect-name Shared channel *channel-name* definitions inconsistent

Severity
8

Explanation

The definition of a shared channel has differing attribute values on the various queue managers in the queue-sharing group. For example, if the type of the channel differs start or stop requests cannot operate correctly.

System action

The request fails.

System programmer response

Change the definitions of the channel so that they are the same on all the queue managers. If the channel type needs changing, you must delete and then redefine the channel.

CSQX489E

csect-name Maximum instance limit *limit* exceeded, channel *channel-name* connection *conn-id*

Severity
8

Explanation

There are too many instances of the channel *channel-name* running to be able to start another. The maximum number allowed is *limit* and is specified in the MAXINST channel attribute.

System action

The channel does not start.

System programmer response

Wait for some of the operating channels to terminate before restarting the channel, or use the ALTER CHANNEL command to increase MAXINST.

CSQX490E

csect-name Maximum client instance limit *limit* exceeded, channel *channel-name* connection *conn-id*

Severity
8

Explanation

There are too many instances of the channel *channel-name* running from the connection *conn-id* to be able to start another. The maximum number allowed is *limit* and is specified in the MAXINSTC channel attribute.

System action

The channel does not start.

System programmer response

Wait for some of the operating channels to terminate before restarting the channel, or use the ALTER CHANNEL command to increase MAXINSTC.

CSQX496I

csect-name Channel *channel-name* stopping because of request by remote exit

Severity
0

Explanation

The channel is closing because the user channel exit at the remote end requested it.

System action

The channel stops. The associated transmission queue might be set to GET(DISABLED) and triggering turned off. For auto-defined channels, the channel does not start.

System programmer response

Note that this puts the channel into STOPPED state. A START CHANNEL command must be issued to restart it.

CSQX498E

csect-name Invalid MQCD field *field-name*, value=*nnn* (X*xxx*)

Severity

8

Explanation

The MQCD structure returned by the channel auto-definition exit had an invalid value in the indicated field. The value is shown in decimal (*nnn*) and hexadecimal (*xxx*).

System action

The channel is not defined.

System programmer response

Correct the channel auto-definition exit.

CSQX500I

csect-name Channel *channel-name* started connection *conn-id*

Severity

0

Explanation

The specified channel has been started.

If *channel-name* is an inbound channel (indicated by *csect-name* containing CSQXRESP) then it was started from connection *conn-id*. If *channel-name* is an outbound channel then *conn-id* will be omitted.

System action

Processing continues.

CSQX501I

csect-name Channel *channel-name* no longer active connection *conn-id*

Severity

0

Explanation

Channel *channel-name* terminated. It is now inactive if it terminated normally when the disconnect interval expired, or stopped if it terminated because of an error or a STOP CHANNEL command.

If *channel-name* was an inbound channel (indicated by *csect-name* containing CSQXRESP) then it was started from connection *conn-id*. If *channel-name* was an outbound channel then *conn-id* will be omitted.

System action

Processing continues.

System programmer response

If the channel is stopped, resolve any error, and issue a START CHANNEL command to restart the channel.

CSQX502E

csect-name Action not allowed for channel *chl-type*(*channel-name*)

Severity

8

Explanation

The action you requested cannot be performed on channel *channel-name*. Some actions are only valid for certain channel types. This channel is a *chl-type* channel type. For example, you can only ping a channel from the end sending the message.

System action

The requested action is not performed.

System programmer response

Check whether the channel name is specified correctly. If it is, check that:

- The channel has been defined correctly
- The connection name identifies the remote end correctly
- For a cluster-receiver channel, the connection name does not specify a generic address
- For TCP/IP connections, the port number specified by the local channel matches that used by the listener at the remote queue manager.

You can use the *csect-name* to determine the action that failed:

Table 425.

csect-name	**action**
CSQXPING	PING CHANNEL
CSQXRESE	RESET CHANNEL
CSQXRESO	RESOLVE CHANNEL
CSQXSTOP	STOP CHANNEL

CSQX503E

csect-name Negotiation failed for channel *channel-name*, type=*last-segment-type*, data=*xxx*, connection *conn-id*

Severity

8

Explanation

Channel *channel-name* could not be established due to a negotiation failure between the local queue manager and the remote end using connection *conn-id*. The last control data received was of type *last-segment-type* and is accompanied by data indicating the error.

System action

The channel is not started.

System programmer response

Examine the console log for the remote end for messages explaining the cause of the negotiation failure.

CSQX504E

csect-name Local protocol error, channel *channel-name* type=*type* data=*xxx*

Severity

8

Explanation

During communications with the remote end, the local message channel agent for channel *channel-name* detected a protocol error.

type shows the type of error that occurred and the incorrect value is shown by *xxx*.

00000001
Missing channel. Define a remote channel. See message CSQX520E for more information.

00000002
Incorrect channel type. Check your definitions. See message CSQX547E for more information.

00000003
Queue manager unavailable. Check the queue manager. See message CSQX524E for more information.

00000004
Message sequence error. Investigate the problem and reset the channel. See message CSQX526E for more information.

00000005
Queue manager terminating. This message might be for information only. See message CSQX525E for more information.

00000006
Unable to store. This message might be for information only. See messages CSQX527E and CSQX544E for more information. Also, check the error log for the remote system. Messages might end up on the remote dead-letter queue.

00000007
User closed. This message might be for information only. See message CSQX528I for more information. The channel is stopping, either because of a STOP CHANNEL command, or the channel initiator is stopping.

00000008
Timeout expired. This message might be for information only. During an MQGET_WAIT the DISCINT times out, so the channel is closed.

00000009
Target queue unknown - contact your IBM support center.

0000000A
Incorrect segment type - contact your IBM support center.

0000000B
Incorrect segment length. Check the remote client. Either the client has sent a segment larger than the buffer it requested, or the requested buffer exceeds the combined payload and header limits.

0000000C
Data not valid - contact your IBM support center.

0000000D
Unexpected segment - contact your IBM support center.

0000000E
Unexpected ID - contact your IBM support center.

0000000F
Unexpected MSH - contact your IBM support center.

00000010
General protocol problem - contact your IBM support center.

00000011
Batch failure - contact your IBM support center.

00000012
Incorrect message length - contact your IBM support center.

00000013
Incorrect segment number - contact your IBM support center.

00000014
Security failure - contact your IBM support center.

00000015
Wrap value error. Use the command ALTER CHANNEL SEQWRAP to align the local or remote channel sequence wrap values. See message CSQX505E for more information.

00000016
Channel unavailable. Check if the remote channel is STOPPED, or otherwise unavailable. See message CSQX558E for more information.

00000017
Closed by exit - contact your IBM support center.

00000018
Cipher spec error. Confirm the SSLCIPH of the channel, and its compatibility if the remote side has been set to SSLFIPS(YES). See message CSQX635E for more information.

00000019
Peer name error. Confirm that SSLPEERNAME on this channel, matches the distinguished name in the certificate of the remote side. See message CSQX636E for more information.

0000001A
SSL/TLS client certificate error. Check the remote channel and see if a certificate has been supplied for SSL/TLS negotiation. See message CSQX637E for more information.

0000001B
RMT RSRCS in recovery. This message is for information only; the condition is transient.

0000001C
SSL/TLS refreshing. This message is for information only; the condition is transient.

0000001D
HOBJ not valid - contact your IBM support center.

0000001E
Conversion ID error - contact your IBM support center.

0000001F
Socket action type not valid - contact your IBM support center.

00000020
Standby queue manager not valid - contact your IBM support center.

00000021
Maximum transmission size not valid. Increase the remote RECEIVER attributes for transmission unit size.

00000022
FAP level not valid - contact your IBM support center.

00000023
Maximum permitted conversions exceeded. The SHARECNV limit has been exceeded. Investigate the remote client and increase the value of SHARECNV.

00000024
Password protection error - contact your IBM support center.

System action

The channel stops. The associated transmission queue might be set to GET(DISABLED) and triggering turned off.

System programmer response

Examine the console log to determine the cause of the failure. This might occur after the channel initiator or queue manager is stopped forcibly or ends abnormally. If it occurs in other cases, contact your IBM support center to report the problem.

CSQX505E

csect-name Sequence wrap values differ, channel *channel-name* local=*local-seqno* remote=*remote-seqno*

Severity
8

Explanation

The sequence number wrap value for channel *channel-name* is *local-seqno*, but the value specified at the remote end is *remote-seqno*. The two values must be the same before the channel can be started.

System action

The channel does not start.

System programmer response

Change either the local or remote channel definition so that the values specified for the message sequence number wrap value are the same.

CSQX506E

csect-name Message receipt confirmation not received for channel *channel-name*

Severity
8

Explanation

The remote end did not accept the last batch of messages.

System action

Channel *channel-name* stops. The associated transmission queue may be set to GET(DISABLED) and triggering turned off.

System programmer response

Determine why the remote end did not accept the last batch of messages. Resolve the problem and restart the channel.

CSQX507E

csect-name Channel *channel-name* is in-doubt, connection *conn-id* (queue manager *qmgr-name*)

Severity
8

Explanation

Channel *channel-name* is in-doubt with the remote end using connection *conn-id*. The associated remote queue manager is *qmgr-name*; in some cases its name cannot be determined and so is shown as '????'.

System action

The requested operation does not complete.

System programmer response

Examine the status of the channel, and either restart a channel to resolve the in-doubt state, or use the RESOLVE CHANNEL command to correct the problem manually.

CSQX511I

csect-name Channel *channel-name* started, connection *conn-id*

Severity
0

Explanation

The specified SVRCONN channel has been started from connection *conn-id*.

System action

Processing continues.

CSQX512I

csect-name Channel *channel-name* no longer active, connection *conn-id*

Severity
0

Explanation

SVRCONN Channel *channel-name* terminated. It is now inactive if it terminated normally when the disconnect interval expired, or stopped if it terminated because of an error or a STOP CHANNEL command.

The SVRCONN *channel-name* was started from connection *conn-id*.

System action

Processing continues.

System programmer response

If the SVRCONN channel is stopped, resolve any error, and issue a START CHANNEL command to restart the channel.

CSQX513E

csect-name Current channel limit exceeded channel *channel-name* connection *conn-id*

Severity
8

Explanation

There are too many channels current to be able to start another. The maximum number allowed is specified in the MAXCHL queue manager attribute. Current channels include stopped and retrying channels as well as active channels.

If *channel-name* was an inbound channel (indicated by *csect-name* containing CSQXRESP) then it was started from connection *conn-id*. If *channel-name* was an outbound channel then *conn-id* will be omitted.

System action

The channel does not start.

System programmer response

Wait for some of the operating channels to terminate before restarting the channel, or use the ALTER QMGR command to increase **MAXCHL**. A change that increases **MAXCHL** will not be effective until the channel initiator has been stopped and restarted. If many of the currently operating channels are server-connection channels, consider limiting the number of those using **MAXINST** or **MAXINSTC** attributes of a server-connection channel. See Server-connection channel limits for more details.

CSQX514E

csect-name Channel *channel-name* is active on *qmgr-name*

Severity

8

Explanation

An operation was requested on a channel that is active. If the channel is shared, it might be active on another queue manager. If the channel is a receiver, a previous instance of it might have been orphaned and therefore still be active.

System action

The request fails.

System programmer response

For operations other than starting the channel, either stop the channel manually, or wait for it to terminate, and try the operation again. It might be necessary to use MODE(FORCE) to stop the channel manually if the Adopt MCA function is not being used. Using the Adopt MCA function avoids the need for manual intervention to handle orphaned receiver channels.

CSQX515I

csect-name Channel *channel-name* changed

Severity

0

Explanation

The channel for which information has been requested is a new instance of the channel. The previous channel instance has ended.

System action

The information shown is for the new channel instance.

CSQX516E

csect-name Error accessing synchronization data, RC=*return-code*

Severity

8

Explanation

There was an error when accessing the channel synchronization data.

If the return code is of the form 10009*nnn* or 20009*nnn*, it is a distributed queuing message code. This is generally associated with message CSQX*nnn*E, which will normally be issued previously.

Otherwise the most likely cause is a shortage of storage.

System action

The channel stops. The associated transmission queue may be set to GET(DISABLED) and triggering turned off.

In some cases, the channel initiator will stop as well.

System programmer response

If the return code is a distributed queuing message code, see the corresponding message explanation for more information. Where no such message is described, see "Distributed queuing message codes" on page 5224 for the corresponding message number.

Restart the channel or the channel initiator. If the problem persists, contact your IBM support center.

CSQX517E

csect-name Error in *q-name* - channel *channel-name* repeated

Severity

8

Explanation

There was more than one set of synchronization information in *q-name* for an instance of channel *channel-name*. This is probably because the channel is a receiver channel, and there are two sender channels with the same name on different queue managers within the same network address that have communicated with it.

System action

The first set of synchronization information for the channel instance is used, and any others are ignored. Errors may occur if the channel is used.

System programmer response

Avoid using the channel. Remove the extra sets of information from the channel synchronization queue, and rename channels so that they have unique names.

If this does not resolve the problem, contact your IBM support center.

CSQX519E

csect-name Channel *channel-name* not defined connection *remote-conn-id*

Severity

8

Explanation

The channel initiator could not find a definition of channel *channel-name*.

The associated remote connection name is *remote-conn-id*. If the request to use the channel is not from an inbound connection, or the remote connection name cannot be determined, *remote-conn-id* will be shown as '????'.

System action

The requested operation fails.

System programmer response

Ensure that the name is specified correctly and the channel definition is available.

The message can also be issued if an automatically defined cluster sender channel (CLUSSDRA) has been deleted as a result of issuing a REFRESH CLUSTER command and a putting application still has a queue object open which is using the channel.

CSQX520E

csect-name Remote channel *channel-name* not defined

Severity

8

Explanation

There is no definition of channel *channel-name* at the remote end.

System action

The channel does not start.

System programmer response

Add an appropriate channel definition at the remote end, and retry the operation.

CSQX523E

csect-name Remote protocol error, channel *channel-name* type=*type* data=*xxx*

Severity

8

Explanation

During communications with the remote end, the remote message channel agent for channel *channel-name* detected a protocol error. *type* shows the type of error that occurred:

0000000A
Incorrect segment type

0000000B
Incorrect length

0000000C
Invalid data

0000000D
Invalid segment

0000000E
Invalid ID

0000000F
Invalid MSH

00000010
General error

00000011
Batch failure

00000012
Incorrect message length

00000013
Incorrect segment number

The data associated with the error (for example, the incorrect value) is shown by *xxx*.

System action

The channel stops. The associated transmission queue might be set to GET(DISABLED) and triggering turned off.

System programmer response

Examine the console log for the remote end to determine the cause of the failure. This might occur after the channel initiator or queue manager is stopped forcibly or ends abnormally. If it occurs in other cases, contact your IBM support center.

CSQX524E

csect-name Remote queue manager unavailable for channel *channel-name*

Severity

8

Explanation

Channel *channel-name* cannot start because the remote queue manager is not currently available.

System action

The channel does not start

System programmer response

Either start the remote queue manager, or retry the operation later.

CSQX525E

csect-name Channel *channel-name* closing because remote queue manager *qmgr-name* is stopping

Severity

8

Explanation

Channel *channel-name* is closing because the remote queue manager *qmgr-name* is stopping. In some cases, the remote queue manager name cannot be determined and so is shown as '????'.

System action

The channel stops. The associated transmission queue might be set to GET(DISABLED) and triggering turned off.

System programmer response

Investigate why the remote queue manager is stopping, if it was not expected.

CSQX526E

csect-name Message sequence error for channel *channel-name*, sent=*msg-seqno* expected=*exp-seqno*

Severity

8

Explanation

The local queue manager does not agree with the remote end on the next message sequence number for channel *channel-name*. The message is normally issued at both the sending and receiving end: at the sending end, *msg-seqno* and *exp-seqno* are unpredictable; at the receiving end, a message had sequence number *msg-seqno* but sequence number *exp-seqno* was expected.

System action

The channel stops. The associated transmission queue might be set to GET(DISABLED) and triggering turned off.

System programmer response

Determine the cause of the inconsistency. It could be that the synchronization information has become damaged, or has been backed out to a previous version. If the problem cannot be resolved, the sequence number can be reset manually at the sending end of the channel using the RESET CHANNEL command. (For some queue managers, it might be necessary to issue the RESET CHANNEL command at the receiving end as well.)

CSQX527E

csect-name Unable to send message for channel *channel-name*

Severity

8

Explanation

The remote end cannot receive the message that is being sent for channel *channel-name*.

System action

The channel stops. The associated transmission queue may be set to GET(DISABLED) and triggering turned off.

System programmer response

Examine the console log for the remote end to determine why the message cannot be received, and then restart the channel.

CSQX528I

csect-name Channel *channel-name* stopping

Severity

0

Explanation

The channel is closing because a STOP CHANNEL command was issued, or because the channel initiator is stopping.

System action

The channel stops. The associated transmission queue may be set to GET(DISABLED) and triggering turned off.

System programmer response

Note that a STOP CHANNEL command puts the channel into STOPPED state. A START CHANNEL command must be issued to restart it.

CSQX531E

csect-name Transmission queue *q-name* for *channel-name* has wrong usage type

Severity

8

Explanation

Queue *q-name* is named as a transmission queue in the channel definition for *channel-name*, but it is not a transmission queue.

System action

The channel does not start.

System programmer response

Ensure the queue name is specified correctly in the channel definition. If it is, alter the queue usage attribute of the queue to that of a transmission queue.

CSQX533I

csect-name Channel *channel-name* is already in requested state

Severity
0

Explanation

A request to stop channel *channel-name* was made, but the channel was already in the specified state, or in the process of reaching that state.

System action

The request is ignored.

CSQX534E

csect-name Channel *channel-name* is stopped

Severity
4

Explanation

The operation requested cannot be performed because the channel is currently stopped.

System action

The request is ignored.

System programmer response

Issue a START CHANNEL command to restart the channel.

CSQX535E

csect-name Channel *channel-name* stopping because exit *exit-name* is not valid

Severity
8

Explanation

The user exit *exit-name* specified for channel *channel-name* is not valid.

System action

The channel stops. The associated transmission queue might be set to GET(DISABLED) and triggering turned off. For auto-defined channels, the channel does not start.

System programmer response

Ensure that the user exit name is specified correctly in the channel definition, and that the user exit program is correct and available. The channel initiator loads exits from the library data sets under the CSQXLIB DD statement of its started task JCL procedure xxxxCHIN.

CSQX536I

csect-name Channel *channel-name* stopping because of request by exit *exit-name*

Severity
0

Explanation

The channel is closing because the user channel exit *exit-name* requested it.

System action

The channel stops. The associated transmission queue may be set to GET(DISABLED) and triggering turned off. For auto-defined channels, the channel does not start.

System programmer response

Note that this puts the channel into STOPPED state. A START CHANNEL command must be issued to restart it.

CSQX539E

csect-name Channel *channel-name* for queue *q-name* is not available

Severity

8

Explanation

A trigger message was received to start a channel *channel-name* to process the transmission queue *q-name*. However, the channel initiator could not find a defined and available channel to start.

System action

The channel does not start.

System programmer response

Ensure that there is a channel defined to process the transmission queue, and that it is not stopped.

CSQX540E

csect-name Unable to commit batch, channel *channel-name* MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity

8

Explanation

An MQCMIT call for the queue associated with channel *channel-name* was unsuccessful.

System action

The channel stops. The associated transmission queue might be set to GET(DISABLED) and triggering turned off.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQX541E

csect-name Invalid CCSIDs for data conversion, *ccsid1* and *ccsid2*

Severity

8

Explanation

Either the local coded character set identifier (CCSID) or the target CCSID is not valid, or is not currently supported, or conversion between the two CCSIDs involved is not supported. (The name of the channel cannot be determined because the invalid CCSID prevents the necessary data conversion being done.)

System action

The channel stops. The associated transmission queue may be set to GET(DISABLED) and triggering turned off.

System programmer response

Ensure that the CCSIDs are valid and that conversion between them is supported. For information about the CCSIDs that are supported, see Codeset names and CCSIDs.

CSQX544E

csect-name Messages for channel *channel-name* sent to remote dead-letter queue

Severity
4

Explanation

During the processing of channel *channel-name*, one or more messages have been put the dead-letter queue at the remote queue manager.

System action

Processing continues.

System programmer response

Examine the contents of the dead-letter queue. Each message is contained in a structure that describes why the message was put to the queue, and to where it was originally addressed.

CSQX545I

csect-name Channel *channel-name* closing because disconnect interval expired

Severity
0

Explanation

The channel is closing because no messages arrived on the transmission queue within the disconnect interval.

System action

The channel ends normally.

CSQX547E

csect-name Remote channel *channel-name* has the wrong type

Severity
8

Explanation

The operation requested cannot be performed because channel *channel-name* on the remote end is not of a suitable type. For example, if the local channel is defined as a sender the remote queue manager must define its corresponding channel as either a receiver or requester.

System action

The requested operation is not performed.

System programmer response

Check that the channel name is specified correctly. If it is, check that:

- The channel definition on the remote end has an appropriate channel type
- The connection name of the local channel identifies the remote end correctly
- For cluster channels, the connection names do not specify a generic address
- For TCP/IP connections, the port number specified by the local channel matches that used by the listener at the remote queue manager.

CSQX548E

csect-name Messages sent to local dead-letter queue, channel *channel-name* reason=*mqrc* (*mqrc-text*)

Severity
4

Explanation

During the processing of channel *channel-name*, one or more messages have been put the dead-letter queue at the local queue manager. *mqrc* shows why, and is one of the following:

- an MQRC_* reason code from an MQPUT or MQPUT1 call
- an MQFB_* feedback code.

System action

Processing continues.

System programmer response

Examine the contents of the dead-letter queue. Each message is contained in a structure that describes why the message was put to the queue, and to where it was originally addressed.

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

For information about MQFB_* feedback codes see the MQMD description in MQMD - Message descriptor.

CSQX549E

csect-name Queue *q-name* for channel *channel-name* is get-inhibited

Severity
8

Explanation

An MQGET failed because the transmission queue had been previously inhibited for gets.

System action

The channel stops. The associated transmission queue might have triggering turned off.

System programmer response

Change the definition of the transmission queue so that it is not inhibited for MQGET calls.

CSQX551E

csect-name Action not supported, channel *channel-name* connection *conn-id* (queue manager *qmgr-name*)

Severity
8

Explanation

The operation requested for channel *channel-name* is not supported by the remote end using the connection *conn-id*. The associated remote queue manager is *qmgr-name*; in some cases its name cannot be determined and so is shown as '????'.

System action

The channel stops. The associated transmission queue may be set to GET(DISABLED) and triggering turned off.

System programmer response

Check that the connection name parameter is specified correctly and that the levels of the queue managers in use are compatible.

CSQX552E

csect-name Security exit data for channel *channel-name* not received, connection *conn-id*

Severity

8

Explanation

The local security user channel exit for channel *channel-name* requested data from the remote security user channel exit, but no data was received. The remote connection was *conn-id*.

System action

The channel stops. The associated transmission queue may be set to GET(DISABLED) and triggering turned off.

System programmer response

Ensure that the security exit for the channel on the remote end has been defined correctly and is available. If it is, check that the exit program operates correctly.

CSQX558E

csect-name Remote channel *channel-name* not available

Severity

8

Explanation

The channel *channel-name* at the remote end is currently stopped or is otherwise unavailable. For example, there might be too many channels current to be able to start it.

System action

The channel does not start.

System programmer response

This might be a temporary situation, and the channel will try again. If not, check the status of the channel at the remote end. If it is stopped, issue a START CHANNEL command to restart it. If there are too many channels current, either wait for some of the operating channels to terminate, or stop some channels manually, before restarting the channel.

CSQX565E

csect-name No dead-letter queue for *qmgr-name*, channel *channel-name*

Severity

8

Explanation

A message could not be delivered normally and there is no dead-letter queue defined for queue manager *qmgr-name*.

You can get this message with a cluster sender channel during message reallocation. During reallocation, the message is got from the transmission queue and put back again. If the transmission queue is full, then the put fails and tries writing the message to the dead letter queue. If the dead letter queue does not exist, message CSQX565E is produced, and the reallocation changes are rolled back. Reallocation does not happen until the queue full problem is resolved.

System action

The channel stops, except in the case where nonpersistent messages are being sent and the NPMCLASS attribute of the channel is set to FAST, when processing continues. The associated transmission queue may be set to GET(DISABLED) and triggering turned off.

System programmer response

Correct the problem that prevented the message from being delivered normally, or define a dead-letter queue for the remote queue manager.

CSQX567E

csect-name Listener unable to register to APPC/MVS, TRPTYPE=LU62 INDISP=*disposition* RC=*return-code* reason=*reason*

Severity

8

Explanation

While starting, the specified LU 6.2 listener could not register as an APPC/MVS server. The return code from APPC/MVS allocate services was *return-code* and the associated reason code was *reason* (both in hexadecimal).

System action

The listener is not started.

System programmer response

See "Communications protocol return codes" on page 5211 for the cause of the return code from APPC/MVS allocate services, and the *Writing Servers for APPC/MVS* manual for more information. Check that the LUNAME queue manager attribute is the same as the PARTNER_LU value for the APPC/MVS symbolic destination used by the listener.

CSQX568E

csect-name Listener unable to unregister from APPC/MVS, TRPTYPE=LU62 INDISP=*disposition* RC=*return-code* reason=*reason*

Severity

8

Explanation

While stopping, the specified LU 6.2 listener could not unregister as an APPC/MVS server. The return code from APPC/MVS allocate services was *return-code* and the associated reason code was *reason* (bothin hexadecimal).

System action

The listener stops. It may not be possible to restart it.

System programmer response

See "Communications protocol return codes" on page 5211 for the cause of the return code from APPC/MVS allocate services and the *Writing Servers for APPC/MVS* manual for more information.

CSQX569E

csect-name Channel *channel-name* exceeded TCP/IP channel limit

Severity

8

Explanation

The number of current TCP/IP channels is the maximum allowed; another channel cannot be started. Current channels include stopped and retrying channels as well as active channels. The

maximum allowed is specified in the TCPCHL queue manager attribute, but may be reduced if a dispatcher fails, or if TCP/IP resources are restricted (as reported by message CSQX118I).

System action

The channel does not start.

System programmer response

If the maximum allowed is zero, TCP/IP communications are not allowed, and no TCP/IP channels can be started. If the maximum allowed is non-zero, wait for some of the operating channels to terminate before restarting the channel, or use the ALTER QMGR command to increase TCPCHL.

CSQX570E

csect-name Channel *channel-name* exceeded LU 6.2 channel limit

Severity

8

Explanation

The number of current LU 6.2 channels is the maximum allowed; another channel cannot be started. Current channels include stopped and retrying channels as well as active channels. The maximum allowed is specified in the LU62CHL queue manager attribute, but may be reduced if a dispatcher fails.

System action

The channel does not start.

System programmer response

If the maximum allowed is zero, LU 6.2 communications are not allowed, and no LU 6.2 channels can be started. If the maximum allowed is non-zero, wait for some of the operating channels to terminate before restarting the channel, or use the ALTER QMGR command to increase LU62CHL.

V 9.0.0

CSQX571E

csect-name Error from PKCS #11 callable service '*func*', RC=*return-code*, reason=*reason*

Severity

8

Explanation

An attempt to use PKCS #11 callable service *func* failed.

System action

The component where the error occurred (message channel agent, supervisor) will continue but the feature being used will be unavailable.

If *func* is CSFPPRF (Pseudo-random function) the feature affected is password protection. If this feature is not being used then this error can be ignored. If this occurs at channel initiator startup, the password protection algorithm uses STCK instead.

System programmer response

For information about the *return-code* and *reason* from the PKCS #11 callable service, see the section on ICSF and cryptographic coprocessor return and reason codes in the *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

For more information about Integrated Cryptographic Service Facility (ICSF), see Using ICSF.

CSQX572E

csect-name Channel *channel-name* stopping because message header is not valid

Severity

8

Explanation

During the processing of channel *channel-name*, a message was found that had an invalid header. The dead-letter queue was defined as a transmission queue, so a loop would have been created if the message had been put there.

System action

The channel stops. The associated transmission queue may be set to GET(DISABLED) and triggering turned off.

System programmer response

Correct the problem that caused the invalid message header.

CSQX573E

csect-name Channel *channel-name* exceeded active channel limit

Severity

8

Explanation

There are too many channels active (transmitting messages) to be able to start another. The maximum number allowed is specified in the ACTCHL queue manager attribute.

System action

The channel does not start.

System programmer response

Either wait for some of the operating channels to terminate, or stop some channels manually, before restarting the channel, or use the ALTER QMGR command to increase ACTCHL. A change that increases ACTCHL will not be effective until the channel inititator has been stopped and restarted.

CSQX574I

csect-name Channel *channel-name* can now start

Severity

0

Explanation

The specified channel was waiting to start, because there were too many channels active (transmitting messages) to be able to start another. One or more of the active channels has terminated, so this channel can now start.

Note: This message is not itself issued, although the corresponding event is generated.

CSQX575E

csect-name Negotiation failed for channel

Severity

8

Explanation

A channel between the local queue manager and the remote end could not be established due to a negotiation failure. The failure was such that the channel name could not be determined: for example, data conversion between the coded character set identifiers (CCSIDs) used by the local and remote ends might not have been possible.

System action

The channel is not started.

System programmer response

Examine the console log for the remote end for messages explaining the cause of the negotiation failure.

V 9.0.0

CSQX576E

csect-name ICSF is not available

Severity

8

Explanation

In order to generate entropy for the password protection algorithm, a call to CSFPPRF (Pseudo-random function) is made which requires the Integrated Cryptographic Service Facility (ICSF) to be available. ICSF was found not to be available.

System action

The password protection algorithm uses STCK instead.

System programmer response

If password protection is being used, start ICSF. If it is not being used, this error message can be ignored.

CSQX578E

csect-name Unable to save status for channel *channel-name*

Severity

8

Explanation

An internal error has occurred.

System action

The channel stops. The associated transmission queue may be set to GET(DISABLED) and triggering turned off.

Information about the error is written to the data set identified by the CSQSNAP DD statement of the channel initiator started task JCL procedure, xxxxCHIN.

System programmer response

Collect the items listed in the Problem Determination section and contact your IBM support center.

CSQX599E

csect-name Channel *channel-name* ended abnormally connection *conn-id*

Severity

8

Explanation

Channel *channel-name* ended abnormally because of a severe problem, as reported in the preceding messages.

If *channel-name* is an inbound channel (indicated by *csect-name* containing CSQXRESP) then it was started from connection *conn-id*. If *channel-name* is an outbound channel then *conn-id* will be omitted.

System action

The channel stops. The associated transmission queue might be set to GET(DISABLED) and triggering turned off.

System programmer response

Investigate the problem reported in the preceding messages. For more information see, Problem determination in DQM.

CSQX608E

csect-name Remote resources in recovery for channel *channel-name*

Severity

8

Explanation

Channel *channel-name* cannot start because resources at the remote queue manager are being recovered.

System action

The channel does not start.

System programmer response

Restart the channel at a later time. If the problem persists examine the console log for the remote end for messages explaining the cause of the problem. This includes an instance of CSQX609E with more details.

CSQX609E

csect-name Resources in recovery, channel *channel-name* MQCC=*mqcc* MQRC=*mqrc* (*mqrc-text*)

Severity

8

Explanation

The message channel agent for the channel could not connect to the queue manager because resources are being recovered.

System action

The channel does not start.

System programmer response

Refer to API completion and reason codes for information about *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form), which come from an MQCONNX request.

CSQX613I

csect-name Channel *channel-name* instance is already in requested state

Severity

0

Explanation

A request to stop a particular instance of channel *channel-name* was made (by specifying a connection name or a remote queue manager name), but the channel instance was already in the specified state, or in the process of reaching that state.

This error will also apply if an attempt is made to stop a SVRCONN channel using the QMNAME parameter. In this case do not use the QMNAME parameter. In order to stop a specific SVRCONN instance use the CONNAME parameter

System action

The request is ignored.

CSQX617I

csect-name SSL key repository refresh not processed, SSL communications unavailable

Severity

0

Explanation

The cached SSL key repository cannot be refreshed in response to a REFRESH SECURITY TYPE(SSL) command because SSL communications are currently unavailble.

System action

0

System programmer response

Investigate why SSL is not available and take action as appropriate. It may be necessary to restart the channel initiator to allow SSL to be used.

CSQX618I

csect-name SSL key repository refresh started

Severity

0

Explanation

The cached SSL key repository is being refreshed in response to a REFRESH SECURITY TYPE(SSL) command.

System action

Message CSQX619I will be issued when the refresh is complete.

CSQX619I

csect-name SSL key repository refresh processed

Severity

0

Explanation

The refresh of the cached SSL key repository is complete.

System action

Channels will be restarted as required.

CSQX620E

csect-name System SSL error, channel *channel-name* connection *conn-id* function '*func*' RC=*return-code*

Severity
8

Explanation

An unexpected SSL communications error occurred for a channel. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'. The remote connection is *conn-id*. *func* is the name of the System SSL function that gave the error, and *return-code* is the return code (in decimal unless *func* is 'gsk_fips_state_set' in which case it is in hexadecimal).

System action

The channel is stopped.

System programmer response

See “Transport Layer Security (TLS) return codes for z/OS” on page 5221 for the cause of the return code from System SSL and refer to the z/OS Cryptographic Services System SSL Programming - SSL Function Return Codes for more information.

CSQX625E

csect-name System SSL error, function '*func*' RC=*return-code*

Severity
8

Explanation

An unexpected SSL communications error occurred for an SSL server subtask. *func* is the name of the System SSL function that gave the error, and *return-code* is the return code (in decimal).

System action

The SSL server subtask terminates.

System programmer response

See “Transport Layer Security (TLS) return codes for z/OS” on page 5221 for the cause of the return code from System SSL and the *System Secure Sockets Layer Programming Guide and Reference* manual for more information.

CSQX629E

csect-name Channel *channel-name* requires ICSF for SSLCIPH(*ciph*)

Severity
8

Explanation

Channel *channel-name* is using a cipherspec *ciph* that requires Integrated Cryptographic Service Facility (ICSF) callable services, but ICSF is not available. Sometimes the channel name and cipherspec are unknown and so are shown as "????". If known, the cipherspec is shown in the message as a 4-character code.

Recognized values are shown in message CSQX631E.

The cipherspecs that use GCM or ephemeral elliptic curve algorithms require ICSF.

System action

The channel will not start.

System programmer response

Ensure ICSF is available, or change the cipherspec that the channel is using to one that does not require ICSF. If you are using ICSF and running the queue manager with SSLFIPS(YES), ensure that ICSF is configured to run in FIPS mode.

For more information, see System SSL RC 455.

CSQX630E

csect-name Channel *channel-name* requires SSL

Severity

8

Explanation

Channel *channel-name* cannot start because it requires SSL, but SSL communications are not currently available.

System action

The channel does not start.

System programmer response

If SSL is required, investigate why it is not available and take action as appropriate. One possible cause, is that there is no certificate available owned by the user who initiated the channel address space. If this is the case, you need to re-configure the user ID to have a certificate with the correct value, by issuing the command **RACDCERT ID(xxxx)**, where *xxxx* is the user ID.

Check that you have the SSL queue manager properties set, for example SSLTASKS must be greater than 0.

If SSL is not required, change the channel definition so that SSL is not used.

CSQX631E

csect-name Cipher specifications differ, channel *channel-name* local=*local-ciph* (*local-protocol*) remote=*remote-ciph* (*remote-protocol*) connection *conn-id*

Severity

8

Explanation

The SSL cipher specification value for channel *channel-name* is *local-ciph* using protocol *local-protocol*, but the value specified at the remote end (from connection *conn-id*) is *remote-ciph* using protocol *remote-protocol*. The cipher specification and protocol values must be the same before the channel can be started. The cipher specification values are shown in the message as four-character codes; common values are:

Table 426. Convert from four-character codes to CipherSpec names

Four-character code	Protocol	CipherSpec name
0001	SSL 3.0	NULL_MD5
0002	SSL 3.0	NULL_SHA
0003	SSL 3.0	RC4_MD5_EXPORT
0004	SSL 3.0	RC4_MD5_US
0005	SSL 3.0	RC4_SHA_US
0006	SSL 3.0	RC2_MD5_EXPORT
0009	SSL 3.0	DES_SHA_EXPORT
0009	TLS 1.0	TLS_RSA_WITH_DES_CBC_SHA
000A	SSL 3.0	TRIPLE_DES_SHA_US
000A	TLS 1.0	TLS_RSA_WITH_3DES_EDE_CBC_SHA
002F	TLS 1.0	TLS_RSA_WITH_AES_128_CBC_SHA
0035	TLS 1.0	TLS_RSA_WITH_AES_256_CBC_SHA

Table 426. Convert from four-character codes to CipherSpec names (continued)

Four-character code	Protocol	CipherSpec name
003B	TLS 1.2	TLS_RSA_WITH_NULL_SHA256
003C	TLS 1.2	TLS_RSA_WITH_AES_128_CBC_SHA256
003D	TLS 1.2	TLS_RSA_WITH_AES_256_CBC_SHA256
C023	TLS 1.2	ECDHE_ECDSA_AES_128_CBC_SHA256
C024	TLS 1.2	ECDHE_ECDSA_AES_256_CBC_SHA384
C027	TLS 1.2	ECDHE_RSA_AES_128_CBC_SHA256
C028	TLS 1.2	ECDHE_RSA_AES_256_CBC_SHA384

System action

The channel does not start.

System programmer response

Change either the local or remote channel definition so that the values specified for the SSL cipher specification are the same.

CSQX632I

csect-name SSL certificate has no associated user ID, remote channel *channel-name*, connection *conn-id* - channel initiator user ID used

Severity

0

Explanation

The certificate sent from the remote end (from connection *conn-id*) during SSL handshaking was accepted, but no user ID could be found associated with it. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'.

Likely causes are that the certificate or a matching certificate name filter are defined to the external security manager (ESM), or that the certificate contains fields that are not understood by the ESM.

System action

The user ID of the channel initiator address space is used as the channel user ID for the channel.

System programmer response

If you are using certificate name filtering, you can create a filter that matches this certificate. See Working with Certificate Name Filters (CNFs) for details on associating a user ID with a certificate.

If the security you want on your channel does not require the use of the SSL mapped certificate user ID, you can define the channel to use Put Authority (**PUTAUT**) with a value of **ONLYMCA** instead of **DEF**, or **ALTMCA** instead of **CTX** and this message is not issued as no security checking for the channel is using the SSL mapped certificate user ID that could not be found. See Receiving channels using TCP/IP for more details about which user IDs are used for security checking on a receiving channel using TCP/IP.

Alternatively, change the **SSLPEER** channel attribute or create a **CHLAUTH** record to prevent this certificate being accepted from the remote channel. See Channel authentication records for more details.

CSQX633E

csect-name SSL certificate for remote channel *channel-name* failed local check, connection *conn-id*

Severity
8

Explanation

The certificate sent from the remote end (from connection *conn-id*) during SSL handshaking could not be validated. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'.

System action

The channel will not start.

System programmer response

Ensure that the SSL certificate connected to the key repository at the remote end is valid, and that the signing certificate(s) have been connected to the key ring on the local queue manager so that the certificate sent can be authenticated.

For full details about SSL certificates and key repositories see Securing.

This error might indicate that the remote end of the channel is configured to send the wrong certificate. Check the certificate label configuration at the remote end of the channel and ensure that the local key repository contains all of the necessary CA certificates.

For more information, refer to System SSL RC 8.

CSQX634E

csect-name SSL certificate failed remote check, channel *channel-name* connection *conn-id*

Severity
8

Explanation

The certificates sent to the remote end using the connection *conn-id* during SSL handshaking could not be validated. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'.

System action

The channel will not start.

System programmer response

Firstly, you need to check that the SSL certificate in the key ring at the local queue manager *qmgr-name* is valid, for example, in TRUST status and not expired.

Secondly, you also need to check that both the signing certificate (for example the certificate from the certificate authority) and the signed certificate have been connected to the key repository on the remote end, so that the certificate sent can be verified at the remote end.

The certificate used is either named on the channel in the CERTLABL attribute, or named on the queue manager in the CERTLABL attribute or CERTQSGL attribute (for a shared channel). If no certificate label is found in any of these attributes, then the certificate is named 'ibmWebSphereMQ*qsg-name*' (for a shared channel) or 'ibmWebSphereMQ*qmgr-name*', or a default certificate in the key ring is used.

For full details about SSL certificates and key repositories see Securing.

For more information, refer to System SSL RC 414.

CSQX635E

csect-name Invalid cipher specification *ciph* for channel *channel-name*

Severity
8

Explanation

The SSL cipher specification value for channel *channel-name* is not valid. The value is shown in the message as a four-character code.

Recognized values are shown in message CSQX631E.

This error can occur if the remote end is configured to use SSLFIPS(YES). Check the errors at the remote end to determine if this is the case.

System action

The channel will not start.

System programmer response

Correct the SSL cipher specification for the channel. If the remote end is configured to only accept FIPS-certified cipher specifications, change the channel to use a FIPS-certified cipher spec. See Specifying CipherSpecs for details on which cipher specifications are FIPS-certified.

For more information, refer to System SSL RCs 402, 412 and 422.

CSQX636E

csect-name Distinguished name does not match peer name, channel *channel-name* name='*dist-name*' connection *conn-id*

Severity
8

Explanation

The distinguished name, *dist-name*, specified in the SSL certificate at the remote end (from connection *conn-id*) does not match the SSL peer name for channel *channel-name*. The distinguished name at the remote end must match the peer name specified (which can be generic) before the channel can be started. In some cases the channel name cannot be determined and so is shown as '????'.

System action

The channel will not start.

System programmer response

This error might indicate that the remote end of the channel is configured to send the wrong certificate. Check the certificate label configuration at the remote end of the channel and ensure that the local key repository contains all of the necessary CA certificates.

To allow this remote end to connect, change the SSL peer name specification for the channel so that it matches the distinguished name in the SSL certificate at the remote end, or obtain the correct certificate for the remote end, as appropriate.

If the SSL Peer name specification needs to match a number of different distinguished names for multiple different remote SSL certificates, consider using channel authentication records to define rules to allow or block specific SSL peer names instead of the SSL Peer name specification on the channel definition. See Channel authentication records for more details.

CSQX637E

csect-name No SSL certificate for remote channel *channel-name*, connection *conn-id*

Severity
8

Explanation

The remote channel (from connection *conn-id*) did not supply a certificate to use during SSL handshaking, but a certificate is required. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'.

System action

The channel will not start.

System programmer response

Ensure that the SSL certificate is connected to the key repository of the remote end; alternatively, if appropriate, change the local channel definition so that its **SSLCAUTH** attribute is set to **OPTIONAL**.

For full details about SSL certificates and key repositories see Securing.

For more information, refer to System SSL RC 403.

CSQX638E

csect-name SSL communications error for channel *channel-name*, connection *conn-id*

Severity

8

Explanation

An unexpected SSL communications error occurred for a channel, as reported in the preceding messages. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'. The remote connection is *conn-id*.

System action

The channel will not start.

System programmer response

Investigate the problem reported in the preceding messages. Review the local and remote console logs for reports of network errors.

For more information, refer to System SSL RC 406.

CSQX639E

csect-name No cipher specification for remote channel *channel-name*, connection *conn-id*

Severity

8

Explanation

No SSL cipher specification was supplied by the remote channel *channel-name* (from connection *conn-id*), but one was required. In some cases the channel name cannot be determined and so is shown as '????'.

System action

The channel will not start.

System programmer response

Change the remote channel definition so that the value specified for the SSL cipher specification is the same as that of the local channel.

CSQX640E

csect-name Invalid peer name, channel *channel-name* attribute=*key-name*

Severity

8

Explanation

The SSL peer name for channel *channel-name* includes a distinguished name attribute key *key-name* which is invalid or unsupported. In some cases the channel name cannot be determined and so is shown as '????'.

System action

The channel will not start.

System programmer response

Correct the SSL peer name for the channel.

CSQX641E

csect-name Cipher specification error for remote channel *channel-name*, connection *conn-id*

Severity

8

Explanation

An error occurred with the SSL cipher specification for remote channel *channel-name* (from connection *conn-id*). In some cases the channel name cannot be determined and so is shown as '????'.

System action

The channel will not start.

System programmer response

Review the remote console log to determine the cipher specification error.

CSQX642E

csect-name No SSL certificate for channel *channel-name*

Severity

8

Explanation

The channel *channel-name* did not supply a certificate to use during SSL handshaking, but a certificate is required by the remote end. In some cases the channel name cannot be determined and so is shown as '????'.

System action

The channel does not start.

System programmer response

Ensure that the key ring of the local queue manager *qmgr-name* has an SSL certificate connected to it which is associated with the queue manager. If you have configured a certificate label, check that the certificate exists.

The certificate used is either named on the channel in the CERTLABL attribute, or named on the queue manager in the CERTLABL attribute or CERTQSGL attribute (for a shared channel). If no certificate label is found in any of these attributes, then the certificate is named 'ibmWebSphereMQ*qsg-name*' (for a shared channel) or 'ibmWebSphereMQ*qmgr-name*', or a default certificate in the key ring is used.

Alternatively, if appropriate, change the remote channel definition so that its SSLCAUTH attribute is set to OPTIONAL.

For full details about SSL certificates and key repositories, see Securing.

CSQX643E

csect-name Peer name error for remote channel *channel-name*, connection *conn-id*

Severity

8

Explanation

An error occurred with the SSL peer name for remote channel *channel-name* (from connection *conn-id*). In some cases the channel name cannot be determined and so is shown as '????'.

System action

The channel will not start.

System programmer response

Review the remote console log to determine the peer name error.

CSQX644E

csect-name Unable to determine peer name for remote channel *channel-name*

Severity

4

Explanation

The peer name associated with the certificate sent from the remote end during SSL handshaking could not be determined. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'.

System action

If the local channel has a peer name specified it does not start.

System programmer response

Ensure that the SSL certificate in the key ring at the local queue manager *qmgr-name* is valid, and that the signing certificate has been connected to the key repository on the remote end so that the certificate sent can be authenticated.

The certificate used is either named on the channel in the CERTLABL attribute, or named on the queue manager in the CERTLABL attribute or CERTQSGL attribute (for a shared channel). If no certificate label is found in any of these attributes, then the certificate is named 'ibmWebSphereMQ*qsg-name*' (for a shared channel) or 'ibmWebSphereMQ*qmgr-name*', or a default certificate in the key ring is used.

Check that the local and remote channel definitions are correct.

For full details about SSL certificates and key repositories, see Securing.

CSQX645E

csect-name Certificate *cert-label* missing for channel *channel-name*

Severity

4

Explanation

An SSL/TLS certificate *cert-label*, or the default certificate cannot be found in the key ring or the certificate is not trusted. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'.

In some cases this message will appear multiple times, once for each affected channel.

System action

The channel does not start.

System programmer response

Ensure that the SSL/TLS certificate named *cert-label* is in the key ring and that it is valid, and that the queue manager is running with OPMODE(NEWFUNC,800).

Alternatively, change the certificate label configuration so that the channel uses a valid certificate.

The certificate used is either named on the channel in the CERTLABL attribute, or named on the queue manager in the CERTLABL attribute or CERTQSGL attribute (for a shared channel). If no certificate label is found in any of these attributes, then the certificate is named 'ibmWebSphereMQ*qsg-name*' (for a shared channel) or 'ibmWebSphereMQ*qmgr-name*', or a default certificate in the key ring is used.

To verify which key ring is in use, issue the following MQSC command:

```
DISPLAY QMGR SSLKEYR
```

To list the certificates that are present in the key ring in use, issue the following RACF command, or an equivalent command in your External Security Manager:

```
RACDCERT ID(chinit-user-id) LISTRING(key-ring-name)
```

https://www-01.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/com.ibm.zos.v2r1.gska100/sssl2msg1000885.htm

CSQX646E

csect-name Error accessing LDAP server for channel *channel-name*

Severity

4

Explanation

While checking CRLs for a channel, an error occurred in setting up the LDAP environment or retrieving an LDAP directory entry. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'.

System action

The channel will not start.

System programmer response

Ensure that the LDAP server is specified and set up correctly, and is running.

For more information, refer to System SSL RC 11.

CSQX658E

csect-name SSL certificate has expired, channel *channel-name* connection *conn-id*

Severity

4

Explanation

The current time is either before the SSL certificate start time or or after the end time. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'. The connection is *conn-id*.

System action

The channel will not start.

System programmer response

Obtain a new certificate if the certifiacte has expired, or wait until the certificate becomes valid if it is not valid yet.

For more information, refer to System SSL RC 401.

CSQX663E

csect-name SSL certificate signature is incorrect, channel *channel-name* connection *conn-id*

Severity
4

Explanation

In the SSL certificate sent from the remote end using the connection *conn-id*, the certificate signature is not correct. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'.

System action

The channel will not start.

System programmer response

Ensure that the SSL certificate connected to the key repository at the remote end is valid.

For more information, refer to System SSL RC 413.

CSQX665E

csect-name Channel *channel-name* stopping because remote SSL socket closed, connection *conn-id*

Severity
4

Explanation

The remote end of a channel using SSL communications (from connection *conn-id*) closed the socket or sent a close notification alert. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'.

System action

The channel stops.

System programmer response

Examine the console log for the remote end to determine the cause of the failure.

For more information, refer to System SSL RC 420.

CSQX666E

csect-name LDAP server unavailable for channel *channel-name*

Severity
4

Explanation

While checking CRLs for a channel, the required LDAP server was not available. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'.

System action

The channel does not start.

System programmer response

Ensure that the LDAP server is running.

For more information, refer to System SSL RC 427.

CSQX673E

csect-name Certificate label *cert-label* not used on channel *channel-name*, remote connection *conn-id*

Severity

8

Explanation

The SSL or TLS channel *channel-name* is configured to use certificate label *cert-label*. However, the remote peer did not send the necessary information to allow the local channel to use the correct certificate. The remote host is *conn-id*.

This error occurs when the local channel definition has a certificate label and the remote peer does not support selection of certificates.

System action

The channel will not start.

System programmer response

Ensure that the remote peer supports certificate label configuration. Refer to Digital certificate labels, understanding the requirements for details of certificate label requirements. Alternatively, alter the local channel definition so that it does not specify a certificate label.

CSQX674E

csect-name Channel *channel-name*specified a weak or broken SSL CipherSpec *sslciph*

Severity

8

Explanation

The channel is unable to start because it is configured to use a CipherSpec that is potentially insecure.

System action

The channel is prevented from starting.

System programmer response

Examine the CipherSpec specified in the SSLCIPH parameter and consider using a more secure CipherSpec.

If you want to re-enable the use of weak CipherSpecs, you can do so by adding a dummy Data Definition (DD) statement named CSQXWEAK to the channel initiator JCL. For example:

```
//CSQXWEAK DD DUMMY
```

If you want to re-enable the disabled SSLv3 support in IBM MQ, you can do so by adding a dummy Data Definition (DD) statement named CSQXSSL3 to the channel initiator JCL. For example:

```
//CSQXSSL3 DD DUMMY
```

You need to specify both of the preceding dummy DD statements, if you want to enable a weak SSLv3-based CipherSpec.

There are alternative mechanisms that can be used to forcibly re-enable weak CipherSpecs, and SSLv3 support,if the Data Definition change is unsuitable. Contact IBM Service for further information.

Attention: Re-enabling CipherSpecs in this manner leaves systems exposed to possible security problems. You should use CipherSpecs that use only the TLS protocol, rather than SSLv3.

CSQX675E

csect-name Unable to complete SSL key repository refresh

Severity

4

Explanation

The refresh of the cached SSL key repository could not be completed because of errors.

System action

The refresh is incomplete.

System programmer response

Examine the console log for messages that might indicate why the refresh could not be started.

CSQX676E

csect-name SSL key repository refresh completed, but some channels not restarted

Severity

4

Explanation

The refresh of the cached SSL key repository has completed, so the latest values and certificates are in use for all SSL channels. However, not all the outbound SSL channels which were running when the refresh was initiated could be restarted after the refresh had completed.

System action

Processing continues.

System programmer response

Examine the console log for messages identifying the channels that did not restart.

CSQX677E

csect-name SSL key repository refresh terminated, waiting for channel *channel-name*

Severity

4

Explanation

The cached SSL key repository is being refreshed, which involves stopping all the channels that use SSL communications. One or more of the channels is taking too long to stop. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'.

System action

The refresh is terminated. Some channels using SSL will have been stopped.

System programmer response

Stop any SSL channels that have not already stopped and issue the REFRESH SECURITY TYPE(SSL) command again.

CSQX678E

csect-name Channel *channel-name* not started, refreshing SSL key repository

Severity

4

Explanation

A channel using SSL communications could not be started because the cached SSL key repository is currently being refreshed. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'.

System action

The channel does not start.

System programmer response

Wait until the refresh has completed and start the channel again.

CSQX679E

csect-name Channel *channel-name* not started, refreshing remote SSL key repository

Severity

4

Explanation

A channel using SSL communications could not be started because the cached SSL key repository is currently being refreshed at the remote end. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'.

System action

The channel does not start.

System programmer response

Wait until the refresh has completed and start the channel again.

CSQX683E

csect-name SSL key repository has no certificates

Severity

4

Explanation

The SSL key repository (that is, the key ring in the external security manager) does not contain any valid certificates.

System action

Channels using SSL communications will not start.

System programmer response

Add the user certificate and any necessary certificate authority (CA) certificates to the key repository. Ensure that existing certificates are valid, have not expired, and are marked as trusted.

For more information, refer to System SSL RC 7.

CSQX684E

csect-name SSL key repository has no CA certificates

Severity

4

Explanation

The SSL key repository (that is, the key ring in the external security manager) does not contain any valid certificate authority (CA) certificates. A channel using SSL communications needs at least one CA or self-signed certificate to perform client authentication.

System action

Channels using SSL communications will not start.

System programmer response

Add the user certificate and any necessary certificate authority (CA) certificates to the key repository. Ensure that existing certificates are valid, have not expired, and are marked as trusted.

For more information, refer to System SSL RC 109.

CSQX685E

csect-name No self-signed certificate for channel *channel-name*, connection *conn-id*

Severity

4

Explanation

A self-signed certificate cannot be validated as it is not in the SSL key repository. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'. The remote connection is *conn-id*.

System action

The channel is not started.

System programmer response

Add the self-signed certificate to the key repository.

Note: Changes to the key repository do not take effect immediately, see When changes to certificates or the key repository become effective on z/OS. If you have already added the self-signed certificate to the key repository, issue a REFRESH SECURITY TYPE(SSL) command or recycle the CHINIT address space.

For more information, refer to System SSL RC 417.

CSQX686E

csect-name SSL private key error for channel *channel-name*

Severity

4

Explanation

The SSL certificate used has no associated private key, or the private key is not available because it key is stored in ICSF and ICSF services are not available. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'.

The certificate used is either named on the channel in the CERTLABL attribute, or named on the queue manager in the CERTLABL attribute or CERTQSGL attribute (for a shared channel). If no certificate label is found in any of these attributes, then the certificate is named 'ibmWebSphereMQ*qsg-name*' (for a shared channel) or 'ibmWebSphereMQ*qmgr-name*', or a default certificate in the key ring is used.

System action

The channel is not started.

System programmer response

Ensure that the private key associated with the SSL certificate used is available. Ensure that the ICSF started task is running if the private key is stored in ICSF.

For more information, refer to System SSL RC 428.

CSQX687E

csect-name SSL certificate revoked by CA for channel *channel-name*, connection *conn-id*

Severity
4

Explanation

The SSL certificate has been revoked by the certificate authority (CA). The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'. The remote connection is *conn-id*.

System action

The channel is not started.

System programmer response

Obtain a new certificate and add it to the key repository.

For more information, refer to System SSL RC 431.

CSQX688E

csect-name No SSL CA certificate for channel *channel-name*, connection *conn-id*

Severity
4

Explanation

The SSL key repository does not contain a certificate for the certificate authority (CA). The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'. The remote connection is *conn-id*.

System action

The channel is not started.

System programmer response

Obtain a certificate for the certificate authority (CA) and add it to the key repository.

For more information, refer to System SSL RC 435.

CSQX689E

csect-name CRL cannot be processed for channel *channel-name*, connection *conn-id*

Severity
4

Explanation

A Certificate Revocation List (CRL) is not valid and cannot be processed. The channel is *channel-name*; in some cases its name cannot be determined and so is shown as '????'. The remote connection is *conn-id*.

System action

The channel is not started.

System programmer response

Contact the certificate authority and obtain a replacement CRL.

For more information, refer to System SSL RC 436.

CSQX690I

csect-name Cipher specifications based on the SSLv3 protocol are disabled.

Severity
4

Explanation

Cipher specifications based on the SSLv3 protocol are not enabled, and channels configured to use those cipher specifications fail when started.

System action

Processing continues.

System programmer response

If you do not need to use cipher specifications based on the SSLv3 protocol, then no action is required.

If you want to re-enable the use of weak CipherSpecs, you can do so by adding a dummy Data Definition (DD) statement named CSQWEAK to the channel initiator JCL. For example:

```
//CSQWEAK DD DUMMY
```

If you want to re-enable the disabled SSLv3 support in IBM MQ, you can do so by adding a dummy Data Definition (DD) statement named CSQXSSL3 to the channel initiator JCL. For example:

```
//CSQXSSL3 DD DUMMY
```

You need to specify both of the preceding dummy DD statements, if you want to enable a weak SSLv3-based CipherSpec.

There are alternative mechanisms that can be used to forcibly re-enable weak CipherSpecs, and SSLv3 support,if the Data Definition change is unsuitable. Contact IBM Service for further information.

Attention: Re-enabling CipherSpecs in this manner leaves systems exposed to possible security problems. You should use CipherSpecs that use only the TLS protocol, rather than SSLv3.

CSQX691I

csect-name Cipher specifications based on the SSLv3 protocol are enabled.

Severity
4

Explanation

Cipher specifications based on the SSLv3 protocol are enabled, and channels can be configured to use those cipher specifications.

System action

Processing continues.

System programmer response

If you need to use cipher specifications based on the SSLv3 protocol, then no action is required.

If you do not need to use cipher specifications based on the SSLv3 protocol, you should remove the override that enables the use of SSLv3.

See message CSQX690I for information on enabling SSLv3.

CSQX692I

csect-name Weak or broken SSL cipher specifications are disabled.

Severity
4

Explanation

Cipher specifications that are known to be weak or broken are not enabled, This includes all SSLv3-based cipher specifications. Channels configured to use those cipher specifications fail when started.

System action

Processing continues.

System programmer response

If you do not need to use broken or weak cipher specifications, no action is required.

If you want to re-enable the use of weak CipherSpecs, you can do so by adding a dummy Data Definition (DD) statement named CSQWEAK to the channel initiator JCL. For example:

```
//CSQWEAK DD DUMMY
```

If you want to re-enable the disabled SSLv3 support in IBM MQ, you can do so by adding a dummy Data Definition (DD) statement named CSQXSSL3 to the channel initiator JCL. For example:

```
//CSQXSSL3 DD DUMMY
```

You need to specify both of the preceding dummy DD statements, if you want to enable a weak SSLv3-based CipherSpec.

There are alternative mechanisms that can be used to forcibly re-enable weak CipherSpecs, and SSLv3 support,if the Data Definition change is unsuitable. Contact IBM Service for further information.

Attention: Re-enabling CipherSpecs in this manner leaves systems exposed to possible security problems. You should use CipherSpecs that use only the TLS protocol, rather than SSLv3.

CSQX693I

csect-name Weak or broken SSL cipher specifications are enabled.

Severity

4

Explanation

Cipher specifications known to be weak or broken are enabled, and channels can be configured to use those cipher specifications.

System action

Processing continues.

System programmer response

If you need to use weak or broken cipher specifications, no action is required.

If you do not need to use weak or broken cipher specifications, you should remove the override that enables the use of weak or broken cipher specifications.

See message CSQX692I for information on enabling weak or broken cipher specifications.

CSQX719E

csect-name Invalid cipher specification *ciph* for FIPS mode for channel *channel-name*

Severity

8

Explanation

The SSL cipher specification value for channel *channel-name* is not FIPS-certified and the queue manager has been configured to run with **SSLFIPS(YES)**. The value is shown in the message as a four-character code.

Recognized values are shown in message CSQX631E.

In some cases, when the channel is responding to an inbound request, the cipher specification cannot be determined and so is shown as '????'.

System action

The channel will not start.

System programmer response

Correct the channel to use a FIPS-certified cipher specification, or if the queue manager should not be running in FIPS mode, alter the queue manager to use **SSLFIPS(NO)**. See Specifying CipherSpecs for details on which cipher specifications are FIPS-certified.

For more information, refer to System SSL RCs 402 and 412.

CSQX772E

csect-name mqapi-call failed, MQRC=*mqrc* (*mqrc-text*)

Severity

8

Explanation

The indicated IBM MQ *mqapi-call* failed for the specified reason code *mqrc*, (*mqrc-text*).

System action

Typically the component in which the error occurs terminates. When the component is a message channel agent, the associated channel is stopped.

System programmer response

Refer to API completion and reason codes for information about *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQX774E

csect-name CHLAUTH cache load failed, all inbound channels blocked

Severity

8

Explanation

The CHLAUTH cache has failed to load. All inbound channels has been blocked from starting until the problem has been fixed. See previous message for the cause of the problem.

System action

All inbound channels are blocked from starting.

System programmer response

Look for the previous related message for the cause of the problem.

CSQX775I

csect-name Channel *channel-name* from *ipaddress* would have been blocked due to userid, Detail: *detail*

Severity

4

Explanation

The inbound channel *channel-name* would have been blocked from address *ipaddress* because the active values of the channel were mapped to a userid that should be blocked. Access is allowed as the channel authentication record is in warning mode.

The active values of the channel were *detail*.

System action

The channel is started.

System programmer response

Examine the channel authentication records to ensure that the correct settings have been configured. If the channel authentication record was not in warning mode the channel would be blocked. The ALTER QMGR **CHLAUTH** switch is used to control whether the channel authentication records are used. The DISPLAY CHLAUTH command can be used to query the channel authentication records.

CSQX776E

csect-name Channel *channel-name* from *ipaddress* has been blocked due to userid, Detail: *detail*

Severity

8

Explanation

The inbound channel *channel-name* was blocked from address *ipaddress* because the active values of the channel were mapped to a userid that should be blocked.

The active values of the channel were *detail*.

System action

The channel is not started.

System programmer response

Examine the channel authentication records to ensure that the correct settings have been configured. The ALTER QMGR **CHLAUTH** switch is used to control whether the channel authentication records are used. The DISPLAY CHLAUTH command can be used to query the channel authentication records.

CSQX777E

csect-name Channel *channel-name* from *ipaddress* has been blocked due to USERSRC(NOACCESS), Detail: *detail*

Severity

8

Explanation

The inbound channel *channel-name* was blocked from address *ipaddress* because the active values of the channel matched a channel authentication record configured with USERSRC(NOACCESS).

The active values of the channel were *detail*.

System action

The channel is not started.

System programmer response

Examine the channel authentication records to ensure that the correct settings have been configured.

The ALTER QMGR **CHLAUTH** switch is used to control whether the channel authentication records are used. The DISPLAY CHLAUTH can be used to query the channel authentication records.

If no host name is shown in the message next to the IP address, and CHLAUTH rules using host names are in place, ensure that your Domain Name Servers can correctly resolve the IP address to a host name and that your queue manager is configured with REVDNS(ENABLED).

CSQX782E

csect-name Connection from address *ipaddress* has been blocked due to matching rule *ip-address-pattern*

Severity

8

Explanation

The inbound connection from the address was blocked because it matches one of the blocked addresses, *ip-address-pattern*, in the channel authentication table.

System action

The channel is not started.

System programmer response

Examine the channel authentication records to ensure that the correct settings have been configured. The ALTER QMGR **CHLAUTH** switch is used to control whether the channel authentication records are used. The DISPLAY CHLAUTH can be used to query the channel authentication records.

CSQX785E

csect-name Channel *channel-name* is configured to not use the dead-letter queue

Severity

8

Explanation

Channel *channel-name* failed to deliver a message to its destination. The report option MQRO_DISCARD_MSG was not specified for the message and the channel has been configured to not use the dead-letter queue through the attribute setting USEDLQ(NO).

System action

The channel either discards the message, or the channel ends, in accordance with the NPMSPEED attribute setting.

System programmer response

Investigate the cause of this error, then either correct the problem that prevented the channel delivering the message, or enable the channel to use the dead-letter queue.

CSQX786I

csect-name Connection from address *ipaddress* would have been blocked due to matching rule *ip-address-pattern*

Severity

4

Explanation

The inbound connection from the address *ipaddress* would have been blocked because it matches one of the blocked addresses, *ip-address-pattern*, in the channel authentication table. Access is allowed as the channel authentication table is in warning mode.

System action

The channel is started.

System programmer response

Examine the channel authentication records to ensure that the correct settings have been configured. If the channel authentication record was not in warning mode the channel would be blocked. The ALTER QMGR **CHLAUTH** switch is used to control whether the channel authentication records are used. The DISPLAY CHLAUTH command can be used to query the channel authentication records.

CSQX787I

csect-name Channel *channel-name* from *ipaddress* would have been blocked due to USERSRC(NOACCESS), Detail: *detail*

Severity

4

Explanation

The inbound channel *channel-name* would have been blocked from address *ipaddress* because the active values of the channel matched a channel authentication record configured with USERSRC(NOACCESS). It was not blocked due to the channel authentication record being in warning mode.

The active values of the channel were *detail*.

System action

The channel is started.

System programmer response

Examine the channel authentication records to ensure that the correct settings have been configured. If the channel authentication record was not in warning mode the channel would be blocked. The ALTER QMGR **CHLAUTH** switch is used to control whether the channel authentication records are used. The DISPLAY CHLAUTH command can be used to query the channel authentication records.

CSQX788I

csect-name DNS lookup for address *address* using function '*func*' took *n* seconds

Severity

4

Explanation

An attempt to resolve address *address* using the '*func*' function call took *n* seconds to complete. This might indicate a problem with the DNS configuration.

System action

Processing continues.

System programmer response

Ensure that the DNS is correctly configured on the local system.

If the address was an IP address then the slow operation was a reverse DNS lookup. Some DNS configurations are not capable of reverse DNS lookups and some IP addresses have no valid reverse DNS entries.

If the problem persists, consider disabling reverse DNS lookups until the issue with the DNS can be resolved.

CSQX790I

csect-name Connection authentication failed for user *user-id* due to CHLAUTH with CHCKCLNT(*chckclnt-value*), Detail: *detail*

Severity

4

Explanation

The user ID *user-id* and its password were checked because the inbound connection matched a channel authentication record with CHCKCLNT(*chckclnt-value*).

The active values of the channel were *detail*. The MATCH(RUNCHECK) mode of the DISPLAY CHLAUTH command can be used to identify the relevant CHLAUTH record.

This message accompanies a previous error to clarify the reason for the user ID and password check.

System action

The channel is not started.

System programmer response

Refer to the previous error for more information.

Ensure that a password is specified by the client application and that the password is correct for the User ID.

Alternatively, to avoid the authentication check you can amend the CHLAUTH record CHCKCLNT attribute. However, allowing unauthenticated remote access is not recommended.

CSQX791E

csect-name Client application *appl-name* from address *ip-address* did not supply a user ID and password, Detail: *detail*

Severity

8

Explanation

The client application *appl-name* running on host *ip-address* did not supply a user ID and password. The channel authentication (CHLAUTH) record for the connection requires a user ID and password, but none was supplied.

The active values of the channel were *detail*. The MATCH(RUNCHECK) mode of the DISPLAY CHLAUTH command can be used to identify the relevant CHLAUTH record.

System action

The channel is not started.

System programmer response

Ensure that the application provides a valid user ID and password, or change the queue manager connection authority (CONNAUTH) configuration to OPTIONAL to allow client applications to connect which have not supplied a user ID and password.

CSQX793E

csect-name The user ID and password for client application *appl-name* from address *ip-address* cannot be checked, Detail: *detail*

Severity

8

Explanation

The user ID and password for the client application *appl-name* running on host *ip-address* cannot be checked. The channel authentication (CHLAUTH) record for the connection requires an authentication check, but the queue manager is not configured to use connection authentication for clients.

The active values of the channel were *detail*. The MATCH(RUNCHECK) mode of the DISPLAY CHLAUTH command can be used to identify the relevant CHLAUTH record.

System action

The channel is not started.

System programmer response

Change the CHLAUTH configuration so that client authentication is not required, or alter the queue manager connection authority (CONNAUTH) configuration to enable client authentication checks.

CSQX830I

csect-name Channel initiator active

Severity
0

Explanation

This is issued in response to the DISPLAY CHINIT command if the channel initiator is active.

CSQX831I

csect-name nn adapter subtasks started, *nn* requested

Severity
0

Explanation

This is issued in response to the DISPLAY CHINIT command, and shows how many adapter subtasks are currently active, and how many were requested by the CHIADAPS queue manager attribute. If the numbers differ, some adapter subtasks have failed and not been restarted, which could reduce processing capacity.

CSQX832I

csect-name nn dispatchers started, *nn* requested

Severity
0

Explanation

This is issued in response to the DISPLAY CHINIT command, and shows how many dispatchers are currently active, and how many were requested by the CHIDISPS queue manager attribute. If the numbers differ, some dispatchers have failed and not been restarted. The number of current TCP/IP and LU 6.2 channels allowed will be reduced proportionately, and other processing capacity may be reduced.

CSQX833I

csect-name nn SSL server subtasks started, *nn* requested

Severity
0

Explanation

This is issued in response to the DISPLAY CHINIT command, and shows how many SSL server subtasks are currently active, and how many were requested by the SSLTASKS queue manager attribute. If the numbers differ, some SSL server subtasks have failed and not been restarted, which could reduce processing capacity.

CSQX836I

csect-name nn Maximum channels - TCP/IP nn, LU 6.2 *nn*

Severity

0

Explanation

This is issued in response to the DISPLAY CHINIT command. It shows the maximum numbers of each type of channel that are allowed.

CSQX840I

csect-name nn channels current, maximum *nn*

Severity

0

Explanation

This is issued in response to the DISPLAY CHINIT command. It shows how many channels are current, and how many are allowed altogether, as requested by the MAXCHL queue manager attribute.

CSQX841I

csect-name nn channels active, maximum nn, including *nn* paused

Severity

0

Explanation

This is issued in response to the DISPLAY CHINIT command. Of the channels that are current, it shows how many are active (transmitting messages), and how many are allowed altogether to be active, by the ACTCHL queue manager attribute. It also shows how many of the active channels are paused, waiting to retry putting a message.

CSQX842I

csect-name nn channels starting, *nn* stopped, *nn* retrying

Severity

0

Explanation

This is issued in response to the DISPLAY CHINIT command. Of the channels that are current, it show how many are:

- waiting to become active, because the limit for active channels has been reached
- stopped, requiring manual intervention
- attempting to reconnect following a temporary error.

CSQX843I

csect-name TCP/IP listener INDISP=*disposition* retrying, for port *port* address *ip-address*

Severity

0

Explanation

This is issued in response to the DISPLAY CHINIT command for each TCP/IP listener that is trying to restart after an error. The channel initiator will attempt to restart the listener, at the intervals specified by the LSTRTMR queue manager attribute.

port and *ip-address* show the port and IP address combination on which it listens; if *ip-address* is '*', it listens on all available IP addresses. *disposition* shows which type of incoming requests the listener handles:

QMGR
those directed to the target queue manager

GROUP
those directed to the queue-sharing group.

CSQX844I

csect-name LU 6.2 listener INDISP=*disposition* retrying, for LU name *name*

Severity
0

Explanation

This is issued in response to the DISPLAY CHINIT command for each LU 6.2 listener that is trying to restart after an error. The channel initiator will attempt to restart the listener at the intervals specified by the LSTRTMR queue manager attribute.

disposition shows which type of incoming requests the listener handles:

QMGR
those directed to the target queue manager

GROUP
those directed to the queue-sharing group.

CSQX845I

csect-name TCP/IP system name is *name*

Severity
0

Explanation

This is issued in response to the DISPLAY CHINIT command, and shows the TCP/IP system name that is being used, as specified in the TCPNAME queue manager attribute.

CSQX846I

csect-name TCP/IP listener INDISP=*disposition* started, for port *port* address *ip-address*

Severity
0

Explanation

This is issued in response to the DISPLAY CHINIT command for each TCP/IP listener that is active.

port and *ip-address* show the port and IP address combination on which it listens; if *ip-address* is '*', it listens on all available IP addresses. *disposition* shows which type of incoming requests the listener handles:

QMGR
those directed to the target queue manager

GROUP
those directed to the queue-sharing group.

CSQX847I

csect-name LU 6.2 listener INDISP=*disposition* started, for LU name *name*

Severity

0

Explanation

This is issued in response to the DISPLAY CHINIT command for each LU 6.2 listener that is active.

disposition shows which type of incoming requests the listener handles:

QMGR
those directed to the target queue manager

GROUP
those directed to the queue-sharing group.

CSQX848I

csect-name TCP/IP listener INDISP=*disposition* not started

Severity

0

Explanation

This is issued in response to the DISPLAY CHINIT command for each TCP/IP listener that is not active.

disposition shows which type of incoming requests the listener handles:

QMGR
those directed to the target queue manager

GROUP
those directed to the queue-sharing group.

System programmer response

If the listener had been started, and was not deliberately stopped, this might be because there was an error in the communications system. The channel initiator will attempt to restart the listener, at the intervals specified by the LSTRTMR queue manager attribute.

CSQX849I

csect-name LU 6.2 listener INDISP=*disposition* not started

Severity

0

Explanation

This is issued in response to the DISPLAY CHINIT command for each LU 6.2 listener that is not active.

disposition shows which type of incoming requests the listener handles:

QMGR
those directed to the target queue manager

GROUP
those directed to the queue-sharing group.

System programmer response

If the listener had been started, and was not deliberately stopped, this might be because there was an error in the communications system. The channel initiator will attempt to restart the listener, at the intervals specified by the LSTRTMR queue manager attribute.

CSQX871I

csect-name Cluster maintenance has been running for *num-mins* minutes, phase *maintenance-phase* has so far processed *num-records* records

Severity

0

Explanation

A queue manager will periodically perform a maintenance cycle to refresh and remove state associated with the clusters it is a member of. This message gives an indication of the progress being made.

System action

For large clusters this maintenance process may take a significant period of time. In such situations this message will be periodically repeated until maintenance has completed, at which time message CSQX872I will be output.

CSQX872I

csect-name Cluster maintenance has completed after *num-mins* minutes, *num-records* records were processed

Severity

0

Explanation

A queue manager will periodically perform a maintenance cycle to refresh and remove state associated with the clusters it is a member of. This message follows one or more instances of message CSQX871I and indicates the cycle has completed.

System action

None

CSQX875I

csect-name REFRESH CLUSTER processing started for cluster cluster-name

Severity

0

Explanation

A REFRESH CLUSTER command has been issued on this queue manager.

In phase one this will discard all locally cached information for the cluster and request new information from other members of the cluster when necessary. Phase two processes the information received. For large cluster configurations this process can take a significant amount of time, especially on full repository queue managers. During this time applications attempting to access cluster resources may see failures to resolve cluster resources. In addition, cluster configuration changes made on this queue manager may not be processed until the refresh process has completed.

System action

Defer any cluster related work on this queue manager until both phases are complete.

Message CSQX442I or CSQX404I will be issued at the end of phase one.

Completion of phase two can be determined when the SYSTEM.CLUSTER.COMMAND.QUEUE has reached a consistently empty state.

CSQX876I

csect-name Cluster cache compression started

Severity

0

Explanation

Periodically cluster management will compress its local cache. Compression can take a significant period of time for certain operations, such as performing a CLUSTER REFRESH. During the compression task, cluster management commands will not be processed.

Once the compression task has completed message CSQX877I will be issued.

CSQX877I

csect-name Cluster cache compression completed

Severity

0

Explanation

The cluster cache compression activity, indicated by message CSQX876I, has now completed.

CSQX878I

csect-name Repository command error, command *command*, cluster object *object-name*, sender *sender-id*, reason *reason*

Severity

8

Explanation

An internal cluster repository command failed to complete successfully. Earlier messages in the log will contain details of the problem. Failure to successfully process a command can leave a cluster in an inconsistent state.

System action

Processing continues

System programmer response

If the problem cannot be resolved, collect the items listed in the Problem Determination section and contact your IBM® support center.

CSQX879E

csect-name Conflicting clustered topic *topic-name* from queue manager *qmgr-name*

Severity

8

Explanation

A conflict has been detected for clustered topic *topic-name*.

Two clustered topics conflict if any of the following conditions are true:

1. They have the same topic string but have a different topic name
2. They have the same topic string, or one is an ancestor of the other in the topic tree, and they have a different cluster name

3. They have the same topic string, or one is an ancestor of the other in the topic tree, and they have incompatible values for the cluster route attribute

System action

The CLSTATE attribute of the clustered topic identified by *topic-name* is set to INVALID and the topic is no longer used by the queue manager.

System programmer response

Review the clustered topics visible to the queue manager and correct any conflicts by modifying or deleting the definitions in error. After updating the topic definitions, ensure all clustered topics have a CLSTATE of ACTIVE on all queue managers in the same cluster.

V 9.0.3

CSQX967I

CSQXBLUR task attached, TCB=*tcb*

Severity
0

Explanation

This is a debug message which is not documented.

V 9.0.3

CSQX968I

CSQXBLUR task detached

Severity
0

Explanation

This is a debug message which is not documented.

Initialization procedure and general services messages (CSQY...): z/OS

V 9.0.3 **CSQY000I**
IBM MQ for z/OS V*n release_type*

V 9.0.3 **Explanation**

This message is issued when the queue manager starts, and shows the release level and release type.

CSQY002I
QUEUE MANAGER STOPPING

Explanation

The STOP QMGR command is accepted. Message CSQ9022I is issued when the queue manager shutdown process has completed. The message is issued either to the originator of the STOP QMGR command, or to the z/OS console from which the START QMGR command was received.

System action

Queue manager shutdown is initiated.

CSQY003I
QUEUE MANAGER IS ALREADY ACTIVE

Explanation

The START QMGR command has not been accepted, because the queue manager is active. Message CSQ9023E is issued after this message.

CSQY004I
QUEUE MANAGER IS ALREADY STOPPING

Explanation

The STOP QMGR command has not been accepted either because the queue manager shutdown is in progress for the specified option (QUIESCE or FORCE), or because the QUIESCE option was specified after a FORCE option had been accepted previously. Message CSQ9023E is issued after this message.

System action

Queue manager shutdown continues.

CSQY005E
QUEUE MANAGER STARTUP TERMINATED, INVALID START COMMAND

Explanation

The queue manager can be started only by a START QMGR command.

System action

Queue manager startup is terminated.

CSQY006E
csect-name INVALID AMODE OR RMODE ATTRIBUTE FOUND FOR LOAD MODULE *module-name*

Explanation

The queue manager initialization procedures found that a module had an invalid AMODE or RMODE attribute when it was loaded. *module-name* is the name of the load module with an invalid addressing or residency mode.

System action

Queue manager startup terminates abnormally.

System programmer response

Verify that all installation and maintenance activities against IBM MQ have been done correctly. If you are unable to correct the problem, contact your IBM support center.

CSQY007E
csect-name QUEUE MANAGER STARTUP TERMINATED, INVALID OPERATING SYSTEM LEVEL

Explanation

The queue manager initialization procedures found that the level of the operating system did not have the function required for correct queue manager operation.

System action

Queue manager startup terminates abnormally.

System programmer response

Verify that the prerequisite, or later, level of the operating system is installed. If you are unable to correct the problem, contact your IBM support center.

CSQY008I
QUEUE MANAGER SHUTDOWN REQUEST NOT ACCEPTED

Explanation

The STOP QMGR command has not been accepted because startup has not completed to the point where shutdown can occur. Message CSQ9023E is issued after this message.

System action

Queue manager startup continues, and the STOP QMGR command is ignored.

CSQY009I
verb-name pkw-name COMMAND ACCEPTED FROM USER(*userid*), STOP MODE(*mode*)

Explanation

This message is issued to record who issued the command to stop IBM MQ, and what type of stop it was. *verb-name* might include the command prefix (CPF). This depends on how the command was entered.

CSQY010E
csect-name LOAD MODULE *module-name* IS NOT AT THE CORRECT RELEASE LEVEL

Explanation

The named load module is not at the correct level for the version of the queue manager that was being used.

System action

If detected by the queue manager, startup terminates abnormally with reason code X'00E80161'. If detected by the channel initiator (*module-name* is CSQXJST), it does not start.

If detected by the AMS enablement module (DRQ0NABL), the queue manager only fails to start if SPLCAP=YES is specified in the system parameters. In this case message CSQY029E is issued.

System programmer response

Verify that the correct IBM MQ program libraries are being used (for the queue manager or channel initiator as appropriate) and that all installation and maintenance activities againstIBM MQ have been done correctly. If the early processing program is incorrect (*module-name* is CSQ3EPX), refresh it by issuing the REFRESH QMGR TYPE(EARLY) command.

If you are unable to correct the problem, contact your IBM support center.

CSQY011E
csect-name COMMAND PREFIX REGISTRATION FAILED. INVALID CHARACTER(S) IN CPF

Explanation

Command prefix registration failed because the command prefix (CPF) contains invalid characters.

System action

The queue manager does not start.

System programmer response

Reissue the z/OS command SETSSI ADD with the correct CPF parameter. Correct the CPF parameter in the record of SYS1.PARMLIB member IEFSSNxx. For information about the parameters, see Updating the subsystem name table.

CSQY012E
csect-name COMMAND PREFIX REGISTRATION FAILED. INVALID CHARACTER(S) IN QUEUE MANAGER NAME

Explanation

Command prefix registration failed because the queue manager name used as the owner of the command prefix (CPF) contains invalid characters.

System action

The queue manager does not start.

System programmer response

Reissue the z/OS command SETSSI ADD with the correct CPF parameter. Correct the CPF parameter in the record of SYS1.PARMLIB member IEFSSNxx. For information about the parameters, see Updating the subsystem name table.

CSQY013E
csect-name COMMAND PREFIX REGISTRATION FAILED. CPF ALREADY DEFINED

Explanation

Command prefix registration failed because the command prefix (CPF) was already defined to z/OS.

System action

The queue manager does not start.

System programmer response

Reissue the z/OS command SETSSI ADD with the correct CPF parameter. Correct the CPF parameter in the record of SYS1.PARMLIB member IEFSSNxx. For information about the parameters, see Updating the subsystem name table.

CSQY014E
csect-name COMMAND PREFIX REGISTRATION FAILED. CPF IS A SUBSET OF A CPF ALREADY DEFINED

Explanation

Command prefix registration failed because the command prefix (CPF) is a subset of a CPF already defined to z/OS.

System action

The queue manager does not start.

System programmer response

Reissue the z/OS command SETSSI ADD with the correct CPF parameter. Correct the CPF parameter in the record of SYS1.PARMLIB member IEFSSNxx. For information about the parameters, see Updating the subsystem name table.

CSQY015E
csect-name COMMAND PREFIX REGISTRATION FAILED. CPF IS A SUPERSET OF A CPF ALREADY DEFINED

Explanation

Command prefix registration failed because the command prefix (CPF) is a superset of a CPF already defined to z/OS.

System action

The queue manager does not start.

System programmer response

Reissue the z/OS command SETSSI ADD with the correct CPF parameter. Correct the CPF parameter in the record of SYS1.PARMLIB member IEFSSNxx. For information about the parameters, see Updating the subsystem name table.

CSQY016E
csect-name SYSTEM ERROR DURING COMMAND PREFIX REGISTRATION

Explanation

A z/OS error occurred during command prefix (CPF) registration.

System action

The queue manager does not start.

System programmer response

Check the z/OS console for other messages relating to the problem.

CSQY017E
csect-name INCORRECT STORAGE PROTECT KEY

Explanation

The queue manager initialization procedures found that the storage protect key was not 7. The most likely causes are that the program properties table (PPT) entry for CSQYASCP has not been specified correctly, or that the IBM MQ program libraries or other libraries in the IBM MQ STEPLIB are not APF authorized.

System action

Queue manager startup terminates abnormally with reason code X'00E80162'.

System programmer response

Check that all the libraries you include in the IBM MQ STEPLIB have been APF-authorized. Also, ensure that you use the actual library name and not the data set alias of the libraries in APF list.

For information about specifying the PPT entry for CSQYASCP and about APF authorization for the IBM MQ program libraries, see Updating the z/OS program properties table.

CSQY018E
csect-name INCORRECT APF AUTHORIZATION

Explanation

The queue manager initialization procedures found that they were not APF authorized. The most likely cause is that one or more of the data sets in the //STEPLIB concatenation is not APF authorized.

System action

Queue manager startup terminates abnormally with reason code X'00E80163'.

System programmer response

Check all the libraries that you include in the IBM MQ STEPLIB are APF-authorized. Also, check that you do not use a data set alias of the libraries in the APF list, use the actual library name instead.

For information about APF authorization for the IBM MQ program libraries, see APF authorize the IBM MQ load libraries.

CSQY019E
csect-name QUEUE MANAGER STARTUP TERMINATED, INVALID PARAMETER MODULE LEVEL, REBUILD *macro-name*

Explanation

The queue manager initialization procedures found that the level of the parameter module (named in the preceding CSQY001I message) is not at the correct level for this version of the queue manager.

CD This can occur when starting a queue manager at a Continuous Delivery (CD) release of IBM MQ, if the system parameter (ZPARM) module has not been updated at a CD release with the value of **OPMODE** set to **OPMODE**=(NEWFUNC,90x), where x is the modification number.

System action

Queue manager startup terminates abnormally with reason code 00E80051.

System programmer response

Rebuild the parameter module ensuring that *macro-name* is recompiled with the same level of code that the queue manager is running with.

CD Ensure that **OPMODE** is set to **OPMODE**=(NEWFUNC,90x), where x is the modification number, for a CD release.

For more information about the macros used to build the parameter module see, Task 17: Tailor your system parameter module.

CSQY020E
csect-name CHANNEL INITIATOR STARTUP TERMINATED, INVALID START COMMAND

Explanation

The channel initiator can be started only by a **START CHINIT** command.

System action

Channel initiator startup is terminated.

System programmer response

Start the channel initiator using the **START CHINIT** command

CSQY021E
csect-name QUEUE MANAGER STARTUP TERMINATED, INSUFFICIENT MEMLIMIT

Explanation

The queue manager initialization procedures found that the configured MEMLIMIT is less than 512MB.

System action

Queue manager startup terminates abnormally.

CSQY022I
QUEUE MANAGER INITIALIZATION COMPLETE

Explanation

This message is issued when the initialization of the queue manager completes normally, and it is ready for use.

CSQY023A
SOME OBJECTS COULD NOT BE MIGRATED, MANUAL RESOLUTION REQUIRED. REPLY TO ACKNOWLEDGE AND CONTINUE STARTUP

Explanation

The queue manager has detected that it was previously running at an earlier version and forward migration has been performed. However, some objects could not be migrated because of locks held by in-doubt transactions. Message CSQI970E is also issued for each object that could not be migrated.

This message is not issued during subsequent restarts of the queue manager whilst it is running at the same version.

System action

Startup is suspended and the queue manager waits for the operator to reply with any single character.

System programmer response

Reply to acknowledge this message and allow queue manager startup to proceed.

Thereafter, additional action is required to complete forward migration of each identified object.

For more information see the description of message CSQI970E.

CSQY024I
IBM MQ AMS for z/OS is not installed, but the system parameter SPLCAP is set to YES

Severity
8

Explanation

The system parameter SPLCAP is set to YES in the queue manager's ZPARM, however, Advanced Message Security has not been installed.

System action

Queue manager startup is terminated.

System programmer response

If Advanced Message Security is required, ensure it has been installed correctly and the queue manager's STEPLIB has been updated to include SDRQAUTH, otherwise update the queue manager's ZPARM to set the system parameter SPLCAP to NO.

CSQY025I
IBM MQ AMS for z/OS is installed.

Severity
0

Explanation

This message indicates that Advanced Message Security is installed.

System action

Queue manager startup continues.

System programmer response

None.

CSQY027I
csect-name AMS STARTING

Severity
0

Explanation

The Advanced Message Security (AMS) address space has been started because the system parameter SPLCAP is set to YES in the queue manager's ZPARM.

System action

Connections to the queue manager are permitted, but MQI calls that might require AMS function are suspended until AMS is available. Further messages are output when the AMS feature initializes.

CSQY028I
csect-name AMS HAS STARTED

Severity
0

Explanation

Advanced Message Security (AMS) initialization has completed successfully.

System action

Applications waiting for AMS function are resumed.

CSQY029E
csect-name QUEUE MANAGER STARTUP TERMINATED, AMS INITIALIZATION FAILED

Severity
12

Explanation

A severe error occurred during initialization of Advanced Message Security (AMS).

System action

The queue manager abnormally terminates with abend code 6C6 and reason 00F00003.

System programmer response

Investigate the problem reported by preceding messages in the job log for the AMS address space (xxxxAMSM). Resolve the problem, then restart the queue manager. If you are unable to resolve the error, contact your IBM support center.

CSQY030E
csect-name QUEUE MANAGER TERMINATING, AMS NOT AVAILABLE

Severity
12

Explanation

The Advanced Message Security (AMS) address space has ended abnormally due to an unrecoverable error.

System action

The queue manager abnormally terminates with abend code 6C6 and reason 00F00003.

System programmer response

Investigate the problem reported by preceding messages in the job log for the AMS address space (xxxxAMSM). Resolve the problem, then restart the queue manager. If you are unable to resolve the error, contact your IBM support center.

CSQY031I
csect-name QUEUE MANAGER WAITING FOR AMS INITIALIZATION

Severity
0

Explanation

The Advanced Message Security (AMS) address space has been started because the system parameter SPLCAP is set to YES in the queue manager's ZPARM. This message is periodically issued until AMS initialization completes.

System action

Processing continues. Connections to the queue manager are permitted, but MQI requests that might require AMS function are suspended until AMS is available.

System programmer response

Investigate the delay in initializing Advanced Message Security by reviewing the messages output in the job log for the AMS address space (xxxxAMSM).

CSQY032E
csect-name QUEUE MANAGER STARTUP TERMINATED, UNABLE TO START AMS

Severity
12

Explanation

The queue manager attempted to start the Advanced Message Security (AMS) address space because the system parameter SPLCAP is set to YES in the queue manager's ZPARM. The AMS address space (xxxxAMSM) failed to start, which might be because another job with the same name is active, or there is an error in the started task JCL.

System action

The queue manager abnormally terminates with abend code 6C6 and reason 00F00003.

System programmer response

Investigate why the AMS address space could not be started. Resolve the problem by terminating an existing address space if one is active, or correct the started task JCL if required, then restart the queue manager.

CSQY033A
csect-name QUEUE MANAGER NOT AVAILABLE, AMS INITIALIZATION ERROR

Severity
12

Explanation

A severe error occurred during initialization of Advanced Message Security (AMS).

System action

Queue manager startup is interrupted. The queue manager accepts commands, but MQI requests that might require AMS function fail with reason code 2063 (MQRC_SECURITY_ERROR).

System programmer response

Investigate the problem reported by preceding messages in the job log for the AMS address space (xxxxAMSM). Resolve the problem, then shutdown and restart the queue manager. If you are unable to resolve the error, contact your IBM support center.

CSQY034I
csect-name QUEUE MANAGER WAITING FOR AMS TO SHUTDOWN

Severity
0

Explanation

The queue manager is stopping and has requested the Advanced Message Security (AMS) address space (xxxxAMSM) ends. This message is periodically issued until AMS shutdown completes.

System action

The queue manager continues to wait for the AMS address space to end.

System programmer response

If this message is repeatedly issued examine the job log for the AMS address space to determine why it has not ended. If the problem cannot be resolved terminate the address space to allow queue manager shutdown to continue.

CSQY035I
csect-name AMS HAS SHUTDOWN

Severity
0

Explanation

The Advanced Message Security (AMS) address space (xxxxAMSM) has ended.

System action

Queue manager shutdown continues.

V 9.0.3

CSQY036I
QMGRPROD= *prod-value*, recording product usage for *product-name*, product ID *product-id*

Explanation

This message is issued when the queue manager starts if SMF 89 product usage records are to be recorded by the queue manager. *product-name* is the descriptive name of the product, and *product-id* is the product ID that is to be used in the SMF 89 data. For example:

- QMGRPROD=MQ, recording product usage for IBM MQ for z/OS, product ID 5655-MQ9 - this indicates that usage data is to be collected for the IBM MQ for z/OS product.
- QMGRPROD=ADVANCEDVUE, recording product usage for IBM MQ Advanced for z/OS, Value Unit Edition, product ID 5655-AV1 - this indicates that usage data is to be collected for the IBM MQ Advanced for z/OS, Value Unit Edition product.

See z/OS MVS Product Management for more information on product usage recording.

V 9.0.3

CSQY037I
Product usage data is not being recorded for *product-name*, product ID *product-id*

Explanation

This message is issued when the queue manager starts, if SMF 89 product usage data is not being recorded by the queue manager. This might be because SMF 89 collection is not activated for the system.

See z/OS MVS Product Management for more information on product usage recording.

V 9.0.3

CSQY038E
csect-name QUEUE MANAGER STARTUP TERMINATED, *product* is not valid for *prod-keyword* in *prod-source*

Explanation

The queue manager initialization procedures found a value *product* for *prod-keyword* in *prod-source* that is not valid.

prod-keyword can be 'QMGRPROD' or 'AMSPROD', and *prod-source* can be 'START COMMAND', 'JCL PARM' or 'CSQ6USGP'.

The message can be issued more than once, if more than one value, that is not valid, is found.

System action

Queue manager startup terminates abnormally with reason code 00E80010.

System programmer response

Correct the value that is not valid:

- If prod-source is 'START COMMAND', see START QMGR for further information.
- If prod-source is 'JCL PARM', see Starting and stopping a queue manager for information on coding the JCL parameter for the queue manager JCL.
- If prod-source is 'CSQ6USGP', see Using CSQ6USGP for information on configuring values using CSQ6USGP.

CSQY100I

csect-name SYSTEM parameters ...

Explanation

The queue manager is being started with the system parameter values shown in the following messages.

System action

Queue manager startup processing continues.

CSQY101I

CSQY102I, CSQY103I, CSQY104I, CSQY105I, CSQY106I, CSQY107I, CSQY108I, CSQY109I, CSQY130I: *csect-name* parms

Explanation

This series of messages shows the system parameter values that the queue manager is using. (Some values are followed by their internal hexadecimal representation in parentheses.) For information about the system parameters for the CSQ6SYSP macro, see Using CSQ6SYSP.

System action

Queue manager startup processing continues.

CSQY110I

csect-name LOG parameters ...

Explanation

The queue manager is being started with the log parameter values shown in the following messages.

System action

Queue manager startup processing continues.

CSQY111I

CSQY112I, CSQY113I, CSQY114I: *csect-name* parms

Explanation

This series of messages shows the log parameter values that the queue manager is using. For information about the log parameters in the CSQ6LOGP macro, see Using CSQ6LOGP.

System action

Queue manager startup processing continues.

CSQY120I

csect-name ARCHIVE parameters ...

Explanation

The queue manager is being started with the archive parameter values shown in the following messages.

System action

Queue manager startup processing continues.

CSQY121I
CSQY122I, CSQY123I, CSQY124I: *csect-name* parms

Explanation

This series of messages shows the archive parameter values that the queue manager is using. For information about the archive parameters in the CSQ6ARVP macro, see Using CSQ6ARVP.

System action

Queue manager startup processing continues.

V 9.0.3

CSQY140I
csect-name USAGE parameters

Explanation

The queue manager is being started with the usage parameter values shown in the following messages.

These values can be overridden by values provided in the queue manager JCL or on the START QMGR command. The resolved values are shown in message CSQY037I and CSQ0619I.

V 9.0.3

CSQY141I
csect-name No USAGE parameters provided

Explanation

No queue manager usage parameters are provided, and the defaults are assumed.

These values can be overridden by values provided in the queue manager JCL or on the START QMGR command. The resolved values are shown in message CSQY037I and CSQ0619I.

V 9.0.3

CSQY142I
csect-name **parms**

Explanation

This message shows the usage parameter values that the queue manager is using. For information about the usage parameters for the CSQ6USGP macro, see Using CSQ6USGP.

CSQY200E
csect-name ARM *request-type* for element *arm-element* type *arm-element-type* failed, rc=*rc* reason=*reason*

Explanation

An ARM request (IXCARM REQUEST=*request-type*) for the specified element failed. *rc* is the return code and *reason* is the reason code (both in hexadecimal) from the call.

System action

None.

System programmer response

See the *z/OS MVS Programming Sysplex Services Reference* manual for information about the return and reason codes from the IXCARM call. If you are unable to solve the problem, contact your IBM support center.

CSQY201I
csect-name ARM REGISTER for element *arm-element* type *arm-element-type* successful

Explanation

The specified element was successfully registered with ARM.

System action

None.

CSQY202E
csect-name ARM registration failed

Explanation

An attempt to register with ARM failed.

System action

Processing continues, but automatic restart is not available.

System programmer response

See the preceding CSQY200E message for more information about the failure.

CSQY203E
csect-name ARM *request-type* for element *arm-element* type *arm-element-type* timed out, rc=*rc* reason=*reason*

Explanation

An ARM request (IXCARM REQUEST=*request-type*) was issued but some predecessor element specified in the ARM policy did not issue an ARM READY request within its specified time interval.

System action

Processing continues.

System programmer response

None required. However, if your program cannot run without the predecessor element, some installation-defined action might be necessary.

CSQY204I
csect-name ARM DEREGISTER for element *arm-element* type *arm-element-type* successful

Explanation

The specified element was successfully deregistered from ARM.

System action

None.

CSQY205I
csect-name ARM element *arm-element* is not registered

Explanation

A STOP QMGR command requested ARM restart, but the queue manager was not registered for ARM.

System action

The queue manager stops normally, but will not be automatically restarted.

System programmer response

Restart the queue manager manually.

CSQY210E
csect-name call-name call for name name-token failed, rc=*rc*

Explanation

During processing for a group connect, a name token services call failed. *rc* is the return code (in hexadecimal) from the call.

System action

If the failure occurs in the batch adapter (*csect-name* CSQBCON or CSQBDSC), the application call will fail with a reason code of MQRC_UNEXPECTED_ERROR. Otherwise (*csect-name* CSQYGRA1), processing continues, but the group connect facility will not be available.

System programmer response

See the *MVS Authorized Assembler Services Reference* manual for information about the return codes from the name token services call. If you are unable to solve the problem, take a stand-alone system dump and contact your IBM support center.

CSQY211I
csect-name Unable to add entry to group connect name table (at *table-addr*)

Explanation

During initialization for the group connect facility, a new entry could not be added to the name table for this queue manager. The most likely cause is that there is already the maximum of 32 queue managers active in the group.

System action

Processing continues, but this queue manager will not be available for group connection.

System programmer response

Reduce the number of active queue managers and restart this queue manager. If this does not solve the problem, contact your IBM support center.

CSQY212E
csect-name Unable to find the group attach table

Explanation

During initialization for the group connect facility, the group attach table could not be found. The most likely causes are that an error occurred during subsystem initialization, or that the subsystem was not initialized with the latest version of the IBM MQ early code.

System action

Processing continues, but the group connect facility will not be available to CICS.

System programmer response

Ensure that the libraries with the latest version, release, or maintenance level of the IBM MQ early code are in the libraries used for the z/OS LPA, and refresh the early code for the queue manager using the IBM MQ command REFRESH QMGR TYPE(EARLY). See the Task 3: Update the z/OS link list and LPA.

CSQY220I

csect-name Queue manager storage usage : local storage : used *mm*MB, free *nn*MB : above bar : used *aabb*,free *cc*

Explanation

This message displays the amount of virtual storage currently used and available:

- in the extended private region (local storage).
- above the Bar (64 bit storage).

  The amount of storage used is displayed in the most appropriate unit (MB / GB) according to the number of bytes, and are approximations. If the amount of storage available exceeds 10 GB, ' >10 GB' is displayed. In all other cases the amount of storage available is displayed in the most appropriate unit. For the amount of storage space available, the total is rounded down to a whole number in the appropriate unit (MB /GB). For example, if the value of 3 GB is displayed the amount of free storage is greater than or equal to 3 GB and less than 4 GB.

This message is logged at queue manager start and then either every hour if the usage does not change or when the memory usage changes (up or down) by more than 2%.

System action

Processing continues. Any special actions taken by IBM MQ or that are required, are indicated by the CSQY221I and CSQY222E messages.

System programmer response

No action is required at this time. However, a frequent occurrence of this message might be an indication that the system is operating beyond the optimum region for the current configuration.

CSQY221I

csect-name Queue manager is short of local storage

Explanation

The queue manager is running short of virtual storage in the extended private region.

System action

Processing continues. Storage contraction processing is performed, which attempts to remove unused storage from internal subpools so that it can be reused in other subpools. This might be necessary after a temporary need for a large amount of storage; for example, an unusually large unit of work being performed.

System programmer response

If only a few of these messages are output then no action is required at this time. However, a frequent occurrence of this message may be an indication that the system is operating beyond the optimum region for the current configuration and should be investigated.

CSQY222E

csect-name Queue manager is critically short of local storage - take action

Explanation

The queue manager is running critically short of virtual storage in the extended private region. Action should be taken to relieve the situation, and to avoid the possible abnormal termination of the queue manager.

System action

Processing continues. Storage contraction processing has been performed, but the remaining unallocated virtual storage is less than a predetermined safe amount. If storage use continues to increase, the queue manager might terminate abnormally in an unpredictable way.

System programmer response

Virtual storage is over-allocated for the current configuration. The following actions can reduce the virtual storage requirement:

- For buffer pools that have the LOCATION parameter set to BELOW, you can reduce buffer pool sizes with the ALTER BUFFPOOL command. Buffer pool statistics can be used to determine buffer pools which are over-allocated.
- Reduce the number of concurrent connections to the queue manager. The DISPLAY CONN command can be used to determine connections which are consuming queue manager resources.

If the problem persists after taking actions described above, it might be an indication of an internal error where storage is not freed (a 'storage leak'). If you suspect this, then collect at least two system dumps of the queue manager, separated by an interval of time, and contact your IBM support center.

CSQY223I
csect-name Queue manager is no longer short of local storage

Explanation

The queue manager is no longer short of virtual storage in the extended private region.

System action

Processing continues. Storage contraction processing has been performed, and the remaining unallocated virtual storage is more than a predetermined safe amount.

CSQY224I
csect-name Queue manager is short of local storage above the bar

Explanation

The queue manager is running short of virtual storage above the bar.

System action

Processing continues. Storage contraction processing is performed, which attempts to remove unused storage from internal subpools so that it can be reused in other subpools. This might be necessary after a temporary need for lots of storage; for example, more than the usual number of messages held on an indexed queue, or an unusually large unit of work being performed.

CSQY225E
csect-name Queue manager is critically short of local storage above the bar - take action

Explanation

The queue manager is running critically short of virtual storage above the bar. Action should be taken to relieve the situation, and to avoid the possible abnormal termination of the queue manager.

System action

Processing continues. Storage contraction processing has been performed, but the remaining unallocated virtual storage is less than a predetermined safe amount. If storage use continues to increase, the queue manager might terminate abnormally in an unpredictable way.

CSQY226I
csect-name Queue manager is no longer short of local storage above the bar

Explanation

The queue manager is no longer short of virtual storage above the bar.

System action

Processing continues. Storage contraction processing has been performed, and the remaining unallocated virtual storage is more than a predetermined safe amount.

CSQY227E

csect-name Unable to allocate storage above the bar using IARV64, RC=rc, reason=reason

Explanation

A request by the queue manager to allocate storage above the bar failed. rc is the return code and reason is the reason code (both in hexadecimal) from the z/OS IARV64 service.

System action

The queue manager will attempt to recover from the error. If recovery is not possible an application or queue manager abend, for example 5C6-00A30042, 5C6-00A31000 or 5C6-00E20045, will occur.

CSQY228E

ACE pool cannot be extended, ACELIM reached

Explanation

The internal storage pool used to manage control blocks representing new connections to the queue manager has reached the limit defined by the ACELIM system parameter.

System action

Queue manager processing continues. New connection requests might have failed, message CSQ3202E or CSM078E give further information about the affected jobs.

System programmer response

Review the configured ACELIM value. It might be useful to use a STATISTICS CLASS(2) trace to establish the normal size of the ACE pool.

See Address space storage for more information.

CSQY270E

csect-name UNRECOGNIZED MESSAGE NUMBER *message-id*

Severity

8

Explanation

An unsuccessful attempt has been made to issue the message *message-id*. This message is issued only if the requested message could not be found in the IBM MQ message directory.

System action

Processing continues as though the requested message had been issued.

System programmer response

Use the message number (*message-id*) and look up the message in this product documentation. If you are using a language other than US English, ensure that you have installed the language feature correctly and that you have the appropriate load library data set concatenations in your job. Apart from that possibility, this might be an MQ system problem; see Troubleshooting and support.

Note: Messages are also used to provide text for constructing panels and reports. If such a message cannot be found, message CSQY270E will appear on the panel or report, generally in truncated form.

CSQY271I

MESSAGE GENERATOR INITIALIZATION PARAMETERS NOT FOUND. DEFAULTS ASSUMED

Severity

4

Explanation

The message generator was unable to access the routing code initialization parameter defined by the CSQ6SYSP macro. Default values defined by that macro are assumed.

System action

Queue manager initialization continues.

System programmer response

It might be necessary to change the CSQ6SYSP macro. For information about the system parameters for the CSQ6SYSP macro, see Using CSQ6SYSP.

CSQY290E

csect-name NO STORAGE AVAILABLE

Severity
4

Explanation

There was insufficient storage available for a system routine. *csect-name* shows the system routine function:

CSQAXDPS, CSQVXDPS
User exits (other than channel)

CSQXARMY
Channel initiator automatic restart

CSQXDCTS, CSQXTRPG
Channel initiator trace

CSQXDMPS
Channel initiator system dump

CSQXLDXS
User channel exits

CSQ2GFRR, CSQ2MFRR
IMS bridge system dump

System action

Processing continues, but the function provided by the system routine will be inhibited. For example, if the routine is CSQXLDXS, then user channel exits will not be available, and channels that use them will not start.

System programmer response

If the problem occurs in the queue manager, increase the size of the its address space, or reduce the number of queues, messages, and threads being used.

If the problem occurs in the channel initiator, increase the size of the its address space, or reduce the number of dispatchers, adapter subtasks, SSL server subtasks, and active channels being used.

CSQY291E

csect-name SDUMPX FAILED, RC=0000*ssrr*, *dump-identifier*

Severity
4

Explanation

The system dump routine was unable to issue a dump; the dump identifier was as shown in the message. *rr* is the return code and *ss* is the reason code (both in hexadecimal) from the z/OS SDUMPX service.

System action

Processing continues.

System programmer response

See the *MVS Authorized Assembler Services Reference* manual for information about the return code and reason code from the SDUMPX request.

CSQY330I
Queue manager has restricted functionality

Explanation

The installation and customization options chosen for IBM MQ do not allow all functions to be used.

System action

Queue manager startup processing continues.

CSQY331E
parm value not allowed - restricted functionality

Explanation

The value specified for the *parm* system parameter is not allowed because the installation and customization options chosen for IBM MQ do not allow all functions to be used.

System action

The queue manager does not start.

CSQY332I
IMS Bridge not available - restricted functionality

Explanation

The IBM MQ-IMS bridge cannot operate because the installation and customization options chosen for IBM MQ do not allow all functions to be used.

System action

The MQ-IMS bridge does not start.

CSQY333E
Command not allowed - restricted functionality

Explanation

The command that was issued is not allowed because the installation and customization options chosen for IBM MQ do not allow all functions to be used.

z/OS For example, the queue manager might have been migrated from a previous version, and is currently operating with OPMODE=COMPAT. The function requested is not available in compatibility mode. See OPMODE on z/OS for more information.

System action

The command is ignored.

CSQY334E
csect-name keyword(*value*) not allowed - restricted functionality

Explanation

The value specified for the keyword is not allowed because the installation and customization options chosen for IBM MQ do not allow all functions to be used.

System action

The command is ignored.

System programmer response

CD

Check for message CSQ5037I, resolve the problem, and restart this queue manager. To access IBM MQ Version 8.0 capabilities, ensure that you have `OPMODE=(NEWFUNC,800)`, or `OPMODE=(NEWFUNC,900)` specified in the **CSQ6SYSP** macro.

CSQY335E

csect-name Channel *channel-name* unusable - restricted functionality

Explanation

The channel cannot be used because the installation and customization options chosen for IBM MQ do not allow all functions to be used.

System action

The requested operation fails.

CSQY336E

csect-name keyword not allowed - restricted functionality

Explanation

The keyword is not allowed because the installation and customization options chosen for IBM MQ do not allow all functions to be used.

z/OS For example, the queue manager might have been migrated from a previous version, and is currently operating with OPMODE=COMPAT. The function requested is not available in compatibility mode; see OPMODE on z/OS for further information.

System action

The command is ignored.

CSQY337E

csect-name keyword value length not allowed - restricted functionality

Explanation

The length of the value specified for the keyword is not allowed because the installation and customization options chosen for IBM MQ do not allow all functions to be used.

z/OS For example, the queue manager might have been migrated from a previous version, and is currently operating with OPMODE=COMPAT. The function requested is not available in compatibility mode; see OPMODE on z/OS for further information.

System action

The command is ignored.

CSQY340E

Queue manager has restricted functionality, but previously had full functionality. Unsupported objects will be deleted (losing messages), invalid attributes will be changed

Explanation

The installation and customization options chosen for IBM MQ do not allow all functions to be used. However, the queue manager has run previously without any functional restriction, and so might have objects and attribute settings that are not allowed with the restricted functionality.

In order to continue, these objects must be deleted (which might mean that messages are lost) and the attributes must be changed. The queue manager does this automatically.

System action

Message CSQY341D is issued and the operator's reply is awaited.

System programmer response

The operator has two options:

- Allow the queue manager to delete the objects and change the attributes, by replying 'Y'.
- Cancel the queue manager, by replying 'N'.

CSQY341D
Reply Y to continue or N to cancel

Explanation

The installation and customization options chosen for IBM MQ have changed, as indicated in the preceding CSQY340E message.

System action

The queue manager waits for the operator's reply

System programmer response

See message CSQY340E.

CSQY342I
Deleting objects and changing attributes - restricted functionality

Explanation

This message is sent if the operator answers 'Y' to message CSQY341D.

System action

The queue manager deletes the objects and changes the attributes that are not allowed with the restricted functionality.

CSQY343I
Queue manager terminating - restricted functionality not accepted

Explanation

This message is sent if the operator answers 'N' to message CSQY341D.

System action

The queue manager does not start.

V 9.0.0

CSQY344E
`csect-name` Loading of channel authentication rules failed, profile `rule-name` is restricted

Severity

8

Explanation

During initialization the queue manager was unable to load the channel authentication rules because profile `rule-name` uses function that is disabled by the queue manager mode of operation (OPMODE).

For example, the channel authentication profile has been defined with a host name but the queue manager mode of operation is OPMODE(COMPAT,800), so version 8 new functions are not available. Only the first profile found is reported so other restricted profiles might also exist.

System action

Queue manager initialization continues, but inbound channels are not permitted to start.

System programmer response

Ensure the queue manager mode of operation is correctly configured. If the configuration is correct, remove channel authentication profiles that use restricted function, then restart the queue manager.

MQ Service Provider messages (CSQZ...): z/OS V 9.0.1

CSQZ0001E

The value of the "{*0*}" attribute for service "{*1*}" is either null, blank, or consists entirely of whitespace.

Explanation

The specified property should have a non-blank value.

User action

Set an appropriate value for the property.

CSQZ0002E

The request data for service "{*0*}" is incorrect for the configured data transformation.

Explanation

The structure of the request data should conform to the data transformation schema.

User action

Ensure the request data conforms to the data transformation schema. Contact the administrator of the service if the schema was not provided.

CSQZ0003E

Service "{*0*}" is stopped and cannot be invoked.

Explanation

The service is currently stopped and cannot be invoked.

User action

Contact the administrator of the service and ask them to start the service. Then resubmit the request.

CSQZ0004E

The request to service "{*0*}" resulted in an unexpected internal error.

Explanation

An unexpected internal error occurred.

User action

Contact the IBM service organization and provide this error message along with any associated information.

CSQZ0005E

A JMS message of unexpected type "{*0*}" has been received while processing a request for service "{*1*}".

Explanation

If a data transformation is configured on the service then either a `javax.jms.TextMessage` or a `javax.jms.BytesMessage` is supported. Otherwise, only a `javax.jms.TextMessage` is supported. In either case the message will get rolled back to the configured reply queue.

User action

Ensure that only messages of a supported type are put to the reply queue.

CSQZ0006E

An unexpected `JMSException` occurred while processing a request for service "{*0*}".

Explanation

An unexpected `JMSException` occurred while processing a request.

User action

Use the information accompanying this message to resolve the problem, then resubmit the request.

CSQZ0007E

An exception occurred while looking up the connection factory or one of the destinations used by service "{*0*}" from JNDI. The JNDI name was "{*1*}". The exception follows: "{*2*}"

Explanation

An exception occurred while looking up JMS resources from JNDI.

User action

Use the information in the exception to resolve the problem.

CSQZ0008E

Service "{*0*}" caught an exception while serializing JSON data. The exception message was "{*1*}"

Explanation

A failure occurred while serializing JSON data.

User action

Use the information in the exception message to resolve the problem, then resubmit the request.

CSQZ0009E

The request to service "{*0*}" contains an incorrect `ibm-mq-md-expiry` value. The value was "{*1*}".

Explanation

The request contains an incorrect `ibm-mq-md-expiry` value.

User action

Change the value of the `ibm-mq-md-expiry` HTTP header to be a valid 32 bit integer, then resubmit the request.

CSQZ0010E

The request to service "{*0*}" contains an incorrect `ibm-mq-md-persistence` value. The value was "{*1*}".

Explanation

The request contains an incorrect `ibm-mq-md-persistence` value.

User action

Change the value of the `ibm-mq-md-persistence` HTTP header to be either false, which means that sent messages are non-persistent, or true, which means that sent messages are persistent.

CSQZ0011E

An unexpected `JMSException` occurred while processing the "`ibm-mq-usr`" HTTP header of a request for service "{0}". The current message is of type "{*1*}", name "{*2*}" and value "{*3*}"

Explanation

An unexpected `JMSException` occurred while processing the "`ibm-mq-usr`" HTTP header.

User action

Use the provided information to resolve the problem, then resubmit the request.

CSQZ0012E

Service "{*0*}" was processing the "`ibm-mq-usr`" HTTP header when a string message was detected which was not surrounded by double quotes. The HTTP header contents were "{*1*}". The error was detected at approximately offset "{*2*}".

Explanation

String message properties in the "`ibm-mq-usr`" HTTP header should be surrounded by double quotes.

User action

Use the provided offset to locate the string message in the header, and ensure it is surrounded by double quotes, then resubmit the request.

CSQZ0013E

Service "{*0*}" was processing the "`ibm-mq-usr`" HTTP header when a boolean message property was detected which had an incorrect value. The HTTP header contents were "{*1*}". The message name was "{*2*}". The value was "{*3*}".

Explanation

A boolean message can only have values of "TRUE" or "FALSE". The specified message property had a different value.

User action

Use the provided information to locate the boolean message with the incorrect value, and change it to either "TRUE" or "FALSE", then resubmit the request.

CSQZ0014E

Service "{*0*}" was processing the "`ibm-mq-usr`" HTTP header when a message of an unexpected type was detected. The HTTP header contents were "{*1*}". The message name was "{*2*}". The type was "{*3*}".

Explanation

A message property of an unexpected type was detected.

User action

Ensure that the message property is one of the following types: boolean, i1, i2, i4, i8, r4, r8, string, then resubmit the request.

CSQZ0015E

Service "{0}" was processing the "ibm-mq-usr" HTTP header when a numeric message was detected which had an incorrect value. The HTTP header contents were "{1}". The message name was "{2}". The type was "{3}". The value was "{4}".

Explanation

The specified value could not be converted into a number of the specified type.

User action

Ensure that the value can be converted into a number of the specified type, then resubmit the request.

CSQZ0016E

Service "{0}" was processing the "ibm-mq-usr" HTTP header when an empty message name was detected. The HTTP header contents were "{1}". The error was detected at offset "{2}".

Explanation

A message with an empty name was detected.

User action

Check that the message has a name, and is correctly formed. Use the provided offset information to locate the message in the header, correct it, then resubmit the request.

CSQZ0017E

Service "{0}" was processing the "ibm-mq-usr" HTTP header and could not find a semi-colon when one was expected. The HTTP header contents were "{1}". The error was detected at offset "{2}".

Explanation

An expected semi-colon could not be located.

User action

Use the provided information to establish the problem and correct it, then resubmit the request.

CSQZ0018E

Service "{0}" is configured to use a topic. The HTTP GET and DELETE methods are not supported in this configuration.

Explanation

Only the HTTP POST method is supported with services that use a topic.

User action

Consider using a different, queue based, service.

CSQZ0019E

The request to service "{0}" contained an incorrect "ibm-mq-gmo-waitInterval" value. The value was "{1}".

Explanation

The request contains an incorrect "ibm-mq-gmo-waitInterval" value.

User action

Change the value of the "ibm-mq-gmo-waitInterval" HTTP header to be a valid 64 bit integer, then resubmit the request.

CSQZ0020E

Service "{0}" is configured to use a queue. The "ibm-mq-pmo-retain" HTTP header is not supported with queues.

Explanation

The request contains the "`ibm-mq-pmo-retain`" HTTP header. This is not supported with services that are configured to use queues.

User action

Delete the "`ibm-mq-pmo-retain`" HTTP header, then resubmit the request.

CSQZ0021E

The request to service "{*0*}" contained an incorrect "{*1*}" HTTP header. The header value was "{*2*}".

Explanation

The specified header was incorrect. If it was prefixed with "`0x:`" then there should be a 24 byte hexadecimal number after the prefix. Otherwise it should be a string, optionally surrounded with double quotes.

User action

Correct the header so that it is correctly formatted, then resubmit the request.

CSQZ0022E

The code page "{*0*}" corresponding to `receiveTextCCSID` "{*1*}" for service "{*2*}" is not supported..

Explanation

The code page is not installed on the server.

User action

Either install the code page or use a CCSID corresponding to a different code page.

CSQZ0023E

The request to service "{*0*}" had an unsupported content type of "{*1*}".

Explanation

The service only supports a content type of application `json` and a character set of utf-8.

User action

Correct either the content type, or character set, of the request and resubmit the request.

CSQZ0024E

The request to service "{*0*}" successfully got a message under a transaction, but the attempt to commit the transaction resulted in the transaction rolling back.

Explanation

HTTP DELETE requests to the service begin a user transaction to minimize the chance of message data being lost. The attempt to commit the transaction failed, resulting in the transaction rolling back.

User action

This is likely to be a temporary error, resubmit the request.

CSQZ0025E

An exception occurred while looking up the connection factory or one of the destinations used by service "{*0*}" from JNDI. The JNDI name was "{*1*}".

Explanation

A failure occurred while looking up JMS resources from JNDI.

User action

Contact the administrator of the service so that they can resolve the problem.

CSQZ0026E

Service "{*0*}" located an object from JNDI, but the object was not of the expected type. The expected type was "{*1*}". The actual type was "{*2*}". The JNDI name was "{*3*}".

Explanation

The object located from JNDI was not of the expected type.

User action

Adjust the configuration of the service so that the correct object can be located.

CSQZ0027E

Service "{*0*}" located an object from JNDI, but the object was not of the expected type. The expected type was "{*1*}". The actual type was "{*2*}". The JNDI name was "{*3*}".

Explanation

The object located from JNDI was not of the expected type.

User action

Contact the administrator of the service so that they can adjust the configuration of the service so that the correct object can be located.

CSQZ0028E

A request to service "{*0*}" resulted in an unexpected internal error.

Explanation

An unexpected internal error occurred.

User action

Contact the IBM service organization, and provide this error message along with the FFDC which will have occurred at the same time.

CSQZ0029E

The `receiveTextCCSID` "{*0*}" used by service "{*1*}" does not exist, or is not supported.

Explanation

The specified CCSID either does not exist or is not supported on the server.

User action

Set the "`receiveTextCCSID`" attribute of the service to a valid, supported CCSID.

CSQZ0030E

Service "{*0*}" has the "**replyDestination**" attribute set but the value of the "**waitInterval**" attribute is 0.

Explanation

The "**waitInterval**" attribute can not have a value of zero if the "**replyDestination**" attribute is set.

User action

Change the value of "**waitInterval**" to a non-zero integer value.

CSQZ0031E

The request to service "{*0*}" failed when performing a data transformation on the request data, prior to sending it to IBM MQ.

Explanation

Data transformation for the request data failed. The data was not sent to IBM MQ.

User action

Check the accompanying stack trace and if the issue cannot be fixed by changing the content of the request data, contact the administrator of the service, as they might need to adjust the configuration of the data transformation.

CSQZ0032E

The request to service "{*0*}" failed when performing a data transformation on the data received from IBM MQ.

Explanation

The data transformation was applied to the data received from IBM MQ, but the transformation failed.

User action

Contact the administrator of the service as they might need to adjust the configuration of the data transformation.

CSQZ0033E

The "`password`" attribute of service "{*0*}" cannot be decoded.

Explanation

An error occurred when decoding the "`password`" attribute. This might be because it was incorrectly formatted.

User action

Ensure that the "`password`" attribute has been correctly encoded. The **`securityUtility`** tool should be used to encode passwords. Use the associated error information in the log file to help resolve the problem.

CSQZ0034E

The combination of "`userName`" and "`password`" attributes of service "{*0*}" is incorrect.

Explanation

Either the "`userName`" and "`password`" attributes must both be blank, or they must both be specified.

User action

Ensure that either the "`userName`" and "`password`" attributes are both blank, or both specified.

CSQZ0035E

Service "{*0*}" is not configured correctly.

Explanation

The service is not configured correctly.

User action

Contact the administrator of the service and ask them to correct the configuration of the service. Then resubmit the request.

CSQZ0036E

The value "{*2*}" of the "{*1*}" attribute of service "{*0*}" can not be converted to the expected type "{*3*}".

Explanation

The attribute of the service has a value of an unexpected type.

User action

Correct the value of the attribute.

CSQZ0037E

The value of the "mqmdFormat" attribute of service "{0}" exceeds the maximum length of 8.

Explanation

The value of the "mqmdFormat" attribute of the service exceeds the maximum length of 8.

User action

Correct the value of the "mqmdFormat" attribute.

CSQZ0038E

The value "{1}" of the "replySelection" attribute of service "{0}" is not one of: "none", "msgIDToCorrelID" or "correlIDToCorrelID".

Explanation

The "replySelection" attribute of the service does not have a valid value.

User action

Correct the value of the "replySelection" attribute.

CSQZ0039E

A service with no 'id' attribute has been discovered, users will not be able to call this service.

Explanation

A service without an id cannot be referenced by users.

User action

Contact the administrator of the service and ask them to correct the configuration of the service. Then resubmit the request.

CSQZ0040E

Service "{0}" cannot be started or stopped because it is not active.

Explanation

An unexpected internal error occurred when trying to start or stop the service.

User action

Contact the IBM service organization and provide this error message along with the FFDC which will have occurred at the same time.

CSQZ0041E

A method on service "{0}" cannot be invoked because the service is inactive.

Explanation

An unexpected internal error occurred when trying to activate the service.

User action

Contact the IBM service organization and provide this error message along with the FFDC which will have occurred at the same time.

CSQZ0042I

"{0}" code level is "{1}".

Explanation

This message is for informational purposes only.

User action

None; this message is for informational purposes only.

Advanced Message Security (CSQ0...): z/OS

CSQ0101E

csect-name Internal message protection error, reason *reason*, diagnostics: *value1,value2*

Severity

8

Explanation

An internal error occurred during message protection processing.

System action

For a put operation, the message is rejected.

For a get operation, the message is moved to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE**.

System programmer response

You should check that the message has valid IBM MQ headers, if not, contact your IBM support center.

CSQ0105E

csect-name Internal error occurred, reason *reason*, diagnostics: *value*

Severity

8

Explanation

An internal error occurred during message protection processing.

System action

The message queue interface (MQI) call fails.

System programmer response

Contact your IBM support center.

CSQ0109E

csect-name SDUMP failed, return code *rc*, reason *reason*

Severity

8

Explanation

An attempt to issue an SDUMP during abend processing failed.

System action

SDUMP diagnostics are not generated.

System programmer response

Review the return code and reason in conjunction with SDUMP documentation to resolve the problem.

CSQ0110I

csect-name AMS abend *abend*, reason *reason*

Severity
8

Explanation

An abend has occurred during message protection processing of type *abend* for reason *reason*.

System action

The message queue interface (MQI) call fails and the IBM MQ subsystem might terminate.

System programmer response

Use the abend and reason code information to resolve the issue.

If the problem cannot be resolved contact your IBM support center.

CSQ0111I

csect-name Module offset *offset*, level *level*

Severity
0

Explanation

The *module* and *level* are reported for diagnostic purposes following an abend during message protection processing.

System action

Processing continues.

System programmer response

None.

CSQ0112I

csect-name PSW SDWAEC1, *SDWAEC1*, SDWAAEC1, *SDWAAEC1*

Severity
0

Explanation

Relevant Program Status Word (PSW) fields are reported following an abend during message protection processing.

System action

Processing continues.

System programmer response

None.

CSQ0113I

csect-name CAB *value1*, *value2*, *value3*, *value4*

Severity
0

Explanation

Relevant internal fields are reported following an abend during message protection processing.

System action

Processing continues.

System programmer response

None.

CSQ0114I

csect-name R0-R3 *gpr0*, *gpr1*, *gpr2*, *gpr3*

Severity
0

Explanation

General purpose registers 0 through 3 are reported following an abend during message protection processing.

System action

Processing continues.

System programmer response

None.

CSQ0115I

csect-name R4-R7 *gpr4*, *gpr5*, *gpr6*, *gpr7*

Severity
0

Explanation

General purpose registers 4 through 7 are reported following an abend during message protection processing.

System action

Processing continues.

System programmer response

None.

CSQ0116I

csect-name R8-R11 *gpr8*, *gpr9*, *gpr10*, *gpr11*

Severity
0

Explanation

General purpose registers 8 through 11 are reported following an abend during message protection processing.

System action

Processing continues.

System programmer response

None.

CSQ0117I

csect-name R12-R15 *gpr12*, *gpr13*, *gpr14*, *gpr15*

Severity
0

Explanation

General purpose registers 12 through 15 are reported following an abend during message protection processing.

System action

Processing continues.

System programmer response

None.

CSQ0118I

csect-name A0-A3 *ar0*, *ar1*, *ar2*, *ar3*

Severity
0

Explanation

Access registers 0 through 3 are reported following an abend during message protection processing.

System action

Processing continues.

System programmer response

None.

CSQ0119I

csect-name A4-A7 *ar4*, *ar5*, *ar6*, *ar7*

Severity
0

Explanation

Access registers 4 through 7 are reported following an abend during message protection processing.

System action

Processing continues.

System programmer response

None.

CSQ0120I

csect-name A8-A11 *ar8*, *ar9*, *ar10*, *ar11*

Severity
0

Explanation

Access registers 8 through 11 are reported following an abend during message protection processing.

System action

Processing continues.

System programmer response

None.

CSQ0121I

csect-name A12-A15 *ar12*, *ar13*, *ar14*, *ar15*

Severity
0

Explanation

Access registers 12 through 15 are reported following an abend during message protection processing.

System action

Processing continues.

System programmer response

None.

CSQ0137I

csect-name SDUMP not taken, suppressed by DAE

Severity
0

Explanation

An SDUMP was suppressed due to Dump Analysis and Elimination (DAE).

System action

Processing continues.

System programmer response

None.

CSQ0151E

csect-name Failed to allocate storage

Severity
8

Explanation

An attempt to allocate storage during message protection processing failed.

System action

The message queue interface (MQI) call fails.

System programmer response

Increase the amount of storage available.

CSQ0174E

csect-name Failed to load module *module*, return code *abncode*, reason *rsncode*

Severity
8

Explanation

An attempt to load a module into storage failed.

System action

The IBM MQ subsystem fails to start.

System programmer response

Use the abend and reason code in conjunction with documentation for the **LOAD** macro to resolve the problem.

CSQ0175E

csect-name Failed to delete module *module*, return code *rc*

Severity
8

Explanation

An attempt to delete a loaded module failed.

System action

The module remains loaded.

System programmer response

Use the return code in conjunction with documentation for the **DELETE** macro to resolve the problem.

CSQ0201E

csect-name Message table not available

Severity
8

Explanation

An attempt to load the message protection component message file failed.

System action

The IBM MQ subsystem fails to start.

System programmer response

Verify that the IBM MQ subsystem has been installed correctly. If the problem persists contact your IBM support center.

CSQ0204I

csect-name AMS is using *use-size* MB of local storage, *free-size* MB free

Severity
0

Explanation

The amount of storage currently used for message protection services is currently *use-size* MB, and a further *free-size* remains free.

System action

Processing continues.

System programmer response

None.

CSQ0209E

csect-name Message for *qname* sent to error queue, MQRC=*mqrc (mqrc-text)*

Severity
4

Explanation

During get processing a protected message on queue *qname* could not be processed for reason *mqrc*, and has been sent to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE** (*mqrc-text* provides the MQRC in textual form).

System action

The message is placed on the error queue and an error is returned to the requesting application.

System programmer response

Examine the message on the error queue and the reason code to determine why the message could not be processed.

You should check the sender and receiver policies. For example, when setting the policy:

- Specify the State or Province using ST=
- The following special characters need escape characters:

```
, (comma)
+ (plus)
" (double quote)
\ (backslash)
< (less than)
> (greater than)
; (semicolon)
```

- If the Distinguished Name contains embedded blanks, you should enclose the DN in double quotation marks.

CSQ0210E

csect-name Failed to redirect message to error queue, MQRC=*mqrc (mqrc-text)*

Severity
8

Explanation

An attempt during message protection processing to put a message to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE**, failed for reason *mqrc* (*mqrc-text* provides the MQRC in textual form).

System action

The get request fails and the message remains on the target queue.

System programmer response

Examine the message on the target queue and the reason code to determine why the message could not be processed or placed on the error queue.

CSQ0213E

csect-name Internal queue close failed MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Severity
8

Explanation

During open processing for a protected queue protection initialization failed and an attempt to internally close the queue failed.

System action

The open request fails.

System programmer response

Examine the completion and reason codes to determine the cause of the failure.

CSQ0214E

csect-name Message protection initialization failed, return code *rc*, reason *reason*

Severity
8

Explanation

During open processing for a protected queue protection initialization failed.

System action

The open request fails.

System programmer response

Examine the completion and reason codes to determine the cause of the failure. For more information, see Messages and codes in the *z/OS Cryptographic Services System SSL Programming* documentation.

CSQ0215E

csect-name Message protection failed, return code *rc*, reason *reason*

Severity
8

Explanation

An attempt to protect a message failed during put processing.

System action

The message is not put to the queue.

System programmer response

Examine the completion and reason codes to determine the cause of the failure. For more information, see Messages and codes in the *z/OS Cryptographic Services System SSL Programming* documentation.

CSQ0216E

csect-name Message unprotection failed, return code *rc*, reason *reason*

Severity
8

Explanation

An attempt to process a protected message during get processing failed.

System action

The message is moved to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE**, if possible.

System programmer response

Examine the completion and reason codes to determine the cause of the failure. For more information, see Messages and codes in the *z/OS Cryptographic Services System SSL Programming* documentation.

CSQ0217E

csect-name Failed to process object *'objname'*

Severity
8

Explanation

An attempt to initialize, protect, or process a protected message failed for the object named by *objname*.

System action

The open, get or put request fails.

System programmer response

Examine preceding or subsequent console messages for more information.

CSQ0218E

csect-name Privacy policy for *qname* invalid. No recipients

Severity
8

Explanation

During open or put1 processing, a privacy policy was stipulated for the object *qname*, but the policy failed to identify any recipients.

System action

The open or put1 request fails.

System programmer response

Modify or delete the protection policy for the object *qname*.

CSQ0219E

csect-name Message verification error for *qname*

Severity
8

Explanation

During put or get processing an attempt to process a message failed due to unexpected header values or offsets.

System action

The put or get operation fails.

For get processing the message is moved to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE**.

System programmer response

Examine the failing message to determine the cause of the problem.

CSQ0220E

csect-name Encryption strength not available

Severity
8

Explanation

During get processing a protected message did not specify an encryption strength.

System action

The get request fails and the message is moved to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE**.

System programmer response

Examine the message on the error queue to determine its origin and why it is not correctly protected.

CSQ0221E

csect-name Message encryption strength *encstr* not valid

Severity

8

Explanation

During get processing a protected message did not have a recognized encryption strength.

System action

The get request fails and the message is moved to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE**.

System programmer response

Examine the message on the error queue to determine its origin and why it does not have a valid encryption strength.

Some encryption algorithms are supported on some platforms, however, not on others.

CSQ0222E

csect-name Message encryption strength *encstr* inconsistent with policy

Severity

8

Explanation

During get processing a protected message did not use an encryption algorithm that matches the expected encryption strength.

System action

The get request fails and the message is moved to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE**.

System programmer response

Examine the message on the error queue to determine its origin and why the encryption algorithm does not match the expected encryption strength.

Some encryption algorithms are supported on some platforms, however, not on others.

CSQ0223E

csect-name Message size *m-size* inconsistent with header size *h-size* or original size *o-size*

Severity

8

Explanation

During get processing a protected message was found to have a header or overall message size that did not match the original unprotected message.

System action

The get request fails and the message is moved to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE**.

System programmer response

Examine the message on the error queue to determine why its lengths are inconsistent with the original unprotected message.

CSQ0224E

csect-name Message buffer length of *m-size* too small

Severity

8

Explanation

During get processing a protected message was of insufficient length to contain a standard protection header and could not be processed.

System action

The get request fails and the message is moved to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE**.

System programmer response

Examine the message on the error queue to determine why it is of insufficient length.

CSQ0225E

csect-name Message header not acceptable, structure identifier is *'strucid'*

Severity

8

Explanation

During get processing a protected message did not have the expected protection header eye-catcher, but instead had *strucid*.

System action

The get request fails and the message is moved to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE**.

System programmer response

Examine the message on the error queue to determine why it has an invalid protection header.

CSQ0226E

csect-name Header version not supported

Severity

8

Explanation

During get processing a protected message did not have the expected protection header version.

System action

The get request fails and the message is moved to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE**.

System programmer response

Examine the message on the error queue to determine why it has an invalid protection header.

CSQ0227E

csect-name Message signature algorithm *sig-alg* not valid

Severity

8

Explanation

During get processing a protected message did not have a recognized signature algorithm.

System action

The get request fails and the message is moved to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE**.

System programmer response

Examine the message on the error queue to determine its origin and why it does not have a valid signature algorithm. Some signature algorithms are supported on some platforms, however, not on others.

CSQ0228E

csect-name Message signature algorithm *sig-alg* inconsistent with policy

Severity

8

Explanation

During get processing a protected message did not use a signature algorithm that matches the expected signature strength.

System action

The get request fails and the message is moved to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE**.

System programmer response

Examine the message on the error queue to determine its origin and why the signature algorithm does not match the expected signature strength. Some signature algorithms are supported on some platforms, however, not on others.

CSQ0229E

csect-name Unable to verify sender distinguished name

Severity

8

Explanation

During get processing the distinguished name of the message sender was not present to be verified.

System action

The get request fails and the message is moved to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE**.

System programmer response

Examine the message on the error queue to determine why it does not have a distinguished name for the sender of the message.

CSQ0230E

csect-name Structure identifier *strucid* invalid for format name *format*

Severity
8

Explanation

During message protection processing a message header did not have the expected value for the message format indicated.

System action

The MQI call fails. For a get request the message is moved to the error queue,**SYSTEM.PROTECTION.ERROR.QUEUE**.

System programmer response

Examine the failing message formats and headers to determine the cause of the problem.

CSQ0231E

csect-name Unrecognized version *version* for structure *strucid*, format name *format*

Severity
8

Explanation

During message protection processing a message header version did not have the expected value for the message format and header indicated.

System action

The MQI call fails. For a get request the message is moved to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE**.

System programmer response

Examine the failing message formats and headers to determine the cause of the problem.

CSQ0232E

csect-name Buffer length insufficient for format name *format*

Severity
8

Explanation

During message protection processing a message length was insufficient to account for the length of a header indicated by the message format.

System action

The MQI call fails. For a get request the message is moved to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE**.

System programmer response

Examine the failing message formats and headers to determine the cause of the problem.

CSQ0233E

csect-name Message *msg-size* of size does not match original size of *orig-size*

Severity
8

Explanation

During get processing a protected message length does not resolve to the original length of the message before it was protected.

System action

The get request fails and the message is moved to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE**.

System programmer response

Examine the message on the error queue to determine why it has a different length than expected.

CSQ0240E

csect-name No storage for error queue processing for *qname*

Severity
8

Explanation

During get processing a message that failed protection processing could not be put to the error queue, **SYSTEM.PROTECTION.ERROR.QUEUE**, due to insufficient storage.

System action

The get request fails and the message remains on the queue *qname*.

System programmer response

Determine the cause of storage shortage and retry the get operation.

CSQ0400I

CSQ0UTIL IBM MQ AMS for z/OS *vrm*

Severity
0

Explanation

The Advanced Message Security policy utility, CSQ0UTIL, is starting for utility version *v*, release *r*, and modlevel *m*.

System action

Processing continues.

System programmer response

None.

CSQ0401I

Queue Manager Protection Policy Utility

Severity
0

Explanation

The Advanced Message Security policy utility, CSQ0UTIL, has started.

System action

Processing continues.

System programmer response

None.

CSQ0402I

Command Name: *command*

Severity
0

Explanation

The Advanced Message Security policy utility is processing the policy command *command*.

System action

Processing continues.

System programmer response

None.

CSQ0403I

Arguments: args

Severity
0

Explanation

The Advanced Message Security policy utility is processing arguments *args* for the current policy command.

System action

Processing continues.

System programmer response

None.

CSQ0404E

Insufficient storage available to perform command

Severity
8

Explanation

The Advanced Message Security policy utility could not allocate storage to process the input command.

System action

The Advanced Message Security policy utility terminates without executing the current input command.

System programmer response

Determine why there is insufficient storage for the policy utility then rerun the utility when the problem has been resolved.

CSQ0405E

An error occurred running command *cmd-number* on line *line number*

Severity
8

Explanation

The Advanced Message Security policy utility encountered an error during the processing of command *cmd-number* at line *line-number* of the input.

System action

The Advanced Message Security policy utility processing fails for the command identified by *cmd-number* at line *line-number*.

System programmer response

Examine the failing command and related messages to determine the cause of the failure.

CSQ0406E

Invalid command found on line *line-number*. Valid commands are SETMQSPL and DSPMQSPL

Severity

8

Explanation

The Advanced Message Security policy utility did not recognize the input command at line *line-number* of the input.

System action

The Advanced Message Security policy utility does not process the command at line *line-number*.

System programmer response

Change the input command to either **SETMQSPL** or **DSPMQSPL**.

CSQ0407E

Quoted string on line *line-number* does not have a terminating quote

Severity

8

Explanation

The Advanced Message Security policy utility could not align matching quotes when processing the input command at line *line-number*.

System action

The Advanced Message Security policy utility does not process the command at line *line-number*.

System programmer response

Change the command at line *line-number* to use consistent and matching quotes in the arguments.

CSQ0408I

cmd-count policy commands have been completed successfully

Severity

0

Explanation

The Advanced Message Security policy utility has successfully processed *cmd-count* commands.

System action

Processing continues.

System programmer response

None.

CSQ0409I

Reached end of input, *num-line* lines read

Severity

0

Explanation

The Advanced Message Security policy utility reached end of input after *num-line* lines.

System action

Processing continues.

System programmer response

None.

CSQ0410E

Error opening SYSIN data set

Severity
8

Explanation

The Advanced Message Security policy utility could not open the standard input (SYSIN DD) to read input commands.

System action

No commands are processed.

System programmer response

Determine why the SYSIN DD is unavailable and resolve the problem, then rerun the policy utility.

CSQ0411E

Unexpected internal error

Severity
8

Explanation

The Advanced Message Security policy utility did not recognize the input command.

System action

The input command is not processed.

System programmer response

Examine the command input and verify that the input expresses a valid command with valid parameters.

CSQ0412I

Policy name: *policy-name*

Severity
0

Explanation

The Advanced Message Security policy utility is displaying information about the policy identified by *policy-name*.

System action

Processing continues.

System programmer response

None.

CSQ0413I

Encryption algorithm: *enc-alg*

Severity
0

Explanation

The Advanced Message Security policy utility is displaying the encryption algorithm *enc-alg* for a given policy.

System action

Processing continues.

System programmer response

None.

CSQ0414I

Recipient DNs: *recipient-dns*

Severity
0

Explanation

The Advanced Message Security policy utility is displaying the recipient distinguished names, *recipient-dns*, for a given policy.

System action

Processing continues.

System programmer response

None.

CSQ0415I

Signature algorithm: *sig-alg*

Severity
0

Explanation

The Advanced Message Security policy utility is displaying the signature algorithm, *sig-alg*, for a given policy.

System action

Processing continues.

System programmer response

None.

CSQ0416I

Signer DNs: *signer-dns*

Severity
0

Explanation

The Advanced Message Security policy utility is displaying the signer distinguished names, *signer-dns*, for a given policy.

System action

Processing continues.

System programmer response

None.

CSQ0417I

Quality of protection: *qop*

Severity
0

Explanation

The Advanced Message Security policy utility is displaying the quality of protection, *qop*, for a given policy.

System action

Processing continues.

System programmer response

None.

CSQ0418I

Toleration: *toleration-flag*

Severity
0

Explanation

The Advanced Message Security policy utility is displaying the toleration flag, *toleration-flag*, for a given policy.

System action

Processing continues.

System programmer response

None.

CSQ0447E

Failed to open EXPORT DD, exporting to STDOUT

Severity
8

Explanation

The Advanced Message Security policy utility could not open the EXPORT DD to process a -export request.

System action

The policy export is sent to STDOUT.

System programmer response

Determine why the EXPORT DD is unavailable and resolve the problem, then rerun the policy utility.

CSQ0448E

Command failed

Severity
8

Explanation

The Advanced Message Security policy utility failed to successfully process an input command.

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Determine the reason for the failing command by examining related messages, and correct the failing input command.

CSQ0449I

Command successful

Severity
0

Explanation

The Advanced Message Security policy utility successfully processed an input command.

System action

Processing continues.

System programmer response

None.

CSQ0450E

Syntax error. Usage: setmqspl -m (qm) -p (policy) -s (sigalg) -a (signer DN) -e (encalg) -r (receiver DN)

Severity
8

Explanation

The Advanced Message Security policy utility failed to interpret a command due to bad command syntax.

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Correct the syntax of the failing command then retry.

CSQ0451E

Invalid queue manager name: *qmgr-name*

Severity
8

Explanation

The Advanced Message Security policy utility encountered an invalid queue manager name value, *qmgr-name*, when processing an input command.

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Correct the queue manager name value in the input command then retry.

CSQ0452E

Invalid policy name: *policy-name*

Severity
8

Explanation

The Advanced Message Security policy utility encountered an invalid policy name when processing an input command.

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Correct the policy name value in the input command then retry.

CSQ0453E

Invalid encryption algorithm

Severity
8

Explanation

The Advanced Message Security policy utility encountered an invalid encryption algorithm when processing an input command.

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Correct the encryption algorithm value in the input command then retry.

CSQ0454E

Invalid signature algorithm

Severity
8

Explanation

The Advanced Message Security policy utility encountered an invalid signature algorithm when processing an input command.

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Correct the signature algorithm value in the input command and retry.

CSQ0455E

Encryption requires the use of a signature algorithm

Severity

8

Explanation

The Advanced Message Security policy utility encountered an invalid command that identified an encryption algorithm, but did not also identify a valid signature algorithm.

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Provide both a valid encryption algorithm and a valid signature algorithm when defining privacy protection policies.

CSQ0456E

Encryption requires a receiver DN to be specified (-r)

Severity

8

Explanation

The Advanced Message Security policy utility encountered an invalid command that identified an encryption algorithm, but did not also identify at least one recipient DN via the -r parameter.

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Provide both an encryption algorithm and at least one recipient DN when defining privacy protection policies.

CSQ0457E

Invalid receiver DN specified: *receiver-dn*

Severity

8

Explanation

The Advanced Message Security policy utility encountered an invalid receiver distinguished name, *receiver-dn*, when processing an input command.

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Correct the receiver distinguished name in the input command then retry.

CSQ0458E

Receiver DN is specified while no encryption is enabled

Severity

8

Explanation

The Advanced Message Security policy utility encountered an invalid command that identified at least one recipient DN, however, did not also identify an encryption algorithm when processing the -e parameter.

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Provide both an encryption algorithm and at least one recipient DN when defining privacy protection policies.

CSQ0459E

Invalid signer DN specified: *signer-dn*

Severity

8

Explanation

The Advanced Message Security policy utility encountered an invalid signer distinguished name, *signer-dn*, when processing an input command.

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Correct the signer distinguished name value in the input command then retry.

CSQ0460E

Signer DN is specified while no signing is enabled

Severity

8

Explanation

The Advanced Message Security policy utility encountered an invalid command that identified at least one signer DN using the -a parameter, however, did not also identify a signature algorithm using the -s parameter.

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Provide both an encryption algorithm and at least one recipient DN when defining privacy protection policies.

CSQ0461E

Queue **SYSTEM.PROTECTION.POLICY.QUEUE** unavailable, MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Severity

8

Explanation

The Advanced Message Security policy utility could not open the policy queue, **SYSTEM.PROTECTION.POLICY.QUEUE**, due to an error identified by *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

System action

Processing ends.

System programmer response

Determine the reason the policy queue is unavailable using the *mqcc* and *mqrc*, then resolve the problem.

CSQ0462E

Failed to retrieve protection policy, MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Severity

8

Explanation

The Advanced Message Security policy utility could not retrieve a policy from the policy queue, **SYSTEM.PROTECTION.POLICY.QUEUE**, due to an error identified by *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Determine the reason the policy could not be retrieved from the policy queue, then resolve the problem.

CSQ0463E

Policy update failed due to concurrent update, MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Severity

8

Explanation

The Advanced Message Security policy utility detected that a policy was changed by another process when it was trying to update or create the same policy, due to an error identified by *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Verify the policy is correct, then update the policy again if necessary.

CSQ0464E

Policy definition not found, MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Severity

8

Explanation

The Advanced Message Security policy utility could not find a policy on the policy queue, **SYSTEM.PROTECTION.POLICY.QUEUE**, when a policy was expected, due to an error identified by *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Determine the reason the policy could not be retrieved from the policy queue, then resolve the problem.

CSQ0465E

An unexpected error occurred, MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Severity

8

Explanation

The Advanced Message Security policy utility encountered an unexpected MQI error when processing an input command, identified by *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Determine the reason the for the MQI error, then resolve the problem.

CSQ0466E

Invalid value specified for toleration flag, specify one of (0, 1)

Severity

8

Explanation

The Advanced Message Security policy utility encountered an invalid value for the toleration parameter.

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Use a value of 0 (false) or 1 (true) for the toleration parameter when creating or modifying a policy.

CSQ0467E

Failed to connect to the queue manager, MQCC=*mqcc* MQRC=*mqrc (mqrc-text)*

Severity

8

Explanation

The Advanced Message Security policy utility could not connect to the input queue manager to process further input commands, due to an error identified by *mqcc* and *mqrc* (*mqrc-text* provides the MQRC in textual form).

System action

Processing ends.

System programmer response

Determine the reason the queue manager is unavailable, then resolve the problem.

CSQ0468I

No policies found

Severity

0

Explanation

The Advanced Message Security policy utility found no policies matching the specified parameters.

System action

Processing continues.

System programmer response

None.

V 9.0.0

CSQ0469E

Invalid value specified for key reuse argument

Severity

8

Explanation

The Advanced Message Security policy utility encountered an invalid value for the key reuse parameter.

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Use a valid value for the key reuse parameter when creating or modifying a policy.

CSQ0470E

Syntax error. Usage: dspmqspl -m (qm) -p (policy) -export

Severity

8

Explanation

The Advanced Message Security policy utility failed to interpret a command due to incorrect syntax.

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Correct the syntax of the failing command then retry.

V 9.0.0

CSQ0471E

Key reuse not valid for policy

Severity
8

Explanation

The Advanced Message Security policy utility encountered an invalid command that specified a non-zero value for the key reuse parameter for a policy that does not allow symmetric key reuse.

System action

The current command is not processed and the Advanced Message Security policy utility attempts to process the next input command, if any.

System programmer response

Either specify a value of `0` for the key reuse parameter (key reuse disabled), or change the policy to use a quality of protection that allows symmetric key reuse, for example, confidentiality.

CSQ0499I

CSQ0UTIL Utility completed return code=*retcode*

Severity
0

Explanation

The Advanced Message Security policy utility, CSQ0UTIL, has completed with return code *retcode*.

System action

Processing continues.

System programmer response

If the utility did not complete successfully refer to other messages in the output to determine the cause of any errors.

CSQ0501I

csect-name SMF recording enabled for record type *record-type*

Severity
0

Explanation

Advanced Message Security has enabled SMF record generation for record type *record-type*.

System action

Processing continues.

System programmer response

None.

CSQ0502I

csect-name SMF recording disabled

Severity

0

Explanation

Advanced Message Security has disabled SMF record generation.

System action

Processing continues.

System programmer response

None.

CSQ0503I

csect-name SMF record write failed, return code *retcode*

Severity

8

Explanation

An attempt the generate an SMF audit record using SMFEWTM failed during message protection processing with return code *retcode*.

System action

The SMF record is not generated.

System programmer response

Examine the *retcode* and documentation for the SMFEWTM macro to determine the cause of the failure.

CSQ0600I

csect-name IBM MQ AMS for z/OS, *version*, *service-level*

Severity

0

Explanation

The Advanced Message Security task is running at version *version* and service level *service-level*.

System action

Processing continues.

System programmer response

None.

CSQ0601I

csect-name Environment variable *varname* has an invalid value, using default *'value'*

Severity

8

Explanation

A Advanced Message Security environment variable, *varname*, was set to an invalid value, resulting in the use of a default value, *value*, for the variable.

System action

Processing continues with the default value for the named environment variable.

System programmer response

Change the environment variable assignment to a valid value if the default is not acceptable.

CSQ0602I

csect-name AMS initialization complete

Severity

0

Explanation

The Advanced Message Security task initialization is complete.

System action

Processing continues.

System programmer response

None.

CSQ0603I

csect-name AMS shutdown requested

Severity

0

Explanation

The Advanced Message Security task has received a shutdown request.

System action

Processing continues.

System programmer response

None.

CSQ0604I

csect-name LOG option processed: *log-option*

Severity

0

Explanation

The Advanced Message Security task processed a LOG command for log option *log-option*.

System action

Processing continues with the new log option.

System programmer response

None.

CSQ0605E

csect-name Incorrect LOG option specified

Severity

8

Explanation

An attempt by the Advanced Message Security task to process a LOG command failed due to an invalid log option.

System action

The LOG command does not take effect.

System programmer response

Correct the LOG option and retry the LOG command.

CSQ0606E

csect-name Unrecognized command: specify **DISPLAY**, **REFRESH**, **LOG**, **SMFTYPE**, **SMFAUDIT** or **STOP**

Severity
8

Explanation

An attempt by the Advanced Message Security task to process a command failed because the command was not recognized.

System action

The command is not executed.

System programmer response

Select a valid command and retry.

Valid commands include **DISPLAY**, **REFRESH**,**LOG**, **SMFTYPE**, **SMFAUDIT** and **STOP**.

CSQ0607E

csect-name Insufficient storage available

Severity
8

Explanation

The Advanced Message Security task failed to allocate storage.

System action

The function being performed by the Advanced Message Security task fails.

System programmer response

Determine the reason there is insufficient storage and correct or increase, as appropriate.

CSQ0608E

csect-name Failed to load policy configuration, MQRC=*mqrc (mqrc-text)*

Severity
8

Explanation

The Advanced Message Security task failed to load the policy configuration for reason *mqrc* (*mqrc-text* provides the MQRC in textual form).

System action

The Advanced Message Security task cannot start.

System programmer response

Use the reason code, *mqrc*, to determine why the policy configuration could not be loaded from the policy queue, **SYSTEM.PROTECTION.POLICY.QUEUE**.

If the queue cannot start, you can define the queue in the CSQINP2 concatenation. Definitions are provided in SCSQPROC as CSQ4INSM.

CSQ0609I

csect-name AMS for z/OS starting, version *version*, level *service-level*

Severity
0

Explanation

Advanced Message Security task has started for product version *version* and service level *service-level*.

System action

Processing continues.

System programmer response

None.

CSQ0610E

csect-name Failed to start policy subtask, error *errcode*, reason *reason*

Severity
8

Explanation

An attempt by the Advanced Message Security task to start the policy configuration subtask failed with errno *errcode* and errno2 *reason*.

System action

The Advanced Message Security task cannot start.

System programmer response

Use the error and reason codes to determine why the policy configuration subtask could not be started, then take corrective action.

CSQ0611E

csect-name Failed to make AMS address space non-swapable, error *errcode*

Severity
8

Explanation

An attempt by the Advanced Message Security task to make itself non-swapable failed with error code *errcode*.

System action

The Advanced Message Security task cannot start.

System programmer response

The error identified by *errcode* is likely to be the return code from the SYSEVENT macro. Use macro documentation to determine the cause of the SYSEVENT failure.

CSQ0612E

csect-name System function *'function'* failed

Severity
8

Explanation

An attempt by the Advanced Message Security task to use a run-time call, *function*, failed.

System action

The Advanced Message Security task cannot continue to process the service it was providing at the time of the failure.

System programmer response

This message is associated with other messages that are generated at the time of failure. Examine these messages for more information, including error codes that might identify the cause of the failure.

CSQ0613E

csect-name AMS initialization error *errno*, reason *errno2*

Severity

8

Explanation

The Advanced Message Security task failed to initialize due to a run-time call failure.

System action

The Advanced Message Security task cannot start.

System programmer response

This message is associated with other messages that are generated at the time of failure. Examine these messages for more information, and use the error codes to determine the cause of the failure.

CSQ0614E

csect-name AMS termination error *errno*, reason *reason*

Severity

8

Explanation

The Advanced Message Security task failed during termination due to a run-time call failure.

System action

The Advanced Message Security task termination continues.

System programmer response

This message is associated with other messages that are generated at the time of failure. Examine these messages for more information, and use the error codes to determine the cause of the failure.

CSQ0615E

csect-name AMS post/wait request failed, reason *reason*

Severity

8

Explanation

An attempt by the Advanced Message Security task to issue a post or wait request failed for reason *reason*.

System action

The Advanced Message Security task cannot continue to process the service it was providing at the time of the failure.

System programmer response

The error identified by *reason* is likely to be the return code from the POST or WAIT macro. Use macro documentation to determine the cause of the failure.

CSQ0616E

csect-name AMS runtime environment initialization failed

Severity
8

Explanation

The Advanced Message Security task failed to initialize.

System action

The Advanced Message Security task cannot start.

System programmer response

Examine associated messages for more information about the failure, then take corrective action.

CSQ0617E

csect-name AMS already active

Severity
8

Explanation

An attempt to start the Advanced Message Security task failed because it was already running.

System action

The Advanced Message Security task cannot start while it is already running.

System programmer response

None.

CSQ0618E

csect-name AMS initialization failed, program not APF authorized

Severity
8

Explanation

An attempt to start the Advanced Message Security task failed because the module, CSQ0DSRV, is not APF authorized.

System action

The Advanced Message Security task cannot start.

System programmer response

Ensure that the AMS task module is APF authorized and retry.

V 9.0.3

CSQ0619I

csect-name AMSPROD=*prod-value*, recording product usage for *product-name* product ID *product-id*

Explanation

This message is issued when AMS starts, if SMF 89 product usage records are to be recorded by AMS. *product-name* is the descriptive name of the product, and *product-id* is the product ID that is to be used in the SMF 89 data. For example:

- AMSPROD=AMS, recording product usage for IBM MQ for z/OS AMS product ID 5655-AM9 - this indicates that usage data will be collected for the IBM MQ for z/OS AMS product.
- AMSPROD=ADVANCEDVUE, recording product usage for IBM MQ Advanced for z/OS, Value Unit Edition product ID 5655-AV1 - this indicates that usage data is to be collected for theIBM MQ Advanced for z/OS, Value Unit Edition product.

See z/OS MVS Product Management for more information on product usage recording.

CSQ0624E

csect-name SMF audit option invalid, defaulting to 'failure'

Severity

8

Explanation

An attempt to process the _AMS_SMF_AUDIT environment variable or an SMFAUDIT command failed because the variable or command value was not recognized.

System action

The variable assignment or command is ignored and the default value 'failure' is used.

System programmer response

Provide a valid variable or command value. Valid values include 'success', 'failure', and 'all'.

CSQ0625E

csect-name SMF record type invalid

Severity

8

Explanation

An attempt to process the _AMS_SMF_TYPE environment variable or an SMFTYPE command failed because the variable or command value was not valid.

System action

The variable assignment or command is ignored.

System programmer response

Provide a valid variable or command value. Valid values include numeric values between 0 and 255 inclusive. The SMFTYPE value represents the SMF record type for SMF record generation. A value of 0 means no SMF record generation is required. The recommended value is 180.

CSQ0626I

csect-name SMF audit type is *audit-type*

Severity

0

Explanation

The Advanced Message Security SMF audit type has been set to *audit-type*.

System action

The new SMF audit type takes effect immediately. If *audit-type* is 'failure', all failing puts/gets to a protected queue are audited. If *audit-type* is 'success', all successful puts/gets to a protected queue are audited. If *audit-type* is 'all', both successful and failing puts/gets to a protected queue are audited.

System programmer response

None.

CSQ0629E

csect-name Unable to create security environment for user *'userid'*, reason *errno*

Severity
8

Explanation

An attempt by the Advanced Message Security task to create a thread-level security environment using pthread_security_np() for user *userid* failed for the reasons indicated by *errno* and *errno2*.

System action

The thread-level security environment is not created, and the AMS function being processed cannot be completed. The MQI call fails.

System programmer response

Examine the *errno* and *errno2* values in conjunction with pthread_security_np() documentation to determine the cause of the failure.

CSQ0630E

csect-name Unable to delete security environment, reason *errno*

Severity
8

Explanation

An attempt by the Advanced Message Security task to delete a thread-level security environment using pthread_security_np() failed for the reason indicated by *errno*.

System action

The thread-level security environment is not deleted. AMS processing continues.

System programmer response

Examine the *errno* value in conjunction with pthread_security_np() documentation to determine the cause of the failure.

CSQ0631E

csect-name AMS not started, product is not enabled

Severity
8

Explanation

An attempt by the Advanced Message Security task to register itself using macro IFAEDREG failed.

System action

The Advanced Message Security task cannot start.

System programmer response

Verify that the PARMLIB IFAPRDxx member has been built with the provided AMS product information, then retry.

CSQ0632E

csect-name AMS deregistration failed, reason *reason*

Severity

8

Explanation

An attempt by the Advanced Message Security task to deregister itself using macro IFAEDDRG failed.

System action

The Advanced Message Security task cannot deregister. Processing continues.

System programmer response

Examine the reason returned by the IFAEDDRG macro in conjunction with macro documentation to determine the cause of the failure.

CSQ0633I

csect-name AMS environment variable values:

Severity

0

Explanation

The Advanced Message Security task identifies its environment variables and their values immediately following this message.

System action

Processing continues.

System programmer response

None.

CSQ0634I

csect-namevariable=value

Severity

0

Explanation

During startup, the Advanced Message Security task issues this message to report an environment variable *variable*, and its value *value*.

System action

Processing continues.

System programmer response

None.

CSQ0635I

csect-name POLICY refresh complete

Severity

0

Explanation

The Advanced Message Security task has refreshed its policy configuration in response to a **REFRESH** command.

System action

Processing continues.

System programmer response

None.

CSQ0636I

csect-name POLICY refresh failed

Severity
8

Explanation

An attempt by the Advanced Message Security task to refresh its policy configuration failed.

System action

The policy configuration is not refreshed.

System programmer response

Examine the console for associated error messages to determine the cause of the failure.

CSQ0637I

csect-name KEYRING refresh complete

Severity
0

Explanation

The Advanced Message Security task has refreshed its keyring configuration in response to a **REFRESH** command.

System action

Processing continues.

System programmer response

None.

CSQ0638E

csect-name KEYRING refresh failed, return code *errno*

Severity
8

Explanation

An attempt by the Advanced Message Security task to refresh its keyring configuration failed for the reason indicated by *errno*.

System action

The keyring configuration is not refreshed.

System programmer response

Examine the console for associated error message to determine the cause of the failure. Use the *errno*, which might represent a System SSL gsk_status to further diagnose the problem.

CSQ0639E

csect-name Incorrect **REFRESH** option, specify KEYRING, POLICY or ALL

Severity
8

Explanation

An attempt by the Advanced Message Security task to process a **REFRESH** command failed because the refresh option was not recognized.

System action

The **REFRESH** command is not processed.

System programmer response

Ensure the **REFRESH** option is KEYRING, POLICY or ALL, depending on which option should be refreshed.

CSQ0640E

csect-name AMS not started correctly

Severity
8

Explanation

The Advanced Message Security task has started incorrectly.

System action

The Advanced Message Security task fails to start.

System programmer response

The Advanced Message Security task can only be started internally by IBM MQ.

CSQ0641I

csect-name **REFRESH** command completed successfully

Severity
0

Explanation

The Advanced Message Security task has successfully processed a **REFRESH** command.

System action

Processing continues.

System programmer response

None.

CSQ0642I

*csect-name***REFRESH** command failed

Severity
8

Explanation

The Advanced Message Security task has failed to successfully process a **REFRESH** command.

System action

The requested **REFRESH** command is not processed.

System programmer response

Examine the console for associated error messages to determine the cause of the problem.

CSQ0648E

csect-name Failed to open AMS key ring, reason *gsk-status*

Severity
8

Explanation

An attempt by the Advanced Message Security task to open its keyring failed for the reason indicated by *gsk-status*.

System action

The AMS keyring is not opened, and the AMS task cannot start.

System programmer response

Examine System SSL documentation related to the gsk_open_keyring() call in conjunction with the *gsk-status* code to determine the cause of the failure.

CSQ0649E

csect-name CRL initialization failed

Severity
8

Explanation

An attempt by the Advanced Message Security task to connect to an LDAP server, based on configuration provided in the CRLFILE DD, failed.

System action

The Advanced Message Security task cannot perform Certificate Revocation List (CRL) checking. The behavior of certificate validation is determined by the System SSL environment variable GSK_CRL_SECURITY_LEVEL. See System SSL documentation for more information.

System programmer response

Check the configuration provided via the CRLFILE DD in the AMS started task JCL and verify that the configuration details are correct.

CSQ0651E

csect-name Failed to open CRL LDAP, *ldap-name*

Severity
8

Explanation

An attempt by the Advanced Message Security task to open an LDAP directory, *ldap-name*, failed.

System action

The Advanced Message Security task cannot perform Certificate Revocation List (CRL) checking against the named LDAP directory. The behavior of certificate validation is determined by the System SSL environment variable GSK_CRL_SECURITY_LEVEL. See System SSL documentation for more information.

System programmer response

Check the configuration provided in the CRLFILE DD in the AMS started task JCL and verify that the configuration details are correct. Verify that the failing directory is available.

CSQ0652I

csect-name CRL checking enabled

Severity

0

Explanation

The Advanced Message Security task has successfully enabled Certificate Revocation List (CRL) checking.

System action

Processing continues.

System programmer response

None.

CSQ0653I

csect-name CRL checking disabled

Severity

0

Explanation

The Advanced Message Security task has successfully disabled Certificate Revocation List (CRL) checking.

System action

Processing continues.

System programmer response

None.

CSQ0660E

csect-name Internal version mismatch

Severity

8

Explanation

The Advanced Message Security task has received a request for data protection services with an unrecognized request version value.

System action

The data protection service cannot be provided.

System programmer response

This error implies that a task other than the Advanced Message Security Interceptor is attempting to exploit AMS data protection services. AMS data protection services are only available by using the AMS Interceptor.

CSQ0699I

csect-name AMS shutdown complete

Severity

0

Explanation

The Advanced Message Security task has shutdown.

System action

Processing continues.

System programmer response

None.

CSQ0996I

csect-name char-diag1, char-diag2, char-diag3, char-diag4, hex-diag1, hex-diag2

Severity
0

Explanation

This message is generated when Advanced Message Security is running in DEBUG mode, as directed by IBM support center, and provides character and hexadecimal diagnostic values to aid in problem resolution.

System action

Processing continues.

System programmer response

None.

CSQ0997I

csect-name char-diag1, char-diag2, char-diag3, hex-diag1, hex-diag2, hex-diag3

Severity
0

Explanation

This message is generated when Advanced Message Security is running in DEBUG mode, as directed by IBM support center, and provides character and hexadecimal diagnostic values to aid in problem resolution.

System action

Processing continues.

System programmer response

None.

CSQ0998I

csect-name char-diag1, char-diag2, hex-diag1, hex-diag2, hex-diag3, hex-diag4

Severity
0

Explanation

This message is generated when Advanced Message Security is running in DEBUG mode, as directed by IBM support center, and provides character and hexadecimal diagnostic values to aid in problem resolution.

System action

Processing continues.

System programmer response

None.

CSQ0999I

csect-name char-diag1, hex-diag1, hex-diag2, hex-diag3, hex-diag4, hex-diag5

Severity

0

Explanation

This message is generated when Advanced Message Security is running in DEBUG mode, as directed by IBM support center, and provides character and hexadecimal diagnostic values to aid in problem resolution.

System action

Processing continues.

System programmer response

None.

Service facilities messages (CSQ1...): z/OS

The value shown for severity in the service facility messages that follow is the value returned as the job-step condition code from the job-step during which the message is issued. If additional messages having higher severity values are issued during the same job-step, the higher value is reflected as the job-step condition code.

Log services return codes

The return codes set by log services are:

0 Successful completion

4 Exception condition (for example, end of file), not an error.

8 Unsuccessful completion due to parameter errors.

12 Unsuccessful completion. Error encountered during processing of a valid request.

CSQ1000I

csect-name IBM MQ for z/OS V*n*

Severity

0

Explanation

This message is issued as the first part of the header to the report issued by the log print utility program.

CSQ1100I

csect-name LOG PRINT UTILITY - *date time*

Severity

0

Explanation

This message is issued as the second part of the header to the report issued by the log print utility program.

CSQ1101I

csect-name UTILITY PROCESSING COMPLETED, RETURN CODE=*rc*

Severity

0

Explanation

The log print utility completed with the return code *rc* indicated. 0 indicates successful completion.

CSQ1102I

SEARCH CRITERIA

Severity

0

Explanation

The search criteria specified for printing the log follow.

CSQ1105I

LOG PRINT UTILITY SUMMARY - *date time*

Severity

0

Explanation

This is issued as a header to the summary data set written by the log print utility.

CSQ1106I

END OF SUMMARY

Severity

0

Explanation

This marks the end of the summary data set written by the log print utility.

CSQ1110E

LIMIT OF 50 STATEMENTS EXCEEDED

Severity

8

Explanation

The limit of 50 input statements allowed by CSQ1LOGP has been exceeded.

System action

Processing is terminated.

System programmer response

Resubmit the job using no more than 50 statements.

CSQ1111E

LIMIT OF 80 TOKENS EXCEEDED

Severity

8

Explanation

The limit of 80 keywords and corresponding value specifications allowed by CSQ1LOGP has been exceeded. A keyword with its value is considered as two tokens.

System action

Processing is terminated.

System programmer response

Resubmit the job using no more than 80 tokens.

CSQ1112E

TOKEN *xxx*... EXCEEDS 48 CHARACTERS

Severity
8

Explanation

An input statement contains the character string beginning *xxx*. This string is not valid because it exceeds 48 characters in length.

System action

Processing is terminated.

System programmer response

Resubmit the job with a valid token.

CSQ1113E

INVALID SYNTAX FOR KEYWORD *kwd*

Severity
8

Explanation

An input statement contains the keyword *kwd*. The value specified for this keyword is not valid, because it is not of the form *kwd*(*value*).

System action

Processing is terminated.

System programmer response

Resubmit the job with the correct form of the keyword.

CSQ1127E

KEYWORD *kwd* UNKNOWN

Severity
8

Explanation

CSQ1LOGP does not recognize the keyword *kwd*.

System action

Processing is terminated.

System programmer response

Check to make sure all keywords are valid and resubmit the job.

CSQ1128E

END OF LOG RANGE SPECIFIED WITHOUT START

Severity
8

Explanation

You cannot specify the end of a search range (RBAEND or LRSNEND) without specifying a beginning of the search range (RBASTART or LRSNSTART).

System action

Processing is terminated.

System programmer response

Resubmit the job providing an RBASTART or LRSNSTART value to correspond to the RBAEND or LRSNEND value given to specify a valid search range.

CSQ1129E

LIMIT OF 10 *kwd* KEYWORDS EXCEEDED

Severity
8

Explanation

The *kwd* keyword appears too many times in the control statements. The limit is 10.

System action

Processing is terminated.

System programmer response

Resubmit the job providing no more than 10 of these keywords.

CSQ1130E

INVALID VALUE FOR KEYWORD *kwd* NUMBER *n*

Severity
8

Explanation

The value for the *n*th occurrence of keyword *kwd* is invalid because it has invalid characters, it is not one of a list of permitted values, or it is too long.

System action

Processing is terminated.

System programmer response

Resubmit the job providing a correct value specification.

CSQ1131E

INVALID VALUE FOR KEYWORD *kwd*

Severity
8

Explanation

The value for the keyword *kwd* is invalid because it has invalid characters, it is not one of a list of permitted values, or it is too long.

System action

Processing is terminated.

System programmer response

Resubmit the job providing a correct value specification.

CSQ1132E

NO VALUE FOR KEYWORD *kwd* NUMBER *n*

Severity
8

Explanation

The *n*th occurrence of keyword *kwd* is not followed by a value.

System action

Processing is terminated.

System programmer response

Resubmit the job providing a correct value specification.

CSQ1133E

NO VALUE FOR KEYWORD *kwd*

Severity
8

Explanation

The keyword *kwd* is not followed by a value.

System action

Processing is terminated.

System programmer response

Resubmit the job providing a correct value specification.

CSQ1134E

KEYWORD EXTRACT REQUIRES AT LEAST ONE OUTPUT DDNAME

Severity
4

Explanation

The keyword extract requires at least one output DDNAME for log records to be extracted.

System action

Processing continues, however, no log records are extracted .

System programmer response

Remove EXTRACT(YES), or alternatively add a DDNAME from the following list: **CSQBACK**, **CSQCMT**, **CSQBOTH**, **CSQINFLT**, **CSQOBJS**. Resubmit the job. For more information see, The log print utility (CSQ1LOGP).

CSQ1135E

KEYWORD *kwd* SPECIFIED MORE THAN ONCE

Severity
8

Explanation

The keyword *kwd* can only be specified once.

System action

Processing is terminated.

System programmer response

Resubmit the job providing only one of these keywords.

CSQ1137I

FIRST PAGE SET CONTROL RECORD AFTER RESTART = *r-rba*

Severity
0

Explanation

r-rba is the log RBA of a record that serves as an implicit indication that a restart occurred just prior to this point.

System action

Processing continues.

CSQ1138E
kwd1 AND *kwd2* CANNOT BOTH BE SPECIFIED

Explanation

kwd1 and *kwd2* cannot both appear in the control statements.

System action

Processing is terminated.

System programmer response

Correct the control statements and rerun the job.

CSQ1139E

SYSSUMRY DD STATEMENT MISSING

Severity
8

Explanation

You requested the SUMMARY option, but did not include the SYSSUMRY DD statement in your JCL.

System action

Processing terminates.

System programmer response

Resubmit the job with a SYSSUMRY DD statement included in the JCL.

CSQ1145E

CURRENT RESTART TIME STAMP OUT OF SEQUENCE - TIME=*date time* LOG RBA=*t-rba*

Severity
4

Explanation

This message indicates that the current log record has a time stamp that is less than the greatest time stamp processed so far. This might be a potential problem.

This message is followed by messages CSQ1147I and CSQ1148I which give the latest time stamp seen.

System action

Processing continues.

System programmer response

Examine the current log to determine whether multiple queue managers are writing to the same log. (Data might be being overwritten.) This might lead to data inconsistencies.

CSQ1146E

CURRENT END CHECKPOINT TIME STAMP OUT OF SEQUENCE - TIME=*date time* LOG RBA=*t-rba*

Severity

4

Explanation

This message indicates that the current log record has a time stamp that is less than the previous time stamp processed. This might be a potential problem.

This message is followed by messages CSQ1147I and CSQ1148I which give the latest time stamp seen.

System action

Processing continues.

System programmer response

Examine the current log to determine whether multiple queue managers are writing to the same log. (Data might be being overwritten.) This might lead to data inconsistencies.

CSQ1147I

LATEST TIME STAMP SEEN SO FAR - TIME=*date time* LOG RBA=*t-rba*

Severity

4

Explanation

This message follows message CSQ1145I or CSQ1146I and gives the latest time stamp seen.

CSQ1148I

MULTIPLE QUEUE MANAGERS MAY BE WRITING TO THE SAME LOG

Severity

4

Explanation

This message follows message CSQ1145I or CSQ1146I to indicate a possible cause of the time stamp problem.

CSQ1150I

SUMMARY OF COMPLETED EVENTS

Severity

0

Explanation

This message heads the summary of completed units of recovery (URs) and checkpoints.

System action

Processing continues.

CSQ1151I

UR CONNID=*cc* THREAD-XREF=*bb* USERID=*aa* TIME=*date time* START=*s-rba* END=*e-rba* DISP=*xx* INFO=*ii*

Severity

0

Explanation

This message describes a unit of recovery that terminated.

cc Connection ID (for example, BATCH)

bb Thread cross-reference ID (for example, JOB xxx)

aa User ID executing the UR

date time
Starting time of the UR

s-rba Log RBA of the first log record associated with the UR (that is, the URID)

e-rba Log RBA of the last log record associated with the UR If the UR is not complete, *e-rba* is shown as '***'

xx Disposition of the UR, values include:

- INFLIGHT
- IN BACKOUT
- IN COMMIT
- INDOUBT
- COMMITTED
- BACKED OUT

ii Status of the data, one of the following:

- COMPLETE, indicating that all page sets modified by this UR have been identified
- PARTIAL, indicating that the list of page sets modified by this UR is incomplete (this is shown if all records associated with a UR are not available, and no checkpoint is found prior to the UR's completion)

If the UR identifying information is not available, it will be shown as '***'.

System action

Processing continues.

CSQ1153I

CHECKPOINT START=*s-rba* END=*e-rba* TIME=*date time*

Severity

0

Explanation

This message describes a complete checkpoint on the log starting at RBA *s-rba* and ending at RBA *e-rba*. If the information is available, CSQ1LOGP also returns the date and time that the checkpoint was completed.

When this message follows message CSQ1157I, it identifies the checkpoint that would be used at restart. If no checkpoint is available, message CSQ1158I is printed instead.

System action

Processing continues.

CSQ1154I

RESTART AT r-*rba* TIME=*date time*

Severity

0

Explanation

A normal restart occurred at log RBA *r-rba*. CSQ1LOGP also returns the date and time of that restart.

System action

Processing continues.

CSQ1155I

CONDITIONAL RESTART AT *r-rba* TIME=*date time*

Severity

0

Explanation

A conditional restart occurred at log RBA *r-rba*. CSQ1LOGP also returns the date and time of that restart.

System action

Processing continues.

CSQ1156I

ALL URS COMPLETE

Severity

0

Explanation

There are no URs outstanding for restart.

System action

Processing continues.

CSQ1157I

RESTART SUMMARY

Severity

0

Explanation

This message heads the summary of the description of work to be performed at restart. Restart information that follows is based on the scope of the log scanned. If you suspect an error in IBM MQ, see Problem determination on z/OS for information about identifying and reporting the problem.

System action

Processing continues.

CSQ1158I

NO CHECKPOINT AVAILABLE - RESTART SUMMARY INCOMPLETE

Severity
0

Explanation

No checkpoint is available within the scope of the log scanned. The information following this message includes:

- URs that have not completed
- Page sets modified by these URs
- Page sets with writes pending

The information cannot be considered complete.

System action

Processing continues.

CSQ1161E

INVALID URE FOUND AT *x-rba*

Severity
4

Explanation

While processing the SUMMARY option, an invalid URE checkpoint record was encountered in the log.

System action

Processing continues.

System programmer response

If the checkpoint record identified in the message is used to restart the queue manager, the restart will be unsuccessful because it will not be able to process the unit of recovery presented by the invalid URE.

Look for other messages that indicate the cause of the problem. If you are unable to resolve the problem, contact your IBM support center.

CSQ1162E

INVALID RURE FOUND AT *x-rba*

Severity
4

Explanation

While processing the SUMMARY option, an invalid RURE checkpoint record was encountered in the log.

System action

Processing continues.

System programmer response

If the checkpoint record identified in the message is used to restart the queue manager, the restart will be unsuccessful because it will not be able to process the unit of recovery presented by the invalid RURE.

Look for other messages that indicate the cause of the problem. If you are unable to resolve the problem, contact your IBM support center.

CSQ1163E

NO CHECKPOINT AVAILABLE DUE TO LOG ERROR - RESTART SUMMARY INCOMPLETE

Severity

4

Explanation

A log error was encountered. CSQ1LOGP marked any checkpoints encountered before the log error as invalid. There were no complete checkpoints following the log error in the specified log range. The information following this message includes:

- URs that have not completed
- Page set modified by these URs
- Page sets with writes pending

This information cannot be considered complete.

System action

Processing continues.

CSQ1165E

UR REQUIRES LOG WHICH IS IN ERROR

Severity

0

Explanation

While processing a UR, information was required from the log, but the log was in error, as indicated by previous messages.

System action

Processing continues.

CSQ1166I

INFORMATION INCOMPLETE FOR UR - LOG TRUNCATED AT *xx*

Severity

0

Explanation

Complete information for the UR is not available within the scope of the log scanned.

System action

Processing continues.

CSQ1209E

END OF LOG RANGE IS LESS THAN START

Severity
8

Explanation

The end log range value (specified by RBAEND or LRSNEND) is less than or equal to the start range value (specified by RBASTART or LRSNSTART).

System action

Processing is terminated.

System programmer response

Resubmit the job providing an RBASTART or LRSNSTART value and a corresponding RBAEND or LRSNEND value to specify a valid search range.

CSQ1210E

LOG READ ERROR RETCODE=*rc* REASON CODE=*reason*

Severity
8

Explanation

An error was detected while attempting to read the log.

System action

Processing is terminated.

CSQ1211E

BAD LOG RBA RETURNED

Severity
8

Explanation

One of the three problems listed in this topic exists:

- The recovery log data set is damaged
- You identified a data set that is not a recovery log data set
- There is a problem with the log print utility

System action

Processing terminates, and a dump is produced.

System programmer response

A common error is to specify the first data set on an archive tape (the Bxxxxxxx data set) as a log data set; it is actually a bootstrap data set (BSDS).

Determine if the problem is your error by dumping the data set and determining if it is a log data set.

CSQ1212I

FIRST LOG RBA ENCOUNTERED = *s-rba*

Severity
0

Explanation

This identifies the RBA of the first log record read.

System action

Processing continues.

CSQ1213I

LAST LOG RBA ENCOUNTERED = *e-rba*

Severity
0

Explanation

This identifies the RBA of the last log record read.

System action

Processing continues.

CSQ1214I

nn LOG RECORDS READ

Severity
0

Explanation

This identifies the number (in decimal) of logical log records read during CSQ1LOGP processing.

System action

Processing continues.

CSQ1215I

NO LOG RECORDS READ

Severity
0

Explanation

CSQ1LOGP read no log records.

Possible explanations are:

- An error has prevented CSQ1LOGP from continuing, therefore no log records have yet been processed (if this is so, an error message should precede this message)
- You specified the active log data sets or archive log data sets out of RBA sequence
- You specified an RBASTART or LRSNSTART value that is greater than any RBA or LRSN in the active and archive data sets available
- You specified a log range using LRSNs, but the queue manager is not in a queue-sharing group.

System action

Processing continues.

CSQ1216E

LOG READ ERROR, RETCODE=*rc*, REASON CODE=*reason*, RBA=*x-rba*

Severity
4

Explanation

An error was encountered while attempting to read the log, indicating that either the log has an error in one of the control intervals (CI), or a data set containing the requested RBA cannot be located. The RBA specification in the message indicates where the error was detected and gives the requested RBA. It will point to:

- The start of the CI if there is a problem with the log control interval definition (LCID), or with any of the general control information within a CI
- The log record in the CI if there is a problem with a log record header (LRH)

If this is the first log record read during this execution of the Log Extractor, and if there is a problem with the LCID, the RBA specification will be all zeros.

Before returning any records, the utility checks the control information (LCID) at the end of a CI, and analyzes the LRH to ensure that all records are properly chained together within the CI. If an error is detected while performing this process, CSQ1LOGP will issue this message, before dumping the entire CI. It will not format individual records within the CI, but will, if possible, continue processing by reading the next CI.

System action

Processing continues.

CSQ1217E

RBA RANGE WARNING, RETCODE=*rc*, REASON CODE=*reason*, PRIOR RBA=*p-rba*, CURRENT RBA=*c-rba*

Severity

4

Explanation

A gap in the log RBA range has been encountered. PRIOR RBA *p-rba* indicates the last good log RBA prior to the gap. CURRENT RBA *c-rba* indicates the log record following the gap, and will be formatted following this message.

System action

Processing continues.

CSQ1218I

nn LOG ERROR MESSAGES

Severity

0

Explanation

CSQ1LOGP distinguishes three classes of errors:

- Code problems existing in the MQ or system code used for CSQ1LOGP. In such cases, abnormal termination with a user completion code of U0153 occurs.
- Incorrect invocation of CSQ1LOGP caused, perhaps, by your having used an incorrect keyword or missed a DD statement. Under these circumstances, CSQ1LOGP issues appropriate error messages, and the program is terminated.
- An error in a particular log CI under the scrutiny of CSQ1LOGP. Such scrutiny is performed before any of the records within the CI are processed. This is an indication of logical damage, and error messages are issued by the utility. The CI or log record in error is printed, and CSQ1LOGP continues to the next CI or log record.

  The count *nn* provided summarizes the number (in decimal) of errors CSQ1LOGP detected while accessing the log.

System action

Processing continues.

CSQ1219I

LOG RECORDS CONTAIN *n* BYTE RBA - QSG(*in-qsg*)

Severity
0

Explanation

This message is issued by CSQ1LOGP to indicate the format of the log records being processed, and whether the queue manager was a member of a queue-sharing group (QSG). The message is issued before any log records are printed, and whenever the format of the log records change.

The value of *n* identifies the log RBA format of the log records being processed, and can be either 6 or 8.

The value of *in-qsg* identifies whether the log records were written by a queue manager that was a member of a QSG, and can be one of the following values:

YES The log records were written by a queue manager that was a member of a QSG

NO The log records were written by a queue manager that was not a member of a QSG

System action
Processing continues

CSQ1220E

ARCHIVE LOG TRUNCATED AT *xxxx* - INVALID LOG RECORDS READ

Severity
4

Explanation

At a restart of the queue manager, an archive log was truncated. This archive log data set could not be physically altered to reflect this truncation, and invalid log records therefore still exist. CSQ1LOGP has already reported this information in the summary report, and cannot retract it. Nor can it disregard the invalid log information already read in order adequately to summarize what has occurred. Therefore, all information up to this point in the log will be summarized, and a new summary report initiated. Consequently, the same UR might be reported twice with different dispositions and different page sets modified.

System action

Processing continues.

System programmer response

To avoid this condition, use the BSDS DD statement instead of the ARCHIVE DD statement.

CSQ1221E

VSAM ERROR, RETCODE=*rc*, REASON CODE=*reason*,VSAM RETURN CODE=*aaaa*, ERROR CODE=*bbbb*

Severity
8

Explanation

A VSAM error was encountered while attempting to read the log.

System action

Processing continues.

CSQ1222E

LOG ALLOCATION ERROR, RETCODE=*rc*, REASON CODE=*reason*, DYNALLOC INFO CODE=*aaaa*, ERROR CODE=*bbbb*

Severity

8

Explanation

An error occurred while dynamically allocating a log data set.

System action

Processing terminates.

CSQ1223E

JFCB READ ERROR, RETCODE=rc, REASON CODE=*reason*, RDJFCB RETURN CODE=*aaaa*

Severity

8

Explanation

An error occurred while trying to read the job file control block.

System action

Processing continues.

CSQ1224I

INFORMATION INCOMPLETE FOR LOG RECORD, CURRENT RBA=*c-rba*, CURRENT URID=*c-urid*

Severity

0

Explanation

Incomplete information for the log record was found within the scope of the logs scanned. An end of log condition was encountered before all segments of a spanned record could be found. CURRENT RBA *c-rba* indicates the log RBA of the record in question. CURRENT URID *c-urid* indicates the UR to which the spanned log record is related. If there is no URID associated with the log record (for instance, a checkpoint record), then this will show zeros.

System action

Processing continues.

System programmer response

If complete information for the identified log record is required, extend the RBA range to be processed until the required log data is available.

CSQ1271I

START OF LOG RANGE SET TO LRSN=*s-lrsn*

Severity

0

Explanation

The LRSN value you specified for the start of the log range is less than the lowest possible LRSN value, which is *s-lrsn*.

System action

Processing continues, using an LRSNSTART value of *s-lrsn*.

CSQ1272I

FIRST LOG LRSN ENCOUNTERED = *s-lrsn*

Severity
0

Explanation

This identifies the LRSN of the first log record read.

System action

Processing continues.

CSQ1273I

LAST LOG LRSN ENCOUNTERED = *e-lrsn*

Severity
0

Explanation

This identifies the LRSN of the last log record read.

System action

Processing continues.

CSQ1275I

LRSN RANGE CAN BE USED ONLY WITH A QUEUE-SHARING GROUP

Severity
0

Explanation

You specified a log range using LRSNs, but CSQ1LOGP read no log records. This could be because the queue manager is not in a queue-sharing group, in which case you cannot use LRSN specifications.

System action

Processing continues.

System programmer response

If the queue manager is not in a queue-sharing group, rerun the job using RBA specifications for the log range.

CSQ1276E

LOG READ ERROR, RETCODE=*rc*, REASON CODE=*reason*, LRSN=*x-lrsn*

Severity
4

Explanation

An error was encountered while attempting to read the log, indicating that either the log has an error in one of the control intervals (CI), or a data set containing the requested LRSN cannot be located. The LRSN specification in the message indicates where the error was detected and gives the requested LRSN. It will point to:

- The start of the CI if there is a problem with the log control interval definition (LCID), or with any of the general control information within a CI

- The log record in the CI if there is a problem with a log record header (LRH)

If this is the first log record read during this execution of the Log Extractor, and if there is a problem with the LCID, the LRSN specification will be all zeros.

Before returning any records, the utility checks the control information (LCID) at the end of a CI, and analyzes the LRH to ensure that all records are properly chained together within the CI. If an error is detected while performing this process, CSQ1LOGP will issue this message, before dumping the entire CI. It will not format individual records within the CI, but will, if possible, continue processing by reading the next CI.

System action

Processing continues.

CSQ1277E

LRSN RANGE WARNING, RETCODE=*rc*, REASON CODE=*reason*, PRIOR LRSN=p-lrsn, CURRENT LRSN=*c-lrsn*

Severity

4

Explanation

A gap in the log LRSN range has been encountered. The PRIOR LRSN specification indicates the last good log LRSN prior to the gap. The CURRENT LRSN specification indicates the log record following the gap, and will be formatted following this message.

System action

Processing continues.

IBM MQ-IMS bridge Messages (CSQ2...): z/OS

CSQ2001I

csect-name OTMA REJECTED MESSAGE - APPLICATION ERROR, SENSE CODE=*code*, XCFGNAME=*gname* XCFMNAME=*mname* TPIPE=*tpipename*

Explanation

Because of an application error, the IBM MQ-IMS bridge received a negative acknowledgment (NAK) from OTMA when sending a message. The information provided in the message is:

gname The XCF group to which the partner belongs.

mname
The member name of the partner.

tpipename
The name of the Tpipe used by the partner.

code The IMS sense code returned by the partner (the first four characters are the sense code).

System action

The message is put to the dead-letter queue, and processing continues.

System programmer response

For information about the sense code from IMS, see the *IMS/ESA Communications and Connections Guide* Version 10, document number SC18-9703, program number 5635-A01.

CSQ2002E

csect-name OTMA CLIENT BID REJECTED, XCFGNAME=*gname* XCFMNAME=*mname*, SENSE CODE=*code*

Explanation

An OTMA client bid command from the IBM MQ-IMS bridge was rejected. *code* is the associated IMS sense code. *gname* and *mname* identify the partner IMS system to which the command was directed.

System action

No connection is made to the IMS system. Connections to other OTMA partners are unaffected.

System programmer response

For information about IMS-OTMA sense codes, see the IMS Messages and Codes.

CSQ2003E

csect-name OTMA REJECTED MESSAGE - SYSTEM ERROR, SENSE CODE=*code*, XCFGNAME=*gname* XCFMNAME=*mname* TPIPE=*tpipename*

Explanation

Because of a system-related error, the IBM MQ-IMS bridge received a negative acknowledgment (NAK) from OTMA when sending a message. The information provided in the message is:

gname The XCF group to which the partner belongs.

mname
The member name of the partner.

tpipename
The name of the Tpipe used by the partner.

code The IMS sense code returned by the partner (the first four characters are the sense code).

System action

If the problem was caused by an environmental error, the IMS bridge returns the message to the queue. Depending on the error described by the sense code, the message send is retried or the queue is closed.

If a severe error occurred, the message is returned to the queue, and the IMS bridge ends abnormally with completion code X'5C6' and reason code X'00F20059'.

System programmer response

For information about IMS-OTMA sense codes, see the IMS Messages and Codes.

CSQ2004E

csect-name ERROR USING QUEUE *q-name*, MQRC=*mqrc (mqrc-text)*

Explanation

The IBM MQ-IMS bridge was unable to open, close, get from, put to, or inquire about a queue.

If *csect-name* is CSQ2QCP0, the problem was with the message queue associated with IMS or the reply-to queue. If *csect-name* is CSQ2QCP1, the problem was with the reply-to queue. If *csect-name* is CSQ2PUTD, the problem was with the dead-letter queue.

If the reason code received is 2042, it is because the IBM MQ-IMS bridge requires exclusive input access (MQOO_INPUT_EXCLUSIVE) to the bridge queue if it is defined with QSGDISP(QMGR), or if it is defined with QSGDISP(SHARED) together with the NOSHARE option.

System action

If the problem was caused by an environmental error, processing continues.

If a severe error occurred, the IMS bridge ends abnormally with completion code X'5C6' and a reason code which shows the particular error.

System programmer response

Refer to API completion and reason codes for information about *mqrc* (*mqrc-text*provides the MQRC in textual form).

CSQ2005I

csect-name ERROR PROCESSING MESSAGE, FEEDBACK=*code*, XCFGNAME=*gname* XCFMNAME=*mname* TPIPE=*tpipename*

Explanation

The IBM MQ-IMS bridge encountered an error while processing a message. *code* is the associated feedback code that will be set in the messagedescriptor. The information provided in the message is:

gname The XCF group to which the partner belongs.

mname
The member name of the partner.

tpipename
The name of the Tpipe used by the partner.

code The IMS sense code returned by the partner.

System action

The message is not processed.

System programmer response

code is one of the following:

291 (MQFB_DATA_LENGTH_ZERO)
A segment length field was zero in the application data of the message.

292 (MQFB_DATA_LENGTH_NEGATIVE)
A segment length field was negative in the application data of the message.

293 (MQFB_DATA_LENGTH_TOO_BIG)
A segment length field was too big in the application data of the message.

294 (MQFB_BUFFER_OVERFLOW)
The value of one of the length fields would overflow the MQ message buffer.

295 (MQFB_LENGTH_OFF_BY_ONE)
The length field was one byte too short.

296 (MQFB_IIH_ERROR)
The MQMD specified MQFMT_IMS, but the message does not begin with a valid MQIIH structure.

298 (MQFB_NOT_AUTHORIZED_FOR_IMS)
The user ID specified in the MQMD was denied access.

3xx IMS sense code xx (where xx is the decimal representation of the IMS sense code). For information about IMS-OTMA sense codes, see the IMS Messages and Codes.

CSQ2006I

csect-name DEAD-LETTER QUEUE UNAVAILABLE, MQRC=*mqrc (mqrc-text)*

Explanation

The IBM MQ-IMS bridge was unable to put a message to the dead-letter queue.

System action

If the message was being sent to IMS, it will be retained on the local IMS queue, and the queue will be disabled. If the message was coming from IMS, a NAK will be sent to IMS so that IMS will retain it and stop sending messages on the Tpipe.

System programmer response

If *mqrc* is 0, there is no dead-letter queue defined; you are strongly recommended not to use the IBM MQ-IMS bridge unless you have a dead-letter queue defined. Otherwise, there is a problem obtaining the name of the queue from the queue manager; refer to API completion and reason codes for information about *mqrc* (*mqrc-text* provides the MQRC in textual form).

CSQ2009I

csect-name PREREQUISITE PRODUCTS FOR IMS BRIDGE NOT AVAILABLE

Explanation

The IBM MQ-IMS bridge cannot operate because:

- The version of z/OS being used is not correct
- The version of IMS being used is not correct
- OTMA support has not been enabled on IMS.
- An incorrect version of the system parameter module (CSQZPARM) is being used.

System action

The MQ-IMS bridge does not start.

System programmer response

Refer to the Planning on z/OS for information about what product levels are required.

If required, recompile CSQZPARM with the correct libraries.

CSQ2010I

csect-name CONNECTED TO PARTNER, XCFGNAME=*gname* XCFMNAME=*mname*

Explanation

The MQ-IMS bridge successfully established a connection to the partner IMS system identified by *gname* and *mname*.

System action

Processing continues; messages can be sent to the partner.

CSQ2011I

csect-name DISCONNECTED FROM PARTNER, XCFGNAME=*gname* XCFMNAME=*mname*

Explanation

The partner IMS system identified by *gname* and *mname* is no longer available, and the connection from the IBM MQ-IMS bridge has ended.

System action

Processing continues; messages can no longer be sent to the partner.

CSQ2012I

csect-name NO UTOKEN SECURITY REQUESTED FOR IMS SIGNON, XCFGNAME=*gname* XCFMNAME=*mname*

Explanation

The IBM MQ-IMS bridge signed-on to the partner IMS system identified by *gname* and *mname*. No UTOKEN security was requested for this session.

System action

Processing continues.

CSQ2013E

csect-name NOT AUTHORIZED FOR IMS SIGNON, XCFGNAME=*gname* XCFMNAME=*mname*

Explanation

The IBM MQ-IMS bridge tried to sign on to the partner IMS system identified by *gname* and *mname*. However, the queue manager not authorized to establish a connection to this IMS system.

System action

No connection is made to the IMS system. Connections to other OTMA partners are unaffected.

CSQ2015I
csect-name IMS BRIDGE ALREADY SUSPENDED, XCFGNAME=*gname* XCFMNAME=*mname*

Explanation

A SUSPEND QMGR FACILITY(IMSBRIDGE) command was issued, but the IBM MQ-IMS bridge to the partner IMS system identified by *gname* and *mname* is already suspended.

System action

None.

CSQ2016I
csect-name IMS BRIDGE NOT SUSPENDED, XCFGNAME=*gname* XCFMNAME=*mname*

Explanation

A RESUME QMGR FACILITY(IMSBRIDGE) command was issued, but the IBM MQ-IMS bridge to the partner IMS system identified by *gname* and *mname* is not suspended.

System action

None.

CSQ2020E
csect-name RESYNCHRONIZATION ERROR

Explanation

A resynchronization error has occurred. The information provided by this message is:

```
  IN TPIPE tpipename  FOR QUEUE q-name,  BY PARTNER, XCFGNAME=gname XCFMNAME=mname,
QMGR SEND=sendseq PARTNER RECEIVE=otmarecvseq,QMGR RECEIVE=recvseq
 PARTNER SEND=otmasendseq,  INDOUBT UNIT OF RECOVERY urid
```

where:

tpipename
The name of the Tpipe which cannot be resynchronized

q-name
The name of the queue for this Tpipe

gname The name of the XCF group to which the Tpipe belongs

mname
The name of the XCF member to which the Tpipe belongs

sendseq
The recoverable sequence number of the message last sent by IBM MQ to the partner, in hexadecimal

otmasendseq
The recoverable sequence number of the message last sent by the partner to IBM MQ, in hexadecimal

recvseq
The recoverable sequence number of the message last received by IBM MQ from the partner, in hexadecimal

otmarecvseq
The recoverable sequence number of the message last received by the partner from IBM MQ, in hexadecimal

urid The identifier of an in-doubt unit of recovery; a value of 0 means that there is no in-doubt unit of recovery.

System action

No messages are sent on the Tpipe.

System programmer response

Use the RESET TPIPE command to reset recoverable sequence numbers, to restart the Tpipe, and, if required, to resolve the unit of recovery.

CSQ2023E
csect-name PARTNER, XCFGNAME=*gname* XCFMNAME=*mname*, CANNOT RESYNCHRONIZE, SENSE CODE=*code*

Explanation

IBM MQ was unable to resynchronize with the partner. The information provided in the message is:

gname The name of the XCF group to which the partner belongs.

mname
The member name of the partner who cannot resynchronize.

code The IMS sense code returned by the partner (the first four characters are the sense code).

System action

The connection to OTMA is stopped

System programmer response

For information about IMS-OTMA sense codes, see the IMS Messages and Codes. Resolve the problem and restart the OTMA connection.

CSQ2024E
csect-name TPIPE *tpipename* IS UNKNOWN TO PARTNER, XCFGNAME=*gname* XCFMNAME=*mname*

Explanation

The Tpipe name was unknown to the partner. The information provided in the message is:

tpipename
The name of the Tpipe which the partner no longer recognizes.

gname The XCF group to which the partner belongs.

mname
The member name of the partner who is resynchronizing

System action

The associated unit of recovery is backed out and processing continues.

System programmer response

If the partner IMS system has been cold started then this message can be considered normal. If the IMS system has not been cold started consider this message as an alert and investigate the partner IMS system.

CSQ2025E

csect-name PARTNER, XCFGNAME=*gname* XCFMNAME=*mname*, CANNOT RESYNCHRONIZE TPIPE *tpipename*, SENSE CODE=*code*

Explanation

The partner was unable to resynchronize the Tpipe. The information provided in the message is:

gname The XCF group to which the partner belongs.

mname
The member name of the partner who is resynchronizing.

tpipename
The name of the Tpipe which the partner cannot resynchronize.

code The IMS sense code returned by the partner.

System action

The Tpipe is stopped.

System programmer response

See the *IMS V10 Communications and Connections* documentation for information about the sense code from IMS. Resolve the problem and restart or reset the Tpipe.

CSQ2026I

csect-name PARTNER, XCFGNAME=*gname* XCFMNAME=*mname*, HAS COLD-STARTED TPIPE *tpipename*

Explanation

The partner has cold started a Tpipe. The information provided in the message is:

gname The XCF group of which the partner is a member.

mname
The member name of the partner who is resynchronizing.

tpipename
The name of the Tpipe which the partner has cold started.

System action

All recoverable sequence numbers are reset to 1, and processing continues.

System programmer response

None.

CSQ2027I

csect-name TPIPE *tpipename* FOR PARTNER, XCFGNAME=*gname* XCFMNAME=*mname*, DOES NOT HAVE AN INDOUBT UNIT OF RECOVERY

Explanation

MQ expected a Tpipe to have an in-doubt unit of recovery. The information provided by the message is:

tpipename
The name of the Tpipe for which there should be a unit of recovery still in doubt

gname The XCF group to which the partner belongs.

mname
The member name of the partner for the Tpipe.

System action

Processing continues.

System programmer response

Collect the following items, and contact your IBM support center.

- Console log
- MQ job log
- IMS job log

CSQ2028I

csect-name QUEUE MANAGER IS NOT CONNECTED TO PARTNER, XCFGNAME=*gname* XCFMNAME=*mname*

Explanation

MQ is not connected to the partner. The information provided in the message is:

gname The group name of the partner.

mname
The member name of the partner.

System action

The command is rejected.

System programmer response

Resubmit the command using the correct XCF group name when IBM MQ is connected to the partner.

CSQ2029I

csect-name TPIPE *tpipename* NOT FOUND FOR PARTNER, XCFGNAME=*gname* XCFMNAME=*mname*

Explanation

The Tpipe could not be found. The information provided in this message is:

tpipename
The name of the Tpipe which could not be found.

gname The XCF group of which the partner is a member.

mname
The member name of the partner for the Tpipe.

System action

The command is rejected.

System programmer response

Resubmit the RESET TPIPE command with the correct Tpipe name.

CSQ2030I

csect-name TPIPE *tpipename* IS STILL OPEN FOR PARTNER, XCFGNAME=*gname* XCFMNAME=*mname*

Explanation

The Tpipe is still open. The information provided by this message is:

tpipename
The name of the Tpipe which is still open.

gname The XCF group name.

mname
The member name of the partner for the Tpipe.

System action

The command is rejected.

System programmer response

The most likely cause of this message is that the RESET TPIPE command was issued with an incorrect Tpipe name or that the command was issued on the wrong queue manager in a queue-sharing group. Resubmit the RESET TPIPE command with the correct Tpipe name.

CSQ2031I
csect-name TPIPE *tpipename* FOR PARTNER, XCFGNAME=*gname* XCFMNAME=*mname*, ACTION REQUIRED FOR INDOUBT UNIT OF RECOVERY

Explanation

A Tpipe has an in-doubt unit of recovery, but no recovery action was specified. The information provided by the message is:

tpipename
The name of the Tpipe which has a unit of recovery still in doubt

gname The XCF group to which the partner belongs.

mname
The member name of the partner for the Tpipe.

System action

Processing continues.

System programmer response

Resubmit the RESET TPIPE command specifying an action (COMMIT or BACKOUT) for the in-doubt unit of recovery.

CSQ2040I

csect-name OTMA MESSAGE FLOOD STATUS=WARNING FOR PARTNER, XCFGNAME=*gname* XCFMNAME=*mname*

Severity
4

Explanation

This message is issued by the IBM MQ-IMS bridge in response to a notification from the partner IMS system, identified by *gname* and *mname*, that an OTMA message flood warning condition exists.

This message indicates that the IMS partner is currently unable to process the volume of transaction requests being sent to it via the IBM MQ-IMS bridge.

System action

Processing continues but the IBM MQ-IMS bridge will slow down the rate at which transaction requests are sent to allow the partner IMS system to process the accumulated backlog.

System programmer response

Review the status of the partner IMS system to determine if any action is required. You can use the **/DISPLAY OTMA** and **/DISPLAY TMEMBER** commands to do this.

Perform a check on the partner IMS system to determine if the message DFS1988W has been issued, identifying the severity of the warning condition.

CSQ2041I

csect-name OTMA MESSAGE FLOOD STATUS=FLOODED FOR PARTNER, XCFGNAME=*gname* XCFMNAME=*mname*

Severity

8

Explanation

This message is issued by the IBM MQ-IMS bridge in response to a notification from the partner IMS system, identified by *gname* and *mname*, that an OTMA message flood condition exists.

This indicates that the IMS partner is currently unable to process the volume of transaction requests being sent to it through the IBM MQ-IMS bridge. No further requests can be sent until the flood condition in IMS has been relieved.

System action

All TPIPEs to the identified partner IMS system are suspended until a notification is received from IMS indicating that the flood condition has been relieved.

Messages can still be put to any IBM MQ-IMS bridge queue with a storage class specifying the identified IMS partner but will remain there until the TPIPES can be resumed.

IBM MQ-IMS bridge queues for other IMS partners are unaffected.

System programmer response

Review the status of the partner IMS system and determine what action is required to relieve the IMS flood condition. You can use the **/DISPLAY OTMA** and **/DISPLAY TMEMBER** commands to do this.

Perform a check on the partner IMS system to determine if the message DFS1989E has been issued, identifying the flood condition.

CSQ2042I

csect-name OTMA MESSAGE FLOOD RELIEVED FOR PARTNER, XCFGNAME=*gname* XCFMNAME=*mname*

Severity

0

Explanation

This message is issued by the IBM MQ-IMS bridge in response to a notification from the partner IMS system, identified by *gname* and *mname*, that an OTMA message flood, or flood warning, condition no longer exists.

System action

If this message follows CSQ2041I, all TPIPEs to the identified partner IMS system that were suspended in response to the flood condition are resumed. The IBM MQ-IMS bridge will gradually increase the rate at which transaction requests are sent until the maximum rate is achieved, or a subsequent flood condition is reported by the partner IMS system.

System programmer response

None required.

Subsystem support messages (CSQ3...): z/OS

CSQ3001E
csect-name - ABNORMAL DISCONNECT FROM SUBSYSTEM INTERFACE

Explanation

An online routine was still supporting SSI calls (IEFSSREQ) even though the queue manager had nearly completed termination or was no longer executing. This occurs with *csect-name* CSQ3RS00 or CSQ3RS0X when the queue manager address space has reached end-of-memory and neither normal termination nor online error recovery routines have successfully completed termination of the queue manager. This occurs with *csect-name* CSQ3SSTM when this condition is discovered during online termination.

System action

The connection is terminated. All IEFSSREQ requests are handled by the IBM MQ early processing program until the queue manager is restarted. An SVC dump is requested.

CSQ3002I
INDOUBT RECOVERY BY *connection-name* STILL IN PROGRESS

Explanation

There might be IBM MQ units of recovery (URs), related to an identified subsystem (*connection-name*), still in doubt after restart synchronization has taken place. (Indoubt URs are those for which commit has been voted by IBM MQ but which have not yet been acknowledged by *connection-name*.)

This message might appear if the *connection-name* subsystem has begun to do new work before having resolved all in-doubt URs. The *connection-name* subsystem is still in the process of resolving the in-doubt URs.

System action

Resources held (locked) by these in-doubt URs are unavailable to any other work units until their status is resolved.

System programmer response

The system programmer or system administrator must determine the correct recovery action to resolve the in-doubt situations. This involves either ensure-commit or backout decisions for all in-doubt URs.

The DISPLAY THREAD command should be used to see the URs still in doubt. It will normally show that all in-doubt URs have now been resolved. If not, the RESOLVE INDOUBT command should be used to resolve the in-doubt URs and to release the resources they hold.

CSQ3004E
SSI DESCRIPTOR GET FAILURE, RC=*rc* REASON=*reason*

Explanation

An internal error has occurred during initialization or termination.

System action

The queue manager terminates.

System programmer response

Ensure that all maintenance has been applied to the IBM MQ program libraries, and then restart the queue manager.

CSQ3006E
'*rmid*' SSI FUNCTION WAS ALREADY ACTIVE WHEN ACTIVATE WAS ATTEMPTED

Explanation

An initialization sequence error has occurred.

System action

The queue manager terminates.

System programmer response

Ensure that all maintenance has been applied to the IBM MQ program libraries, and then restart the queue manager.

CSQ3007E
'*rmid*' SSI FUNCTION WAS ALREADY INACTIVE WHEN DEACTIVATE WAS ATTEMPTED

Explanation

A termination sequence error has occurred.

System action

Termination continues.

System programmer response

Ensure that all maintenance has been applied to the IBM MQ program libraries.

CSQ3008E
csect-name - ABNORMAL DISCONNECT FOR PROGRAM REQUEST HANDLER(S)

Explanation

One or more resource managers are still supporting application program calls through their program request handler, even though the queue manager had almost completed termination, or was no longer executing. This occurs when the queue manager address space has gone to end of memory and neither normal termination nor online error recovery routines have successfully completed termination.

System action

The connection is terminated. All application program support requests are rejected with an indication that the queue manager is not active. An SVC dump is requested.

System programmer response

If the problem persists, collect the following items, and contact your IBM support center:

- System dump
- Printout of SYS1.LOGREC

CSQ3009E
error-info

Explanation

An internal error has occurred in RRS exit processing. The message contains error information that will be needed to resolve the problem.

System action

Processing continues, but RRS coordination is no longer available to the queue manager. It will probably be necessary to restart the queue manager or RRS.

CSQ3011I
csect-name Coordinator RRS is cold-starting and has lost its log. In-doubt IBM MQ threads need manual resolution

Explanation

IBM MQ has participant responsibility for in-doubt threads. RRS, the commit coordinator, has informed the queue manager that it lost all knowledge of IBM MQ in-doubt threads. The in-doubt threads at this queue manager must be manually resolved with the RESOLVE INDOUBT command.

System action

Processing continues.

System programmer response

A list of in-doubt threads where RRS is the coordinator can be displayed using the DISPLAY THREAD command for in-doubt type threads by specifying RRSBATCH as the connection name.

The decision to commit or back out the logical unit of work should be coordinated with any other participant RRS Recoverable Resource Managers. The existence of other participants might not be easy to determine. The information might be available in the RRS recovery log even though information has been lost.

At this queue manager, all in-doubt threads coordinated by RRS must be resolved with the RESOLVE INDOUBT command. Locked data remains unavailable until resolution. Threads that were already resolved with this command are discarded. Threads not yet resolved are discarded after resolution with the command.

The commit or back out decision provided using the RESOLVE INDOUBT command for a logical unit of work is propagated to all downstream participants, if any.

CSQ3013I

csect-name Queue manager was restarted on the wrong system so cannot connect to RRS. There are unresolved URs where IBM MQ is a participant

Explanation

The queue manager has one or more in-doubt threads and is unable to connect to RRS to resolve these in-doubt units of recovery (URs).

System action

Processing continues.

CSQ3014I

csect-name In-doubt RRS URID=rrs-*urid* is unknown to IBM MQ. URID recorded for IBM MQ by RRS=mq-*urid*

Explanation

The queue manager is restarting with RRS where IBM MQ is a participant and RRS is the coordinator. RRS has a unit of recovery (UR) that the queue manager should be a participant in, but it has no knowledge of the RRS unit of recovery, with an ID of *rrs-urid*. RRS has recorded the IBM MQ URID as *mq-urid*.

System action

Restart with RRS continues.

System programmer response

This message might indicate a problem in IBM MQ or RRS, or it might be produced because of one of the following prior actions:

- A conditional restart was performed that resulted in the loss of part or all of the IBM MQ log. This conditional restart might have happened at any time in the past.
- The RESOLVE INDOUBT command was used to resolve the IBM MQ UR with ID *mq-urid*.

If one of these occurred, the message can be ignored. If neither occurred, there might be a problem in IBM MQ or RRS.

If the *mq-urid* appears to be a valid log RBA, use the log print utility (CSQ1LOGP) with the SUMMARY option and URID options using the *mq-urid* value. If this finds the UR, the disposition will indicate whether it was committed or backed out. If possible, use the RRS ISPF interface to commit or back out the RRS URID so that they match.

If you suspect an error in IBM MQ, collect the items listed in the Problem Determination section and contact your IBM support center.

CSQ3016I
csect-name RRS has lost data from its log

Explanation

The queue manager is restarting with RRS and RRS has lost some portion of its log.

System action

Restart with RRS continues.

System programmer response

IBM MQ might not be able to resolve in-doubt units of recovery successfully with RRS because of the loss of RRS log data.

CSQ3017I
csect-name RRS function *call-name* failed, RC=*rc*

Explanation

During queue manager restart, the RRS function specified by *call-name* issued a return code *rc* indicating a failure.

System action

Processing continues, but RRS functions will not be available. For example, connections using the RRS adapter will not be allowed, and queue-sharing group facilities will not work.

System programmer response

Investigate the RRS return code from the function specified and resolve the problem. Then restart the queue manager.

CSQ3100I
csect-name - SUBSYSTEM *ssnm* READY FOR START COMMAND

Explanation

The queue manager has terminated, and can be restarted when required.

CSQ3101E
csect-name - INVALID EARLY PROCESSING PARAMETER

Explanation

The z/OS command SETSSI ADD or the subsystem definition record in the IEFSSNxx member of SYS1.PARMLIB for the IBM MQ subsystem specified the early processing initialization parameter incorrectly. The name must be CSQ3EPX.

The failing subsystem name is provided in message IEF759I, which follows this message.

System action

The IBM MQ subsystem with the indicated name is not available.

System programmer response

Correct the parameter fields in the record of SYS1.PARMLIB member IEFSSNxx. For information about the parameters, see Update SYS1.PARMLIB members.

CSQ3102E
csect-name - INVALID COMMAND PREFIX

Explanation

The z/OS command SETSSI ADD or the subsystem definition record in the IEFSSNxx member of SYS1.PARMLIB for the IBM MQ subsystem specified the command prefix initialization parameter incorrectly.

The failing subsystem name is provided in message IEF759I, which follows this message.

System action

The IBM MQ subsystem with the indicated name is not available.

System programmer response

Correct the parameter fields in the record of SYS1.PARMLIB member IEFSSNxx. For information about the parameters, see Update SYS1.PARMLIB members.

CSQ3104I
csect-name - TERMINATION COMPLETE

Explanation

The queue manager has terminated. The actual z/OS termination of the queue manager address spaces might have completed earlier. This message is presented for every termination, normal or abnormal.

CSQ3105E
csect-name - UNABLE TO LOAD EARLY PROCESSING PROGRAM 'CSQ3EPX'. *ssnm* IS NOT AVAILABLE

Explanation

Subsystem initialization or early processing refreshing for the IBM MQ subsystem failed because the initialization program (CSQ3INI) could not locate the early processing program (CSQ3EPX).

For subsystem initialization, the program must be either in the linkpack area (LPA) or in a library which is in the link list. For early processing refreshing, the program must be in the LPA.

System action

Subsystem initialization or early processing refreshing ends abnormally. IBM MQ subsystem *ssnm* is not available.

CSQ3106E
csect-name - QUEUE MANAGER STOPPED. COMMAND NOT PROCESSED - *command-text*

Explanation

A command was received which cannot be processed due to one of the following:

- The queue manager has not been started (this could be because the START QMGR command was not entered correctly)
- The command was queued for processing while the queue manager was starting, but startup terminated with an error
- The queue manager terminated before the command could be processed

System action

The command is not processed.

CSQ3107E
csect-name - COMMAND REJECTED. REQUESTER NOT AUTHORIZED

Explanation

A command was received from a console that does not have the correct authority.

System action

The command is not processed. This message is sent to the console that entered the command.

System programmer response

Verify that this console should be used for entering IBM MQ commands. If so, authorize it for IBM MQ commands by using z/OS services.

Note: If IBM MQ security is not activated, this check is still performed. This authorization is the z/OS console authority, and is not related to the external security manager. The user ID that entered the IBM MQ command must have OPERPARM AUTH with SYS, ALL, or MASTER console authority.

CSQ3108E

csect-name - COMMAND REJECTED. COMMAND FACILITY PATH UNAVAILABLE

Explanation

A command was received, but the path from z/OS consoles to the IBM MQ command processor is unavailable. It might still be possible to enter commands in other ways. You can also receive this message if the early code for the queue manager was being refreshed when the command was issued.

System action

The command is not processed. This message is delivered to the console that entered the command.

System programmer response

The console command facility is available again the next time the queue manager is started.

If the command was rejected because the early code for the queue manager was being refreshed when you issued it, wait until message CSQ3110I is issued to indicate that the early code has successfully refreshed before you issue the command again.

CSQ3109E

csect-name - UNABLE TO OBTAIN SUBSYSTEM AFFINITY TABLE INDEX FOR SUBSYSTEM *ssnm*. IEFSSREQ RC=*nn*

Explanation

IBM MQ was unable to obtain a subsystem affinity table index for the named subsystem. z/OS did not recognize the named subsystem name as a known subsystem. If this message is issued, a serious error has occurred in z/OS or IBM MQ.

In the message, *nn* is the return code from the IEFSSREQ z/OS service. *ssnm* is the name of the IBM MQ subsystem undergoing IPL-time initialization.

System action

IBM MQ ends abnormally with completion code X'5C6' and reason code X'00F30104'. The IBM MQ subsystem with the indicated name is not available for this IPL of z/OS.

System programmer response

Try to perform an IPL of the z/OS system. If the problem persists, see Problem determination on z/OS for information about identifying and reporting the problem.

CSQ3110I

csect-name - SUBSYSTEM *ssnm* INITIALIZATION COMPLETE

Explanation

Either:

- IBM MQ subsystem initialization is complete, following z/OS IPL processing or the z/OS command SETSSI ADD.
- The IBM MQ early processing program has been successfully refreshed, following a REFRESH QMGR TYPE(EARLY) command.

CSQ3111I

csect-name - EARLY PROCESSING PROGRAM IS V*n* LEVEL *l*

Explanation

This message shows the level of the early processing program that is being used.

The level is of the form *nnn-mmm* and indicates the capability of the early code.

nnn is incremented for each new release of the product and *mmm* can be incremented from time to time when PTFs add maintenance to the early code.

The early code level used must have a capability level corresponding with the highest release of the product you intend to run on an LPAR. You can use the *nnn* value to confirm the level installed.

Corresponding values of *nnn* are:

- **005**: IBM WebSphere MQ for z/OS Version 7.0.1
- **006**: IBM WebSphere MQ for z/OS Version 7.1
- **007**: IBM MQ for z/OS Version 8.0

CSQ3112E

csect-name - INVALID CPF SCOPE

Explanation

The z/OS command SETSSI ADD or the subsystem definition record in the IEFSSNxx member of SYS1.PARMLIB for the IBM MQ subsystem specified the CPF scope initialization parameter incorrectly.

The failing subsystem name is provided in message IEF759I, which follows this message.

System action

The IBM MQ subsystem with the indicated name is not available.

System programmer response

Correct the parameter fields in the record of SYS1.PARMLIB member IEFSSNxx. For information about the parameters, see Update SYS1.PARMLIB members.

CSQ3113E

csect-name - COMMAND PREFIX REGISTRATION FAILED. INVALID CHARACTER(S) IN CPF

Explanation

Command prefix registration failed because the command prefix (CPF) contains invalid characters.

System action

The IBM MQ subsystem with the indicated name is not available.

System programmer response

Correct the CPF parameter in the record of SYS1.PARMLIB member IEFSSNxx. For information about the parameters, see Update SYS1.PARMLIB members.

CSQ3114E

csect-name - COMMAND PREFIX REGISTRATION FAILED. INVALID CHARACTER(S) IN SUBSYSTEM NAME

Explanation

Command prefix registration failed because the subsystem name used as the owner of the command prefix (CPF) contains invalid characters.

System action

The IBM MQ subsystem with the indicated name is not available.

System programmer response

Correct the CPF parameter in the record of SYS1.PARMLIB member IEFSSNxx. For information about the parameters, see Update SYS1.PARMLIB members.

CSQ3115E

csect-name - COMMAND PREFIX REGISTRATION FAILED. CPF ALREADY DEFINED

Explanation

Command prefix registration failed because the command prefix (CPF) was already defined to z/OS.

System action

The IBM MQ subsystem with the indicated name is not available.

System programmer response

Correct the CPF parameter in the record of SYS1.PARMLIB member IEFSSNxx. For information about the parameters, see Update SYS1.PARMLIB members.

CSQ3116E

csect-name - COMMAND PREFIX REGISTRATION FAILED. CPF IS A SUBSET OF A CPF ALREADY DEFINED

Explanation

Command prefix registration failed because the command prefix (CPF) is a subset of a CPF already defined to z/OS.

System action

The IBM MQ subsystem with the indicated name is not available.

System programmer response

Correct the CPF parameter in the record of SYS1.PARMLIB member IEFSSNxx. For information about the parameters, see Update SYS1.PARMLIB members.

CSQ3117E

csect-name - COMMAND PREFIX REGISTRATION FAILED. CPF IS A SUPERSET OF A CPF ALREADY DEFINED

Explanation

Command prefix registration failed because the command prefix (CPF) is a superset of a CPF already defined to z/OS .

System action

The IBM MQ subsystem with the indicated name is not available.

System programmer response

Correct the CPF parameter in the record of SYS1.PARMLIB member IEFSSNxx. For information about the parameters, see Update SYS1.PARMLIB members.

CSQ3118E
csect-name - SYSTEM ERROR DURING COMMAND PREFIX REGISTRATION

Explanation

A z/OS error occurred during command prefix (CPF) registration.

System action

The MQ subsystem with the indicated name is not available.

System programmer response

Check the z/OS console for other messages relating to the problem.

CSQ3119E
csect-name call-name call for group attach table failed, rc=*rc*

Explanation

During initialization for the group connect facility, a name token services call failed. *rc* is the return code (in hexadecimal) from the call.

System action

Processing continues, but the group connect facility will not be available to CICS.

System programmer response

See the *MVS Authorized Assembler Services Reference* manual for information about the return codes from the name token services call. If you are unable to solve the problem, take a stand-alone system dump and contact your IBM support center.

CSQ3120E
csect-name - IXCQUERY ERROR FOR XCF GROUP *group-name* APPLID= *applid*, RC= *rc* REASON= *reason*

Explanation

A CICS region with APPLID *applid* attempted to connect to a queue-sharing group. During processing of the request an IXCQUERY call failed with return code *rc* and reason code *reason*.

The XCF group for which the IXCQUERY request was performed is identified by *group-name*.

System action

The request by CICS to connect to the queue-sharing group fails with the reason code MQRC_UNEXPECTED_ERROR.

System programmer response

See the *z/OS MVS Sysplex Services Reference* manual for an explanation of the IXCQUERY return and reason codes. If you are unable to solve the problem, contact your IBM support center.

CSQ3201E
ABNORMAL EOT IN PROGRESS FOR USER=*user* CONNECTION-ID=*conn-id* THREAD-XREF=*thread-xref* JOBNAME=*jobname* ASID=*asid* TCB=*tcb*

Explanation

Abnormal termination processing has been started for the agent with the values for the USER, CONNECTION-ID, THREAD-XREF, JOBNAME, ASID and TCB shown. These values are the last known set of identifiers for the terminating agent.

The abnormal termination might be the result of an error in the allied agent's address space or the result of the z/OS command CANCEL issued by the operator.

The value for the USER, the THREAD-XREF or both might be blank. The values for the USER, CONNECTION-ID, THREAD-XREF, JOBNAME and ASID are the last values established to IBM MQ for this connection and might represent the current activity of the agent. The TCB value is the address of the TCB that is terminating. Previous IBM MQ work by this agent might have completed successfully.

This message, CSQ3201E, is written to the z/OS console after the agent has been removed from the service task work queue at the time that termination processing begins.

System action

The agent was previously queued to a service task for termination processing. This message indicates that the agent has been taken from the queue for processing. Any uncommitted changes will be backed out.

System programmer response

See the Problem Determination section of this message. The z/OS commands CANCEL and FORCE will have no effect. Do not cancel IBM MQ. If an extensive backout is in progress, the subsequent queue manager restart might take a very long time due to additional log activity.

CSQ3202E
CONNECTION FOR *jobname* FAILED, INSUFFICIENT ECSA STORAGE TO CREATE ACE

Explanation

jobname attempted to connect to IBM MQ using the MQCONN, or MQCONNX, API call.

There was insufficient common storage available to build the control blocks to represent the connection and to the connection attempt failed.

There might be a system wide ECSA shortage, or the storage available for creating new queue manager connections might be limited by the ACELIM system parameter.

This message can be seen for batch applications, including RRS applications; for example, Db2 stored procedures and WebSphere Application Server.

System action

The MQCONN or MQCONNX API call, used by *jobname* returns MQCC_FAILED, together with reason code MQRC_Q_MGR_NOT_AVAILABLE 2059

Queue manager processing continues.

CSQ3580E
CONNECTION FOR '*ssi-call*' GAVE RC=*rc*, REASON=*reason*

Explanation

A nonzero return code has been returned to CSQ3AMI2 from the connect to subsystem interface (SSI) call. The variables in the message indicate which SSI call is involved and the actual return and reason codes associated with it.

System action

The current task is ended abnormally with a system completion code of X'5C6' and a reason code of X'00F30580'. The queue manager terminates.

System programmer response

Restart the queue manager. Note the values contained in the message, and contact your IBM support center.

Db2 manager messages (CSQ5...): z/OS

CSQ5001I
csect-name Connected to Db2 *db2-name*

Explanation

The queue manager has successfully established a connection to the named Db2 subsystem.

System action

Processing continues.

System programmer response

None.

CSQ5002E
csect-name Connection to Db2 using *connect-name* failed, RC=*return-code* reason=*reason*

Explanation

The queue manager's attempt to establish a connection to the named Db2 subsystem failed.

System action

Queue manager startup is terminated.

System programmer response

This is normally an authorization error.

Consult the *Db2 for z/OS Messages and Codes* manual for an explanation of the codes and attempt to resolve the problem.

CSQ5003A
csect-name Connection to Db2 using *connect-name* pending, no active Db2

Explanation

The queue manager is waiting for an eligible Db2 subsystem to become active so that a connection can be established. Alternatively, RRS is inactive or was started after the Db2 subsystems.

System action

The queue manager waits for an eligible Db2 subsystem to become active.

System programmer response

Check whether the Db2 subsystem(s) are active. If not then start them. If they are active, ensure RRS is active and check that it was started prior to the Db2 subsystems.

CSQ5004E
csect-name Db2 table entry for queue manager in queue-sharing group *qsg-name* is missing or incorrect

Explanation

During startup the queue manager was unable to find its entry in the Db2 administration tables, or the entry was incorrect.

System action

The queue manager terminates with completion code X'6C6' and reason code X'00F50013'.

System programmer response

Check that a queue manager record exists in the Db2 tables for the Db2 data-sharing group specified. Check the QSGDATA system parameter specifies the correct Db2 data-sharing group. If so, check that a queue manager entry exists in the CSQ.ADMIN_B_QMGR table.

If you are migrating from a previous release of IBM MQ, check also that you have updated the Db2 tables to the format for the current release. For information about migration and compatibility between releases, see Maintaining and migrating.

CSQ5005E
csect-name Queue manager release level is incompatible with queue-sharing group

Explanation

The release level of the queue manager that is being started is incompatible with that of other members of the queue-sharing group.

System action

The queue manager terminates with completion code X'6C6' and reason code X'00F50029'.

System programmer response

Verify that the correct load libraries are being used and that the queue-sharing group information in the system parameters has been specified correctly. Also use the queue-sharing group utility (CSQ5PQSG) to verify that the queue manager has been defined correctly in the Db2 administration tables, using the MIGRATE QSG option. Ensure that you use the same version of IBM MQ for the utility, as was used for running the queue manager.

For information about migration and compatibility between releases, see Migrating queue-sharing groups.

If the MIGRATE QSG option results show queue managers that no longer exist, but are still in the Db2 tables, use the REMOVE QMGR option or, if necessary, the FORCE QMGR option.

CSQ5006E
csect-name Data-sharing groups differ

Explanation

A mismatch has been detected between the Db2 data-sharing group specified on the QSGDATA system parameter and the queue manager entry in the CSQ.ADMIN_B_QMGR table.

System action

The queue manager terminates with completion code X'6C6' and reason code X'00F50006'.

System programmer response

The queue-sharing group name specified on the QSGDATA system parameter must match that in which the queue manager is defined in the Db2 CSQ.ADMIN_B_QMGR table.

CSQ5007E
csect-name RRSAF function *function* failed for plan *plan-name*, RC=*return-code* reason=*reason* syncpoint code=*sync-code*

Explanation

A non-zero or unexpected return code was returned from an RRSAF request. The Db2 plan involved was *plan-name*.

System action

If the error occurs during queue manager startup or reconnect processing, the queue manager terminates with completion code X'6C6' and reason code X'00F50016'. Otherwise, an error message is issued and processing continues.

System programmer response

Determine the cause of the error using the RRS return and reason code from the message.

Consult the *Db2 for z/OS Messages and Codes* manual for an explanation of the codes and attempt to resolve the problem.

CSQ5008E
csect-name Db2 *db2-name* is not a member of data-sharing group *dsg-name*

Explanation

The Db2 subsystem to which the queue manager has connected is not a member of the Db2 data-sharing group specified on the QSGDATA system parameter.

System action

The queue manager terminates with completion code X'6C6' and reason code X'00F50007'.

System programmer response

Ensure that the Db2 subsystem to which the queue manager has connected is a member of the data-sharing group specified on the QSGDATA system parameter.

Issue the Db2 command DIS GROUP to the Db2 subsystem and check the data-sharing group name matches the data-sharing group name on the QSGDATA system parameter.

CSQ5009E
csect-name SQL error for table *table-name*, code=*SQL-code* state=*SQL-state*, data=*d1 d2 d3 d4 d5*

Explanation

A non-zero or unexpected SQL return code was returned from a Db2 SQL request.

System action

The requested operation fails. Processing continues, but the failed request may result in further errors occurring. In some circumstances, the queue manager terminates with completion code X'6C6' and reason code X'00F50014'.

System programmer response

Determine the reason for the SQL error and correct the problem.

Consult the *Db2 for z/OS Messages and Codes* manual to determine the reason for the SQL error.

CSQ5010E
csect-name XCF IXCQUERY member error, RC=*return-code* reason=*reason*

Explanation

The queue manager received an unexpected return code from an IXCQUERY request.

System action

The queue manager terminates with completion code X'6C6' and reason code X'00F50017'.

System programmer response

Determine the reason for the unexpected error and correct the problem.

Consult the *z/OS MVS Programming: Sysplex Services Reference* manual for an explanation of the return and reason code from the IXCQUERY request.

This message may occur if one or more of the queue managers in a queue-sharing group (QSG) do not have a member entry in the XCF group for the QSG.

Enter the following z/OS command substituting the QSG name for xxxx:

```
D XCF,GRP,CSQGxxxx,ALL
```

This will list the members of the XCF group. If any queue managers are defined as a member of the QSG, but do not have an entry in the XCF Group, use the ADD QMGR command of the CSQ5PQSG utility to restore the XCF group entry for that queue manager. The utility should be run for each queue manager which does not have an entry in the XCF group.

CSQ5011E
csect-name XCF IXCJOIN group error, RC=*return-code* reason=*reason*

Explanation

The queue manager received an unexpected return code from an IXCJOIN request.

System action

The queue manager terminates with completion code X'6C6' and reason code X'00F50019'.

System programmer response

Determine the reason for the unexpected error and correct the problem.

Consult the *z/OS MVS Programming: Sysplex Services Reference* manual for an explanation of the return and reason code from the IXCJOIN request.

CSQ5012E
csect-name XCF IXCQUIES group error, RC=*return-code* reason=*reason*

Explanation

The queue manager received an unexpected return code from an IXCQUIES request.

System action

The queue manager terminates with completion code X'6C6' and reason code X'00F50021'.

System programmer response

Determine the reason for the unexpected error and correct the problem.

Consult the *z/OS MVS Programming: Sysplex Services Reference* manual for an explanation of the return and reason code from the IXCQUIES request.

CSQ5013E
csect-name XCF IXCSETUS error, RC=*return-code* reason=*reason*

Explanation

The queue manager received an unexpected return code from an IXCSETUS request.

System action

The queue manager terminates with completion code X'6C6' and reason code X'00F50018'.

System programmer response

Determine the reason for the unexpected error and correct the problem.

Consult the *z/OS MVS Programming: Sysplex Services Reference* manual for an explanation of the return and reason code from the IXCSETUS request.

CSQ5014I
csect-name Connection to *db2-name* lost, Db2 terminated abnormally

Explanation

The queue manager received an abnormal termination notification from the Db2 subsystem to which it is connected.

System action

The queue manager will clean up its connection to the Db2 subsystem and attempt to reconnect. If a Db2 group attach name was specified on the QSGDATA system parameter a connection to a different Db2 may occur.

System programmer response

Determine the reason for the Db2 abnormal termination. Correct the problem and attempt to restart the Db2 subsystem.

CSQ5015I

csect-name Connection to *db2-name* lost, Db2 shut down forcibly

Explanation

The queue manager received a STOP FORCE termination notification from the Db2 subsystem to which it is connected.

System action

The queue manager will clean up its connection to the Db2 subsystem and attempt to reconnect. If a Db2 group attach name was specified on the QSGDATA system parameter a connection to a different Db2 may occur.

System programmer response

Determine the reason for the Db2 forcible stop. Restart the Db2 subsystem.

CSQ5016I

csect-name Connection to *db2-name* quiescing, Db2 terminating

Explanation

The queue manager received a STOP QUIESCE termination notification from the Db2 subsystem to which it is connected.

System action

The queue manager will quiesce all Db2 server tasks and disconnect from the Db2 subsystem so that it can shut down. It will then attempt to reconnect. If a Db2 group attach name was specified on the QSGDATA system parameter a connection to a different Db2 may occur.

System programmer response

Restart the Db2 subsystem so that shared queue operations can resume.

CSQ5019I

csect-name Disconnected from Db2 *db2-name*

Explanation

The queue manager has successfully disconnected from the Db2 subsystem.

System action

If the disconnect is due to a Db2 STOP MODE(QUIESCE) the queue manager will attempt to reconnect to the Db2 subsystem.

System programmer response

None.

CSQ5020E

csect-name SQL error, table *table-name* not defined in Db2

Explanation

The queue manager attempted to access one of its Db2 tables. Db2 has returned an SQL code indicating the table does not exist.

System action

The request fails and processing continues.

System programmer response

Check that all MQ tasks to set up the Db2 environment completed successfully and that the correct Db2 data-sharing group name was specified on the QSGDATA system parameter.

CSQ5021E
csect-name SQL error, table *table-name* index not built in Db2

Explanation

The queue manager has attempted to access one of its Db2 tables. Db2 has returned an SQL code indicating that the index for the specified table has not been built.

System action

The request fails and processing continues.

System programmer response

Check that all IBM MQ tasks to set up the Db2 environment completed successfully and that the correct Db2 data-sharing group name was specified on the QSGDATA system parameter.

CSQ5022I
csect-name Pending connection to Db2 using *connect-name* ended, queue manager terminating

Explanation

The outstanding connection pending request to Db2 has been terminated due to a STOP QMGR request.

System action

The pending connect to Db2 is canceled and queue manager termination continues.

System programmer response

None.

CSQ5023E
csect-name SQL error, failed to access table *table-name*

Explanation

An attempt by the queue manager to access one of its tables was returned an SQL code indicating that access to the named resource failed.

System action

The request fails and processing continues.

System programmer response

This message will be followed by message CSQ5009E which contains full details of the information returned from Db2 which should be used in conjunction with messages on the Db2 log to diagnose the problem.

The most likely cause of this problem is contention for a Db2 resource, especially on a heavily-used system. If so, the problem is temporary; retry the action that gave the error.

If not, and the problem persists, determine from the message and the Db2 log the resource concerned and perform the recovery actions necessary to unlock the resource. Such a problem could be caused by a Db2 failure while updating one of the Db2 tables, which will be indicated in the Db2 log.

CSQ5024E
csect-name Unable to update queue manager status, RC=*return-code*

Explanation

During startup and shutdown processing the queue manager attempts to update its status in the CSQ.ADMIN_B_QMGR table. This attempt failed.

System action

None. Startup/shutdown processing continues.

System programmer response

None.

CSQ5025E
csect-name SQL error, function *function* code=*SQL-code*

Explanation

A call to the SQL function specified by *function* returned a non-zero code specified by *SQL-code*.

System action

Processing continues.

System programmer response

Note the values contained in the message, and contact your IBM support center. Consult the *Db2 for z/OS Messages and Codes* manual for more information about the error code.

CSQ5026E
csect-name Unable to access Db2, RRS is not available

Explanation

The queue manager tried to access Db2, but RRS is not available.

System action

If this occurs during queue manager initialization, the queue manager waits for RRS to become available.

If this occurs at other times, the queue manager terminates its connection to Db2, and then tries to reconnect. Some queue-sharing group functions will not be available until RRS is restarted and the connection to Db2 is reestablished.

System programmer response

Start (or restart) RRS.

CSQ5027E
csect-name SQL error for table *table-name*, deadlock or timeout occurred (code=*SQL-code*)

Explanation

An SQL call returned a non-zero code indicating that a deadlock or timeout condition occurred.

System action

The request fails and processing continues.

System programmer response

Retry the command or application involved. If the problem persists, contact your IBM support center. Consult the *Db2 for z/OS Messages and Codes* manual for more information about the error code.

CSQ5028E
csect-name Unable to access Db2, RRS connection limit exceeded

Explanation

The queue manager tried to access Db2, but RRS has reached the limit of allowed concurrent connections (IDENTIFYs).

System action

If this message occurs during queue manager initialization, the queue manager waits for an RRS connection to become available.

If this message occurs at other times, the queue manager terminates its connection to Db2, and then tries to reconnect. Some queue-sharing group functions are not available until RRS is restarted and the connection to Db2 is reestablished.

System programmer response

Adjust the RRS connection limit if required, then start (or restart) RRS.

Ensure that the Db2 system parameter controlling the maximum number of concurrent users and connections is correct. The Db2 parameter is Max Batch connect (CTHREAD) on the thread management panel DSNTIPE.

See the *Db2 for z/OS* documentation for an explanation of this Db2 parameter to resolve the problem.

CSQ5029E
csect-name Operation on Db2 table *table-name* failed

Explanation

An operation requested for the named Db2 table failed. For example, the table might be full, or there might be insufficient storage available to perform the request.

This is most likely to occur when writing data to one of the tables that IBM MQ uses to store large shared messages.

System action

Message CSQ5009E is issued giving details of the associated SQL error codes. The requested operation fails and processing continues. The message or other data is not written to the table.

System programmer response

Investigate the cause of the problem as indicated by the SQL codes in message CSQ5009E.

If the table is one of the tables used for storing large shared messages, and the problem is due to insufficient storage, try the operation again later, as the condition might be temporary. If the problem is because the table is full, remove some of the messages; for example, start an application that retrieves and processes the messages. Use the MQ DISPLAY GROUP command to check if there are any obsolete messages in the table space, and delete them. If necessary, increase the size of the table.

CSQ5032I
csect-name Connection to Db2 *db2-name* in data-sharing group *dsg-name* is suspended

Explanation

This is issued in response to a SUSPEND QMGR FACILITY(Db2) command if it completed successfully.

System action

All Db2 activity is suspended for the queue manager named, and the connection to Db2 is broken.

System programmer response

Use the RESUME QMGR FACILITY(Db2) command when ready to resume Db2 activity.

CSQ5033I

csect-name Connection to Db2 *db2-name* in data-sharing group *dsg-name* is resumed

Explanation

The RESUME QMGR FACILITY(Db2) command completed successfully, reestablishing the connection to Db2.

System action

Db2 activity is resumed for the queue manager named.

CSQ5034I

csect-name Suspend or resume Db2 request pending

Explanation

A SUSPEND or RESUME QMGR FACILITY(Db2) command was issued, but such a request is already pending.

System action

None.

System programmer response

Wait until the pending request completes, then reissue the command if necessary.

CSQ5035I

csect-name Connection to Db2 *db2-name* in data-sharing group *dsg-name* already suspended

Explanation

A SUSPEND QMGR FACILITY(Db2) command was issued, but the connection to the named Db2 subsystem is already suspended.

System action

None.

CSQ5036I

csect-name Connection to Db2 *db2-name* in data-sharing group *dsg-name* not suspended

Explanation

A RESUME QMGR FACILITY(Db2) command was issued, but the connection to the named Db2 subsystem is not suspended.

System action

None.

CSQ5037I

csect-name New function not available, incompatible queue managers in the queue-sharing group

Explanation

An attempt was made to start the queue manager in new function mode, but some queue managers in the queue-sharing group are either not at a version that is sufficient to coexist with the new functions provided in this level of code, have not been started in new function mode, or do not have compatible QSGDATA parameters.

System action

Processing continues, but certain functions will be unavailable.

System programmer response

Ensure that all of the queue managers in the queue-sharing group have been started in new function mode at the appropriate version, then restart the queue manager. See OPMODE on z/OS for more information on running in new function mode

CSQ5038I

csect-name Service task service-task has been unresponsive since `hh.mm.ss.nnnnnn`. Check for problems with Db2

Explanation

The queue manager has detected a service task *service-task* that is taking too long to process a request that started at `hh.mm.ss.nnnnnn`.

System action

Processing continues, but certain functions might be unavailable.

System programmer response

Investigate if there are any problems with Db2 or RRS that prevent them responding to IBM MQ requests. For example, the Db2 CTHREAD limit has been exceeded, or Db2 is running slowly because it is short of resources like CPU, I/O capacity, or storage; or Db2 is waiting for log space.

CSQ5039I

`csect-name` SQL error information

Explanation

An SQL error has occurred. Additional diagnostic information direct from Db2 follows.

System action

See the preceding Db2 manager error message.

System programmer response

Determine the reason for the SQL error and correct the problem.

CSQ5100I

DISPLAY GROUP report ...

Explanation

This message is the initial response to the DISPLAY GROUP command. It is followed by message CSQ5102I which is a formatted report of the queue managers in the group.

System action

Processing continues normally.

CSQ5102I

Queue managers in group *group-name*

Explanation

This message is part of the responses to the DISPLAY GROUP command. It provides information about each queue manager in the group, as follows:

```
Name Num Prefix  Status    Ver  Db2    Connection  name num cpf     qmgr-stat vrm db2-id conn-stat
⋮
End of queue managers report
```

where:

name The name of the queue manager.

num The internally generated number of the queue manager in the group.

cpf The command prefix of the queue manager.

qmgr-stat
The current status of the queue manager:

ACTIVE
The queue manager is running.

INACTIVE
The queue manager is not running, having terminated normally.

FAILED
The queue manager is not running, having terminated abnormally.

CREATED
The queue manager has been defined to the group, but has not yet been started.

UNKNOWN
The status cannot be determined.

vrm The function level of the queue manager. The value is a 3-digit number, where:

v is the version number

r is the release number

m is the modification number.

db2-id The name of the Db2 subsystem or group attachment to which the queue manager connects.

conn-stat
The current status of the connection to Db2:

ACTIVE
The queue manager is running and connected to Db2.

PENDING
The queue manager is running but not connected because Db2 has terminated normally.

FAILED
The queue manager is running but not connected because Db2 has terminated abnormally.

INACTIVE
The queue manager is not running and not connected to Db2.

UNKNOWN
The status cannot be determined.

Exceptionally, the last line might be either:

```
Report terminated, too many lines
```

if the report was generated in response to a command from a z/OS console and more than 253 response lines were generated. Only 253 response lines are returned.

```
Report terminated
```

if there was an error in obtaining the information. The error is described in the following messages.

System action

Processing continues normally.

CSQ5103I

Obsolete messages in Db2 for group *group-name*

Explanation

Messages are normally deleted automatically from Db2, but in exceptional circumstances obsolete messages can remain. This identifies such messages, as follows:

```
LEID msg-id
⋮
End of messages report
```

where:

msg-id
is the identifier of the message.

Exceptionally, the last line might be either:

```
Report terminated, too many lines
```

if the report was generated in response to a command from a z/OS console and more than 253 response lines were generated. Only 253 response lines are returned.

```
Report terminated
```

if there was an error in obtaining the information.

System action

Processing continues normally.

System programmer response

Delete the obsolete messages from Db2. For example, use SPUFI to issue the SQL command

```
DELETE FROM CSQ.ADMIN_B_MESSAGES     WHERE QSGNAME = 'group-name' AND     LEID = 'msg-id';
```

CSQ5113I

Queue manager is not in a queue-sharing group

Severity

0

Explanation

A command that requires a queue-sharing group was entered, but the queue manager is not in a group.

System action

The command is not actioned.

CSQ5116E

call-name call failed, rc=*rc* reason=*reason*

Severity

8

Explanation

During processing for a DISPLAY GROUP command, a coupling facility services call used to get information failed. *rc* is the return code and *reason* is the reason code (both in hexadecimal) from the call.

System action

Processing is terminated. A following message is issued to identify which type of information was being obtained.

System programmer response

See the *z/OS MVS Programming Sysplex Services Reference.* manual for information about the return and reason codes from the call.

CSQ5117E

Information not available for group *group-name* - reason

Severity

8

Explanation

During processing for a DISPLAY GROUP command, information could not be obtained for the group, for the *reason* indicated:

ERROR
A coupling facility services call failed, as indicated in the preceding CSQ5116E message.

CHANGED
The group size has changed.

System action

Processing is terminated.

System programmer response

Resolve the problem accordingly.

Generalized command preprocessor messages (CSQ9...): z/OS

CSQ9000E

'*keyword*' appears more than once

Explanation

The named keyword appears more than once in the command. This message will be issued for each occurrence of the keyword after the first.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command correctly. See Building command scripts for information about the rules for building commands.

CSQ9001E

'*keyword*' is invalid

Explanation

The named keyword is unknown or undefined. It might be misspelled, or it might not be applicable to the command being processed.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command correctly. See MQSC commands for information about the command.

CSQ9002E

Unbalanced parentheses following '*keyword*'

Explanation

An invalid combination of parentheses has been found following the keyword *keyword*. A closing parenthesis must follow an opening parenthesis before any other opening parenthesis occurs.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command correctly. See Building command scripts for information about the rules for building commands.

CSQ9003E
'*keyword*' parameter contains unbalanced apostrophes

Explanation

An odd number of apostrophes is present in a parameter value of keyword *keyword*. If the parameter is a quoted string, it must have one apostrophe at each end of the string. If an apostrophe is to appear within the string, two adjacent apostrophes must be entered. If the parameter is a hexadecimal value, it must be entered as X'hex-characters'.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command correctly. See Building command scripts for information about the rules for building commands.

CSQ9004E
'*keyword*' parameter specifies range (:) incorrectly

Explanation

A parameter of keyword *keyword* specifies a range of values incorrectly. The character used to denote a range is a colon (:); the format is *lower-limit*:*upper-limit*.

System action

Processing for the command is terminated.

System programmer response

See MQSC commands to verify that the command you are using allows a range for the given keyword. Correct the error, and reissue the command.

CSQ9005E
'*keyword*' parameter does not satisfy generic rules

Explanation

For the keyword *keyword*, parameter values can be generic, but the value specified does not conform to the rules for a generic value. The value does not conform to these rules due to one of the following reasons:

- The value contains an asterisk (*) which is not the last character.
- The value contains a question mark (?) or colon (:).
- The keyword is WHERE and the value is a single asterisk.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, correct the keyword parameter, and reenter the command. See MQSC commands for a description of the keyword and how to enter the command.

CSQ9006E
'*keyword*' parameter uses asterisk (*) incorrectly

Explanation

For the keyword *keyword*, an asterisk (*) was used in a parameter value. Either:

- The asterisk was not the last or only character in the value. Incorrect examples are NAME(BL*CK) and NAME(*LUE); a correct specification is NAME(BL*) or NAME(*).
- There is a list of parameter values, for example DETAIL(1,*).

System action

Processing for the command is terminated.

System programmer response

See MQSC commands to verify that the command you are using allows specification of '*' for the given keyword. Correct the error, and reissue the command.

CSQ9007E
Either '*keyword1*' or '*keyword2*' must be specified

Explanation

The command requires that either keyword *keyword1* or keyword *keyword2* is specified, but neither keyword was entered on the command. One of the two keywords must be present in order for the command to be processed.

System action

Processing for the command is terminated.

System programmer response

Reissue the command and include whichever keyword is appropriate. See MQSC reference for descriptions of the two keywords, and for information about the rules for building commands.

CSQ9008E
'*keyword*' may not be negated

Explanation

The negation characters (NO) appear in front of the keyword *keyword*, but negating this keyword is not allowed.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command correctly. See Building command scripts for further information about this command.

CSQ9009E
'*keyword*' not specified

Explanation

The keyword *keyword* must be present, but it was not entered. This keyword must be present in order for the command to process properly.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command including the specified keyword. See MQSC commands for further information about this command.

CSQ9010E
Required parameter for '*keyword*' not specified

Explanation

For the keyword *keyword*, either:

- One or more parameters must be specified, but no parameter was entered.
- A fixed number of parameters must be specified, but fewer parameters were entered.

For example, the keyword USERDATA must have a parameter that is a character string. Entering USERDATA() is meaningless; you must either enter a string (for example, USERDATA(MY_DATA)), or if you want to remove this attribute, you must enter USERDATA(' ').

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, supply appropriate parameters for the specified keyword, and reissue the command. See MQSC commands for further information about this command.

CSQ9011E
Parameter(s) not allowed for '*keyword*'

Explanation

No parameters can be specified for the keyword *keyword*. This message is issued for each invalid parameter, so it can be issued more than once for a command.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, correct the error, and reissue the command. See Building command scripts for details on how to enter the command.

CSQ9012E
'*keyword*' parameter is not hexadecimal

Explanation

Parameter values for the keyword *keyword* must be hexadecimal values. Hexadecimal characters are the numeric digits 0 through 9 and the letters A through F, in either uppercase or lowercase. The value can optionally be specified using the hexadecimal string notation X'hex characters'; for example, *keyword*(123ABC) and *keyword*(X'123ABC') are synonymous.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command, ensuring that the parameters for the named keyword are hexadecimal values.

CSQ9013E
'*keyword*' parameter '*parameter-value*' length is more than *nn*

Explanation

The parameter value *parameter-value* for keyword *keyword* exceeds the limit of *nn* characters in length.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry. See MQSC commands for a list of acceptable parameters. Correct the error, and reissue the command.

CSQ9014E
More than *nn* parameter(s) for '*keyword*'

Explanation

Too many parameters have been specified for the keyword *keyword*. At most *nn* parameters can be specified. In addition to entering too many parameters, this could also be caused by a missing closing parenthesis that has not yet been detected.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command, using no more than the specified limit of parameters for the given keyword. See MQSC reference for further details, and for information about the rules for building commands.

If this error occurs while you are using connection names with the CSQUTIL program you must enclose certain variables within single quotation marks. See CSQUTIL for more information.

CSQ9015E
Parameter '*parameter-value*' is unacceptable for '*keyword*'

Explanation

The parameter value *parameter-value* is not an acceptable value for keyword *keyword*. Either:

- The keyword parameter can be one of a set of character values, but the value specified is not one of them.
- The keyword parameter can be a bounded numeric value, but the value specified is outside the bounds.
- The keyword parameter can be either numeric or one of a set of character values, but the value specified is neither numeric nor one of the set.
- The keyword is WHERE and the first parameter (the filter keyword) is not one of the acceptable keywords for the command.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command correctly. See MQSC reference for a list of acceptable values, and for information about the rules for building commands.

CSQ9016E
'*cmd*' command request not authorized

Explanation

The command requires a level of authorization that you do not have, either for the command itself, or for the resource that it is operating on.

System action

The command is not executed. Processing is terminated.

System programmer response

Contact the system programmer responsible for system security, and request that this person grant you authorization to use the command. Otherwise, you must have someone who is authorized issue the command for you.

CSQ9017E
Failure while processing '*cmd*' command

Explanation

The command preprocessor ended abnormally while processing the command shown in the message. The error is recorded in SYS1.LOGREC, and an SVC dump is requested. The command might have partially completed. Look at any previous response messages to determine what has been done.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command. If it fails again, collect the items listed in the Problem Determination section, and contact your IBM support center.

CSQ9018E
csect-name Insufficient storage to process '*cmd*' command

Explanation

The command preprocessor was unable to obtain sufficient storage to complete processing of any response messages generated by the command.

System action

Processing for the command is terminated abnormally.

System programmer response

If the problem persists, you might need to increase the region size used by your queue manager or channel initiator, or you might need to reduce the number of jobs running in your system.

CSQ9019E
'*cmd*' command is invalid

Explanation

The command, which starts with *cmd*, is invalid. This could be because:

- the command verb is unknown
- no keywords were specified, or none were specified that are valid as a secondary keyword for the command
- there is syntax error at the start of the command

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command correctly. See MQSC reference for the correct command format, and for information about the rules for building commands.

CSQ9020E
'*keyword1*' and '*keyword2*' cannot both be specified

Explanation

The command does not allow keyword *keyword1* and keyword *keyword2* to be specified together.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command, omitting the inappropriate keyword. See MQSC reference for descriptions of the two keywords and how to enter the command.

CSQ9022I

csect-name '*cmd*' NORMAL COMPLETION

Explanation

All synchronous processing for the command completed successfully. Any tasks executing asynchronously on behalf of the command might still be executing when this message is displayed.

System action

Synchronous processing for the command is complete.

CSQ9023E

csect-name '*cmd*' ABNORMAL COMPLETION

Explanation

The command has not completed successfully. The command has issued one or more error messages prior to this message.

System action

Processing for the command has ended.

System programmer response

Follow the instructions for the other messages associated with the error.

CSQ9025E

'*parameter-value*' is unacceptable with 'WHERE' parameter '*filter-keyword*'

Explanation

The parameter values for the WHERE keyword are incompatible. The WHERE keyword must have three parameters, *filter-keyword*, *operator*, and *filter-value*. The error is one of the following:

- The operator parameter is not appropriate for the type of parameter values that the filter keyword requires. For example, the filter keyword requires one of a set of parameter values, but the operator is not EQ or NE.
- The filter value parameter exceeds the length limit for parameter values of the filter keyword.
- The filter value parameter is not a value that is valid as a value of the filter keyword. For example:
  - The filter keyword requires a numeric parameter value but the filter value parameter is not numeric.
  - The filter keyword requires one of a set of parameter values but the filter value parameter is not one of them.
  - The filter keyword requires a bounded numeric parameter value but the filter value parameter is outside the bounds.
  - The filter keyword requires an object or system name, but the filter value parameter does not consist only of characters that are valid for such a name.

Depending on the error, *parameter-value* may be the operator parameter or the filter value parameter.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command correctly. See MQSC reference for information about the parameters for the WHERE keyword.

CSQ9026E
'*keyword*' parameter does not satisfy name rules

Explanation

Parameter values for the keyword *keyword* are names, and therefore must consist only of characters that are valid for the particular type of name, object name or system name. The valid object name characters are uppercase A-Z, lowercase a-z, numerics 0-9, period (.), forward slash (/), underscore (_), and percent sign (%). The valid system name characters are uppercase A-Z, and numerics 0-9; the first character must not be numeric.

This message is issued if the name specified contains invalid characters, or if the name is all blank in cases where an all-blank name is not allowed.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command ensuring that the parameters for the named keyword are of the required type. See MQSC reference for a description of the keyword and how to enter the command.

CSQ9028E
'*keyword*' parameter is not numeric

Explanation

Parameter values for the keyword *keyword* must consist of numeric values only.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command ensuring that the parameters for the named keyword are of the required type. See MQSC reference for a description of the keyword and how to enter the command.

CSQ9029E
csect-name Failure while processing a command

Explanation

An error occurred while processing a command. The command might or might not have been executed. The error has been recorded in the system error log (the SYS1.LOGREC data set), and an SVC dump was attempted.

You can get this message if you have insufficient ECSA.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command. If you cannot resolve the problem, collect the items listed in the Problem Determination section, and contact your IBM support center.

CSQ9030E
'*keyword*' parameter may not be generic

Explanation

The parameter for the keyword *keyword* specifies a generic value using an asterisk (for example, ABC*), but a generic value is not allowed for that keyword.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, correct the keyword parameter, and reenter the command. See MQSC reference for a description of the keyword and how to enter the command.

CSQ9031E
Syntax error following '*keyword*'

Explanation

The text that follows the named keyword contains invalid syntax. This is typically caused by specifying an incorrect sequence of special characters, such as equals (=), comma (,), colon (:), or parentheses.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, examining the text following the named keyword. Ensure that you have followed the rules for command entry, and reenter the command. See Building command scripts for information about the rules for building commands.

CSQ9032E
Requested function is not available

Explanation

An attempt was made to invoke a command processor that was not loaded.

System action

The requested function is not performed.

System programmer response

Verify the command entry, to determine which command caused the error.

CSQ9033E

Command exceeds allowable length

Severity
8

Explanation

The command is so large that its internal form has exceeded the maximum length allowed. The size of the internal form of the command is affected by both the length, and the complexity of the command. (For example, an attempt has been made to use the operations and control panels to create a namelist containing too many names.)

This message could also be caused by commands entered through one of the following:

- the initialization input data sets
- the COMMAND function of the utility program CSQUTIL
- a user-written program that puts commands onto the system-command input queue, SYSTEM.COMMAND.INPUT

System action

Processing of the command is terminated.

System programmer response

If you are using the operations and control panels to define a namelist, use the edit facility to reduce the number of names in the list. If you are entering a command from elsewhere, determine which command caused the error, and verify the syntax of that command from MQSC commands. Correct the command.

CSQ9034E

Command cannot be issued using command server

Severity

8

Explanation

An attempt was made to issue a command using the command server. The command cannot be issued in that way.

The command server is used by commands entered through one of the following:

- the COMMAND function of CSQUTIL
- the CSQINPX initialization input data set of the channel initiator
- a user-written program that puts commands onto the system-command input queue, SYSTEM.COMMAND.INPUT

System action

The command is ignored.

CSQ9035E

csect-name Required keyword not specified

Severity

8

Explanation

The command requires one of a set of alternative keywords to be specified, but none was.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command correctly. See MQSC reference for the proper format of the command, and for information about the rules for building commands.

CSQ9036E

Command with '*keyword*(*parameter-value*)' not allowed when queue manager is active

Severity

8

Explanation

The command has the specified parameter value for keyword *keyword*. The command with this keyword and value can be issued only when the queue manager is not active.

System action

The command is ignored.

System programmer response

See MQSC commands for information about how to use the command.

CSQ9037E

Command must be issued from *ddname*

Severity

8

Explanation

An attempt was made to issue a command from the specified initialization input data set. The command cannot be issued from that data set.

System action

The command is ignored.

System programmer response

See MQSC commands for information about how to use the command.

CSQ9038E

Command must be issued from console

Severity

8

Explanation

An attempt was made to issue a command from other than the z/OS console or its equivalent. The command can only be issued in that way.

System action

The command is ignored.

System programmer response

Issue the command from the z/OS console; it cannot be issued from elsewhere.

If you issued the **DEFINE PSID** command from the console, you must include the additional DSN parameter for the command to complete successfully.

See MQSC commands for information about how to use the command.

CSQ9039E

Command cannot be issued from console

Severity

8

Explanation

An attempt was made to issue a command from the z/OS console or its equivalent. The command cannot be issued in that way.

System action

The command is ignored.

System programmer response

See MQSC commands for information about how to use the command.

CSQ9040E

Command cannot be issued from *ddname*

Severity
8

Explanation

An attempt was made to issue a command from the specified initialization input data set. The command cannot be issued from that data set.

System action

The command is ignored.

System programmer response

See MQSC commands for information about how to use the command.

CSQ9041E

Command not allowed during restart

Severity
8

Explanation

An attempt was made to issue a command before restart had completed, but the command cannot be issued at that time. This could be because the command was in the CSQINP1 initialization input data set.

System action

The command is ignored.

System programmer response

If the command was in the CSQINP1 initialization input data set, delete it.

CSQ9042E

Command with '*keyword*()' cannot be issued from *ddname*

Severity
8

Explanation

The command was issued with the specified keyword from an initialization input data set. The command with this keyword cannot be issued from that data set.

System action

The command is ignored.

System programmer response

See MQSC commands for information about how to use the command.

CSQ9045E
'*keyword*' has parameter(s) and is a 'WHERE' parameter

Explanation

The command specifies the WHERE keyword with a filter keyword parameter *keyword*. That keyword is also specified explicitly with with parameters, which is not allowed.

System action

Processing for the command is terminated.

System programmer response

Verify the command entry, and reissue the command correctly. See MQSC reference for information about the parameters for the WHERE keyword.

IBM MQ for z/OS codes

z/OS

Each component of IBM MQ for z/OS can issue codes and each component uses a unique two character hexadecimal identifier for its messages. Use this topic to identify and interpret the codes for IBM MQ for z/OS components.

The following code types are described:

Connection manager codes (X'94'): z/OS

If a connection manager reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- A description of the actions that led to the error or, if applicable, either a listing of the application program or the input string to a utility program that was being run at the time of the error.
- The console output for the period leading up to the error.
- The queue manager job log.
- The system dump resulting from the error.
- If you are using CICS, the CICS transaction dump output.
- Appropriate IBM MQ, z/OS, Db2, CICS, and IMS service levels.
- If you are using the IBM MQ Operations and Control panels, the ISPF panel name.

00940001

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6', and the queue manager terminates.

System programmer response

Collect the items listed in “Diagnostics” and contact your IBM support center.

Restart your queue manager.

00940003

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” and contact your IBM support center.

00940004

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5019 and contact your IBM support center.

00940007

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5019 and contact your IBM support center.

00940008

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6', and the queue manager terminates.

System programmer response

Collect the items listed in “Diagnostics” on page 5019 and contact your IBM support center.

Restart your queue manager.

00940028

A requested diagnostic trap has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

This should only occur if the IBM support center have requested that a dump be captured to aid in problem diagnosis

Collect the items listed in “Diagnostics” on page 5019 and contact your IBM support center.

0094002B

An internal error has occurred during ALESERV processing.

System action

The current execution unit terminates with completion code X'5C6'. The failing return code from ALESERV will be in register 2 of the dump.

System programmer response

Collect the items listed in “Diagnostics” on page 5019 and contact your IBM support center.

Restart the queue manager.

Topic Manager codes ('X'A3'): z/OS

If a topic manager reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- A description of the actions that led to the error or, if applicable, either a listing of the application program or the input string to a utility program that was being run at the time of the error.
- The console output for the period leading up to the error.
- The queue manager job log.
- The system dump resulting from the error.
- If you are using CICS, the CICS transaction dump output.
- Appropriate IBM MQ, z/OS, Db2, CICS, and IMS service levels.
- If you are using the IBM MQ Operations and Control panels, the ISPF panel name.

00A30001, 00A30002,00A30052, 00A30053, 00A30054, 00A30061, 00A30062, 00A30064, 00A30065, 00A30066, 00A31000

An internal error has occurred while processing a command.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” and contact your IBM support center.

00A30042

An internal error has occurred while processing a command.

If this error occurs in conjunction with a CSQY227E message then the problem is a lack of 64 bit storage.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” and contact your IBM support center.

You should consider raising the value of the MEMLIMIT parameter. For more information, see Address space storage.

00A30072, 00A30073, 00A30074, 00A30075, 00A30076, 00A30077

An internal error occurred during commit processing.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” and contact your IBM support center.

Batch adapter codes (X'C2'): z/OS

00C20001

The CSQBSRV program has detected a request for a nonexistent function. CSQBSRV is invoked from batch and RRS-batch applications via a stub such as CSQBSTUB, CSQBRRSI, or CSQBRSTB.

System action

The application program ends abnormally, but MQ continues processing.

System programmer response

The most likely cause of this problem is incompatible versions of CSQBSRV and the stub. If this is not the cause of the problem, obtain the diagnostic items listed in this topic, and contact your IBM support center.

- Application program listing
- Queue manager job log
- PSW and registers at point of failure

00C20009

The task which started an asynchronous IBM MQ thread (for asynchronous message consumption or asynchronous event listening) has ended before the asynchronous thread which it started had ended. This abend is raised on the asynchronous IBM MQ thread, because processing cannot continue after the resources allocated by the original thread have been released.

System action

The application program ends abnormally, but IBM MQ continues processing.

System programmer response

Ensure that an MQDISC is called for all connections which are used to start asynchronous threads before termination of the task which created the connection.

00C2000A, 00C2000B, 00C2000C, 00C2000D, 00C2000E, 00C2000F

An internal error has occurred while processing an MQCRTMH call.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Obtain the diagnostic items listed in this topic, and contact your IBM support center.

- An application program listing.
- The queue manager job log.
- The PSW and registers at point of failure.

Coupling Facility codes (X'C5'): z/OS

If a coupling facility reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center. Restart the queue manager if necessary.

Diagnostics

- A description of the actions that led to the error or, if applicable, either a listing of the application program or the input string to a utility program that was being run at the time of the error.
- The console output for the period leading up to the error.
- The queue manager job log.
- The system dump resulting from the error.
- If you are using CICS, the CICS transaction dump output.
- Appropriate IBM MQ, z/OS, Db2, CICS, and IMS service levels.
- If you are using the IBM MQ Operations and Control panels, the ISPF panel name.
- A dump of the coupling facility structure.

00C50006

A backup or recovery of a CF structure failed because the queue manager is not connected to a Db2 subsystem.

System action

CF structure backup or recovery processing is terminated.

System programmer response

Configure the Db2 subsystem so that the queue manager can connect to it.

00C50012

CF structure processing failed, because the CF structure became full during the action.

System action

CF structure processing is terminated.

System programmer response

Increase the size of the CF structure.

00C50014

An unexpected reason code was returned by the Db2 subsystem that the queue manager is connected to.

System action

The current operation is terminated.

System programmer response

Investigate the cause of the error, as reported in the preceding messages.

00C50050

The CF structure is being recovered and cannot be used until the recovery is complete.

System action

Processing of the command is terminated.

System programmer response

Wait for the recovery of the structure to complete, then reissue the command. Use the DISPLAY CFSTATUS command to view the status of the CF structures.

00C50064

A backup or recovery of a CF structure failed either because the installation and customization options chosen for IBM MQ do not allow the queue manager to use structures at the required level, or because the level of the structure is not supported by the current command level.

For example, the queue manager may have been migrated from a previous version and currently operating with OPMODE=COMPAT, and CF structures at the level of the structure being processed are not available in compatibility mode.

See OPMODE on z/OS for further information.

System action

CF structure backup or recovery processing is terminated.

00C5004F

This reason code is issued in message CSQM090E when a command has failed. It indicates that a request has been issued for a CF structure, but the request cannot be performed, as explained in the accompanying more specific message.

Severity

4

System action

The command is ignored.

System programmer response

Refer to the description of the accompanying message.

00C5005B

CF structure recovery failed because an error occurred when reading the BSDS of another queue manager in the queue-sharing group.

System action

CF structure recovery processing is terminated.

System programmer response

Check the log for recovery log manager messages that indicate the reason for the error.

00C50D00

A backup of a CF structure failed because a required SMDS data set is not available.

System action

CF structure backup processing is terminated.

System programmer response

Ensure that all SMDS data sets used for the CF structure are available, then reissue the backup command. A **RECOVER CFSTRUCT** command can be used to restore these data sets if this is required.

00C51001, 00C51004, 00C51005, 00C51006, 00C5100A, 00C51019, 00C5101A, 00C5101B, 00C5101C, 00C5001D

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C51021, 00C51022, 00C51023, 00C51024, 00C50025, 00C51026, 00C51027, 00C51028, 00C51029, 00C5002A, 00C5102B, 00C5102C, 00C5102D, 00C5102E, 00C5002F

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C50030, 00C51031, 00C51032, 00C51033, 00C51034, 00C50035, 00C51036, 00C51037, 00C51038, 00C51039, 00C5003A, 00C5103A, 00C5103B, 00C5103C, 00C5103D, 00C5103E, 00C5003F

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C50040, 00C51041, 00C51042, 00C51043, 00C51044, 00C50045, 00C51046, 00C51047

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager may terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C51051, 00C51052, 00C51053, 00C51054, 00C50055, 00C51056

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager may terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C51090, 00C51092, 00C51093

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C51094, 00C51095, 00C51096, 00C51097

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

00C510A1, 00C510A2, 00C510A3, 00C510A4, 00C500A5, 00C510A6, 00C510A7, 00C510A8, 00C510A9, 00C500AA

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C510AB

The CF structure has failed or connection to it has been lost.

System action

This might be issued in response to a command, in which case processing of the command is terminated. Otherwise, the current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Restart the queue manager if necessary. Recover the structure; if the error occurred in response to a command, reissue it.

00C510AC, 00C510AD

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C51100, 00C51101, 00C51102, 00C51103, 00C51104, 00C51105, 00C51106, 00C51107, 00C51108, 00C51109, 00C5110A, 00C5110B, 00C5110C, 00C5110D, 00C5110E, 00C5110F

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager may terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C51110, 00C51111, 00C51112, 00C51113, 00C51114, 00C51115, 00C51116, 00C51117, 00C51118, 00C51119, 00C5111A, 00C5111B, 00C5111C, 00C5111D, 00C5111E, 00C5111F

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C51120, 00C51121, 00C51122, 00C51123, 00C51124, 00C51125, 00C51126, 00C51127, 00C51128, 00C51129, 00C5112A, 00C5112B, 00C5112C, 00C5112D, 00C5112E, 00C5112F

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C51130, 00C51131, 00C51132, 00C51133, 00C51134, 00C51135, 00C51136, 00C51137, 00C51138, 00C51139, 00C5113A, 00C5113B, 00C5113C, 00C5113D, 00C5113E, 00C5113F

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager may terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C51140, 00C51141, 00C51142, 00C51143, 00C51144, 00C51145, 00C51146, 00C51147, 00C51148, 00C51149, 00C5114A, 00C5114B, 00C5114C, 00C5114D, 00C5114E, 00C5114F

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager may terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C51150, 00C51151, 00C51152, 00C51153, 00C51154, 00C51155, 00C51156, 00C51157, 00C51158, 00C51159, 00C5115A, 00C5115B, 00C5115C, 00C5115D, 00C5115E, 00C5115F

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C51160, 00C51161, 00C51162, 00C51163, 00C51164, 00C51165, 00C51166, 00C51167, 00C51168, 00C51169, 00C5116A, 00C5116B, 00C5116C, 00C5116D, 00C5116E, 00C5116F

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C51170, 00C51171, 00C51172, 00C51174, 00C51175, 00C51176, 00C51177, 00C51178, 00C51179, 00C5117A, 00C5117B, 00C5117C, 00C5117D, 00C5117E, 00C5117F

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C51173

An internal error has occurred.

System action

The internal task performing recovery of a CFSTRUCT terminates with completion code x'5C6'.

System programmer response

This error is often, but not exclusively, associated with space issues in the coupling facility.

Ensure that sufficient space is available in the cfstructure.

A common source of error is that the INITSIZE and SIZE values do not match in the CFRM policy. During normal use, the structure has expanded through AUTOALTER processing and the structure backup being restored reflects this size.

However, a new structure has been allocated with the INITSIZE attribute that is too small.If storage issues are not indicated, then collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

00C51180, 00C51181, 00C51182, 00C51184, 00C51185, 00C51186, 00C51187, 00C51188, 00C51189, 00C5118A, 00C5118B, 00C5118C, 00C5118D, 00C5118E, 00C5118F

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager may terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C51183

An internal error has occurred.

System action

The internal task performing recovery of a CFSTRUCT terminates with completion code x'5C6'.

System programmer response

This error is often, but not exclusively, associated with space issues in the coupling facility.

Ensure that sufficient space is available in the cfstructure.

A common source of error is that the INITSIZE and SIZE values do not match in the CFRM policy. During normal use, the structure has expanded through AUTOALTER processing and the structure backup being restored reflects this size.

However, a new structure has been allocated with the INITSIZE attribute that is too small.If storage issues are not indicated, then collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

00C51190, 00C51191, 00C51192, 00C51193, 00C51194, 00C51195, 00C51196, 00C51197, 00C51198, 00C51199, 00C5119A, 00C5119B, 00C5119C, 00C5119D, 00C5119E, 00C5119F

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager may terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C511A0, 00C511A1, 00C511A2, 00C511A3, 00C511A4, 00C511A5, 00C511A6, 00C511A7, 00C511A8, 00C511A9, 00C511AA, 00C511AB, 00C511AC, 00C511AD, 00C511AE, 00C511AF

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager may terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C511B0, 00C511B1, 00C511B2, 00C511B3, 00C511B4, 00C511B5, 00C511B6, 00C511B7, 00C511B8, 00C511B9, 00C511BA, 00C511BB, 00C511BC, 00C511BD, 00C511BE, 00C511BF

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager may terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C511C0, 00C511C1, 00C511C2, 00C511C3, 00C511C4, 00C511C5, 00C511C6, 00C511C7, 00C511C8, 00C511C9, 00C511CA, 00C511CB, 00C511CC, 00C511CD, 00C511CE, 00C511CF

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager may terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C511D0, 00C511D1, 00C511D2, 00C511D3, 00C511D4, 00C511D5, 00C511D6, 00C511D7, 00C511D8, 00C511D9, 00C511DA, 00C511DB, 00C511DC, 00C511DD, 00C511DE, 00C511DF

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C511E0, 00C511E1, 00C511E2, 00C511E3, 00C511E4, 00C511E5, 00C511E6, 00C511E7, 00C511E8, 00C511E9, 00C511EA, 00C511EB, 00C511EC, 00C511ED, 00C511EE, 00C511EF

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C511F0, 00C511F1, 00C511F2, 00C511F3, 00C511F4, 00C511F5, 00C511F6, 00C511F7, 00C511F8, 00C511F9, 00C511FA, 00C511FB, 00C511FC, 00C511FD, 00C511FE, 00C511FF

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5023 and contact your IBM support center.

Restart the queue manager if necessary.

00C53000

The queue manager cannot use the administration structure because its size is less than the minimum that IBM MQ requires.

System action

The queue manager terminates with completion code X'6C6'.

System programmer response

Increase the size of the admininstration structure. See message CSQE022E for more information.

00C53001

The queue manager has detected a mismatch between the queue-sharing group creation timestamp in the Db2 tables and the creation timestamp associated with the structure name in message CSQE029E.

System action

The queue manager terminates, a record is written to SYS1.LOGREC and a dump is taken.

System programmer response

Verify the queue manager, queue-sharing group and data-sharing group configuration and determine whether a queue manager has configured to connect to a different Db2 data-sharing group.

If the queue manager and queue-sharing group configuration is correct then the structure should be deallocated. Having verified that there are only failed-persistent connections remaining to the structure, deallocate it with the z/OS command

```
SETXCF FORCE,STRUCTURE,STRNAME=ext-struc-name
```

(In this command, *ext-struc-name* is formed by prefixing the IBM MQ structure name from message CSQE029E with the queue-sharing group name.)

00C53002

The queue manager cannot use the administration structure because the administration structure is full and remains full despite repeated attempts to wait for space to become available.

System action

The queue manager terminates with completion code X'5C6'.

System programmer response

Increase the size of the administration structure. See message CSQE038E for more information.

Message generator codes (X'C6'): z/OS

00C60001

IBM MQ received return code X'20' when issuing a WTO request to display a console message. This means that there are no message buffers for either Multiple Console Support (MCS) or JES3, or there is a JES3 WTO staging area excess. The WTO request is terminated. The current console message and all subsequent informational console messages are ignored until the problem is corrected.

System action

A record is written to SYS1.LOGREC. A retry is requested and execution continues. IBM MQ resumes issuing console messages when the condition is corrected.

00C60004

The queue manager was unable to load the message table (CSQFMTAB).

System action

The queue manager terminates.

System programmer response

Ensure that the message table is in the required library (SCSQANLx, where x is your national language letter), that it is referenced correctly, and that all the libraries in the concatenation are APF authorized. Restart the queue manager.

00C60005

An internal error has occurred.

System action

The queue manager is terminated, and a dump is produced.

System programmer response

Restart the queue manager.

Collect the following diagnostic items and contact your IBM support center:

- Queue manager job log
- System dump resulting from the error

00C60006

The MQ utility program was unable to load its message table (CSQFSTAB).

System action

The utility program ends abnormally.

System programmer response

Check the console for messages indicating why CSQFSTAB was not loaded. Ensure that the message table is in the required library (SCSQANLx, where x is your national language letter), and that it is referenced correctly, and resubmit the job.

The utility program attempts to load this module from the library data sets under the STEPLIB DD statement of the utility address space.

00C60007

The IBM MQ CICS adapter was unable to load its message table (CSQFCTAB).

System action

The IBM MQ CICS adapter server task terminates.

System programmer response

Check the console for messages indicating why CSQFCTAB was not loaded. Ensure that the message table is in the required library (SCSQANLx or SCSQSNLx, where x is your national language letter), and that it is referenced correctly.

CSQCSERV attempts to load this module from the library data sets under the STEPLIB DD statement of the CICS address space.

00C60008

The IBM MQ utility program was unable to load its message table (CSQFLTAB).

System action

The utility program ends abnormally.

System programmer response

Check the console for messages indicating why CSQFLTAB was not loaded. Ensure that the message table is in the required library (SCSQANLx, where x is your national language letter), and that it is referenced correctly, and resubmit the job.

The utility program attempts to load this module from the library data sets under the STEPLIB DD statement of the utility address space.

00C6000A

The IBM MQ early processing program was unable to load its message table (CSQ3ECMX).

System action

The queue manager terminates.

System programmer response

Ensure that the message table in the required library (SCSQSNLx, where x is your national language letter), and that it is referenced correctly, and perform an IPL of your z/OS system or use the z/OS command SETSSI ADD to restart the queue manager.

00C6000B

The distributed queuing component was unable to load its message table (CSQFXTAB).

System action

The channel initiator ends.

System programmer response

Check the console for messages indicating why CSQFXTAB was not loaded. Ensure that the message table is in the required library (SCSQANLx, where x is your national language letter), that it is referenced correctly, and that all the libraries in the concatenation are APF authorized. Restart the channel initiator.

00C6000C

The IMS trigger monitor was unable to load its message table (CSQFSTAB).

System action

The trigger monitor ends.

System programmer response

Check the console for messages indicating why CSQFSTAB was not loaded. Ensure that the message table is in the required library (SCSQANLx, where x is your national language letter), and that it is referenced correctly, and restart the trigger monitor.

00C600F0

The Advanced Message Security component was unable to load its message table (CSQF0TAB).

System action

The Advanced Message Security component fails to start during queue manager startup.

System programmer response

Check the console for messages indicating why CSQF0TAB was not loaded. Ensure that the message table is in the required library (SCSQANLx, where x is your national language letter), that it is referenced correctly, and that all the libraries in the concatenation are APF authorized. Restart the queue manager.

Functional recovery manager codes (X'C7'): z/OS

00C70010

While trying to recover from an error, an internal consistency check indicated a storage overlay, or an internal error.

System action

Control is percolated to the z/OS recovery termination manager, and a dump is requested.

System programmer response

Retain the dump, and contact your IBM support center for assistance.

Restart the queue manager if necessary.

00C70020

A critical procedure recovery routine has ended abnormally, causing a secondary abnormal end.

System action

Control is percolated to the z/OS recovery termination manager, and in some cases the queue manager terminates abnormally. A dump is produced for both the primary and secondary errors.

System programmer response

Retain both dumps, and contact your IBM support center for assistance.

Restart the queue manager if necessary.

00C70030

A request to z/OS to establish an ESTAE produced a non-zero return code.

System action

A dump is requested.

System programmer response

The return code from z/OS is captured in register 14. See the *MVS Assembler Services Reference* manual for an explanation of the return code.

00C70040

This abnormal end reason code was caused by an internal IBM MQ error.

System action

Control is percolated to the z/OS recovery termination manager, and a dump is requested.

System programmer response

Retain the dump, and contact your IBM support center for assistance.

Restart the queue manager if necessary.

Security manager codes (X'C8'): z/OS

If a security manager reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- A description of the actions that led to the error or, if applicable, either a listing of the application program or the input string to a utility program that was being run at the time of the error.
- The console output for the period leading up to the error.
- The queue manager job log.
- The system dump resulting from the error.
- If you are using CICS, the CICS transaction dump output.
- Appropriate IBM MQ, z/OS, Db2, CICS, and IMS service levels.
- If you are using the IBM MQ Operations and Control panels, the ISPF panel name.
- The security command issued before the error.

00C80001

An attempt to obtain storage for the security manager was unsuccessful.

Note: This could indicate a system-wide storage problem.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the return code from the storage failure.

System programmer response

Check that you are running with the recommended region size, and if not, reset your system and restart the queue manager. If this is not the cause of the problem, use these items to diagnose the cause of the problem:

- Queue manager job log
- Information about any other storage-related problems
- System dump resulting from the error

00C80002

An attempt to obtain storage for the security manager was unsuccessful.

Note: This error code could indicate a system-wide storage problem.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the return code from the storage failure.

System programmer response

Check that you are running with the suggested region size, and if not, reset your system and restart the queue manager. If this is not the cause of the problem, use these items to diagnose the cause of the problem:

- Queue manager job log
- Information about any other storage-related problems
- System dump resulting from the error

00C80003

An attempt to obtain a storage subpool for the security manager was unsuccessful.

Note: This error code could indicate a system-wide storage problem.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the return code from the storage failure.

System programmer response

Check that you are running with an appropriate region size, and if not, reset your system and restart the queue manager. If the region size is not the cause of the problem, use these items to diagnose the cause of the problem:

- Queue manager job log
- Information about any other storage-related problems
- System dump resulting from the error

00C80004

An internal error has occurred.

System action

The queue manager is terminated, and a dump is produced.

System programmer response

Collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

Restart the queue manager.

00C8000A

A severe error has occurred during a SAF RACROUTE REQUEST=STAT call to the external security manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. Check your security configuration (for example, that the required classes are installed and active). If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C8000B

A severe error has occurred during a SAF RACROUTE REQUEST=EXTRACT call to the external security manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the entity being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. For information about setting IBM MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C8000C

A severe error has occurred during a SAF RACROUTE REQUEST=LIST (create) call to the external security manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class, and register 3 the address of the entity, being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C8000D

An unexpected return code has been received from one of the following SAF calls to the external security manager (ESM) during security switch processing at queue manager initialization time:

- RACROUTE REQUEST=EXTRACT
- RACROUTE REQUEST=LIST
- RACROUTE REQUEST=STAT

System action

Message CSQH004I is produced containing the return codes from SAF and the ESM. The queue manager is terminated, and a dump is produced. Register 2 contains the address of the return codes.

System programmer response

See your ESM documentation for information about the return codes that appear in message CSQH004I (in the job log) or the dump. For information about setting IBM MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C8000E

An unexpected setting for the subsystem security switch was encountered.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the control block containing the switch setting.

System programmer response

Collect the items listed in "Diagnostics" on page 5035, together with a note of what you expected the switch to be set to, and whether you had defined a profile for it or not, and contact your IBM support center.

00C8000F

An internal error has occurred.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class involved at the time of the error.

System programmer response

Collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

Restart the queue manager.

00C80010

An attempt to obtain storage for the security manager was unsuccessful.

Note: This error code could indicate a system-wide storage problem.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced. Register 2 contains the return code from the storage failure.

System programmer response

Check that you are running with the suggested region size, and if not, reset your system and restart the queue manager. If this is not the cause of the problem, use the items listed in “Diagnostics” on page 5035, together with information about any other storage-related problems, to diagnose the cause of the problem. If you are unable to resolve the problem, contact your IBM support center.

00C80011

An attempt to obtain a storage subpool for the security manager was unsuccessful.

Note: This error code could indicate a system-wide storage problem.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced. Register 2 contains the return code from the storage failure.

System programmer response

Check that you are running with the suggested region size, and if not, reset your system and restart the queue manager. If this is not the cause of the problem, use the items listed in “Diagnostics” on page 5035, together with information about any other storage-related problems, to diagnose the cause of the problem. If you are unable to resolve the problem, contact your IBM support center.

00C80012

An attempt to obtain storage for the security manager was unsuccessful.

Note: This error code could indicate a system-wide storage problem.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced. Register 2 contains the return code from the storage failure.

System programmer response

Check that you are running with the suggested region size, and if not, reset your system and restart the queue manager. If this is not the cause of the problem, use the items listed in “Diagnostics” on page 5035, together with information about any other storage-related problems, to diagnose the cause of the problem. If you are unable to resolve the problem, contact your IBM support center.

00C80013

An internal error has occurred while processing a security request.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

00C80020

An attempt to obtain storage for the security manager was unsuccessful.

Note: This error code could indicate a system-wide storage problem.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced. Register 2 contains the return code from the storage failure.

System programmer response

Check that you are running with the suggested region size, and if not, reset your system and restart the queue manager. If this is not the cause of the problem, use the items listed in "Diagnostics" on page 5035, together with information about any other storage-related problems, to diagnose the cause of the problem. If you are unable to resolve the problem, contact your IBM support center.

00C80024

An internal error has occurred while processing a command.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced.

System programmer response

Collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80025

An internal error has occurred while processing a command.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced.

System programmer response

Collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80026

An internal error has occurred while processing a command.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced.

System programmer response

Collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80027

An unrecognized keyword was encountered whilst processing a REFRESH SECURITY command.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced. Register 2 contains the address of the keyword causing the problem.

System programmer response

Collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80028

An attempt to obtain a storage subpool for the security manager was unsuccessful. This might have occurred during the processing of an ALTER SECURITY command, a REFRESH SECURITY command, or during the automatic security timeout processing.

Note: This could indicate a system-wide storage problem.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced. Register 2 contains the return code from the storage failure.

System programmer response

Use the items listed in "Diagnostics" on page 5035, together with information about any other storage-related problems, to diagnose the cause of the problem. If you are unable to resolve the problem, contact your IBM support center.

00C80029

A severe error has occurred during a SAF RACROUTE REQUEST=STAT call to the external security manager (ESM) during security switch processing for a REFRESH SECURITY command.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced. Register 2 contains the address of the class being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. Check your security configuration (for example, that the required classes are installed and active). If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80031

A severe error has occurred during a SAF RACROUTE REQUEST=LIST (create) call to the external security manager (ESM) during the processing for a REFRESH SECURITY command.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced. Register 2 contains the address of the class, and register 3 the address of the entity, being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. Check your security configuration (for example, that the required classes are installed and active). If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80032

An unexpected return code has been received from one of the following SAF calls to the external security manager (ESM) during the processing of a REFRESH SECURITY command:

- RACROUTE REQUEST=LIST (create)
- RACROUTE REQUEST=LIST (delete)
- RACROUTE REQUEST=STAT

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced. Register 2 contains the address of the return codes from SAF, and the ESM.

Note: If the error occurred on a STAT call, the error is preceded by a CSQH004I message containing the return codes from SAF, and the ESM.

System programmer response

See your ESM documentation for information about the return codes from SAF and the ESM. For information about setting IBM MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80033

An unexpected setting for the subsystem security switch was encountered during the processing of a REFRESH SECURITY command.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced.

System programmer response

Collect the items listed in "Diagnostics" on page 5035, together with a note of what you expected the switch to be set to, and whether you had defined a profile for it or not, and contact your IBM support center.

00C80034

An internal error has occurred.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced. Register 2 contains the address of the class invoked at the time of the check.

System programmer response

Collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80035

A severe error has occurred during a SAF RACROUTE REQUEST=STAT call to the external security manager (ESM) during security switch processing for a REFRESH SECURITY command.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced. Register 2 contains the address of the class being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. Check your security configuration (for example, that the required classes are installed and active). If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80036

A severe error has occurred during a SAF RACROUTE REQUEST=EXTRACT call to the external security manager (ESM) during security switch processing for a REFRESH SECURITY command.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced. Register 2 contains the address of the entity being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. For information about setting IBM MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

00C80037

A severe error has occurred during a SAF RACROUTE REQUEST=LIST (create) call to the external security manager (ESM) during the processing for a REFRESH SECURITY command.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced. Register 2 contains the address of the class, and register 3 the address of the entity, being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

00C80038

An unexpected return code has been received from one of the following SAF calls to the external security manager (ESM) during the processing of a REFRESH SECURITY command.

- RACROUTE REQUEST=LIST (create)
- RACROUTE REQUEST=LIST (delete)
- RACROUTE REQUEST=EXTRACT
- RACROUTE REQUEST=STAT

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced. Register 2 contains the address of the return codes from SAF, and the ESM.

Note: If the error occurred on a STAT call, the error is preceded by a CSQH004I message containing the return codes from SAF, and the ESM.

System programmer response

See your ESM documentation for information about the return codes from SAF and the ESM. For information about setting IBM MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

00C80039

An attempt to obtain a storage subpool for a security manager user entry block was unsuccessful. This could have occurred during either security timeout processing, or REFRESH SECURITY command processing.

Note: This could indicate a system-wide storage problem.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced. Register 2 contains the return code from the storage failure.

System programmer response

Use the items listed in “Diagnostics” on page 5035, together with information about any other storage-related problems, to diagnose the cause of the problem. If you are unable to resolve the problem, contact your IBM support center.

00C80040

A severe error has occurred during security timeout processing. An unexpected return code has been received from the IBM MQ timer component.

Note: This could indicate a system-wide problem with the timer component, or the system timer.

System action

Messages CSQH009I and CSQH010I are issued. The current execution unit terminates with a completion code of X'5C6', and a dump is produced. Register 2 contains the return code from the timer component that caused the problem.

System programmer response

Use the items listed in "Diagnostics" on page 5035, together with information about any other timer-related problems, to diagnose the cause of the problem. If you are unable to resolve the problem, contact your IBM support center.

00C80041

A severe error has occurred during security timeout processing for an ALTER SECURITY command. An unexpected return code has been received from the IBM MQ timer component.

Note: This could indicate a system-wide problem with the timer component, or the system timer.

System action

Message CSQH010I is issued. The current execution unit terminates with a completion code of X'5C6' and a dump is produced. Register 2 contains the return code from the timer component that caused the problem.

System programmer response

Use the items listed in "Diagnostics" on page 5035, together with information about any other timer-related problems, to diagnose the cause of the problem. If you are unable to resolve the problem, contact your IBM support center.

00C80042

A severe error has occurred during security initialization when trying to start the security timer. An unexpected return code has been received from the IBM MQ timer component.

Note: This could indicate a system-wide problem with the timer component, or the system timer.

System action

Message CSQH010I is issued. The queue manager terminates and a dump is produced. Register 2 contains the return code from the timer component that caused the problem.

System programmer response

Use the items listed in "Diagnostics" on page 5035, together with information about any other timer-related problems, to diagnose the cause of the problem. If you are unable to resolve the problem, contact your IBM support center.

00C80043

A severe error has occurred whilst processing a DISPLAY SECURITY command. A parameter has been entered on the SECURITY keyword, but this is invalid.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

00C80044

A severe error has occurred whilst processing an ALTER SECURITY command. A parameter has been entered on the SECURITY keyword, but this is invalid.

System action

The current execution unit terminates with a completion code of X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

00C80045

A severe error has occurred because the last security refresh did not complete successfully.

System action

The current execution unit terminates with error reason code X'5C6', and a dump is produced.

System programmer response

If you are able to fix the cause of the problem, you must refresh the security again before you can continue. If you are unable to solve the problem, collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

00C80046

An attempt to obtain a storage subpool for the security manager Utoken blocks was unsuccessful.

This indicates that there could be a wider ranging problem relating to storage availability.

System action

The queue manager is terminated and a dump is produced.

System programmer response

Use the items listed in “Diagnostics” on page 5035, together with information about any other storage-related problems, to diagnose the cause of the problem.

00C80047

An attempt to obtain a storage block for a security manager Utoken block was unsuccessful.

This indicates that there could be a wider ranging problem relating to storage availability.

System action

The current execution unit terminates with X'5C6' and a dump is produced.

System programmer response

Use the items listed in “Diagnostics” on page 5035, together with information about any other storage-related problems, to diagnose the cause of the problem. Contact your IBM support center if you need help.

00C80050

A severe error has occurred during a SAF RACROUTE REQUEST=STAT call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. Check your security configuration (for example, that the required classes are installed and active). If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

00C80051

A severe error has occurred during a SAF RACROUTE REQUEST=EXTRACT call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the entity being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. For information about setting IBM MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

00C80052

A severe error has occurred during a SAF RACROUTE REQUEST=LIST (create) call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class, and register 3 the address of the entity, being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

00C80053

An unexpected return code has been received from one of the following SAF calls to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

- RACROUTE REQUEST=EXTRACT
- RACROUTE REQUEST=LIST
- RACROUTE REQUEST=STAT

System action

Message CSQH004I is produced containing the return codes from SAF and the ESM. The queue manager is terminated, and a dump is produced. Register 2 contains the address of the return codes.

System programmer response

See your ESM documentation for information about the return codes that appear in message CSQH004I (in the job log) or the dump. For information about setting IBM MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

00C80054

An unexpected setting for the subsystem security switch was encountered.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the control block containing the switch setting.

System programmer response

Collect the items listed in “Diagnostics” on page 5035, together with a note of what you expected the switch to be set to, and whether you had defined a profile for it or not, and contact your IBM support center.

Restart the queue manager.

00C80055

An internal loop count was exceeded during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class being checked at the time of the error.

System programmer response

Collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

Restart the queue manager.

00C80060

A severe error has occurred during a SAF RACROUTE REQUEST=STAT call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. Check your security configuration (for example, that the required classes are installed and active). If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

00C80061

A severe error has occurred during a SAF RACROUTE REQUEST=EXTRACT call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the entity being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. For information about setting IBM MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

00C80062

A severe error has occurred during a SAF RACROUTE REQUEST=LIST (create) call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class, and register 3 the address of the entity, being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80063

An unexpected return code has been received from one of the following SAF calls to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

- RACROUTE REQUEST=EXTRACT
- RACROUTE REQUEST=LIST
- RACROUTE REQUEST=STAT

System action

Message CSQH004I is produced containing the return codes from SAF and the ESM. The queue manager is terminated, and a dump is produced. Register 2 contains the address of the return codes.

System programmer response

See your ESM documentation for information about the return codes that appear in message CSQH004I (in the job log) or the dump. For information about setting IBM MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80064

An unexpected setting for the subsystem security switch was encountered.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the control block containing the switch setting.

System programmer response

Collect the items listed in "Diagnostics" on page 5035, together with a note of what you expected the switch to be set to, and whether you had defined a profile for it or not, and contact your IBM support center.

Restart the queue manager.

00C80065

An internal loop count was exceeded during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class being checked at the time of the error.

System programmer response

Collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

Restart the queue manager.

00C80070

A severe error has occurred during a SAF RACROUTE REQUEST=STAT call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. Check your security configuration (for example, that the required classes are installed and active). If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80071

A severe error has occurred during a SAF RACROUTE REQUEST=EXTRACT call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the entity being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. For information about setting IBM MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80072

A severe error has occurred during a SAF RACROUTE REQUEST=LIST (create) call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class, and register 3 the address of the entity, being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80073

An unexpected return code has been received from one of the following SAF calls to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

- RACROUTE REQUEST=EXTRACT
- RACROUTE REQUEST=LIST
- RACROUTE REQUEST=STAT

System action

Message CSQH004I is produced containing the return codes from SAF and the ESM. The queue manager is terminated, and a dump is produced. Register 2 contains the address of the return codes.

System programmer response

See your ESM documentation for information about the return codes that appear in message CSQH004I (in the job log) or the dump. For information about setting IBM MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80074

An unexpected setting for the subsystem security switch was encountered.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the control block containing the switch setting.

System programmer response

Collect the items listed in "Diagnostics" on page 5035, together with a note of what you expected the switch to be set to, and whether you had defined a profile for it or not, and contact your IBM support center.

00C80075

An internal loop count was exceeded during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class being checked at the time of the error.

System programmer response

Collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

Restart the queue manager.

00C80080

A severe error has occurred during a SAF RACROUTE REQUEST=STAT call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. Check your security configuration (for example, that the required classes are installed and active). If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80081

A severe error has occurred during a SAF RACROUTE REQUEST=EXTRACT call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the entity being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. For information about setting IBM MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80082

A severe error has occurred during a SAF RACROUTE REQUEST=LIST (create) call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class, and register 3 the address of the entity, being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80083

An unexpected return code has been received from one of the following SAF calls to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

- RACROUTE REQUEST=EXTRACT
- RACROUTE REQUEST=LIST
- RACROUTE REQUEST=STAT

System action

Message CSQH004I is produced containing the return codes from SAF and the ESM. The queue manager is terminated, and a dump is produced. Register 2 contains the address of the return codes.

System programmer response

See your ESM documentation for information about the return codes that appear in message CSQH004I (in the job log) or the dump. For information about setting IBM MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80084

An unexpected setting for the subsystem security switch was encountered.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the control block containing the switch setting.

System programmer response

Collect the items listed in "Diagnostics" on page 5035, together with a note of what you expected the switch to be set to, and whether you had defined a profile for it or not, and contact your IBM support center.

00C80090

A severe error has occurred during a SAF RACROUTE REQUEST=STAT call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. Check your security configuration (for example, that the required classes are installed and active). If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

00C80091

A severe error has occurred during a SAF RACROUTE REQUEST=EXTRACT call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the entity being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. For information about setting IBM MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

00C80092

A severe error has occurred during a SAF RACROUTE REQUEST=LIST (create) call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class, and register 3 the address of the entity, being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

00C80093

An unexpected return code has been received from one of the following SAF calls to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

- RACROUTE REQUEST=EXTRACT
- RACROUTE REQUEST=LIST
- RACROUTE REQUEST=STAT

System action

Message CSQH004I is produced containing the return codes from SAF and the ESM. The queue manager is terminated, and a dump is produced. Register 2 contains the address of the return codes.

System programmer response

See your ESM documentation for information about the return codes that appear in message CSQH004I (in the job log) or the dump. For information about setting IBM MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

00C80094

An unexpected setting for the subsystem security switch was encountered.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the control block containing the switch setting.

System programmer response

Collect the items listed in "Diagnostics" on page 5035, together with a note of what you expected the switch to be set to, and whether you had defined a profile for it or not, and contact your IBM support center.

Restart the queue manager.

00C80095

An internal loop count was exceeded during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class being checked at the time of the error.

System programmer response

Collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

Restart the queue manager.

00C80100

A severe error has occurred during a SAF RACROUTE REQUEST=STAT call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. Check your security configuration (for example, that the required classes are installed and active). If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80101

A severe error has occurred during a SAF RACROUTE REQUEST=EXTRACT call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the entity being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. For information about setting IBM MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80102

A severe error has occurred during a SAF RACROUTE REQUEST=LIST (create) call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class, and register 3 the address of the entity, being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80103

An unexpected return code has been received from one of the following SAF calls to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

- RACROUTE REQUEST=EXTRACT
- RACROUTE REQUEST=LIST
- RACROUTE REQUEST=STAT

System action

Message CSQH004I is produced containing the return codes from SAF and the ESM. The queue manager is terminated, and a dump is produced. Register 2 contains the address of the return codes.

System programmer response

See your ESM documentation for information about the return codes that appear in message CSQH004I (in the job log) or the dump. For information about setting IBM MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80104

An unexpected setting for the subsystem security switch was encountered.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the control block containing the switch setting.

System programmer response

Collect the items listed in "Diagnostics" on page 5035, together with a note of what you expected the switch to be set to, and whether you had defined a profile for it or not, and contact your IBM support center.

Restart the queue manager.

00C80105

An internal loop count was exceeded during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class being checked at the time of the error.

System programmer response

Collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

Restart the queue manager.

00C80200

A severe error has occurred during a SAF RACROUTE REQUEST=STAT call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. Check your security configuration (for example, that the required classes are installed and active). If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80201

A severe error has occurred during a SAF RACROUTE REQUEST=EXTRACT call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the entity being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. For information about setting IBM MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80202

A severe error has occurred during a SAF RACROUTE REQUEST=LIST (create) call to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class, and register 3 the address of the entity, being checked at the time of the error.

System programmer response

See your ESM documentation for information about any return codes that appear in the job log. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80203

An unexpected return code has been received from one of the following SAF calls to the External Security Manager (ESM) during security switch processing at queue manager initialization time.

- RACROUTE REQUEST=EXTRACT
- RACROUTE REQUEST=LIST
- RACROUTE REQUEST=STAT

System action

Message CSQH004I is produced containing the return codes from SAF and the ESM. The queue manager is terminated, and a dump is produced. Register 2 contains the address of the return codes.

System programmer response

See your ESM documentation for information about the return codes that appear in message CSQH004I (in the job log) or the dump. For information about setting MQ security switches, see Switch profiles. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5035 and contact your IBM support center.

00C80204

An unexpected setting for the subsystem security switch was encountered.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the control block containing the switch setting.

System programmer response

Collect the items listed in “Diagnostics” on page 5035, together with a note of what you expected the switch to be set to, and whether you had defined a profile for it or not, and contact your IBM support center.

Restart the queue manager.

00C80205

An internal loop count was exceeded during security switch processing at queue manager initialization time.

System action

The queue manager is terminated, and a dump is produced. Register 2 contains the address of the class being checked at the time of the error.

System programmer response

Collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

Restart the queue manager.

00C80206

An unexpected setting for the request type was encountered on an authentication request.

System action

The current execution unit terminates with a completion code of X'5C6' and a dump is produced. Register 2 contains the request type in error.

System programmer response

Collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

Restart the queue manager.

00C80207

An unexpected setting for the request type was encountered on an authentication request.

System action

The queue manager terminates and a dump is produced. Register 2 contains the request type in error.

System programmer response

Collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

Restart the queue manager.

00C81000

A severe error has occurred while processing a REFRESH SECURITY command.

System action

The current execution unit terminates with error reason code X'5C6', and a dump is produced. Register 2 contains the address of the control block involved in the error.

System programmer response

Collect the items listed in “Diagnostics” on page 5035 and contact your IBM support center.

Data manager codes (X'C9'): z/OS

If a data manager reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- A description of the actions that led to the error or, if applicable, either a listing of the application program or the input string to a utility program that was being run at the time of the error.
- The console output for the period leading up to the error.
- The queue manager job log.
- The system dump resulting from the error.
- If you are using CICS, the CICS transaction dump output.
- Appropriate IBM MQ, z/OS, Db2, CICS, and IMS service levels.
- If you are using the IBM MQ Operations and Control panels, the ISPF panel name.

00C90100

The object IBM MQ was trying to create was too large to be stored.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” and contact your IBM support center.

00C90200

A page set page retrieved was not valid.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” and contact your IBM support center.

00C90201

A page set page retrieved was not valid. The page was not a header page.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” and contact your IBM support center.

00C90202

A page set page retrieved was not valid. The page was not a data page.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” and contact your IBM support center.

00C90300

MQ was unable to start a unit of recovery for this execution unit.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C90301

An internal logging error has occurred for the current execution unit.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C90400

The data manager has detected in invalid log record.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C90401

The data manager has detected an invalid log record subtype.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C90500

The data manager was asked to make a change to some data in a page, but the change would have extended beyond the specific data item.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C90600

The data manager was unable to locate a specific logical record within a data page. The record was required for an update, or to insert a new record immediately after.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C90700

The data manager was unable to locate its *resource access list entry* (RALE).

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C90800

The data manager was requested to put a message on a queue, but told to give the message an invalid priority.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C90900

The data manager was asked to retrieve a logical record from a page, but on retrieving it discovered that the record is invalid.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C90A00

The data manager was asked to carry out a value logging operation with an invalid length field.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C90B00

The space reclamation routines have been asked to deallocate a page that is not in a state to be deallocated.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C90C00

An object type description passed to the data manager is not valid.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C90D00

A page set that was originally page set n is being presented as being a different page set, probably because the started task JCL procedure for the queue manager has been changed. Register 0 contains the identifier of the page set in error, and register 2 contains the identifier it was previously associated with.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Check the started task JCL procedure for the queue manager, and undo any changes to the CSQPnnnn DD statements that specify the page sets. Restart the queue manager. If the problem persists, or no changes have been made to these statements, collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C90D01

Your data set is not recognized as an IBM MQ page set. This may be for one of the following reasons.

- The data set has not been formatted
- You are attempting to start the queue manager with an older version of IBM MQ and the backward migration PTF for that release is not installed
- You are attempting to start the queue manager with an older version of IBM MQ, but the queue manager had been running with OPMODE=NEWFUNC at the higher release, and fallback to an earlier release is not possible

Register 0 contains the identifier of the page set in error.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Investigate the reason code and take one of the following actions:

- format the page set
- install appropriate backward migration PTFs
- start the queue manager with the correct level of code

00C90D02

This reason code is caused by one of the following:

- An attempt to use a page set that is a valid IBM MQ page set, but does not belong to this queue manager
- An attempt to change the subsystem name

Neither of these actions is allowed.

Register 0 contains the identifier of the page set in error.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

If you were attempting to use a page set from another queue manager, correct the error. Do not attempt to change the name of your queue manager.

00C90D03

An internal error has occurred during processing of an MQGET call with the Mark Skip Backout option.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C90D04

During restart, the queue manager detected that a page set has been truncated. This is probably because the data set allocated during restoration of a backup was smaller than required to hold the backed up data, and so the data has been truncated. It might also occur if page set 0 is larger than the maximum supported page set size.

System action

The identifier of the page set in error is put in register 0. Restart is terminated.

System programmer response

Reallocate the data set correctly, restore the backed up data if necessary, and restart the queue manager.

00C90E00

The data manager was passed an invalid parameter describing the location of a logical record within a data page and page set.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C90F00

The data manager was requested to update a logical record within a page, but the record had been deleted previously.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C91000

The data manager was asked to retrieve a message from an object that was not a local queue.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C91094, 00C91095, 00C91096, 00C91097

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C91101

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C91102

MQ received a return code indicating an error from the RRS ATRSROI service.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

The return code from ATRSROI is in register 3. See the *MVS Programming: Resource Recovery* manual for information about the return code.

00C91104

The data manager was requested to carry out a browse message operation, but the required lock was not held.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C91200

The internal data manager locate-object routine could not find the object it was seeking during UNDO processing.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C91300

During queue manager startup, an attempt was made to recover an object, the length of which exceeds a single data page. However, one of the intermediate data pages was not available, and IBM MQ was unable to recover the object.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C91400

The data manager was unable to access the header page (page 0) of one of the page sets.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced. The number of the page set with a header page that was unreadable is held in register 2.

System programmer response

1. Check for a preceding IEC161I, CSQP060E, or CSQP011E message relating to page set mentioned in register 2.
2. Check the following:
   - For the page set mentioned in register 2, is the appropriate CSQPnnnn DD statement present in the started task JCL procedure for the queue manager, xxxxMSTR?
   - Does this DD statement reference a genuine data set? DD DUMMY statements are not allowed for page sets.
   - Is DEFINE PSID(nn) present in the CSQINP1 initialization input data set?
3. If you are still unable to resolve the problem, collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C91500

During queue manager startup, the data manager was following a chain of objects on disk, and requested the next data page in the chain from the buffer manager. However, the buffer manager could not supply this page.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C91600

During restart, the data manager rebuilds its in-storage structures from page set data. On rebuilding an object, data manager discovered that the object already exists.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C91700, 00C91800

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C91900

During restart, data manager has detected an error in the rebuild of its in-storage object structures.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C91B01

During restart, the data manager found a queue with messages that are apparently located in a newly added page set. This is probably because the queue manager was run with a page set offline, and a new page set was formatted to replace the original one. This will lead to data loss.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C91C00

A delete purge request has been issued but the object type is not a local queue.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C91D00

A lock request has failed during an attempt to lock all pages associated with a long catalog object, or a long message.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C91E00

During a request issued by CSQIPUT5 or CSQIPUT6, an attempt to obtain a page level lock was unsuccessful.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C91F00

During a request issued by CSQIPUT5 or CSQIPUT6, an attempt to obtain a record level lock was unsuccessful.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C92000

An attempt to obtain a page level lock on the owner page relating to an object or message was unsuccessful.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C92100

An attempt to obtain a page level lock while trying to insert data was unsuccessful.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C92200

An attempt to obtain a record level lock while trying to insert data was unsuccessful.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C92300

An attempt to obtain a record level lock while trying to amend data was unsuccessful.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C92400

An attempt to get a lock on object type concatenated with object name within CSQIMGE1 was unsuccessful.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C92500, 00C92600, 00C92700, 00C92800, 00C92900, 00C92A00, 00C92B00, 00C92C00, 00C92D00, 00C92E00, 00C92F00, 00C93000

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C93100

A keyed read queue has encountered an error. A problem has occurred in the hash-table structure for the queue.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C93200, 00C93300

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C93500

IBM MQ was extending a page set at startup, based on log records from earlier dynamic page set extend operations. (IBM MQ does this so that any media recovery operation will have the required number of pages available in the page set.)

The page set could not be extended to the required RBA value.

The contents of the relevant registers are as follows:

R0 The number of the page set that could no longer be extended

R2 The logged page number that IBM MQ was trying to extend to

R3 The high page number at restart. This is the base from which IBM MQ was extending.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Create a larger page set, using multiple disk volumes if required, with a larger secondary extent value. The high page number of the page set should at least match that shown in register 2 in the dump.

00C93700

A queue contains messages, but the storage class named in the queue definition does not exist. This is an error.

This reason code is issued on queue manager restart if it is **not** the first time the queue manager has been started after migration to a new version.

Register 2 contains the first 4 characters of the storage class name, and register 3 contains characters 5 through 8.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the dump and a listing of your page set 0 and contact your IBM support center.

00C93800

A queue contains messages, which are on a page set other than that defined by the storage class named by the queue.

This reason code is issued on queue manager restart if it is **not** the first time the queue manager has been started after migration to a new version. It is preceded by one or more instances of message CSQI028E.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the dump and a listing of your page set 0 and contact your IBM support center.

00C93900

During MQPUT processing, IBM MQ was unable to acquire a lock on the storage class of the queue.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C93A00

During MQGET processing, IBM MQ was unable to acquire a lock on the queue it was processing.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C93B00

During MQPUT processing, IBM MQ was unable to acquire a lock on the queue it was processing.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C93C00

During MQGET processing, IBM MQ was unable to retrieve a message page from a queue it was processing.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C93D00, 00C93E00, 00C93F00, 00C94000, 00C94100

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C94200

MQ received a return code indicating an error from the RRS ATREINT service. This can occur if RRS is stopped when running an IBM MQ application linked with an RRS stub.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

The return code from ATREINT is in register 3. See the *MVS Programming: Resource Recovery* manual for information about the return code.

00C94300

MQ received a return code indicating an error from the RRS ATRSIT service.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

The return code from ATRSIT is in register 3. See the *MVS Programming: Resource Recovery* manual for information about the return code.

00C94400

MQ received a return code indicating an error from the RRS ATRSPID service.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

The return code from ATRSPID is in register 3. See the *MVS Programming: Resource Recovery* manual for information about the return code.

00C94500, 00C94501, 00C94502

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C94503

A page set that has been the subject of the RESETPAGE function had not previously been through a clean shutdown of the queue manager. Using this page set for subsequent IBM MQ processing would lead to data integrity problems.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Check the page sets that are defined to the queue manager. One or more of the page sets has been the subject of a RESETPAGE operation. Do not run the RESETPAGE operation against page sets that are either of the following:

- Fuzzy page set backups
- From a queue manager that has terminated abnormally

If you are unable to solve the problem, collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C94505

An internal error has occurred.

An attempt to restart with a log from another queue manager was detected. The queue-sharing group name recorded in the log during checkpoint does not match the name of the queue-sharing group in the queue manager using that log. If the correct log is being used, you can perform the change only after a clean shutdown of the queue manager, that is, after a quiesce.

Message CSQI060E is issued before this error occurs.

System action

Restart is terminated abnormally with completion code X'5C6' and a dump is produced.

System programmer response

Restart the queue manager using the correct logs and BSDS, or change the QSGDATA system parameter. Note that you cannot change the name of the queue-sharing group that a queue manager uses unless it has been shut down normally.

The following registers in the dump contain helpful values:

- R0 = the queue-sharing group name recorded in the log
- R2 = the queue-sharing group name in the running queue manager

00C94506

An internal error has occurred.

An attempt to restart with a log from another queue manager was detected. The shared queue manager identifier recorded in the log during checkpoint does not match the shared queue manager identifier in the queue manager using that log. If the correct log is being used, the entry in the Db2 CSQ.ADMIN_B_QMGR table for this queue manager has been corrupted.

Message CSQI061E is issued before this error occurs.

System action

Restart is terminated abnormally with completion code X'5C6' and a dump is produced.

System programmer response

Restart the queue manager using the correct logs and BSDS. If the correct logs are being used, correct the entry for the queue manager in the Db2 CSQ.ADMIN_B_QMGR table. If you cannot resolve the problem, contact your IBM support center for assistance.

The following registers in the dump contain helpful values:

- R0 = the queue manager identifier recorded in the log
- R2 = the queue manager identifier in the running queue manager

00C94507

An internal error has occurred during processing of Mark Skip Backout.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C94510

A request was made to a coupling facility resource manager service within IBM MQ. The coupling facility resource manager service returned an unexpected return code.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C94511

An attempt to obtain storage for the data manager's use was unsuccessful. This indicates there could be a wider-ranging problem relating to storage availability.

System action

The queue manager is terminated and a dump is produced.

System programmer response

Check that you are running with the recommended region size, and if not, reset your system and restart the queue manager. If this is not the cause, use these items to diagnose the cause of the problem:

- Queue manager job log
- Information about any other storage-related problems
- System dump resulting from the error

00C94512

A request was made to a Db2 resource manager service within IBM MQ. The Db2 resource manager service returned an unexpected return code.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C94513

A request was made to a coupling facility resource manager service within IBM MQ. The coupling facility resource manager service returned an unexpected return code.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C9451A

A request was made to a Db2 resource manager service within IBM MQ during restart. The Db2 resource manager service returned an unexpected return code related to a locked table condition.

System action

The queue manager terminates with completion code X'5C6', and a dump is produced.

System programmer response

Restart the queue manager. If you started several queue managers at the same time, try restarting them one at a time to alleviate this condition.

If the problem persists, collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

00C9FEEE

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5056 and contact your IBM support center.

Recovery log manager codes (X'D1'): z/OS

If a recovery log manager reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- The console output for the period leading up to the error.
- The system dump resulting from the error.
- If you are using CICS, the CICS transaction dump output.
- Appropriate IBM MQ, z/OS, Db2, CICS, and IMS service levels.
- A printout of SYS1.LOGREC, if the reason code is issued by an active queue manager.
- If the reason code is issued by an active queue manager, a CSQ1LOGP detail report containing the log records associated with the problem.
- Contents of the BSDS. Obtain a listing by running the Print Log Map utility (CSQJU004).
- The recovery log manager standard diagnostic information, which is provided in the SYS1.LOGREC variable recording area (VRA) of the system diagnostic work area (SDWA) for many of the reason codes:

  MODID
  Name of the module issuing the error.

  LEVEL
  Change level.

  COMPONENT
  Subcomponent identifier of the recovery log manager.

  REGISTERS
  General purpose registers (GPRs) 0-15 at time of the abend.

00D10010

The end log range value specified on an invocation of the log print utility (CSQ1LOGP) is less than or equal to the start range value.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Correct the log range input control parameters specified in the invocation of the log print utility.

For more information about log services, refer to CSQ1LOGP.

00D10011

An invocation of the log print utility (CSQ1LOGP) was unable to obtain the storage required to perform the request.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

It is probable that the REGION parameter on the EXEC statement of the job control language (JCL) for this invocation is too small. Increase the REGION size, and resubmit the log print request.

For more information about log services, refer to Address space storage.

00D10012

An invocation of the log print utility (CSQ1LOGP) was unsuccessful because the job control language (JCL) for this invocation did not specify either the use of the bootstrap data set (BSDS) or, in the absence of the BSDS, the active or archive log data sets.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Correct the JCL and resubmit the log print request.

For more information about BSDS, refer to Managing the BSDS.

00D10013

An invocation of the log print utility (CSQ1LOGP) resulted in a VSAM error while attempting to open the bootstrap data set (BSDS).

This reason code, and the VSAM return code are issued with message CSQ1221E.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Refer to the *DFSMS/MVS Macro Instructions for Data Sets* to determine the meaning of the VSAM OPEN error. Take appropriate action, and resubmit the log print request.

00D10014

The job control language (JCL) for an invocation of the log print utility (CSQ1LOGP) specified the use of the bootstrap data set (BSDS), but the utility control statements did not specify values for RBASTART and RBAEND.

The RBASTART and RBAEND values must be specified when using the BSDS, although they are not required when using the active or archive logs.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Either:

- Continue to use the BSDS, but change the utility control statements to specify values for RBASTART and RBAEND

- Change the JCL to use the active and archive data sets instead

For more information, refer to CSQ1LOGP.

00D10015

An invocation of the log print utility (CSQ1LOGP) was unsuccessful because the record format of the bootstrap data set is incompatible with this release of the log print services.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Ensure that the correct release of the log print services are used with the appropriate BSDS record format.

For more information, refer to CSQ1LOGP.

00D10019

An invocation of the log print utility (CSQ1LOGP) resulted in a VSAM error while attempting to open the bootstrap data set (BSDS). The error was determined to be one which could be corrected by use of a VSAM access method services (AMS) VERIFY call, but the VERIFY call was also unsuccessful.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Collect the following items, and contact your IBM support center:

- A copy of the user's job control language (JCL) that was used to invoke the log print utility (CSQ1LOGP)
- The log data sets that the user was attempting to print

00D10020

The log print utility (CSQ1LOGP) issued this message because the end of data has been reached (that is, the end of the log, or the end of the user-specified data sets, or the user-specified RBAEND value has been reached).

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

This is not an error. This reason code denotes a normal end of data condition. No action is necessary.

For more information, refer to CSQ1LOGP.

00D10021

An invocation of the log print utility (CSQ1LOGP) encountered a gap in the log RBA range when switching log data sets. This indicates that log records might be missing.

Normally, a continuous set of log records is supplied as input by the ACTIVE and ARCHIVE DDnames (or the BSDS DDname if you are using the bootstrap data set (BSDS) to access the log data sets) in the job control language (JCL) used to invoke the utility. If a log data set was removed from the JCL, this condition will arise.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

If the log data set was not removed intentionally, check the JCL to ensure that the log data sets are specified in ascending RBA value order. If you are using the BSDS to access the log data sets, use the print log map utility (CSQJU004) to examine the RBA ranges as recorded in the BSDS, and note any RBA gaps that might have resulted from the deletion of an active or archive log data set.

If it appears that a log error might have occurred, see Active log problems for information about dealing with problems on the log.

00D10022

An invocation of the log print utility (CSQ1LOGP) encountered a gap in the log RBA range when switching log data sets. This indicates that log records might be missing. The log RBA of the next record following the gap is greater than the RBAEND value specified in the utility control statements.

Normally, a continuous set of log records is supplied as input by the ACTIVE and ARCHIVE DDnames (or the BSDS DDname if using the bootstrap data set (BSDS) to access the log data sets) in the job control language (JCL) used to invoke the utility. If a log data set was removed from the JCL, this condition will arise.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Check the JCL and the RBAEND value specified in the utility control statements.

If a log data set was not removed intentionally, check that the log data sets are specified in ascending RBA value order. If using the BSDS to access log data sets, use the print log map utility (CSQJU004) to examine the RBA ranges as recorded in the BSDS, and note any RBA gaps that might have resulted from the deletion of an active or archive log data set.

If it appears that a log error might have occurred, see Active log problems for information about dealing with problems on the log.

00D10024

An invocation of the log print utility (CSQ1LOGP) encountered a log RBA sequence error. The RBA of the previous log record is greater than the RBA of the current log record.

Normally, a continuous set of log records is supplied as input by the ACTIVE and ARCHIVE DDnames (or the BSDS DDname if using the bootstrap data set (BSDS) to access the log data sets) in the job control language (JCL) used to invoke the utility. If a log data set appears out of sequence, this condition will arise.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Check the JCL to ensure that the log data sets are specified in ascending RBA value order. If using the BSDS to access the log data sets, use the print log map utility (CSQJU004) to examine the RBA ranges associated with each archive and active log data set. If both archive and active log data sets are used, the first archive log data set must contain the lowest log RBA value. If necessary, adjust the concatenation of the archive and active log data sets in the JCL to ensure that log records are read in ascending RBA sequence, and resubmit the log print request.

If it appears that a log error might have occurred, see Active log problems for information about dealing with problems on the log.

00D10025

An invocation of the log print utility (CSQ1LOGP) resulted in a VSAM GET error while attempting to read the active log data set.

This reason code, and the VSAM return and reason codes are issued in message CSQ1221E.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Refer to the *DFSMS/MVS Macro Instructions for Data Sets* to determine the meaning of the VSAM GET error and the RPL error code. Take appropriate action to correct the error, and resubmit the log print request.

00D10026

An invocation of the log print utility (CSQ1LOGP) was unsuccessful because an RBA value within the range specified by RBASTART and RBAEND could not be located on a log data set.

This reason code, and the RBA value that could not be located are issued with message CSQ1216E

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Check the utility control statements to ensure that the RBASTART and RBAEND values have not exceeded the lower or upper bounds of the RBAs available on all the active or archive log data sets defined by DDnames in the JCL.

If you are using the BSDS to access the log data sets, use the print log map utility (CSQJU004) to examine the RBA ranges associated with each archive and active log data set.

Correct the JCL and utility control statements as necessary, and resubmit the log print request.

For more information, refer to CSQ1LOGP.

00D10027

An invocation of the log print utility (CSQ1LOGP) resulted in a VSAM GET error while attempting to read the bootstrap data set (BSDS).

This reason code, and the VSAM return and reason codes, are issued with message CSQ1221E.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Refer to the *DFSMS/MVS Macro Instructions for Data Sets* manual to determine the meaning of the VSAM GET error and the RPL error code. Take appropriate action to correct the error and resubmit the log print request.

00D1002A

An invocation of the log print utility (CSQ1LOGP) was unsuccessful because an RBA value has been requested in an active log data set that has previously not been opened. A VSAM OPEN error occurred while attempting to open the active log data set.

This reason code, and the VSAM return and reason codes, are issued in message CSQ1221E.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Refer to the *DFSMS/MVS Macro Instructions for Data Sets* manual to determine the meaning of the VSAM OPEN error and the ACB error code. Take appropriate action to correct the error, and resubmit the log print request.

00D1002B

An invocation of the log print utility (CSQ1LOGP) was unsuccessful because an RBA value has been requested in an active log data set that has previously not been opened. A VSAM OPEN error occurred while attempting to open the active log data set. The VSAM OPEN error was determined to be one that could be corrected, however, a system error occurred while executing a z/OS TESTCB macro to determine whether the active log data set in question was a VSAM ESDS (entry-sequenced data set) or a VSAM LDS (linear data set).

This reason code, and the VSAM return and reason codes are issued in message CSQ1221E.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Refer to the *DFSMS/MVS Macro Instructions for Data Sets* manual to determine the meaning of the VSAM OPEN error and the ACB error code. Take appropriate action to correct the error, and resubmit the log print request.

If the problem persists, collect the following items, and contact your IBM support center:

- A copy of the job control language (JCL) used to invoke the log print utility (CSQ1LOGP)
- The log data sets that the user was attempting to print

00D1002C

An invocation of the log print utility (CSQ1LOGP) was unsuccessful because an RBA value has been requested in a active log data set that has previously not been opened. A VSAM OPEN error occurred while attempting to open the active log data set. The VSAM OPEN error was determined to be one which could be corrected by use of a VSAM access method services (AMS) VERIFY call, but the VERIFY call was unsuccessful.

This reason code, and the VSAM return and reason codes are issued with message CSQ1221E.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Refer to the *DFSMS/MVS Macro Instructions for Data Sets* manual to determine the meaning of the VSAM OPEN error and the ACB error code. Take appropriate action to correct the error, and resubmit the log print request.

00D1002D

An invocation of the log print utility (CSQ1LOGP) was unsuccessful because an RBA value has been requested in an active log data set that has previously not been opened. A VSAM OPEN error occurred while attempting to open the active log data set. The VSAM OPEN error was corrected by use of a VSAM access method services (AMS) VERIFY call, but a subsequent attempt to reposition the VSAM pointer back to the beginning of the active log data set (using the VSAM AMS POINT call) was unsuccessful.

This reason code and the VSAM return and reason codes are issued with message CSQ1221E.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Refer to the *DFSMS/MVS Macro Instructions for Data Sets* manual to determine the meaning of the VSAM OPEN error and the ACB error code. Take the appropriate action to correct the error, and resubmit the print log request.

00D10030

An invocation of the log print utility resulted in an internal error.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Collect the following items, and contact your IBM support center:

- A copy of the job control language (JCL) used to invoke the log print utility
- The log data sets that the user was attempting to print

00D10031

An invocation of the log print utility (CSQ1LOGP) was unsuccessful because an RBA value has been requested in a log data set that has previously not been opened. The job control language (JCL) has specified that the bootstrap data set (BSDS) be used as the guide to determine which data sets are required. An attempt to allocate the appropriate data set dynamically (using z/OS SVC 99) was unsuccessful.

This reason code, and the dynamic allocation information and error codes (S99INFO and S99ERROR) are issued with message CSQ1222E.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Refer to the *MVS Authorized Assembler Services Guide* manual to determine the meaning of the SVC 99 information and error codes. Take the appropriate action to correct the error, and resubmit the log print request.

00D10040

An invocation of the log print utility (CSQ1LOGP) was unsuccessful because an RBA value has been requested in an archive log data set (on tape) that has previously not been opened. An attempt was made to open the second file on the archive log tape (the first file normally contains the bootstrap data set) but this was unsuccessful because the archive log data set was not the second file on the archive log tape. The read job file control block (RDJFCB) macro was then invoked to attempt to change the data set sequence number from the default value of 2 to a value of 1, before attempting to open the second file again, but the macro invocation resulted in an error.

This reason code, and the RDJFCB return code are issued in message CSQ1223E.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Refer to the *MVS/ESA DFP System Programming Reference* manual to determine the meaning of the RDJFCB error code. Take the appropriate action to correct the error, and resubmit the log print request.

00D10044

An invocation of the log print utility (CSQ1LOGP) was unsuccessful because an RBA value has been requested in an archive log data set that has previously not been opened. An attempt to open the archive log data set resulted in a QSAM (queued sequential access method) error.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Check the console for messages indicating the cause of the QSAM error. Take the appropriate action to correct the error, and resubmit the log print request.

00D10048

An invocation of the log print utility (CSQ1LOGP) was unsuccessful because a QSAM (queued sequential access method) GET error occurred while reading an archive log data set.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Check the console for messages indicating the cause of the QSAM error. Take the appropriate action to correct the error, and resubmit the log print request.

00D10050

An invocation of the log print utility (CSQ1LOGP) was unsuccessful because the bootstrap data set (BSDS) was erroneously specified as one of the archive data sets in the job control language (JCL).

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set.

System programmer response

Examine the JCL, and remove the occurrence of the BSDS data set as one of the concatenated ARCHIVE data sets. Resubmit the log print request.

For more information, refer to Archive log problems.

00D10061

An invocation of the log print utility (CSQ1LOGP) succeeded, but an unexpected physical record length was encountered for the log record control interval (CI) for an active or archive log data set.

The data on the log data set might have been corrupted after it was written by IBM MQ. The data in the log data set might still be usable, but with caution.

The length of a log CI in an active log data set is expected to be 4096 bytes. The length of a log CI in an archive log data set is expected to be 4089 bytes.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set. The log print request has completed. This reason code is issued as a warning.

System programmer response

Ensure that the ACTIVE and ARCHIVE DDnames in the job control language (JCL) refer to active and archive logs correctly.

If the problem persists, collect the following items, and contact your IBM support center:

- A copy of the job control language (JCL) used to invoke the log print utility (CSQ1LOGP)

- The log data set that the user was trying to print

00D10062

An invocation of the log print utility (CSQ1LOGP) succeeded, but the first log record segment could not be found for a middle spanned log record segment.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set. The log print request has completed. This reason code is issued as a warning.

System programmer response

Several possibilities exist for the cause of this condition:

- The recovery log manager component of IBM MQ did not originally construct the log record header (LRH) properly
- The LRH for the log record segment was damaged after it was written by IBM MQ
- The application program continued to process after being informed about a gap in the log RBA values (reason code X'00D10021')

Determine if the LRH of the log record segment is truly in error by looking at the record segments directly preceding and after the record segment in question.

Take the appropriate action to correct the error, and resubmit the log print request. If the problem persists, collect the following items, and contact your IBM support center:

- A copy of the job control language (JCL) used to invoke the lop print utility (CSQ1LOGP)
- The log data set that the user was attempting to print

00D10063

An invocation of the log print utility (CSQ1LOGP) succeeded, but the first log record segment could not be found for a last spanned log record segment.

System action

No error is issued by log services, and no information is written to SYS1.LOGREC data set. The log print request has completed. This reason code is issued as a warning.

System programmer response

Several possibilities exist for the cause of this condition:

- The recovery log manager component of IBM MQ did not originally construct the log record header (LRH) properly
- The LRH for the log record segment was damaged after it was written by IBM MQ
- The application program continued to process after being informed about a gap in the log RBA values (reason code X'00D10021')

Determine if the LRH of the log record segment is truly in error by looking at the record segments directly before and after the record segment in question.

Take the appropriate action to correct the error, and resubmit the log print request. If the problem persists, collect the following items, and contact your IBM support center:

- A copy of the job control language (JCL) used to invoke the lop print utility (CSQ1LOGP)
- The log data set that the user was attempting to print

00D10114

IBM MQ failed to read or write member information in the queue-sharing group table, CSQ.ADMIN_B_QSG.

System action

Queue manager initialization terminates.

System programmer response

Investigate Db2 SQL errors reported in the queue manager job log immediately preceding this error, to determine the cause. It is most likely due to incorrect table setup, plans not bound or insufficient authority to execute DB2 plans.

00D10120

The BSDS version is not allowed by the installation and customization options chosen for IBM MQ. For example, the queue manager might have been migrated from a previous version, and is currently operating with OPMODE=COMPAT.

System action

Queue manager startup terminates.

System programmer response

Either enable the required functions using OPMODE, or if this is the first attempt to start the queue manager with a BSDS that has been converted to a new version, restore the BSDS to the original version.

00D10121

The BSDS is not valid. A non-valid BSDS is the result of a failure during a previous attempt to run the BSDS conversion utility.

System action

Queue manager startup terminates.

System programmer response

The procedure for running the BSDS conversion utility involves renaming the original BSDS. Restore the BSDS to the original pre-conversion copy by renaming the data sets, then try the conversion again.

When the conversion is successful, try the program that issued the error message again.

00D10122

The BSDS version is not supported by this release of IBM MQ.

System action

Queue manager startup, or the process that was accessing the BSDS, terminates.

System programmer response

Start the queue manager at a version of IBM MQ that supports the BSDS version.

You can determine the version of a BSDS by running the print log map utility (CSQJU004)

00D10250

An unrecoverable error occurred while updating either the BSDS or the z/OS catalog to reflect changes in active log data sets.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The queue manager then terminates abnormally.

System programmer response

Obtain the SYS1.LOGREC and SVC dump. Correct the error, and restart the queue manager.

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem. In addition, see the description of reason code X'00D10252' for details of the information recorded in the variable recording area (VRA) of the system diagnostic work area (SDWA).

Examine the console log for a CSQJxxxx message preceding this error to determine whether the error was a BSDS error or a z/OS catalog update error. If you cannot resolve the problem, contact your support center.

00D10251

An unrecoverable error occurred in the log buffer writer.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The queue manager then terminates abnormally.

System programmer response

Obtain the SYS1.LOGREC and the SVC dump. This error is usually caused by a previous error that was recorded on SYS1.LOGREC and produced an SVC dump. The SYS1.LOGREC entries and SVC dump should be examined to determine the primary error that occurred.

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem. In addition, see the description of reason code X'00D10252' for details of the information recorded in the variable recording area (VRA) of the system diagnostic work area (SDWA).

If you cannot resolve the problem, contact your support center.

00D10252

This reason code is used to define the format of the information recorded in the variable recording area (VRA) of the system diagnostic work area (SDWA).

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump.

System programmer response

Obtain the SYS1.LOGREC and SVC dump.

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem. In addition, the following information is contained in the VRA of the SDWA:

- Reason code X'00D10252' stored with VRA key 6.
- The log buffer writer recovery tracking area is stored with VRA key 10.

00D10253

An application program check occurred in an MVCP instruction that attempted to move a parameter list or other data from the caller's address space to the queue manager address space.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump.

System programmer response

Obtain the SYS1.LOGREC and SVC dump. You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem.

Examine the area from which data was to be moved. It might be in the wrong key, or the address might be the cause of the problem. The incorrect instruction has a DA opcode and indicates the registers showing address and length to be moved.

00D10254

An application program check occurred in an MVCS instruction that attempted to move data from the queue manager address space to the caller's address space.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump.

System programmer response

Obtain the SYS1.LOGREC and SVC dump. You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem.

Examine the area to which data was to be moved. It might be in the wrong key, or the address might be the cause of the problem. The incorrect instruction has a DB opcode and indicates the registers showing address and length to be moved.

00D10257

The log RBA has reached or exceeded the value FFF800000000 (if 6-byte log RBAs are in use) or FFFFFFC000000000 (if 8-byte log RBAs are in use). The queue manager is terminated because the log RBA range has reached a CRITICAL level where the available range is too small for the queue manager to continue.

System action

The queue manager terminates with reason code 00D10257.

System programmer response

You need to reset the logs before the queue manager can be restarted. If you do not perform this action, the queue manager will abend once again after the next log data set switch.

For information on how to reset the logs using the CSQUTIL utility program, see RESETPAGE.

If your queue manager is using 6-byte log RBAs, consider converting the queue manager to use 8-byte log RBAs. See Planning to increase the maximum addressable log range for further information.

00D10261

While scanning the records and record segments in a log control interval (CI), it was discovered that the forward record chain was broken. This condition is the result of an incorrect record length in the log record header of some record in the log CI.

System action

This reason code can be issued by an active queue manager as the log buffers are scanned before they are written to the active log, or by the IBM MQ log services GET processor as a CI is retrieved from a user-specified active or archive log data set.

If the reason code is issued by an active queue manager, a diagnostic record is written to SYS1.LOGREC, and an SVC dump is requested.

- If the error was detected by CSQJOFF1, the archiving of the active log data set is terminated and the faulty active log data set is marked 'stopped'
- If the error was detected by CSQJR005, message CSQJ012E is issued and the calling agent is terminated
- If the error was detected by CSQJW009, message CSQJ012E is issued and the queue manager is terminated
- If the error was detected by CSQJW107, the queue manager is terminated

If this reason code is issued as the result of IBM MQ log services GET processing, no error is issued and no information is written to the SYS1.LOGREC data set.

System programmer response

You might find the items listed in “Diagnostics” on page 5070 useful in resolving the problem. If you are unable to solve the problem, contact your IBM support center.

00D10262

While scanning a log control interval (CI), the offset to the last record or record segment in the CI was found to be incorrect.

System action

This reason code can be issued by an active queue manager as the log buffers are scanned before they are written to the active log, or by the IBM MQ log services GET processor as a CI is retrieved from a user-specified active or archive log data set.

If the reason code is issued by an active queue manager, a diagnostic record is written to SYS1.LOGREC, and an SVC dump is requested.

- If the error was detected by CSQJOFF1, the archiving of the active log data set is terminated and the faulty active log data set is marked 'stopped'
- If the error was detected by CSQJR005, message CSQJ012E is issued and the calling agent is terminated
- If the error was detected by CSQJW009, message CSQJ012E is issued and the queue manager is terminated
- If the error was detected by CSQJW107, the queue manager is terminated

If this reason code is issued as the result of IBM MQ log services GET processing, no error is issued, and no information is written to the SYS1.LOGREC data set.

System programmer response

You might find the items listed in “Diagnostics” on page 5070 useful in resolving the problem. If you are unable to solve the problem, contact your IBM support center.

00D10263

While scanning a log control interval (CI), the VSAM RDF/CIDF control information was found to be incorrect.

System action

This reason code can be issued by an active queue manager as the log buffers are scanned before they are written to the active log, or by the IBM MQ log services GET processor as a CI is retrieved from a user-specified active or archive log data set.

If the reason code is issued by an active queue manager, a diagnostic record is written to SYS1.LOGREC, and an SVC dump is requested.

- If the error was detected by CSQJOFF1, the archiving of the active log data set is terminated and the faulty active log data set is marked 'stopped'
- If the error was detected by CSQJR005, message CSQJ012E is issued and the calling agent is terminated
- If the error was detected by CSQJW009, message CSQJ012E is issued and the queue manager is terminated
- If the error was detected by CSQJW107, the queue manager is terminated

If this reason code is issued as the result of IBM MQ log services GET processing, no error is issued, and no information is written to the SYS1.LOGREC data set.

System programmer response

You might find the items listed in “Diagnostics” on page 5070 useful in resolving the problem. If you are unable to solve the problem, contact your IBM support center.

00D10264

While scanning a log control interval (CI), the beginning log RBA of the CI was not the expected RBA.

System action

This reason code can be issued by an active queue manager as the log buffers are scanned before they are written to the active log, or by the IBM MQ log services GET processor as a CI is retrieved from a user-specified active or archive log data set.

If the reason code is issued by an active queue manager, a diagnostic record is written to SYS1.LOGREC, and an SVC dump is requested.

- If the error was detected by CSQJOFF1, the archiving of the active log data set is terminated and the faulty active log data set is marked 'stopped'
- If the error was detected by CSQJR005, message CSQJ012E is issued and the calling agent is terminated
- If the error was detected by CSQJW009, message CSQJ012E is issued and the queue manager is terminated
- If the error was detected by CSQJW107, the queue manager is terminated

If this reason code is issued as the result of IBM MQ log services GET processing, no error is issued, and no information is written to the SYS1.LOGREC data set.

System programmer response

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem. If you are unable to solve the problem, contact your IBM support center.

00D10265

While scanning the records and record segments in a log control interval (CI), it was discovered that the backward record chain was broken. This condition is the result of an incorrect record length in the log record header of some record in the log CI.

System action

This reason code can be issued by an active queue manager as the log buffers are scanned before they are written to the active log, or by the IBM MQ log services GET processor as a CI is retrieved from a user-specified active or archive log data set.

If the reason code is issued by an active queue manager, a diagnostic record is written to SYS1.LOGREC, and an SVC dump is requested.

- If the error was detected by CSQJOFF1, the archiving of the active log data set is terminated
- If the error was detected by CSQJR005, message CSQJ012E is issued and the calling agent is terminated
- If the error was detected by CSQJW009, message CSQJ012E is issued and the queue manager is terminated
- If the error was detected by CSQJW107, the queue manager is terminated

If this reason code is issued as the result of IBM MQ log services GET processing, no error is issued, and no information is written to SYS1.LOGREC data set.

System programmer response

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem. If you are unable to solve the problem, contact your IBM support center.

00D10266

While scanning a log control interval (CI), a unit of recovery ID or LINK RBA in some record was found to be inconsistent with the beginning log RBA of the CI.

System action

This reason code can be issued by an active queue manager as the log buffers are scanned before they are written to the active log, or by the IBM MQ log services GET processor as a CI is retrieved from a user-specified active or archive log data set.

If the reason code is issued by an active queue manager, a diagnostic record is written to SYS1.LOGREC, and an SVC dump is requested.

- If the error was detected by CSQJOFF1, the archiving of the active log data set is terminated and the faulty active log data set is marked 'stopped'
- If the error was detected by CSQJR005, message CSQJ012E is issued and the calling agent is terminated
- If the error was detected by CSQJW009, message CSQJ012E is issued and the queue manager is terminated
- If the error was detected by CSQJW107, the queue manager is terminated

If this reason code is issued as the result of IBM MQ log services GET processing, no error is issued, and no information is written to SYS1.LOGREC data set.

System programmer response

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem. If you are unable to solve the problem, contact your IBM support center.

00D10267

While scanning a log control interval (CI), a middle or last spanned record segment was not the first segment contained in the log CI.

System action

This reason code can be issued by an active queue manager because the log buffers are scanned before they are written to the active log, or by the IBM MQ log services GET processor because a CI is retrieved from a user-specified active or archive log data set.

If the reason code is issued by an active queue manager, a diagnostic record is written to SYS1.LOGREC, and an SVC dump is requested.

- If the error was detected by CSQJOFF1, the archiving of the active log data set is terminated and the faulty active log data set is marked 'stopped'
- If the error was detected by CSQJR005, message CSQJ012E is issued and the calling agent is terminated
- If the error was detected by CSQJW009, message CSQJ012E is issued and the queue manager is terminated
- If the error was detected by CSQJW107, the queue manager is terminated

If this reason code is issued as the result of IBM MQ log services GET processing, no error is issued, and no information is written to the SYS1.LOGREC data set.

System programmer response

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem. If you are unable to solve the problem, contact your IBM support center.

00D10268

While scanning a log control interval (CI), a first or middle spanned record segment was not the last segment contained in the log CI.

System action

This reason code can be issued by an active queue manager as the log buffers are scanned before they are written to the active log, or by the IBM MQ log services GET processor as a CI is retrieved from a user-specified active or archive log data set.

If the reason code is issued by an active queue manager, then a diagnostic record is written to SYS1.LOGREC, and an SVC dump is requested.

- If the error was detected by CSQJOFF1, the archiving of the active log data set is terminated and the faulty active log data set is marked 'stopped'
- If the error was detected by CSQJR005, message CSQJ012E is issued and the calling agent is terminated
- If the error was detected by CSQJW009, message CSQJ012E is issued and the queue manager is terminated
- If the error was detected by CSQJW107, the queue manager is terminated

If this reason code is issued as the result of IBM MQ log services GET processing, no error is issued, and no information is written to the SYS1.LOGREC data set.

System programmer response

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem. If you are unable to solve the problem, contact your IBM support center.

00D10269

An unrecoverable error was found in one of the buffers, while moving the current log buffer to the static write buffer in preparation for the physical write to the active log.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The queue manager then terminates.

System programmer response

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem. If you are unable to solve the problem, contact your IBM support center.

00D10270

A LOG WRITE request completed unsuccessfully because the length of the log record header was not as expected. This is an internal error.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem.

Examine the SYS1.LOGREC, console log, and SVC dump for information about prior errors during LOG WRITE processing.

If you are unable to solve the problem, contact your IBM support center.

00D10327

A LOG READ completed unsuccessfully because of an invalid log LOGRBA. A log read, MODE(DIRECT) with a requested RBA does not match the start of a log record.

System action

An SVC dump is requested and the execution unit ends abnormally. If the log read error occurs during queue manager startup then the queue manager ends abnormally.

System programmer response

Log read with MODE(DIRECT) is most commonly used in the queue manager for verifying that the start RBA of a unit of work can be found on the log, before a sequential (maybe backward)

read of the log data to recover locks on an in-doubt unit of work, or to back out a unit of work. It indicates that the queue manager is being started with incomplete log data available.

If you suspect an error in IBM MQ, collect the following data and contact IBM support:

- The BSDS
- All active and archive logs
- The SVC dump created by this error

00D1032A

An unsuccessful completion of a LOG READ has occurred. BSDS does not map the specified RBA into a log data set. Either the BSDS is in error, or the log data set has been deleted.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem.

00D1032B

Completion of a LOG READ was unsuccessful, because an error occurred while attempting to allocate a log data set.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem.

Examine LOGREC and SVC dump information. Also, examine any prior messages with a CSQJ prefix from recovery log manager allocation processing.

00D1032C

A LOG READ completed unsuccessfully, because an error occurred while opening or closing a log data set.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem.

Examine LOGREC and SVC dump information. Also, examine prior messages from recovery log manager open/close processing. These messages have a prefix of CSQJ.

00D1032E

A LOG READ completed unsuccessfully due to an internal error.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem. Examine the SYS1.LOGREC and SVC dump information.

00D10340

An unsuccessful completion of a LOG READ has occurred. This reflects an internal recovery log manager (RLM) logic error.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem.

Examine the SYS1.LOGREC, console log and SVC dump for information about prior errors during LOG READ processing.

If you cannot solve the problem, contact your IBM support center.

00D10341

A LOG READ completed unsuccessfully because an error was detected during a Forward READ of the log record. This is an internal error.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem.

Examine the SYS1.LOGREC, console log and SVC dump for information about prior errors during LOG READ processing.

If you cannot solve the problem, contact your IBM support center.

00D10342

A LOG READ completed unsuccessfully because an error was detected during a backward READ of a log record. This is an internal error.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem.

Examine the SYS1.LOGREC, console log and SVC dump for information about prior errors during LOG READ processing.

If you cannot solve the problem, contact your IBM support center.

00D10343

A LOG READ completed unsuccessfully because an error was detected during a READ of a log record due to an invalid CI offset. This is an internal error.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem.

Examine the SYS1.LOGREC, console log and SVC dump for information about prior errors during LOG READ processing.

If you cannot solve the problem, contact your IBM support center.

00D10345

A LOG READ completed unsuccessfully because an error was received from a CATALOG LOCATE request for an archive log data set. The requested archive log data set might have been uncataloged or deleted.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem. Examine the SYS1.LOGREC and SVC dump.

00D10348

The maximum retry count was exceeded while attempting to read a log RBA.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

Check the console log for related errors. This problem might occur if the user has specified an archive or active log data set to the BSDS with an incorrect RBA range.

If you cannot solve the problem, contact your IBM support center.

00D10354

A LOG READ request completed successfully but the length of the log record header was not as expected. This is an internal error.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem.

Examine the SYS1.LOGREC, console log, and SVC dump for information about prior errors during LOG READ processing.

If you are unable to solve the problem, contact your IBM support center.

00D10406

The bootstrap data set access service received a request with an invalid function code.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

You might find the items listed in "Diagnostics" on page 5070 useful in resolving the problem. If you cannot solve the problem, contact your IBM support center.

00D10410

An unsuccessful completion of a READ BSDS RECORD has occurred. An error has been returned from VSAM.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

Check the console log for return codes from VSAM.

If you are unable to resolve the problem, note these values, collect the items listed in "Diagnostics" on page 5070, and contact your IBM support center.

00D10411

An unsuccessful completion of a WRITE UPDATE BSDS RECORD has occurred. An error has been returned from VSAM.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

Check the console log for return codes from VSAM.

If you are unable to resolve the problem, note these values, collect the items listed in "Diagnostics" on page 5070, and contact your IBM support center.

00D10412

An unsuccessful completion of a WRITE INSERT BSDS RECORD has occurred. An error has been returned from VSAM.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

Check the console log for return codes from VSAM.

If you are unable to resolve the problem, note these values, collect the items listed in "Diagnostics" on page 5070, and contact your IBM support center.

00D10413

An unsuccessful completion of a DELETE BSDS RECORD has occurred. An error has been returned from VSAM.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

Check the console log for return codes from VSAM.

If you are unable to resolve the problem, note these values, collect the items listed in "Diagnostics" on page 5070, and contact your IBM support center.

00D10419

An error was returned from the z/OS GETDSAB service.

System action

The current utility terminates abnormally.

System programmer response

Contact your IBM support center.

00D1041A

An error was returned from the z/OS SWAREQ service.

System action

The current utility terminates abnormally.

System programmer response

Contact your IBM support center.

00D1041B

The Db2 subsystem that a utility has connected to does not meet the minimum system requirements for this version of IBM MQ for z/OS.

System action

The current utility terminates abnormally.

System programmer response

Ensure that the Db2 data-sharing group name, and subsystem ID provided in the parameters to the utility are correct, and that the Db2 subsystem meets the system requirements for this version of IBM MQ for z/OS.

See IBM MQ prerequisites for a link to the IBM MQ for z/OS requirements web page.

00D10700

An error completion code was returned by SETLOCK OBTAIN.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

You might find the items listed in “Diagnostics” on page 5070 useful in resolving the problem. In the dump, register 0 contains the return code from SETLOCK OBTAIN.

00D10701

An error completion code was returned by SETLOCK RELEASE.

System action

An execution unit writes a record to SYS1.LOGREC and requests an SVC dump. The execution unit then terminates abnormally.

System programmer response

You might find the items listed in “Diagnostics” on page 5070 useful in resolving the problem. In the dump, register 0 contains the return code from SETLOCK RELEASE.

Lock manager codes (X'D3'): z/OS

If a lock manager reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- A description of the actions that led to the error or, if applicable, either a listing of the application program or the input string to a utility program that was being run at the time of the error.
- The console output for the period leading up to the error.
- The queue manager job log.
- The system dump resulting from the error.
- If you are using CICS, the CICS transaction dump output.
- Appropriate WebSphere MQ, z/OS, Db2, CICS, and IMS service levels.

00D301F1

An attempt to obtain storage was unsuccessful. This is probably because there is insufficient storage in your region.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Check that you are running in a region that is large enough. If not, reset your system and restart the queue manager. If this is not the cause of the problem, collect the items listed in "Diagnostics" and contact your IBM support center.

00D301F2

An attempt to obtain storage was unsuccessful. This is probably because there is insufficient storage in your region.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Check that you are running in a region that is large enough. If not, reset your system and restart the queue manager. If this is not the cause of the problem, collect the items listed in "Diagnostics" and contact your IBM support center.

00D301F3

An attempt to obtain storage was unsuccessful. This is probably because there is insufficient storage in your region.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Check that you are running in a region that is large enough. If not, reset your system and restart the queue manager. If this is not the cause of the problem, collect the items listed in "Diagnostics" and contact your IBM support center.

00D301F4

An attempt to obtain storage was unsuccessful. This is probably because there is insufficient storage in your region.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Check that you are running in a region that is large enough. If not, reset your system and restart the queue manager. If this is not the cause of the problem, collect the items listed in "Diagnostics" on page 5091 and contact your IBM support center.

00D301F5

An attempt to obtain storage was unsuccessful. This is probably because there is insufficient storage in your region.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Check that you are running in a region that is large enough. If not, reset your system and restart the queue manager. If this is not the cause of the problem, collect the items listed in "Diagnostics" on page 5091 and contact your IBM support center.

00D302F1, 00D302F2, 00D302F3, 00D302F4, 00D302F5, 00D303F1, 00D303F2, 00D303F3, 00D304F1, 00D305F1, 00D306F1

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in "Diagnostics" on page 5091 and contact your IBM support center.

00D31094, 00D31095, 00D31096, 00D31097

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Collect the items listed in "Diagnostics" on page 5091 and contact your IBM support center.

Message manager codes (X'D4'): z/OS

If a message manager reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- A description of the actions that led to the error or, if applicable, either a listing of the application program or the input string to a utility program that was being run at the time of the error.
- The console output for the period leading up to the error.
- The queue manager job log.
- The system dump resulting from the error.
- If you are using CICS, the CICS transaction dump output.
- Appropriate IBM MQ, z/OS, Db2, CICS, and IMS service levels.
- If you are using the IBM MQ Operations and Control panels, the ISPF panel name.

00D40001, 00D40002

An internal error has occurred while processing a command.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in "Diagnostics" and contact your IBM support center.

00D40003, 00D40004, 00D40007

An internal error has occurred while processing a DEFINE or ALTER command for a queue.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in "Diagnostics" and contact your IBM support center.

00D40008

An internal error has occurred while processing a DEFINE or ALTER command for a process.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in "Diagnostics" and contact your IBM support center.

00D40009

An internal error has occurred while processing a DEFINE or ALTER command for a queue.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in "Diagnostics" and contact your IBM support center.

00D4000A, 00D4000B, 00D4000C

An internal error has occurred while processing a command.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D4000D

An internal error has occurred while attempting to establish a processing environment for the command processors.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D4000E, 00D4000F

An internal error has occurred while attempting to establish a processing environment.

System action

The current execution unit terminates with completion code X'5C6'

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D40010

An internal error has occurred while processing a command.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D40011, 00D40012, 00D40013, 00D40014

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D40015

An attempt to write a trigger message to the initiation queue or the dead-letter queue was unsuccessful because of an internal error (for example, a storage overwrite).

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D40016, 00D40017, 00D40018, 00D4001A, 00D4001B, 00D4001C, 00D4001D, 00D4001E, 00D4001F

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D40020, 00D40021, 00D40022, 00D40023, 00D40024, 00D40025

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D40026

An internal error has occurred while processing a DEFINE CHANNEL or ALTER command for a channel.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D40027, 00D40028, 00D40029, 00D4002A, 00D4002B, 00D4002C

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D4002D

An attempt to write a message to a queue was unsuccessful because of an internal error (for example, a storage overwrite).

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D4002E

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D4002F

An internal error has occurred while processing a channel command.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D40030

The report option requested in a message was not recognized.

System action

The current execution unit terminates with completion code X'5C6'. A dump is produced.

System programmer response

Correct the value of the report option field (the value specified is given in register 2).

00D40031, 00D40032

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D40033

An internal error has occurred while processing a STGCLASS command.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D40034, 00D40035, 00D40036, 00D40037, 00D40038, 00D40039

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D4003B

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093. Also collect details of the queue-sharing group (QSG) and of the queue managers active, as well as the queue managers defined to the queue-sharing group at the time of the error. This information can be obtained by entering the following z/OS commands:

```
D XCF,GRP
```

to display a list of all QSGs in the coupling facility.

```
D XCF,GRP,qsg-name,ALL
```

to display status about the queue managers defined to queue-sharing group qsg-name. Contact your IBM support center.

00D4003C, 00D4003D

An internal error has occurred while processing a DEFINE CFSTRUCT or ALTER CFSTRUCT or DELETE CFSTRUCT command.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in "Diagnostics" on page 5093 and contact your IBM support center.

00D4003E

An internal error has occurred while processing an AUTHINFO command.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in "Diagnostics" on page 5093 and contact your IBM support center.

00D4003F

An internal error has occurred while processing a DEFINE MAXSMSGS or ALTER QMGR command.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in "Diagnostics" on page 5093 and contact your IBM support center.

00D40040

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in "Diagnostics" on page 5093 and contact your IBM support center.

00D40042

An internal processing error has occurred. The repository cannot locate an object that it has been asked to release.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in "Diagnostics" on page 5093 and contact your IBM support center.

00D40043, 00D40044, 00D40045, 00D40046, 00D40047, 00D40048

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D40049

An internal processing error has occurred while attempting to create the queue manager object during end restart processing.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D40050

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. The IGQ agent then attempts to recover.

System programmer response

If the IGQ agent fails to recover properly, an attempt could be made to disable the SYSTEM.QSG.TRANSMIT.QUEUE to force the IGQ agent to enter retry, or if this fails, the IGQ agent task can be restarted by issuing an ALTER QMGR IGQ(ENABLED) command or by restarting the queue manager.

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D40051, 00D40052

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D40053

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 together with a dump of the coupling facility list structure that the shared queue is defined to use, and contact your IBM support center.

00D40054

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093. Also collect details of the queue-sharing group (QSG) and of the queue managers active, as well as the queue managers defined to the queue-sharing group at the time of the error. This information can be obtained by entering the following z/OS commands:

```
D XCF,GRP
```

to display a list of all QSGs in the coupling facility.

```
D XCF,GRP,qsg-name,ALL
```

to display status about the queue managers defined to queue-sharing group qsg-name. Contact your IBM support center.

00D40055, 00D40056

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D40060

While performing Shared Channel Recovery Processing, DB2 was found to be inactive.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Check why Db2 related tasks are unavailable.

The recovery process is terminated; some channels might have been recovered, while others have not. Any channels that were not recovered will be recovered when the recovery process next runs; alternatively, they can be restarted manually. For more information about recovery and restart mechanisms used by IBM MQ, see Recovery and restart.

00D40062, 00D40064, 00D40065, 00D40066

An internal error has occurred during shared channel recovery.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

The recovery process is terminated; some channels may have been recovered, while others have not. Any channels that were not recovered will be recovered when the recovery process next runs; alternatively, they can be restarted manually. For more information about recovery and restart mechanisms used by IBM MQ, see Recovery and restart.

00D40067

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D40068

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'. In some cases, the queue manager might terminate with completion code X'6C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

Restart the queue manager if necessary.

00D40069

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093. Also collect details of the queue-sharing group (QSG) and of the queue managers active, as well as the queue managers defined to the queue-sharing group at the time of the error. This information can be obtained by entering the following z/OS commands:

```
D XCF,GRP
```

to display a list of all QSGs in the coupling facility.

```
D XCF,GRP,qsg-name,ALL
```

to display status about the queue managers defined to queue-sharing group qsg-name. Contact your IBM support center.

00D40070

An internal error has occurred involving the cluster cache.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and the channel initiator job log, and contact your IBM support center.

00D40071, 00D40072, 00D40073, 00D40074, 00D40075, 00D40076, 00D40077, 00D40078, 00D40079, 00D4007A, 00D4007B, 00D4007C, 00D4007D, 00D4007E, 00D4007F

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

Restart the queue manager if necessary.

00D40080

An internal error has occurred invloving the cluster cache.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and the channel initiator job log, and contact your IBM support center.

00D40081

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

Restart the queue manager if necessary.

00D40082

An internal error has occurred invloving the cluster cache.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and the channel initiator job log, and contact your IBM support center.

00D40083

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

Restart the queue manager if necessary.

00D40084

An internal error has occurred when opening a managed destination queue.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

Restart the queue manager if necessary.

00D40085

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

Restart the queue manager if necessary.

00D40086, 00D40087

An internal error has occurred while processing a DEFINE or ALTER command for a subscription.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D40091

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

Restart the queue manager if necessary.

00D4009C

An internal error has occurred while processing an **ALTER SMDS** or **RESET SMDS** command.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D4009D

An internal error has occurred while processing a **START SMDSCONN** or **STOP SMDSCONN** command.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

00D400B1

While putting a message, an error was detected in the chaining of message headers.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Check the chaining fields (**CodedCharSetId**, **Encoding**, and **Format**) in the MQMD and headers for the problem message to determine which values are invalid or inconsistent.

At each point in the header chain, the field values must correctly describe the data in the next header:

- The **Format** field identifies the correct format of the next header
- The **CodedCharSetId** field identifies the character set of text fields in the next header

- The **Encoding** field identifies the numeric encoding of numeric fields in the next header

V 9.0.3

00D400D0

Unable to get below the line storage for data control blocks when attempting to open the QM INI (CSQMQMIN) data set.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in "Diagnostics" on page 5093 and contact your IBM support center.

00D401F1

Whilst processing a get message request, the specified search type (message identifier or correllation identifier) was found to be in error. This indicates a data corruption error.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in "Diagnostics" on page 5093 and contact your IBM support center.

00D44001

This reason code is issued in message CSQM090E when a command has failed. This code indicates that an object of the specified name exists, but is of a different subtype; it might not necessarily have the same disposition in the queue-sharing group. This can only occur with subtypes of queues or channels. Message CSQM099I is also issued, indicating the object in error.

Severity

8

System action

The command is ignored.

System programmer response

Reissue the command, ensuring that all object subtypes are correct.

00D44002

This reason code is issued in message CSQM090E when a command has failed. This code indicates that the object specified on the request could not be located. Message CSQM094I or message CSQM125I is also issued, indicating the object in error.

It is also issued in message CSQM086E, indicating that the queue manager object could not be located.

Severity

8

System action

For CSQM090E, the command is ignored. For CSQM086E, the queue manager fails to restart.

System programmer response

If you are using a queue-sharing group, check that DB2 is available and not suspended. Define the object in question. For the queue manager, reissue the START QMGR command to restart the queue manager.

Note: An object of the same name and type, but of a different disposition, might already exist. If you are dealing with a queue or channel object, an object of the same name, but of a different subtype, might already exist.

00D44003

This reason code is issued in message CSQM090E when a command has failed. This code indicates that the object specified on the request already exists. This will only arise when trying to define a new object. Message CSQM095I is also issued.

Severity

8

System action

The command is ignored.

System programmer response

Use the object in question.

00D44004

This reason code is issued in message CSQM090E when a command has failed. This code indicates that one or more of the keywords on the command failed the parameter validation rules that apply to them. One or more other more specific messages are also issued, indicating the reason for the validation failure.

Severity

8

System action

The command is ignored.

System programmer response

Refer to the more specific associated message to determine what the error is.

00D44005

This reason code is issued in message CSQM090E when a command has failed. This code indicates that one of the following situations has occurred:

- The object specified on the request is currently open. This typically happens when an object is in use through the API or a trigger message is being written to it, but it could also arise because the object specified is in the process of being deleted. For a local queue, it can occur because there are messages currently on the queue. Message CSQM101I or CSQM115I is also issued.
- A request has been issued for a local queue, but this queue has incomplete units of recovery outstanding for it. Message CSQM110I is also issued.
- An alter, delete, or define request was made against a storage class that is in use (that is, there is a queue defined as using the storage class, and there are messages currently on the queue. Message CSQM101I is also issued.
- An ALTER CFSTRUCT command was issued and an associated shared queue has messages or uncommitted message activity.

Severity

8

System action

The command is ignored.

System programmer response

Refer to the description of message CSQM101I, CSQM110I, or CSQM115I as appropriate.

00D44006

This reason code is issued in message CSQM090E when a command has failed. This code indicates that a request has been issued to delete a local queue. The PURGE option has not been specified, but there are messages on the queue. Message CSQM103I is also issued.

Severity

8

System action

The command is ignored.

System programmer response

If the local queue must be deleted, even though there are messages on it, reissue the command with the PURGE option.

00D44007

This reason code is issued in message CSQM090E when a command has failed. This code indicates that a request has been issued for a local queue that is dynamic, but this queue has been flagged for deletion. Message CSQM104I is also issued.

Severity

8

System action

The command is ignored.

System programmer response

None, the local queue will be deleted as soon as possible.

00D44008

This reason code is issued in message CSQM090E when a command has failed. This code indicates that the object specified on the request needs updating because the IBM MQ version has changed, but that this cannot be done because the object is currently open. Message CSQM101I is also issued.

Severity

8

System action

The command is ignored.

System programmer response

Wait until the object is closed and reissue the command.

00D44009

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM112E or message CSQM117E indicating the object in error. It is also issued in message CSQM086E during queue manager restart.

This code indicates that a request has been issued for an object, but the object information could not be accessed because of an error on page set zero.

Severity

8

System action

The command is ignored or the queue manager fails to restart.

System programmer response

Check for error messages on the console log that might relate to the problem. Verify that page set zero is set up correctly; refer to Managing page sets for information about this.

00D4400A

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM113E indicating the object in error. It is also issued in message CSQM086E during queue manager restart. This code indicates that a request has been issued for an object, but page set zero is full.

Severity

8

System action

The command is ignored or the queue manager fails to restart.

System programmer response

Increase the size of page set zero. Refer to Managing page sets for information about how to do this.

00D4400B

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM114E. This code indicates that a request has been issued for a local queue, but no more local queues could be defined. There is an implementation limit of 524 287 for the total number of local queues that can exist. For shared queues, there is a limit of 512 queues in a single coupling facility structure, and 512 structures altogether.

For the indexes used by shared queues (OBJ_QUEUE_IX1 and OBJ_QUEUE_IX2), 48 KB of space allocation is sufficient for 512 queues.

Severity

4

System action

The command is ignored.

System programmer response

Delete any existing queues that are no longer required.

00D4400C

This reason code is issued in message CSQM090E when a command has failed. It indicates that the command is not allowed for a particular subtype of an object, as shown in the accompanying more specific message.

Severity

4

System action

The command is ignored.

System programmer response

Reissue the command with the object name specified correctly.

00D4400D

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM127I. This code indicates that a request was issued specifying a namelist as a list of cluster names, but there are no names in the namelist.

Severity

8

System action

The command is ignored.

System programmer response

Specify a namelist that is not empty.

00D4400E

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM112E or message CSQM117E indicating the object in error. It is also issued in message CSQM086E during queue manager restart. This code indicates that a request has been issued for an object, but that a page set that it requires is not defined.

Severity

8

System action

The command is ignored or the queue manaager fails to restart.

System programmer response

Ensure that the necessary page set is defined in the initialization input data set CSQINP1, and has a DD statement in the queue manager started task JCL procedure. Restart the queue manager.

00D4400F

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM112E or message CSQM117E indicating the object in error. It is also issued in message CSQM086E during queue manager restart. This code indicates that a request has been issued for an object, but that a page set that it requires is not open.

Severity

8

System action

The command is ignored or the queue manager fails to restart.

System programmer response

Ensure that the necessary page set is defined in the initialization input data set CSQINP1, and has a DD statement in the queue manager started task JCL procedure. Restart the queue manager.

00D44010

This reason code is issued in message CSQM090E when a command has failed. This code indicates that a request was issued to change the default transmission queue for the queue manager, but the queue is already in use.

Severity

8

System action

The command is ignored.

System programmer response

Wait until the queue is no longer in use, or choose another queue.

00D44011

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM128E. This code indicates that a request was issued that required a message to be sent to a command queue, but the message could not be put.

Severity

8

System action

The command is ignored.

System programmer response

Resolve the problem with the command queue.

00D44013

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM160I indicating the object in error.

Severity

8

System action

The command is ignored.

System programmer response

See message CSQM160I for more information.

00D44014

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM161I.

Severity

8

System action

The command is ignored.

System programmer response

See message CSQM161I for more information.

00D44015

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM164I indicating the object in error.

Severity

8

System action

The command is ignored.

System programmer response

See message CSQM164I for more information.

00D44016

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM163I indicating the object in error.

Severity

8

System action

The command stops processing.

System programmer response

See message CSQM163I for more information.

00D44017

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM112E or message CSQM117E indicating the object in error. It is also issued in message CSQM086E during queue manager restart.

This code indicates that a request has been issued for an object, but the object information could not be accessed because coupling facility structure has failed.

Severity

8

System action

The command is ignored or the queue manager fails to restart.

System programmer response

Check for error messages on the console log that might relate to the problem. Use the RECOVER CFSTRUCT command to recover the coupling facility structure.

00D44018

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM112E or message CSQM117E indicating the object in error. It is also issued in message CSQM086E during queue manager restart.

This code indicates that a request has been issued for an object, but the object information could not be accessed because there is an error or inconsistency in the coupling facility information.

Severity

8

System action

The command is ignored or the queue manager fails to restart.

System programmer response

Check for error messages on the console log that might relate to the problem. Check that Db2 is available and not suspended. If the problem persists, it may be necessary to restart the queue manager.

00D44019

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM112E or message CSQM117E indicating the object in error. It is also issued in message CSQM086E during queue manager restart.

This code indicates that a request has been issued for an object, but the object information could not be accessed because Db2 is not available or is suspended.

Severity

8

System action

The command is ignored or the queue manager fails to restart.

System programmer response

Check for error messages on the console log that might relate to the problem. Check that Db2 is available and not suspended.

00D44023

This reason code is issued in message CSQM090E and is accompanied by message CSQM117E when a command cannot be executed because a CF structure is not available.

System action

The command is ignored.

System programmer response

See reason code MQRC_CF_STRUC_IN_USE (2346, X'092A') for more information.

00D4001B

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM182E.

Severity

8

System action

The command is ignored.

System programmer response

See message CSQM182E for more information.

00D4001C

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM183E.

Severity

8

System action

The command is ignored.

System programmer response

See message CSQM183E for more information.

00D4001D

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM185E.

Severity

8

System action

The command is ignored.

System programmer response

See message CSQM185E for more information.

00D4001E

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM186E.

Severity

8

System action

The command is ignored.

System programmer response

See message CSQM186E for more information.

00D4401F

This reason code is issued in message CSQM090E when a command has failed, and is accompanied by message CSQM190E.

Severity

8

System action

The command is ignored.

System programmer response

See message CSQM190E for more information.

00D44020

This reason code is issued in message CSQM090E when a PUBSUB command cannot be executed because PUBSUB is disabled.

System action

The command is ignored.

System programmer response

See message CSQM292I for more information.

00D4F001

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5093 and contact your IBM support center.

Command server codes (X'D5'): z/OS

If a command server reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- A description of the actions that led to the error or, if applicable, either a listing of the application program or the input string to a utility program that was being run at the time of the error.
- The console output for the period leading up to the error.
- The queue manager job log.
- The system dump resulting from the error.
- Appropriate WebSphere MQ, z/OS, Db2, CICS, and IMS service levels.
- Any trace information collected.
- If message CSQN104I or CSQN202I was issued was issued, return and reason codes from the message.

00D50101

During initialization, the command server was unable to obtain storage. This is probably because there is insufficient storage in your region.

System action

Message CSQN104I is sent to the console containing this reason code and the return code from the internal storage macro. None of the commands in the initialization data set currently being processed are performed. Queue manager startup continues.

Note: If there is a storage problem, startup might not be successful.

System programmer response

Check that you are running in a region that is large enough, and if not, reset your system and restart the queue manager. If this is not the cause of the problem, collect the following items and contact your IBM support center:

- Return and reason codes from CSQN104I message
- Trace of startup (if available)

00D50102

The command preprocessor ended abnormally while processing a command in the initialization input data set.

System action

Message CSQ9029E is produced, followed by message CSQN103I with this code as the return code, and a reason code of -1 indicating that the command was not processed, and a dump is produced. The next command is processed.

System programmer response

Look in the output data set to determine the command in error. Check that the command is correctly formed, that it applies to a valid object.

If the command is correct, collect the items listed in “Diagnostics” and contact your IBM support center.

00D50103

During initialization, an internal error occurred.

System action

Message CSQN104I is sent to the z/OS console, indicating the return and reason codes from the internal macro. The command server stops, without processing any commands.

System programmer response

Review the job log for messages about other errors that might be related. If you are unable to solve the problem, collect the items listed in "Diagnostics" on page 5112, and contact your IBM support center.

00D50104

An internal error occurred during initialization.

System action

Message CSQN104I is sent to the z/OS console, indicating the return and reason codes from the internal macro. The command server stops, without processing any commands.

System programmer response

Stop and restart the queue manager.

Collect the items listed in "Diagnostics" on page 5112 and contact your IBM support center.

00D50105

An internal error has occurred.

System action

The command server terminates, and a dump is produced.

System programmer response

Stop and restart the queue manager.

Collect the items listed in "Diagnostics" on page 5112 and contact your IBM support center.

00D50201

The command server was unable to obtain storage while starting. This return code typically occurs because there is insufficient storage in your region.

System action

Message CSQN202I is sent to the z/OS console, indicating the return code from the internal storage macro. The command server stops, without processing any commands.

System programmer response

Check that you are running in a region that is large enough, and if not, reset your system and restart the queue manager. If this is not the cause of the problem, collect the items listed in "Diagnostics" on page 5112 and contact your IBM support center.

00D50202

An internal error has occurred.

System action

Message CSQN202I is sent to the z/OS console, indicating the return code from the internal macro. The command server stops, without processing any commands.

System programmer response

Review the job log for messages about other errors that might be related. If you are unable to solve the problem, collect the items listed in "Diagnostics" on page 5112 and contact your IBM support center.

00D50203

An internal error has occurred.

System action

Message CSQN202I is sent to the z/OS console, indicating the return code from the internal macro. The command server stops, without processing any commands.

System programmer response

Issue the START CMDSERV command to restart the command server.

Collect the items listed in "Diagnostics" on page 5112 and contact your IBM support center.

00D50208

The command server was unable to obtain storage during startup.

System action

Message CSQN202I is sent to the z/OS console, indicating the return code from the internal macro. The command server stops, without processing any commands.

System programmer response

Check that you are running in a region that is large enough, and if not, reset your system and restart the queue manager. If this is not the cause of the problem, collect the items listed in "Diagnostics" on page 5112 and contact your IBM support center.

00D50209

The command preprocessor ended abnormally while processing a command from the command server.

System action

Message CSQN205I is put onto the reply-to queue with COUNT=1, RETURN=00D50209, and REASON=-1 indicating that the command has not been processed. The command server processes the next command.

System programmer response

Check that the command is correctly formed, that it applies to a valid object.

If the command is correct, collect the items listed in "Diagnostics" on page 5112 and contact your IBM support center.

00D5020C

While waiting for a command, the command server did not recognize the reason for the end of the wait. This is because it was not one of the following:

- The arrival of a message
- The STOP CMDSERV command

System action

Messages CSQN203I and CSQN206I are sent to the console, containing the return and reason codes from the request function, and the ECB list.

The command server is terminated and a dump is produced.

System programmer response

Issue the START CMDSERV command to restart the command server.

Collect the items listed in "Diagnostics" on page 5112 and contact your IBM support center.

00D5020E

The command processor attempted to get a command from the system-command-input queue, but the attempt was unsuccessful because of an internal error.

System action

The command server continues processing. Message CSQN203I is written to the console containing the return and reason codes from the API call.

System programmer response

Collect the items listed in "Diagnostics" on page 5112 and contact your IBM support center.

00D5020F

The command processor got a command from the system-command-input queue, but was unable to process it because the message was not of type MQMT_REQUEST.

System action

The command processor processes the next command message.

00D50210

The command processor got a command from the system-command-input queue, but was unable to process it because the command message was of length zero.

System action

The command processor processes the next command message.

00D50211

The command processor got a command from the system-command-input queue, but was unable to process it because the command message consisted of blank characters only.

System action

The command processor processes the next command message.

00D50212

The command processor got a command from the system-command-input queue, but was unable to process it because the command message was greater than 32 762 characters long.

System action

The command processor processes the next command message.

00D54000

An internal error has occurred.

System action

The command server is terminated and a dump is produced.

System programmer response

Issue the START CMDSERV command to restart the command server.

Collect the items listed in "Diagnostics" on page 5112 and contact your IBM support center.

00D54nnn

The command processor got a command from the system-command-input queue, but was unable to process it because the command message indicated that data conversion was required and an error occurred during conversion. *nnn* is the reason code (in hexadecimal) returned by the MQGET call.

System action

The command processor processes the next command message.

System programmer response

Refer to API completion and reason codes for information about the reason code *nnn*.

Buffer manager codes (X'D7'): z/OS

If a buffer manager reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- A description of the actions that led to the error or, if applicable, either a listing of the application program or the input string to a utility program that was being run at the time of the error.
- The console output for the period leading up to the error.
- The queue manager job log.
- The WebSphere MQ active log data set.
- The system dump resulting from the error.
- If you are using CICS, the CICS transaction dump output.
- Appropriate WebSphere MQ, z/OS, Db2, CICS, and IMS service levels.

00D70101

An attempt to obtain storage for a buffer manager control block (the PANC) was unsuccessful. This is probably because there is insufficient storage in your region.

System action

The queue manager is terminated, an entry is written to SYS1.LOGREC, and a dump is produced. Registers 2 and 0 contain the return and reason codes from the STORAGE or GETMAIN request.

System programmer response

Check that you are running in a region that is large enough, and if not, reset your system and restart the queue manager. If this does not resolve the problem, note the register values, and contact your IBM support center.

00D70102

The name of the queue manager being restarted does not match the name recorded in a prior checkpoint log record.

System action

The queue manager is terminated, an entry is written to SYS1.LOGREC, and a dump is produced. This is preceded by message CSQP006I. Register 0 contains the name found in the log record. Register 2 contains the name of the queue manager being restarted.

System programmer response

Change the started task JCL procedure xxxxMSTR for the queue manager to name the appropriate bootstrap and log data sets.

The print log utility, CSQ1LOGP, can be used to view checkpoint records. You might also find the MQ active log data set useful for problem determination.

00D70103

An attempt to obtain storage for a buffer manager control block (a PSET) was unsuccessful.

System action

The queue manager is terminated, an entry is written to SYS1.LOGREC, and a dump is produced. Registers 2 and 0 contain the return and reason codes from the STORAGE or GETMAIN request.

System programmer response

Restart the queue manager.

Note the register values, and contact your IBM support center.

00D70104

An attempt to obtain storage for a buffer manager control block (a BHDR) was unsuccessful.

System action

The queue manager is terminated, an entry is written to SYS1.LOGREC, and a dump is produced. Registers 2 and 0 contain the return and reason codes from the STORAGE or GETMAIN request.

System programmer response

Restart the queue manager.

Note the register values, and contact your IBM support center.

00D70105

An internal error has occurred during dynamic page set expansion.

System action

The current page set extend task is terminated, an entry is written to SYS1.LOGREC, and a dump is produced. No further attempt will be made to expand the page set until the queue manager is restarted. Subsequent dynamic page set extend requests for other page sets are processed.

System programmer response

Collect the items listed in "Diagnostics" on page 5116 and contact your IBM support center.

00D70106

An internal error has occurred.

System action

An entry is written to SYS1.LOGREC, and a dump is produced.

System programmer response

Collect the items listed in "Diagnostics" on page 5116 and contact your IBM support center.

00D70108

An attempt to obtain storage for the buffer pool was unsuccessful.

System action

The queue manager is terminated, an entry is written to SYS1.LOGREC, and a dump is produced. Register 2 contains the return code from the STORAGE, GETMAIN or IARV64 GETSTOR request. Register 3 contains the buffer pool number.

System programmer response

Provide sufficient storage for the number of buffers specified in the DEFINE BUFFPOOL command.

If the buffer pool is backed by page fixed storage, that is it has a PAGECLAS of FIXED4KB, check that there is enough real storage available on the system. For more information, see Address space storage.

If it is not possible to rectify the problem:

- Alter the definition of the buffer pool in the CSQINP1 data set, to include the REPLACE attribute, and specify a smaller number of buffers, or
- Change the PAGECLAS attribute to 4KB.

00D7010A

An internal storage error has occurred.

System action

The queue manager is terminated, an entry is written to SYS1.LOGREC, and a dump is produced. Registers 2 and 3 contain the return and reason codes from the IARV64 GETSTOR request. Register 4 contains the buffer pool number.

System programmer response

Increase the value of the MEMLIMIT parameter.

00D70112

A critical process could not be started during queue manager initialization. This could be because there is insufficient storage in your region.

System action

The queue manager is terminated, an entry is written to SYS1.LOGREC, and a dump is produced. Register 0 contains the reason code for the error.

System programmer response

Check that you are running in a region that is large enough. If not, reset your system and restart the queue manager. If this does not resolve the problem, note the completion code and the reason code and contact your IBM support center.

00D70113

A critical process could not be started during queue manager initialization. This could be because there is insufficient storage in your region.

System action

The queue manager is terminated, an entry is written to SYS1.LOGREC, and a dump is produced. Register 0 contains the reason code for the error.

System programmer response

Check that you are running in a region that is large enough. If not, reset your system and restart the queue manager. If this does not resolve the problem, note the completion code and the reason code and contact your IBM support center.

00D70114

An internal cross-component consistency check failed.

System action

The request is terminated, an entry is written to SYS1.LOGREC, and a dump is produced. Register 0 contains the value in error.

System programmer response

Note the completion code and the reason code, collect the MQ active log data set, and contact your IBM support center.

00D70116

An I/O error has occurred.

System action

An entry is written to SYS1.LOGREC, and a dump is produced. Register 0 contains the Media Manager reason code from an MMCALL call. In some circumstances, the queue manager will terminate. (This depends on the nature of the error, and the page set on which the error occurred.)

System programmer response

Restart the queue manager if necessary.

See the *MVS/DFP Diagnosis Reference* manual for information about return codes from the Media Manager. If you do not have access to the required manual, contact your IBM support center, quoting the Media Manager reason code.

You might also find the MQ active log data set useful for problem determination.

00D70118

A page was about to be written to a page set, but was found to have improper format. The executing thread is terminated. (If this is the deferred write processor, the queue manager is terminated)

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Restart the queue manager. If the problem persists collect the items listed in “Diagnostics” on page 5116 and contact your IBM support center.

00D70120

No buffers are available to steal. An executing thread needed a buffer in a buffer pool to bring a page in from the page set. The buffer pool is overcommitted, and despite attempts to make more buffers available, including writing pages to the page set, no buffers could be released.

System action

The current execution unit terminates with completion code X'5C6'. The API request is terminated with reason code MQRC_UNEXPECTED_ERROR, with the aim of reducing demand for the buffer pool.

System programmer response

Determine the problem buffer pool from preceding CSQP019I and CSQP020E messages. Review the size of the buffer pool with the DISPLAY USAGE command. Consider increasing the size of the buffer pool using the ALTER BUFFPOOL command.

00D70122

An unrecoverable error has occurred during check point.

System action

The queue manager is terminated, an entry is written to SYS1.LOGREC, and a dump is produced. Register 0 contains the reason code for the error.

System programmer response

Restart the queue manager.

Note the completion code and the reason code, collect the MQ active log data set, and contact your IBM support center.

00D70133

An internal consistency check failed.

System action

The request is terminated, an entry is written to SYS1.LOGREC, and a dump is produced.

System programmer response

Note the completion code and the reason code, collect the MQ active log data set, and contact your IBM support center.

00D70136

A critical process could not be started during queue manager initialization. This could be because there is insufficient storage in your region.

System action

The queue manager is terminated, an entry is written to SYS1.LOGREC, and a dump is produced. Register 0 contains the reason code for the error.

System programmer response

Check that you are running in a region that is large enough. If not, reset your system and restart the queue manager. If this does not resolve the problem, note the completion code and the reason code and contact your IBM support center.

00D70137

A critical process could not be started during queue manager initialization. This could be because there is insufficient storage in your region.

System action

The queue manager is terminated, an entry is written to SYS1.LOGREC, and a dump is produced. Register 0 contains the reason code for the error.

System programmer response

Check that you are running in a region that is large enough. If not, reset your system and restart the queue manager. If this does not resolve the problem, note the completion code and the reason code and contact your IBM support center.

00D70139

An attempt to allocate 64 bit storage for internal use failed.

System action

The queue manager is terminated.

System programmer response

Raise the value of the MEMLIMIT parameter. For more information, see Address space storage.

00D7013A

An attempt to allocate storage for internal use failed. Register 2 contains the return code from the STORAGE request.

System action

The queue manager is terminated.

System programmer response

Provide sufficient storage. For more information, see Address space storage.

00D7013B

An internal consistency check failed.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect buffer manager problem determination information, and contact your IBM support center.

Recovery manager codes (X'D9'): z/OS

If a recovery manager reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- A description of the actions that led to the error or, if applicable, either a listing of the application program or the input string to a utility program that was being run at the time of the error.
- The console output for the period leading up to the error.
- The queue manager job log.
- The system dump resulting from the error.
- A printout of SYS1.LOGREC.
- If you are using CICS, the CICS transaction dump output.
- Appropriate IBM MQ, z/OS, Db2, CICS, and IMS service levels.

00D90000

A recovery manager module received control from its FRR for retry and found an invalid retry point identifier. The name of the module in which the error occurred appears in the SYS1.LOGREC entry showing this reason code in register 15.

System action

Standard diagnostic information is provided. The error is recorded in SYS1.LOGREC, an SVC dump is scheduled, and queue manager termination is requested. The termination reason code reflects the function for which retry was unsuccessfully attempted.

System programmer response

This is a secondary error. Obtain a copy of SYS1.LOGREC and the SVC dump for this error and for the original problem that resulted in the retry attempt. Examine the SYS1.LOGREC information and the dumps from both the original and the secondary error to determine if the recovery parameter area was damaged or if retry incorrectly restored registers for the mainline module.

Restart the queue manager.

00D90002

The recovery manager startup notification routine received an error return code from the recovery log manager when attempting to read a recovery manager status table (RMST) record from the bootstrap data set (BSDS) in one of the following cases:

- When reading the record containing the RMST header. The first copy was successfully read, but the second copy could not be found.
- When reading records containing the RMST entries. A *no record found* condition was encountered before all entries were read.
- When reading either a header record or an entry record. The record exceeded its expected length.

This is an IBM MQ error.

System action

The recovery manager has no functional recovery routine (FRR) in place when this error occurs. It relies on its invoker, the facility startup function, to perform SYS1.LOGREC recording and to request a dump. The queue manager terminates with a X'00E80100' reason code.

System programmer response

The queue manager determined that the BSDS that it was reading has been corrupted. If you are running in a dual BSDS environment, determine which BSDS is corrupt, and follow the procedures described in Recovering the BSDS to recover it from the valid BSDS.

Similarly, if you are running in a single BSDS environment, refer to Recovering the BSDS, which describes the procedures needed to recover your BSDS from an archived BSDS.

00D92001

The checkpoint/restart serial controller FRR invoked queue manager termination, because an unrecoverable error was detected while processing a request.

This is a queue manager termination reason code.

System action

Queue manager termination is initiated. Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the associated error.

System programmer response

Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error, and follow the instructions associated with it.

Restart the queue manager.

00D92003

The restart request servicer FRR invoked queue manager termination, because an unrecoverable error was detected while processing a restart request.

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Obtain a copy of SYS1.LOGREC and the SVC dump for the original error and follow the instructions associated with it.

Restart the queue manager.

00D92004

The shutdown checkpoint controller FRR invoked queue manager termination, because an unrecoverable error was detected while processing a shutdown checkpoint request.

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Obtain a copy of SYS1.LOGREC and the SVC dump for the original error and follow the instructions associated with it.

Restart the queue manager.

00D92011

An internal error has occurred.

System action

The checkpoint process will end abnormally to prevent a damaged URE from being written out to the log, and the queue manager will be terminated. This is to prevent the loss or incorrect processing of an IBM MQ unit of recovery (UR). Restart will use the previous checkpoint and apply all the IBM MQ log records up to the point of the problem. Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is scheduled.

System programmer response

Restart the queue manager.

Collect the items listed in "Diagnostics" on page 5121 and contact your IBM support center.

00D92012

An internal error has occurred.

System action

The checkpoint process will end abnormally to prevent a damaged RURE from being written out to the log, and the queue manager will be terminated. This is to prevent the loss or incorrect processing of an IBM MQ unit of recovery. Restart will use the previous checkpoint and apply all the IBM MQ log records up to the point of the problem. Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is scheduled.

System programmer response

Restart the queue manager.

Collect the items listed in "Diagnostics" on page 5121 and contact your IBM support center.

00D92021

An internal error has occurred.

System action

The restart processing ends abnormally, which terminates the queue manager. This is to prevent the loss or incorrect processing of a IBM MQ unit of recovery.

System programmer response

Do not attempt to restart the queue manager until the error is resolved.

The log has become corrupted. If you are running with dual logging, try to start the queue manager from the undamaged log.

If you are unable to do achieve this, use the following procedure (you will lose all updates since your last back up):

1. Restore your page sets from the last set of full backups. The queue manager must have been shut down cleanly before taking the backup copies of the page sets.
2. Clear the logs by following the procedure detailed in Resetting the queue manager's log.

See CSQUTIL utility for information about restarting the queue manager from one log when using dual logging, and using the CSQUTIL utility. If you are unable to resolve the problem, contact your IBM support center.

00D92022

An internal error has occurred.

System action

The restart processing ends abnormally, which terminates the queue manager. This is to prevent the loss or incorrect processing of a IBM MQ unit of recovery.

System programmer response

Do not attempt to restart the queue manager until the error is resolved.

The log has become corrupted. If you are running with dual logging, try to start the queue manager from the undamaged log.

If you are unable to do achieve this, use the following procedure (you will lose all updates since your last back up):

1. Restore your page sets from the last set of full backups. The queue manager must have been shut down cleanly before taking the backup copies of the page sets.
2. Clear the logs by following the procedure detailed in Resetting the queue manager's log.

See CSQUTIL utility for information about restarting the queue manager from one log when using dual logging, and using the CSQUTIL utility. If you are unable to resolve the problem, contact your IBM support center.

00D92023

During queue manager restart in 6 byte log RBA mode, a log record has been encountered that is written with an 8 byte log RBA.

System action

The restart processing ends abnormally, which terminates the queue manager. This is to prevent the loss, or incorrect processing, of a IBM MQ unit of recovery.

System programmer response

Do not attempt to restart the queue manager until the error is resolved.

The queue manager might have been started with an incorrect log or BSDS. Ensure that the queue manager is started with the correct log and BSDS data sets. If this was not the cause of the problem, the log or BSDS has become corrupted.

To recover from a corrupted log or BSDS, if you are running with dual logging or dual BSDS, try to start the queue manager from the undamaged log.

If you are unable to do achieve this, use the following procedure. Note, that by carrying out this procedure, you will lose all updates since your last back up:

1. Restore your page sets from the last set of full backups. The queue manager must have been shut down cleanly before taking the backup copies of the page sets.
2. Clear the logs by following the procedure detailed in Resetting the queue manager's log.

If necessary, when clearing the logs ensure that you convert the BSDS to the previous version, using the BSDS conversion utility CSQJUCNV.

See CSQUTIL utility for information about using the CSQUTIL utility. If you are unable to resolve the problem, contact your IBM support center.

00D93001

The commit/backout FRR invoked queue manager termination, because an unrecoverable error was detected during 'must-complete' processing for phase 2 of a commit-UR request.

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Obtain a copy of SYS1.LOGREC and the SVC dump for the original error and follow the instructions associated with it.

Restart the queue manager.

00D93011

A subcomponent of IBM MQ invoked commit when the agent state was invalid for commit-UR invocation. Commit-UR was requested for an agent that was modifying data. Either commit-UR or backout-UR was already in process, or the recovery structure (URE) was damaged.

System action

Abnormal termination of the agent results, including backing out (backout-UR) of its activity to the previous point of consistency. This releases all locks held by the agent for its resources.

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is scheduled. Additional information, identified in the SDWA variable recording area (VRA) by reason code X'00D9CCCC', is added to the VRA.

If the agent was in a 'must-complete' state (in-commit or in-backout), the queue manager is also terminated with reason code X'00D93001'. When the queue manager is next restarted, recoverable activity for this agent (such as an ensure-backout or ensure-commit UR) is handled to complete the commit or backout process.

System programmer response

This is an IBM MQ error. Examine the SYS1.LOGREC data and the dump to establish whether either commit-UR was invoked incorrectly or the control structure that reflects the state was damaged.

00D93012

A subcomponent of IBM MQ invoked commit when the agent state was invalid for commit-UR invocation. Commit-UR was invoked for an agent that was only retrieving data. Either commit-UR or backout-UR was already in process, or the ACE progress state field was damaged.

System action

Abnormal termination of the agent results, including backing out (backout-UR) of its activity to the previous point of consistency. This releases all locks held by the agent for its resources.

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is scheduled. Additional information, identified in the SDWA variable recording area (VRA) by reason code X'00D9CCCC', is added to the SDWA VRA.

System programmer response

This is an IBM MQ error. Examine the SYS1.LOGREC data and the dump to establish whether either commit-UR was invoked incorrectly or the control structure was damaged.

00D93100

This reason code indicates that an IBM MQ allied agent does not need to participate in the Phase-2 (Continue Commit) call, because all required work has been accomplished during the Phase-1 (Prepare) call.

This reason code is generated by the recovery manager when it is determined that an IBM MQ allied agent has not updated any IBM MQ resource since its last commit processing occurred.

System action

The 'yes' vote is registered with the commit coordinator.

System programmer response

None should be required because this is not an error reason code. This reason code is used for communication between components of IBM MQ.

00D94001

The commit/backout FRR invoked queue manager termination, because an unrecoverable error was detected during 'must-complete' processing for a backout-UR request.

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Obtain a copy of SYS1.LOGREC and the SVC dump for the original error and follow the instructions associated with it.

Restart the queue manager.

00D94011

A subcomponent of IBM MQ invoked backout at a point when the agent state is invalid for invoking the function that backs out units of recovery. Either backout-UR or commit-UR phase-2 was already in process, or the agent structure was damaged.

System action

Abnormal termination of the agent results and, because the agent is in a 'must-complete' state, the queue manager is terminated with reason code X'00D94001'. When the queue manager is restarted, recoverable activity for this agent is handled to complete the commit or backout process.

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is scheduled. Additional information, identified in the SDWA variable recording area (VRA) by reason code X'00D9AAAA', is added to the SDWA VRA.

System programmer response

This is an IBM MQ error. Examine the SYS1.LOGREC data and the dump to establish whether commit-UR was invoked incorrectly or the control structure was damaged.

00D94012

During backout, the end of the log was read before all the expected log ranges had been processed. The error is accompanied by an abnormal termination with reason code X'00D94001'.

This could be because the queue manager has been started with a system parameter load module that specifies OFFLOAD=NO rather than OFFLOAD=YES.

System action

The agent is abnormally terminated with completion code X'5C6'. Because the agent is in a must-complete state, the queue manager is terminated with reason code X'00D94001' and message CSQV086E.

Standard diagnostic information is recorded in SYS1.LOGREC. and an SVC dump is requested.

System programmer response

Run the print log map utility to print the content of both BSDSs. Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error. At the time of the error, registers 3 and 4 contain the 8-byte relative byte address (RBA) of the beginning of this unit of recovery. IBM MQ must read the log back to this point to complete the backout of this unit of recovery.

To restart the queue manager, you must add the missing archive log data sets back to the BSDS with the change log inventory utility, and increase the MAXARCH parameter in the CSQ6LOGP macro (the system parameter module log initialization macro) to complete the backout.

If the missing archive log is not available, or if archiving was not active, the queue manager cannot be restarted unless the log data sets and page sets are all reinitialized or restored from backup copies. Data will be lost as a result of this recovery action.

00D95001

The recovery manager's common FRR invoked queue manager termination, because an unrecoverable error was detected during checkpoint processing.

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Obtain a copy of SYS1.LOGREC and the SVC dump for the original error and follow the instructions associated with it.

Restart the queue manager.

00D95011

The recovery manager checkpoint FRR invoked queue manager termination, because an unrecoverable error was detected while performing its checkpoint functions.

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error and follow the instructions associated with it.

Restart the queue manager.

00D96001

The recovery manager's restart FRR invoked queue manager termination, because an unrecoverable error was detected during the restart processor processing.

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error and follow the instructions associated with it.

Restart the queue manager.

00D96011

The restart participation FRR invoked queue manager termination, because an unrecoverable error was detected while processing log records during restart.

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error and follow the instructions associated with it.

Restart the queue manager when the problem has been corrected.

00D96021

The queue manager was terminated during restart because an error occurred while attempting to read the log forward MODE(DIRECT). It is accompanied by a recovery log manager error X'5C6' with a reason code describing the specific error.

Each time a portion of the log is skipped, a 'read direct' is used to validate the beginning RBA of the portion that is read.

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Run the print log map utility to print the contents of both BSDSs. Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error. Follow instructions for the accompanying recovery log manager error. If possible, remove the cause of original error and restart the queue manager. If you cannot correct the error, contact your IBM support center.

00D96022

The restart FRR invoked abnormal termination, because, while reading the log forward during restart, the end-of-log was read before all recovery log scopes had been processed. It is followed by an abnormal termination with the same reason code (X'00D96022').

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the error before queue manager termination is initiated.

System programmer response

Run the print log map utility to print the contents of both BSDSs. Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error. At the time of the error, registers 2 and 3 (as shown in the dump or in SYS1.LOGREC) contain the relative byte address (RBA) of the last log record that was read before end-of-log was encountered. Follow instructions for the accompanying recovery log manager error. If you cannot correct the error, contact your IBM support center.

00D96031

The restart FRR invoked queue manager termination, because an error occurred while attempting to read the log backward MODE(DIRECT). It is accompanied by a recovery log manager error X'5C6' with a reason code describing the specific error.

Each time a portion of the log is skipped, a 'read direct' is used to validate the beginning RBA of the portion that is read.

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Run the print log map utility to print the contents of both BSDSs. Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error. Follow instructions for the accompanying recovery log manager error. See the accompanying error reason code.

Restart the queue manager.

00D96032

During restart, the end of the log was read before all the expected log ranges had been processed. The error is accompanied by an abnormal termination with the same reason code (X'00D96032').

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC. An SVC dump is requested. The queue manager is terminated with message CSQV086E.

System programmer response

Run the print log map utility to print the contents of both BSDSs. See Finding out what the BSDS contains for more information.

Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error. At the time of the error, registers 2 and 3 contain the relative byte address (RBA) of the last log record that was read before end-of-log was encountered. Determine where the log went.

00D97001

The agent concerned was canceled while waiting for the RECOVER-UR service to complete.

System action

The RECOVER-UR function is completed. Abnormal termination of the requesting agent occurs. Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested.

The condition that caused cancellation of the agent was installation initiated (for example, a *forced* termination of the queue manager).

00D97011

The queue manager was terminated during RECOVER-UR because an unrecoverable error was detected during RECOVER-UR (CSQRRUPR) recovery processing.

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested. queue manager terminates with message CSQV086E and return code X'00D97011'.

System programmer response

Determine the original error. If the error is log-related, see Active log problems before restarting the queue manager.

00D97012

The RECOVER-UR request servicer FRR invoked queue manager termination, because an unrecoverable error was detected while attempting to recover a unit of recovery.

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error and follow the instructions associated with it.

Restart the queue manager.

00D97021

The RECOVER-UR FRR invoked queue manager termination, because an error occurred while attempting to read the log MODE(DIRECT) during forward processing. It is accompanied by a recovery log manager error X'5C6' with a reason code describing the specific error.

Each time a portion of the log is skipped, a 'read direct' is used to validate the beginning RBA of the portion that is read.

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Run the print log map utility to print the contents of both BSDSs. Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error. Follow instructions for the accompanying recovery log manager error. See the accompanying error reason code.

Restart the queue manager.

00D97022

The RECOVER-UR invoked abnormal termination because end-of-log was reached before all ranges had been processed for forward recovery. This error is accompanied by an abnormal termination with the same reason code (X'00D97022').

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Run the print log map utility to print the contents of both BSDSs. Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error. At the time of the error, registers 2 and 3 contain the relative byte address (RBA) of the last log record that was read before end-of-log was encountered. Follow instructions for the accompanying recovery log manager error.

Restart the queue manager.

00D97031

The RECOVER-UR FRR invoked queue manager termination, because an error occurred during an attempt to read the log MODE(DIRECT) while reading the log backward. It is accompanied by a recovery log manager error X'5C6' with a reason code describing the specific error.

Each time a portion of the log is skipped, a 'read direct' is used to validate the begin-scope RBA of the portion that is read.

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Run the print log map utility to print the contents of both BSDSs. Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error. See the accompanying error reason code. Follow instructions for the accompanying recovery log manager error.

Restart the queue manager.

00D97032

The RECOVER-UR invoked abnormal termination because end-of-log was reached before all ranges had been processed for backward recovery. This error is accompanied by an abnormal termination with the same reason code (X'00D97032').

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Run the print log map utility to print the contents of both BSDSs. Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error. At the time of the error, registers 2 and 3 contain the relative byte address (RBA) of the last log record that was read before end-of-log was encountered. Follow instructions for the accompanying recovery log manager error.

Restart the queue manager.

00D98001

The recovery manager's common FRR invoked queue manager termination, because an unrecoverable error was detected during indoubt-UR processing.

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error and follow the instructions associated with it.

Restart the queue manager.

00D98011

The FRR for the resolved-indoubt-UR request servicer invoked queue manager termination, because an unrecoverable error was detected processing a request.

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error. See the accompanying error reason code.

Restart the queue manager.

00D98021

The resolved indoubt FRR invoked queue manager termination because of an error while attempting to read the log MODE(DIRECT) during forward recovery. It is accompanied by a recovery log manager error X'5C6' with a reason code describing the specific error.

Each time a portion of the log is skipped, a 'read direct' is used to validate the beginning RBA of the portion that is read.

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Run the print log map utility to print the contents of both BSDSs. Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error. See the accompanying error reason code. Follow instructions for the accompanying recovery log manager error.

Restart the queue manager.

00D98022

Resolved indoubt invoked abnormal termination when end-of-log was reached before all ranges had been processed for forward recovery. This error is accompanied by abnormal termination with the same reason code (X'00D98022').

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Run the print log map utility to print the contents of both BSDSs. Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error. At the time of the error, registers 2 and 3 contain the relative byte address (RBA) of the last log record that was read before end-of-log was encountered. Follow instructions for the accompanying recovery log manager error.

Restart the queue manager.

00D98031

The resolved indoubt FRR invoked queue manager termination, because an error occurred during an attempt to read the log MODE(DIRECT) while reading the log backward. It is accompanied by a recovery log manager error X'5C6' with a reason code describing the specific error.

Each time a portion of the log is skipped, a 'read direct' is used to validate the begin-scope RBA of the portion that is read.

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Run the print log map utility to print the contents of both BSDSs. Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error. See the accompanying error reason code. Follow instructions for the accompanying recovery log manager error.

Restart the queue manager.

00D98032

The resolved indoubt FRR invoked abnormal termination when end-of-log was reached before all ranges had been processed for backward recovery. This error is accompanied by abnormal termination with the same reason code (X'00D98032').

This is a queue manager termination reason code.

System action

Standard diagnostic information is recorded in SYS1.LOGREC, and an SVC dump is requested for the original error before queue manager termination is initiated.

System programmer response

Run the print log map utility to print the contents of both BSDSs. Obtain a copy of the SYS1.LOGREC and the SVC dump for the original error. At the time of the error, registers 2 and 3 contain the relative byte address (RBA) of the last log record that was read before end-of-log was encountered. Follow instructions for the accompanying recovery log manager error.

Restart the queue manager.

00D99001

The checkpoint RBA in the conditional restart control record, which is deduced from the end RBA or LRSN value that was specified, is not available. This is probably because the log data sets available for use at restart do not include that end RBA or LRSN.

System action

The queue manager terminates.

System programmer response

See message CSQR015E.

00D99104

Queue manager restart detected that backward migration of messages was required. For backward migration to be possible, there must be no uncommitted units of recovery present at the end of restart. During restart, however, a decision was made not to force commit a detected indoubt unit of work. The decision is based on the response to message `CSQR021D`, or by the presence of a service parm which prevents the `CSQR021D` WTOR from being issued.

System action

Queue manager restart is terminated.

System programmer response

Either restart the queue manager with a higher level of code so that backward migration is not required, or, allow indoubt units of work to be force committed during restart.

00D9AAAA

This reason code identifies additional data stored in the system diagnostic work area (SDWA) variable recording area (VRA) following an error during backout-UR.

System action

Data is stored in the field indicated by VRA key 38 following the EBCDIC string 'RMC-COMMIT/BACKOUT'. This information is useful for IBM service personnel.

System programmer response

Quote this code, and the contents of the VRA field indicated by key 38 when contacting your IBM support center.

00D9BBBB

This reason code identifies additional data stored in the system diagnostic work area (SDWA) variable recording area (VRA) following an error during begin-UR.

System action

Data is stored in the field indicated by VRA key 38. This information is useful for IBM service personnel.

System programmer response

Quote this code, and the contents of the VRA field indicated by key 38 when contacting your IBM support center.

00D9CCCC

This reason code identifies additional data stored in the system diagnostic work area (SDWA) variable recording area (VRA) following an error during commit-UR.

System action

Data is stored in the field indicated by VRA key 38 following the EBCDIC string 'RMC-COMMIT/ABORT'. This information is useful for IBM service personnel.

System programmer response

Quote this code, and the contents of the VRA field indicated by key 38 when contacting your IBM support center.

00D9EEEE

This reason code identifies additional data stored in the system diagnostic work area (SDWA) variable recording area (VRA) following an error during end-UR.

System action

Data is stored in the field indicated by VRA key 38. This information is useful for IBM service personnel.

System programmer response

Quote this code, and the contents of the VRA field indicated by key 38 when contacting your IBM support center.

Storage manager codes (X'E2'): z/OS

If a storage manager reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- A description of the actions that led to the error or, if applicable, either a listing of the application program or the input string to a utility program that was being run at the time of the error.
- The console output for the period leading up to the error.
- The queue manager job log.
- The system dump resulting from the error.
- A printout of SYS1.LOGREC.
- If you are using CICS, the CICS transaction dump output.
- Appropriate WebSphere MQ, z/OS, Db2, CICS, and IMS service levels.

00E20001, 00E20002

An internal error has occurred.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Collect the items listed in "Diagnostics" and contact your IBM support center.

00E20003

A request for storage indicated that sufficient storage in the private area was not available.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Increase region size.

If you are unable to solve the problem by increasing the region size, collect the items listed in "Diagnostics" and contact your IBM support center.

00E20004

A request for storage indicated that sufficient storage was not availab le because of pool size limits.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Increase pool sizes.

If you are unable to solve the problem by increasing the pool sizes, collect the items listed in "Diagnostics" and contact your IBM support center.

00E20005, 00E20006, 00E20007, 00E20008, 00E20009

An internal error has occurred.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Collect the items listed in "Diagnostics" on page 5135 and contact your IBM support center.

00E2000A

A request to get storage was unsuccessful.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Increase the region size.

If increasing the region size does not help you solve the problem, collect the items listed in "Diagnostics" on page 5135 and contact your IBM support center.

00E2000B

A request to get storage was unsuccessful.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Increase region size.

If increasing the region size does not help you solve the problem, collect the items listed in "Diagnostics" on page 5135 and contact your IBM support center.

00E2000C

A request for storage indicated that sufficient storage was not available because of pool size limits.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Increase pool sizes.

If increasing the pool size does not help you solve the problem, collect the items listed in "Diagnostics" on page 5135 and contact your IBM support center.

00E2000D, 00E2000E

An internal error has occurred.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

The most likely cause of the problem is a storage overlay or an invalid storage request from a queue manager component. A product other than MQ could cause the storage overlay problem.

Collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E2000F, 00E20010, 00E20011, 00E20012

An internal error has occurred.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E20013

A request to get storage was unsuccessful.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Increase region size.

If increasing the region size does not help you to solve the problem, collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E20014

An internal error has occurred.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E20015

A request for storage indicated that 8K bytes of private area storage in subpool 229 was not available.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

There is probably a shortage of private area storage in the address space in which the problem occurred. Increase maximum private storage.

If increasing the maximum private storage does not solve the problem, collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E20016

A request for storage indicated that sufficient storage in subpool 229 was not available.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Increase region size.

If increasing the region size does not help you resolve the problem, collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E20017, 00E20018, 00E20019

An internal error has occurred.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E2001A

An error has occurred with the z/OS ESTAE.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested. Register 15 contains the return code from the z/OS ESTAE.

System programmer response

Collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E2001B

The 'setlock obtain' function issued a nonzero return code.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E2001D, 00E2001E

An internal error has occurred.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E2001F

There was insufficient storage in the common service area (CSA) to satisfy a request for storage.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Run the monitoring tools available at your installation to review your CSA usage.

Increase the CSA size.

If increasing the CSA size does not solve the problem, collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E20020

There was insufficient storage in the private area to satisfy a request for storage.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Increase region size.

If increasing the region size does not solve the problem, collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E20021

There was insufficient storage in the common service area (CSA) to satisfy a request for storage.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Run the monitoring tools available at your installation to review your CSA usage.

Increase the CSA size.

If increasing the size of the CSA does not solve the problem, collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E20022

There was insufficient storage in the common service area (CSA) to satisfy a request for storage.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Run the monitoring tools available at your installation to review your CSA usage.

Increase the CSA size.

If increasing the size of the CSA does not solve the problem, collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E20023

There was insufficient storage in the private area was to satisfy a request for storage.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Increase region size.

If increasing the region size does not solve the problem, collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E20024

There was insufficient storage in the common service area (CSA) to satisfy a request for storage.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Run the monitoring tools available at your installation to review your CSA usage.

Increase the CSA size.

If increasing the CSA size does not solve the problem, collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E20025

There was insufficient storage in the common service area (CSA) to satisfy a request for storage.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Run the monitoring tools available at your installation to review your CSA usage.

Increase the CSA size.

If increasing the CSA size does not solve the problem, collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E20026

A request for storage indicated that 4K bytes of private area storage in subpool 229 was not available.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

There is probably a shortage of private area storage in the address space in which the problem occurred. Increase region size.

If increasing the region size does not solve the problem, collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E20027, 00E20028, 00E20029, 00E2002A

An internal error has occurred.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Collect the items listed in “Diagnostics” on page 5135 and contact your IBM support center.

00E2002B

This reason code code is used to force percolation when an error is encountered while in storage manager code and the storage manager has been called recursively.

System programmer response

Refer to the originating error code.

00E20042, 00E20043, 00E20044, 00E20045

An internal error has occurred.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Collect the items listed in "Diagnostics" on page 5135 and contact your IBM support center.

00E20046

There was insufficient storage in a 64-bit storage pool to satisfy a request.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Increase the MEMLIM for the queue manager and restart it. If the problem persists collect the items listed in "Diagnostics" on page 5135 and contact your IBM support center.

00E20047

An internal error has occurred.

System action

The invoker is abnormally terminated. Diagnostic information is recorded in SYS1.LOGREC, and a dump is requested.

System programmer response

Collect the items listed in "Diagnostics" on page 5135 and contact your IBM support center.

Timer services codes (X'E3'): z/OS

00E30001

An internal error has occurred.

System programmer response

Collect the system dump, any trace information gathered and the related SYS1.LOGREC entries, and contact your IBM support center.

00E30002

This reason code was issued because an attempt to call the z/OS macro STIMERM was unsuccessful. The return code from STIMERM is in register 9.

System programmer response

Analyze the system dump, correct the problem from the information contained in the dump, and restart the queue manager.

For information about the STIMERM macro, see the *MVS Programming: Assembler Services Reference* manual.

Agent services codes (X'E5'): z/OS

If an agent services reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- A description of the actions that led to the error or, if applicable, either a listing of the application program or the input string to a utility program that was being run at the time of the error.
- The console output for the period leading up to the error.
- The queue manager job log.
- The system dump resulting from the error.
- A printout of SYS1.LOGREC.
- If you are using CICS, the CICS transaction dump output.
- Appropriate WebSphere MQ, z/OS, Db2, CICS, and IMS service levels.

00E50001, 00E50002

An internal error has occurred.

System action

The requesting execution unit is ended abnormally.

System programmer response

Collect the items listed in "Diagnostics" and contact your IBM support center.

00E50004, 00E50005, 00E50006, 00E50007, 00E50008, 00E50009, 00E50012

An internal error has occurred.

System action

The requesting execution unit is ended abnormally. A record is written to SYS1.LOGREC and an SVC dump is requested.

System programmer response

Collect the items listed in "Diagnostics" and contact your IBM support center.

00E50013

An MQ execution unit has been ended abnormally.

System action

The agent CANCEL processing continues.

System programmer response

This reason code might be issued as a result of any abnormal termination of a connected task, or a STOP QMGR MODE(FORCE) command. No further action is required in such cases.

If the error results in the termination of the queue manager, and you are unable to resolve the problem, collect the items listed in "Diagnostics" and contact your IBM support center.

00E50014

An internal error has occurred.

System action

An entry is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50015

An internal error has occurred.

System action

The operation is retried once. If this is not successful, the queue manager is terminated with reason code X'00E50054'.

A SYS1.LOGREC entry and an SVC dump are taken.

System programmer response

Restart the queue manager if necessary.

Collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50029

The agent services function which establishes the MQ tasking structure ends abnormally with this reason code following the detection of a load module which was loaded without the 31-bit addressing capability. This is preceded by message CSQV029E.

System action

Queue manager start-up is terminated.

System programmer response

See message CSQV029E.

00E50030, 00E50031, 00E50032, 00E50035, 00E50036

An internal error has occurred.

System action

The requesting execution unit is ended abnormally. The error is recorded on SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50040

Queue manager termination was invoked following an unrecoverable error while processing a terminate allied agent request at the *thread*, or *identify* level.

System action

The queue manager is terminated.

System programmer response

Restart the queue manager.

Scan the system log and the contents of SYS1.LOGREC for MQ errors occurring immediately before the system termination message CSQV086E. Follow the problem determination procedures for the specific errors. If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50041

Queue manager termination was invoked following an unrecoverable error while processing a terminate agent request.

System action

The queue manager is terminated.

System programmer response

Restart the queue manager.

Scan the system log and the contents of SYS1.LOGREC for MQ errors occurring immediately before the system termination message CSQV086E. Follow the problem determination procedures for the specific errors. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50042, 00E50044

An internal error has occurred.

System action

The current execution unit is ended abnormally. A record is written to SYS1.LOGREC and an SVC dump is requested.

System programmer response

Collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50045

Queue manager termination was invoked following an unrecoverable error while processing a create allied agent service request at the *thread*, or *identify* level.

System action

The queue manager is terminated.

System programmer response

Restart the queue manager.

Scan the system log and the contents of SYS1.LOGREC for MQ errors occurring immediately before the termination message CSQV086E. Follow the problem determination procedures for the specific errors. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50046

Queue manager termination was invoked following an unrecoverable error while processing a create agent structure request.

System action

The queue manager is terminated.

System programmer response

Restart the queue manager.

Scan the system log and the contents of SYS1.LOGREC for MQ errors occurring immediately before the termination message CSQV086E. Follow the problem determination procedures for the specific errors. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50047

An internal error has occurred.

System action

The queue manager is terminated.

System programmer response

Restart the queue manager.

Scan the system log and the contents of SYS1.LOGREC for MQ errors occurring immediately before the termination message CSQV086E. Follow the problem determination procedures for the specific errors. If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50050

An internal error has occurred.

System action

The requesting execution unit is ended abnormally.

An X'00E50054' recovery reason code is placed in the SDWACOMU field of the SDWA, indicating that synchronization services was responsible for queue manager termination.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50051

An internal error has occurred.

System action

The queue manager is ended abnormally with a X'5C6' completion code and this reason code.

An X'00E50054' recovery reason code is placed in the SDWACOMU field of the SDWA indicating that synchronization services was responsible for queue manager termination.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50052

The z/OS cross-memory lock (CML) could not be released.

System action

The queue manager is ended abnormally with a X'5C6' completion code and this reason code.

An X'00E50054' recovery reason code is placed in the SDWACOMU field of the SDWA indicating that synchronization services was responsible for queue manager termination.

A record is written to SYS1.LOGREC and an SVC dump is produced.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50054

The queue manager is ended abnormally by the synchronization services recovery routine when an unrecoverable error is encountered during recovery processing for the SUSPEND, CANCEL, RESUME, or SRB REDISPATCH functions. This is a queue manager termination reason code.

One of the following conditions was encountered during recovery processing for the requested function:

- Unable to complete resume processing for an SRB mode execution unit that was suspended at time of error

- Errors were encountered during primary recovery processing causing entry to the secondary recovery routine
- Recovery initiated retry to mainline suspend/resume code caused retry recursion entry into the functional recovery routine
- Unable to obtain or release the cross-memory lock (CML) of the queue manager address space either during mainline processing or during functional recovery processing (for example, reason code X'00E50052')

System action

The queue manager is terminated. This reason code is associated with a X'6C6' completion code indicating that synchronization services was responsible for termination.

System programmer response

Restart the queue manager.

Scan the system log and the contents of SYS1.LOGREC for MQ errors occurring immediately before the system termination message CSQV086E. Follow the problem determination procedures for the specific errors. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50055

The synchronization services functional recovery routine was unable to successfully complete resume processing for a suspended TCB mode execution unit. The resume processing was requested by the CANCEL or RESUME functions.

System action

Because the suspended TCB mode execution unit must not be permitted to remain in a suspended state, the recovery routine invokes the z/OS CALLRTM (TYPE=ABTERM) service to end the execution unit abnormally with a X'6C6' completion code. Depending upon which execution unit was terminated, the queue manager might be ended abnormally.

System programmer response

Restart the queue manager if necessary.

Scan the system log and the contents of SYS1.LOGREC for MQ errors occurring immediately before the end of the execution unit. Follow the problem determination procedures for the specific errors. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50059

An internal error has occurred.

System action

If the module detecting the error is CSQVSDC0, it will be retried once. If validation is unsuccessful, the queue manager is terminated abnormally with a X'00E50054' reason code.

A SYS1.LOGREC entry and an SVC dump are requested.

System programmer response

Restart the queue manager.

Collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50062

An internal error has occurred.

System action

The allied task is ended abnormally.

System programmer response

Collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50063

An internal error has occurred.

System action

The task is ended abnormally.

System programmer response

Collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50065

An internal error has occurred.

System action

The execution unit is ended abnormally.

System programmer response

Collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50069

This reason code is issued during recovery processing for the suspend function when executing in SRB mode under the recovery routine established by the z/OS SRBSTAT(SAVE) service. Because the recovery routine established by this service is the only routine in the FRR stack at the time of error, normal RTM percolation to the invoking resource manager recovery routine is not possible.

After recovery processing for the initial error has successfully completed, the RTM environment is exited through retry to a routine that restores the original FRR stack. This routine terminates abnormally with completion code X'5C6' and this reason code. This causes entry into the original recovery routine established during suspend initialization.

System action

After this is intercepted by the original suspend recovery routine, a SYS1.LOGREC entry and SVC dump are requested to document the original error. The original recovery reason code is placed in the SDWACOMU field of the SDWA indicating the actions performed during recovery processing of the initial error. Control is then returned to the invoking resource manager's recovery routine through RTM percolation.

System programmer response

Because this is used only to permit the transfer of the initial recovery reason code to the invoking resource manager's recovery routine, no further recovery actions are required for this reason code. Diagnostic information for the initial error encountered can be obtained through the SYS1.LOGREC and SVC dump materials provided.

00E50070

To enable an internal task to terminate itself, the task has ended abnormally. This is not necessarily an error.

System action

The task is ended abnormally.

If the service task is ended abnormally with a completion code of X'6C6', no SVC dump is taken.

System programmer response

The error should be ignored if it happens in isolation, however, if it occurs in conjunction with other problems, these problems should be resolved.

If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50071

An internal error has occurred.

System action

The internal task is ended abnormally.

System programmer response

Collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50072

An internal error has occurred.

System action

The queue manager is ended abnormally.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50073

An internal error has occurred.

System action

The current execution unit is ended abnormally. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50074

This reason code is issued in response to a nonzero return code from ATTACH during an attempt to create an internal task.

System action

The ATTACH is retried. A record is written to SYS1.LOGREC, and an SVC dump is requested. If a problem occurs again, the queue manager is terminated.

System programmer response

Restart the queue manager if necessary.

Register 2, in the SDWA, contains the return code from the ATTACH request. If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50075, 00E50076, 00E50077, 00E50078

An internal error has occurred.

System action

The requesting execution unit is terminated. The queue manager might also be terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager if necessary.

Collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50079

An internal error has occurred. This can occur if the allied address space is undergoing termination.

System action

The requesting execution unit is ended abnormally. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50080, 00E50081

An internal error has occurred.

System action

An SVC dump is requested specifying a completion code of X'5C6' and this reason code. No record is written to SYS1.LOGREC. Execution continues.

System programmer response

Collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50094, 00E50095, 00E50096, 00E50097, 00E50100

An internal error has occurred.

System action

The requesting recovery routine is ended abnormally. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50101

MQ was unable to establish an ESTAE.

System action

The error is passed on to a subsystem support subcomponent (SSS) ESTAE. Probably, the queue manager is ended abnormally. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

The inability to establish an ESTAE is normally due to insufficient free space in the local system queue area (LSQA) for an ESTAE control block (SCB). If necessary, increase the size of the queue manager address space.

Restart the queue manager.

Review the associated SVC dump for usage and free areas in the LSQA subpools belonging to the system services address space. If you are unable to solve the problem, collect the items listed in “Diagnostics” on page 5142 and contact your IBM support center.

00E50102

An unrecoverable error occurred while canceling all active agents during processing of the STOP QMGR MODE(FORCE) command. This is a queue manager termination reason code.

System action

The queue manager is ended abnormally. A record is written to SYS1.LOGREC.

System programmer response

Restart the queue manager.

You might find the items listed in “Diagnostics” on page 5142 useful in resolving the problem. Review the SYS1.LOGREC entries for errors immediately preceding queue manager termination.

00E50500

A z/OS LOCAL or CML lock could not be obtained during queue manager abnormal termination processing.

System action

The execution unit is ended abnormally. The error is recorded on SYS1.LOGREC, and abnormal queue manager termination is completed under a different execution unit if possible.

System programmer response

Restart the queue manager if necessary.

You might find the items listed in “Diagnostics” on page 5142 useful in resolving the problem.

00E50501

A z/OS LOCAL or CML lock could not be released during queue manager abnormal termination processing.

System action

The execution unit is ended abnormally. The error is recorded on SYS1.LOGREC. Queue manager termination is completed under a different execution unit if possible.

System programmer response

Restart the queue manager.

You might find the items listed in “Diagnostics” on page 5142 useful in resolving the problem.

00E50502

A z/OS LOCAL lock could not be obtained during queue manager abnormal termination processing.

System action

The execution unit is ended abnormally. The error is recorded on SYS1.LOGREC, and abnormal queue manager termination is completed under a different execution unit if possible.

System programmer response

Restart the queue manager.

You might find the items listed in “Diagnostics” on page 5142 useful in resolving the problem.

00E50503

A z/OS LOCAL lock could not be released during queue manager abnormal termination processing.

System action

The execution unit is ended abnormally. The error is recorded on SYS1.LOGREC, and abnormal queue manager termination is completed under a different execution unit if possible.

System programmer response

Restart the queue manager.

You might find the items listed in "Diagnostics" on page 5142 useful in resolving the problem.

00E50504

This reason code is used to define the format of the information recorded in the SDWA variable recording area (VRA) by the queue manager termination processor. The code identifies additional information provided in the VRA for errors encountered in module CSQVATRM.

System action

Recording of the error encountered during queue manager termination continues.

System programmer response

None.

00E50505

This reason code is used to define the format of the information recorded in the SDWA variable recording area (VRA). The code identifies additional information provided in the VRA for errors encountered in module CSQVATR4.

System action

Recording of the error encountered during queue manager termination continues.

System programmer response

None.

00E50701

A problem occurred during Commit Phase-1. This is used to effect backout, deallocation, and end-UR processing.

System action

The queue manager is ended abnormally. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50702

An error occurred while processing in SRB mode which could not be recovered.

SRB mode processing is often used internally by the queue manager to ensure data integrity and consistency of internal state. Where recovery is not possible, the queue manager is terminated with this reason code.

Most occurrences are due to internal errors which should be reported to IBM service for further investigation.

The error is also known to occur where log data sets have been reformatted, without reformatting the pagesets (so they still contain active data). This situation can be resolved by user action.

System action

The queue manager is ended abnormally with this reason code. An SVC dump of the original error was requested by the recovery routine for CSQVEUS2 and a record written to SYS1.LOGREC.

System programmer response

Restart the queue manager.

Scan the SYS1.LOGREC entries looking for one or more MQ errors immediately prior to the queue manager termination. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50703

This queue manager termination reason code is used following an error while attempting to resume a suspended execution unit. The successful completion of resume processing was 'indoubt'.

System action

The queue manager is ended abnormally. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

You might find the items listed in "Diagnostics" on page 5142 useful in resolving the problem.

00E50704

An internal error has occurred.

System action

The queue manager is terminated with this reason code. Additionally, if no SDWA was provided to the recovery routine, a dump is requested.

System programmer response

Restart the queue manager.

Scan the SYS1.LOGREC entries looking for one or more MQ errors immediately prior to the queue manager termination. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50705

An internal error has occurred.

System action

The queue manager is ended abnormally.

System programmer response

Restart the queue manager.

Collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50706

An internal error has occurred.

System action

The queue manager is terminated with this reason code. Additionally, if no SDWA was provided to the recovery routine, a dump is requested. A record is written to SYS1.LOGREC.

System programmer response

Restart the queue manager.

Scan the SYS1.LOGREC entries looking for one or more MQ errors immediately prior to the queue manager termination. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50707

An ESTAE could not be established.

System action

The queue manager is ended abnormally. A record is written to SYS1.LOGREC.

System programmer response

Review the usage and the free areas in the LSQA subpool of the queue manager address space. If necessary, increase the private area size of the address space.

Restart the queue manager.

If queue manager termination was requested by module CSQVRCT, a standard SVC dump was requested. If insufficient private storage is the cause of the problem, other MQ resource managers might have ended abnormally.

If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50708

An error occurred while connecting an allied agent to the queue manager address space. The connection must complete so that the allied agent can be terminated.

System action

The queue manager is terminated with this reason code. An SVC dump of the original error was requested and a record entered into SYS1.LOGREC.

System programmer response

Restart the queue manager.

Scan the SYS1.LOGREC entries looking for one or more MQ errors immediately prior to the queue manager termination.

00E50709

An internal error has occurred.

System action

The queue manager is ended abnormally.

System programmer response

Restart the queue manager.

Scan the SYS1.LOGREC entries for one or more MQ errors occurring immediately prior to the queue manager termination. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50710

An internal error has occurred.

System action

The queue manager is terminated with this reason code. An SVC dump of the original error was requested and a record entered into SYS1.LOGREC.

System programmer response

Restart the queue manager.

Scan the SYS1.LOGREC entries looking for one or more MQ errors immediately prior to the queue manager termination. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50711

An internal error has occurred.

System action

The queue manager is terminated with this reason code. An SVC dump of the original error was requested and a record entered into SYS1.LOGREC.

System programmer response

Restart the queue manager.

Scan the SYS1.LOGREC entries looking for one or more MQ errors immediately prior to the queue manager termination. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50712

An error occurred in a latch manager function attempting to terminate the holder of an MQ latch. The holder's task has been set nondispatchable by z/OS and a CALLRTM to terminate this task was unsuccessful.

System action

The queue manager is terminated with this reason code. An SVC dump of the error is requested and a record entered into SYS1.LOGREC. Register 3 at time of error contains the latch-holder's TCB address in the home address space and register 4 contains the return code from CALLRTM.

System programmer response

Restart the queue manager.

You might find the items listed in "Diagnostics" on page 5142 useful in resolving the problem. Scan the SYS1.LOGREC entries for one or more MQ errors immediately prior to the queue manager termination.

00E50713

An internal error has occurred.

System action

The queue manager is ended abnormally. An SVC dump is requested by the queue manager termination processor and a record is written to SYS1.LOGREC.

System programmer response

Restart the queue manager.

Scan the SYS1.LOGREC entries for one or more MQ errors occurring immediately prior to the queue manager termination. It might be necessary to analyze the SVC dump requested. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50715

Queue manager termination was requested following an unrecoverable error in an SRB mode execution unit.

System action

The SRB-related task was ended abnormally as a result of SRB to TCB percolation. The queue manager is ended abnormally.

System programmer response

Restart the queue manager.

You might find the items listed in "Diagnostics" on page 5142 useful in resolving the problem. Scan the SYS1.LOGREC entries for one or more MQ errors occurring immediately prior to the queue manager termination.

00E50717

An internal error has occurred.

System action

The queue manager is ended abnormally.

System programmer response

Restart the queue manager.

Scan the SYS1.LOGREC entries for one or more MQ errors occurring immediately prior to the queue manager termination. If an error preceded the queue manager termination request, diagnostic information can be obtained through SYS1.LOGREC and SVC dump materials. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50719

An internal error has occurred.

System action

The queue manager is ended abnormally.

System programmer response

Restart the queue manager.

Scan the SYS1.LOGREC entries for one or more MQ errors occurring immediately prior to the queue manager termination. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5142 and contact your IBM support center.

00E50725

Queue manager termination was requested because of an unrecovered error in a scheduled SRB-mode execution unit.

System action

The SRB-related task was ended abnormally, due to SRB to TCB percolation. The queue manager is ended abnormally.

System programmer response

Restart the queue manager.

You might find the items listed in "Diagnostics" on page 5142 useful in resolving the problem. Scan the SYS1.LOGREC entries for one or more MQ errors occurring immediately prior to the queue manager termination. If necessary, analyze the SVC dump requested by queue manager termination.

00E50727

A secondary error occurred during agent services functional recovery processing. This is a queue manager termination reason code.

System action

The queue manager is ended abnormally.

System programmer response

Restart the queue manager.

You might find the items listed in "Diagnostics" on page 5142 useful in resolving the problem. Scan the SYS1.LOGREC entries for one or more MQ errors occurring immediately prior to the queue manager termination.

Instrumentation facilities codes (X'E6'): z/OS

If an instrumentation facilities reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- The console output for the period leading up to the error.
- The system dump resulting from the error.
- A printout of SYS1.LOGREC.
- Appropriate WebSphere MQ, z/OS, Db2, CICS, and IMS service levels.

00E60008

An internal error has occurred.

System action

The function being traced is ended abnormally. The queue manager remains operational.

System programmer response

Collect the items listed in "Diagnostics" and contact your IBM support center.

00E60017

This code is an internal code used by the dump formatter.

System action

The request is ended abnormally.

System programmer response

Collect the items listed in "Diagnostics" and contact your IBM support center.

00E60085, 00E60086, 00E60087, 00E60088, 00E60089

An internal error has occurred.

System action

The request is end abnormally.

System programmer response

Collect the items listed in "Diagnostics" and contact your IBM support center.

00E60100 through 00E60199

The reason codes X'00E60100' through X'00E60199' are used by the instrumentation facility component (IFC) when a trace event occurs for which IBM service personnel have requested a dump using the IFC selective dump service aid.

System action

The agent might be retried or terminated, depending upon the serviceability dump request.

System programmer response

The reason code is issued on the occurrence of a specified trace event. An SVC dump is taken to the SYS1.DUMPxx data set. Problem determination methods depend on the condition that IBM service personnel are attempting to trap.

00E60701

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5156 and contact your IBM support center.

00E60702, 00E60703

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5156 and contact your IBM support center.

Distributed queuing codes (X'E7'): z/OS

If a distributed queuing reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- A description of the actions that led to the error or, if applicable, either a listing of the application program or the input string to a utility program that was being run at the time of the error.
- The channel definitions being used
- If the error affected a message channel agent, a listing of any user channel exit programs used by the message channel agent.
- The console output for the period leading up to the error.
- The queue manager job log.
- The channel initiator job log.
- The system dump resulting from the error.
- Appropriate WebSphere MQ, z/OS, Db2, CICS, and IMS service levels.

00E70001

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5157 and contact your IBM support center.

00E70002

No adapter subtasks are active. They have failed many times and so have not been restarted.

System action

The channel initiator terminates.

System programmer response

Investigate the adapter subtask failure problems, as reported in the messages associated with each failure.

00E70003

No dispatchers are active. Either all the dispatchers failed to start, or all the dispatchers have failed many times and so have not been restarted.

System action

The channel initiator terminates.

System programmer response

Investigate the dispatcher failure problems, as reported in the messages associated with each failure.

00E70004

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5157 and contact your IBM support center.

00E70007

An attempt by an adapter subtask to obtain some storage failed.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Increase the size of the channel initiator address space, or reduce the number of dispatchers, adapter subtasks, SSL server subtasks, and active channels being used.

00E70008, 00E70009, 00E7000A

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5157 and contact your IBM support center.

00E70011

The channel initiator was unable to load the module CSQXBENT.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Check the console for messages indicating why CSQXBENT was not loaded. Ensure that the module is in the required library, and that it is referenced correctly.

The channel initiator attempts to load this module from the library data sets under the STEPLIB DD statement of its started task JCL procedure xxxxCHIN.

00E70013

Some adapter subtasks were requested, but none could be attached.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Investigate the adapter subtask attach problems, as reported in the messages associated with each failure. If you cannot resolve the problems, collect the items listed in "Diagnostics" on page 5157 and contact your IBM support center.

00E70015

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in "Diagnostics" on page 5157 and contact your IBM support center.

00E7001D

During startup, the channel initiator was unable obtain some storage below 16M.

System action

The channel initiator ends.

System programmer response

Investigate the cause of the problem.

00E7001E, 00E7001F

An internal error has occurred.

System action

The channel initiator terminates with completion code X'5C6'.

System programmer response

Restart the channel initiator.

Collect the items listed in "Diagnostics" on page 5157 and contact your IBM support center.

00E70020

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Check the console for preceding error messages. If the problem cannot be resolved, collect the items listed in "Diagnostics" on page 5157 and contact your IBM support center.

00E70021, 00E70022, 00E70023, 00E70024, 00E70025

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5157 and contact your IBM support center.

00E70031

An internal error has occurred. A lock is currently held by a task that has terminated.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Determine why the terminated task did not free the lock. This might be due to a previous error. If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5157 and contact your IBM support center.

00E70032

An internal error has occurred. An attempt to update information held in the coupling facility failed.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5157, together with details of the queue-sharing group and of the queue managers active, as well as the queue managers defined to the queue-sharing group at the time. This information can be obtained by entering the following z/OS commands:

```
D XCF,GRP
```

to display a list of all queue-sharing groups in the coupling facility

```
D XCF,GRP,qsg-name,ALL
```

to display status about the queue managers defined to the queue-sharing group.

Contact your IBM support center.

00E70033

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5157 and contact your IBM support center.

00E70052

No SSL server subtasks are active. They have failed many times and so have not been restarted.

System action

The channel initiator terminates.

System programmer response

Investigate the SSL server subtask failure problems, as reported in the messages associated with each failure.

00E70053

Some SSL server subtasks were requested, but none could be attached.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Investigate the SSL server subtask attach problems, as reported in the messages associated with each failure. If you cannot resolve the problems, collect the items listed in “Diagnostics” on page 5157 and contact your IBM support center.

00E7010C

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5157 and contact your IBM support center.

00E7010E

The dispatcher detected an inconsistency in the linkage stack.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

The most likely cause is incorrect use of the linkage stack by a user exit; exits must issue any MQ API calls and return to the caller at the same linkage stack level as they were entered. If exits are not being used, or if they do not use the linkage stack, collect the items listed in “Diagnostics” on page 5157 and contact your IBM support center.

00E7010F, 00E7014A

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5157 and contact your IBM support center.

00E7014C

An internal error has occurred. This can be caused by the channel initiator failing to stop when running against a previous instance of the queue manager and attempting to connect to a later instance of the queue manager.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5157, terminate then restart the channel initiator and contact your IBM support center.

00E7014D

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5157 and contact your IBM support center.

00E7014F

An internal error has occurred. This is normally as a result of some previous error.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Check the console for preceding error messages reporting a previous error, and take the appropriate action for resolving that error. If there is no previous error, collect the items listed in “Diagnostics” on page 5157 and contact your IBM support center.

00E7015A, 00E70214, 00E70216, 00E70226, 00E70231, 00E70232, 00E70233, 00E70501, 00E70522, 00E70543, 00E70546, 00E70553

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5157 and contact your IBM support center.

00E70054, 00E70055, 00E70056

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5157 and contact your IBM support center.

00E70057, 00E70058

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5157 and contact your IBM support center.

00E70708

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5157 and contact your IBM support center.

Initialization procedure and general services codes (X'E8'): z/OS

If an initialization procedure reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- The console output for the period leading up to the error.
- The queue manager job log.
- The system dump resulting from the error.
- A printout of SYS1.LOGREC.
- The system parameter load module.
- The initialization procedure.
- The started task JCL procedure for this queue manager.
- Appropriate WebSphere MQ, z/OS, Db2, CICS, and IMS service levels.

00E80001

An internal error has occurred.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” and contact your IBM support center.

00E80002

The queue manager address space was not started correctly or an error occurred during z/OS IEFSSREQ processing.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested. Register 9 contains the address of an 8-byte field that contains the following diagnostic information:

- Bytes 1 through 4 - subsystem name
- Bytes 5 through 8 - contents of register 15 that contains the return code set by the z/OS IEFSSREQ macro

System programmer response

You might find the items listed in “Diagnostics” useful in resolving the problem.

00E80003, 00E80004, 00E80005, 00E80006

An internal error has occurred.

System action

A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Collect the items listed in “Diagnostics” and contact your IBM support center.

00E8000E

An ESTAE could not be established for the queue manager address space control task.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested. Register 9 contains the address of a 4-byte field that contains the ESTAE macro return code.

System programmer response

Restart the queue manager.

You might find the items listed in "Diagnostics" on page 5163 useful in resolving the problem.

00E8000F

Invalid startup parameters were specified. This was probably caused by an attempt to start the queue manager by som means other than a START QMGR command.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5163 and contact your IBM support center.

V 9.0.3

00E80010

An invalid product was specified. This abend is preceded by one or more instances of message CSQY038E. See this message for more details.

System action

The queue manager is terminated.

System programmer response

Locate the related CSQY038E messages for the queue manager, and correct the issue described in each of those messages.

00E80011

The address space could not be made non-swappable.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

You might find the items listed in "Diagnostics" on page 5163 useful in resolving the problem.

00E80012

An internal error has occurred.

System programmer response

Collect the items listed in "Diagnostics" on page 5163 and contact your IBM support center.

00E80013, 00E8001F, 00E8002F

An internal error has occurred.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” on page 5163 and contact your IBM support center.

00E80031

An unsupported input parameter was detected for allied address space initialization.

System action

The caller's task is ended abnormally. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Collect the items listed in “Diagnostics” on page 5163 and contact your IBM support center.

00E80032

An unsupported input parameter was detected for allied address space termination.

System action

The caller's task is ended abnormally. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Collect the items listed in “Diagnostics” on page 5163 and contact your IBM support center.

00E80033

This reason code accompanies a X'6C6' completion code. This module detected that the queue manager was terminating.

System action

The caller's task is ended abnormally with code X'6C6'. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

You might find the items listed in “Diagnostics” on page 5163 useful in resolving the problem.

00E8003C

An internal error has occurred.

System action

The caller's task is ended abnormally. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Collect the items listed in “Diagnostics” on page 5163 and contact your IBM support center.

00E8003D

An internal error has occurred.

System action

Abnormal termination of the queue manager is initiated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” on page 5163 and contact your IBM support center.

00E8003E

An ESTAE could not be established in an address space about to be initialized as an MQ allied address space.

System action

The caller's task is ended abnormally. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5163 and contact your IBM support center.

00E8003F

An internal error has occurred.

System action

The caller's task is ended abnormally. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Collect the items listed in “Diagnostics” on page 5163 and contact your IBM support center.

00E80041

An internal error has occurred.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” on page 5163 and contact your IBM support center.

00E80042, 00E8004F

An internal error has occurred.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” on page 5163 and contact your IBM support center.

00E80051

An error was detected in the command that was used to start the queue manager.

System action

The queue manager is terminated.

System programmer response

Reenter the command if it was entered incorrectly.

If you are unable to resolve the problem, contact your IBM support center.

00E80052, 00E80053, 00E80054, 00E80055

An internal error has occurred.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

Collect the items listed in "Diagnostics" on page 5163 and contact your IBM support center.

00E80057

An error occurred while trying to start a queue manager address space. One possible cause of this problem is an error in the started task JCL procedure for the queue manager.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

You might find the items listed in "Diagnostics" on page 5163 useful in resolving the problem.

00E80058

An error occurred during command prefix registration.

System action

The queue manager ends abnormally.

System programmer response

See the accompanying CSQYxxx messages for information about the cause of the problem.

Restart the queue manager after correcting the problem.

00E8005F, 00E80061, 00E8006F, 00E8007F

An internal error has occurred.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

Collect the items listed in "Diagnostics" on page 5163 and contact your IBM support center.

00E80081

An invalid load module was detected.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested. Register 9 contains the address of an 8-byte field that contains the name of the module in error.

System programmer response

Check that the installation process was successful.

Restart the queue manager after resolving the problem.

If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5163 and contact your IBM support center.

00E80084

A resource manager provided notification of an error during queue manager startup notification processing.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested. Register 9 contains the address of a 4-byte field that contains the RMID of the resource manager that requested queue manager termination.

System programmer response

Look for error messages indicating the cause of the problem.

Restart the queue manager after resolving the problem.

If you are unable to solve the problem, collect the items listed in "Diagnostics" on page 5163, together with the contents of the BSDS and a GTF trace, and contact your IBM support center.

00E8008F, 00E80091, 00E8009F, 00E800AF, 00E800B1

An internal error has occurred.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

Collect the items listed in "Diagnostics" on page 5163 and contact your IBM support center.

00E800B2

The queue manager initialization procedure found that the version of ZPARM loaded was compiled for a higher release of IBM WebSphere MQ, and specifies NEWFUNC.

System action

Startup is terminated.

System programmer response

Check whether the correct ZPARM has been loaded at initialization.

The existence of this version of ZPARM implies that the queue manager has been running at a higher release of the product.

Check to see if the queue manager has been started on a higher release of the product. If this is the case, you have inadvertently started IBM WebSphere MQ with the wrong version of the product libraries.

It might still be possible to rebuild ZPARM using the macros from SCSQMACS for the current release of the product.

Note, however, that a queue manager that has been operating with `OPMODE(NEWFUNC,nnn)` at a higher release of the product cannot be subsequently started at a lower release of the product.

00E800CE

An ESTAE could not be established.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested. Register 9 contains the address of a 4-byte field that contains the ESTAE macro return code.

System programmer response

Restart the queue manager.

You might find the items listed in "Diagnostics" on page 5163 useful in resolving the problem.

00E800D1

An internal error has occurred.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

Collect the items listed in "Diagnostics" on page 5163 and contact your IBM support center.

00E800D2

An error was encountered while attempting to obtain the z/OS LOCAL lock.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

You might find the items listed in "Diagnostics" on page 5163 useful in resolving the problem.

00E800D3

An error was encountered while attempting to release the z/OS LOCAL lock.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

You might find the items listed in "Diagnostics" on page 5163 useful in resolving the problem.

00E800DF

An internal error has occurred.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” on page 5163 and contact your IBM support center.

00E80100

The queue manager was ended abnormally because the queue manager address space control task ESTAE was entered. This reason code is issued for all completion codes, except for the X'5C6' completion code.

The queue manager is unable to determine the cause of the error.

System action

Termination of the queue manager is initiated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager after resolving the problem.

The subcomponent that caused the error is unknown. This reason code might be returned if the queue manager is unable to find the system parameter load module you specified on the START QMGR command (the default name is CSQZPARM). Check that the module you specified is available.

This reason code is also issued if the queue manager is canceled by the z/OS command CANCEL. If this is the case, determine why the queue manager was canceled.

You might find the items listed in “Diagnostics” on page 5163, together with the contents of the BSDS and a GTF trace, useful in resolving the problem.

00E8011D

An internal error has occurred.

System action

Termination of queue manager is initiated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” on page 5163 and contact your IBM support center.

00E8011E

The allied address space task primary ESTAE detected that the secondary ESTAE could not be established.

System action

Abnormal termination of allied address space is continued. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

You might find the items listed in “Diagnostics” on page 5163 useful in resolving the problem.

00E8011F

The allied address space task primary ESTAE was entered without a subsystem diagnostic work area (SDWA) provided by z/OS RTM.

System action

Abnormal termination of the allied address space is continued. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

You might find the items listed in “Diagnostics” on page 5163 useful in resolving the problem.

00E8012D

An internal error has occurred.

System action

Abnormal termination of queue manager is initiated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” on page 5163 and contact your IBM support center.

00E8012F

The allied address space task secondary ESTAE was entered without a subsystem diagnostic work area (SDWA) provided by z/OS .

System action

Continue with the abnormal termination of the allied address space. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

You might find the items listed in “Diagnostics” on page 5163 useful in resolving the problem.

00E80130

The FRR that protects the START QMGR/STOP QMGR command processor function was entered while a valid STOP QMGR command was being processed.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

You might find the items listed in “Diagnostics” on page 5163 useful in resolving the problem.

00E80140

An internal error has occurred.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” on page 5163 and contact your IBM support center.

00E80150, 00E80151

An invalid module was detected.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested. Register 9 contains the address of a 12-byte field that contains the following diagnostic information:

- Bytes 1 through 8 contain the name of the load module that contains the initialization entry point list with the invalid entry

System programmer response

Restart the queue manager after resolving the problem.

Check that the installation process was successful. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5163 and contact your IBM support center.

00E8015F

An internal error has occurred.

System action

The queue manager is terminated. A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Restart the queue manager.

Collect the items listed in "Diagnostics" on page 5163 and contact your IBM support center.

00E80160

The queue manager initialization procedures found that a load module had an invalid AMODE or RMODE attribute.

System action

Queue manager startup is terminated.

System programmer response

See message CSQY006E.

00E80161

The queue manager initialization procedures found that a load module was not at the correct level for the version of the queue manager that was being started.

System action

Queue manager startup is terminated.

System programmer response

See message CSQY010E.

00E80162

The queue manager initialization procedures found that the storage protect key was not 7. The most likely cause is that the program properties table (PPT) entry for CSQYASCP has not been specified correctly.

System action

Queue manager startup is terminated.

System programmer response

Restart the queue manager after resolving the problem.

For information about specifying the PPT entry for CSQYASCP, see Update the z/OS program properties table.

00E80163

The queue manager initialization procedures found that they were not APF authorized. The most likely cause is that one or more of the data sets in the //STEPLIB concanentation is not APF authorized.

System action

Queue manager startup is terminated.

System programmer response

Restart the queue manager after resolving the problem.

For information about APF authorization for the MQ load libraries, see APF authorize the WebSphere MQ load libraries

00E80170

An internal error has occurred.

System action

The request is ignored.

System programmer response

Collect the items listed in "Diagnostics" on page 5163 and contact your IBM support center.

System parameter manager codes (X'E9'): z/OS

If a system parameter manager reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- The console output for the period leading up to the error.
- The queue manager job log.
- The system dump resulting from the error.
- A printout of SYS1.LOGREC.
- The system parameter load module.
- The initialization procedure.
- The started task JCL procedure for this queue manager.
- Appropriate WebSphere MQ, z/OS, Db2, CICS, and IMS service levels.

00E90101

An error has occurred while trying to open MQ resources. The most likely cause is that a customized system parameter load module specified on the START QMGR command is not available.

System action

A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Check that the system parameter load module you specified on the START QMGR command (the default name is CSQZPARM) is available for use. If it is, collect the items listed in "Diagnostics" and contact your IBM support center.

00E90201

An internal error has occurred while attempting to open MQ resources.

System action

A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Collect the items listed in “Diagnostics” on page 5173 and contact your IBM support center.

00E90202

An error has occurred while attempting to open MQ resources. The most likely cause is that a customized system parameter load module specified on the START QMGR command (the default name is CSQZPARM) has been built incorrectly.

System action

A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Check that the system parameter load module that you specified is available, and that it was linked correctly. See CSQ4ZPRM for sample link-edit JCL. and for information about the system parameter modules, see Tailor your system parameter module.

Restart the queue manager. If the problem persists, collect the items listed in “Diagnostics” on page 5173 and contact your IBM support center.

00E90203

An internal error has occurred while attempting to verify descriptor control information in MQ resources.

System action

A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Collect the items listed in “Diagnostics” on page 5173 and contact your IBM support center.

00E90301

An internal error has occurred while attempting to close MQ resources.

System action

A record is written to SYS1.LOGREC, and an SVC dump is requested.

System programmer response

Collect the items listed in “Diagnostics” on page 5173 and contact your IBM support center.

Service facilities codes (X'F1'): z/OS

00F10001, 00F10002, 00F10003, 00F10004, 00F10005, 00F10006, 00F10007, 00F10008, 00F10009, 00F10010, 00F10011, 00F10012, 00F10013, 00F10014, 00F10015, 00F10016, 00F10017, 00F10018

An internal error has been detected in the CSQ1LOGP log print utility.

System action

A dump is requested. The utility ends abnormally with completion code X'5C6'.

System programmer response

Collect the following diagnostic items and contact your IBM support center:

- Utility report output
- System dump resulting from the error, if any
- The WebSphere MQ, z/OS, Db2, CICS, and IMS service levels

00F10100

An internal error has been detected in the CSQ1LOGP log print utility.

System action

A dump is requested. The utility ends abnormally with completion code X'5C6'.

System programmer response

Resubmit the job.

Contact your IBM support center if the problem persists.

00F10101

The stand-alone log read function returned an invalid RBA. See the explanation for message CSQ1211E.

System action

A dump is requested. The utility ends abnormally with completion code X'5C6'.

System programmer response

If you determine that the data set is a log data set and that it is not damaged, contact your IBM support center.

IBM MQ-IMS bridge codes (X'F2'): z/OS

If an IBM MQ-IMS bridge reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- A description of the actions that led to the error or, if applicable, either a listing of the application program or the input string to a utility program that was being run at the time of the error.
- The console output for the period leading up to the error.
- The queue manager job log.
- The IMS job logs.
- The system dump resulting from the error.
- Appropriate IBM MQ, z/OS, Db2, CICS, and IMS service levels.

00F20001, 00F20002, 00F20003, 00F20004, 00F20005, 00F20006, 00F20007, 00F20008, 00F20009, 00F2000A, 00F2000B, 00F2000C, 00F2000D, 00F2000E, 00F2000F, 00F20010, 00F20011

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5175 and contact your IBM support center.

00F20012

The IBM MQ-IMS bridge received a bad return code from IXCQUERY macro.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Registers 3 and 4 contain the return and reason codes from XCF. Refer to the *MVS Programming: Sysplex Services Reference* for information about these codes.

00F20013

The IBM MQ-IMS bridge received a bad return from IXCJOIN macro.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Registers 3 and 4 contain the return and reason codes from XCF. Refer to the *MVS Programming: Sysplex Services Reference* for information about these codes.

00F20014

The IBM MQ-IMS bridge received a bad return from IXCCREAT macro.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Registers 3 and 4 contain the return and reason codes from XCF. Refer to the *MVS Programming: Sysplex Services Reference* for information about these codes.

Use the IMS DIS OTMA command to see if the OTMACON member name is already in use. This can be caused by specifying the IMS system instead of the queue manager name in the OTMACON member name.

00F20015, 00F20016

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5175 and contact your IBM support center.

00F20017

The IBM MQ-IMS bridge received a bad return from IXCLEAVE macro.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Registers 3 and 4 contain the return and reason codes from XCF. Refer to the *MVS Programming: Sysplex Services Reference* for information about these codes.

00F20018

The IBM MQ-IMS bridge received a bad return from IXCDELET macro.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Registers 3 and 4 contain the return and reason codes from XCF. Refer to the *MVS Programming: Sysplex Services Reference* for information about these codes. Contact your IBM support center to report the problem.

00F20019, 00F2001A, 00F2001B, 00F2001C, 00F2001D, 00F2001E, 00F2001F, 00F20020, 00F20021, 00F20022

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in "Diagnostics" on page 5175 and contact your IBM support center.

00F20023

The IBM MQ-IMS bridge received a bad return code from IXCMSGO.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Registers 2 and 3 contain the return and reason codes from XCF. Refer to the *MVS Programming: Sysplex Services Reference* for information about these codes.

00F20024, 00F20026, 00F20027, 00F20029, 00F2002A, 00F2002B

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in "Diagnostics" on page 5175 and contact your IBM support center.

00F2002C

The IBM MQ-IMS bridge received a bad return code from IXCMSGO.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Registers 2 and 3 contain the return and reason codes from XCF. Refer to the *MVS Programming: Sysplex Services Reference* for information about these codes.

00F2002D, 00F2002E

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5175 and contact your IBM support center.

00F20030

The IBM MQ-IMS bridge received a bad return code from IXCMSGO.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Registers 2 and 3 contain the return and reason codes from XCF. Refer to the *MVS Programming: Sysplex Services Reference* for information about these codes.

00F20031

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5175 and contact your IBM support center.

00F20032

The IBM MQ-IMS bridge received a bad return code from IXCMSGO.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Registers 2 and 3 contain the return and reason codes from XCF. Refer to the *MVS Programming: Sysplex Services Reference* for information about these codes.

00F20035, 00F20036, 00F20037, 00F20038, 00F20039, 00F2003A, 00F2003B, 00F2003D, 00F2003E, 00F2003F, 00F20040

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in “Diagnostics” on page 5175 and contact your IBM support center.

00F20041

The IBM MQ-IMS bridge received an MQOPEN error.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Contact your IBM support center to report the problem.

00F20042

The IBM MQ-IMS bridge received an MQCLOSE error.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Contact your IBM support center to report the problem.

00F20043

The IBM MQ-IMS bridge received an MQGET error.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Contact your IBM support center to report the problem.

00F20044

The IBM MQ-IMS bridge received an MQPUT error.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Contact your IBM support center to report the problem.

00F20045

The IBM MQ-IMS bridge received an MQOPEN error.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Contact your IBM support center to report the problem.

00F20046

The IBM MQ-IMS bridge received an MQCLOSE error.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Contact your IBM support center to report the problem.

00F20047

The IBM MQ-IMS bridge received an MQGET error.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Contact your IBM support center to report the problem.

00F20048

The IBM MQ-IMS bridge received an MQPUT error.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Contact your IBM support center to report the problem.

00F20049

The IBM MQ-IMS bridge received an MQPUT1 error.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Contact your IBM support center to report the problem.

00F2004A, 00F2004B, 00F2004C, 00F2004D, 00F2004E, 00F2004F, 00F20050, 00F20051, 00F20052, 00F20053, 00F20054, 00F20055, 00F20056, 00F20057

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Collect the items listed in "Diagnostics" on page 5175 and contact your IBM support center.

00F20058

The IBM MQ-IMS bridge received an MQPUT1 error.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Contact your IBM support center to report the problem.

00F20059

The IBM MQ-IMS bridge received a severe sense code in an IMS negative response.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

The IMS sense code is given in message CSQ2003I.

00F20069

The IBM MQ-IMS bridge received an error when trying to resolve an in-doubt unit of recovery.

System action

The current execution unit terminates with completion code X'5C6', and a dump is produced.

System programmer response

Contact your IBM support center to report the problem.

Subsystem support codes (X'F3'): z/OS

Many of the following reason codes are returned in register 15 at the time of an abnormal termination with completion code X'0Cx', and not as the reason code for a completion code of X'5C6'. This is indicated in the descriptions that follow.

If a subsystem support reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- A description of the actions that led to the error or, if applicable, either a listing of the application program or the input string to a utility program that was being run at the time of the error.
- The console output for the period leading up to the error.
- The queue manager job log.
- The system dump resulting from the error.
- A printout of SYS1.LOGREC.
- Appropriate WebSphere MQ, z/OS, Db2, CICS, and IMS service levels.

00F30003, 00F30004, 00F30005

An internal error has occurred.

System action

The request is not processed. A dump is taken, and an entry is written in SYS1.LOGREC.

System programmer response

Collect the items listed in "Diagnostics" and contact your IBM support center.

00F30006

An internal error has occurred.

System action

The request is not processed.

System programmer response

Collect the items listed in "Diagnostics" and contact your IBM support center.

00F30007, 00F30008

An internal error has occurred.

System action

The request is not processed. A dump is taken, and an entry is written in SYS1.LOGREC.

System programmer response

Collect the items listed in "Diagnostics" and contact your IBM support center.

00F30014

An internal error has occurred.

System action

The requester's task is ended abnormally with completion code X'5C6'. A dump is taken, and an entry is written in SYS1.LOGREC.

System programmer response

Collect the items listed in "Diagnostics" and contact your IBM support center.

00F30027, 00F30030 ,00F30032, 00F30033, 00F30038

An internal error has occurred.

System action

The request is not processed. A dump is taken, and an entry is written in SYS1.LOGREC.

System programmer response

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30042

An internal error has occurred.

System action

A dump is taken, and an entry is written in SYS1.LOGREC.

System programmer response

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30048

An internal error has occurred.

System action

The request is not processed. A dump is taken, and an entry is written in SYS1.LOGREC.

System programmer response

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30052

The recovery coordinator for the caller has already terminated, so the connection from the caller to MQ has been terminated.

System action

The request is not processed. The connection from the caller to MQ is terminated.

The caller might reconnect to MQ when the recovery coordinator has been restarted.

System programmer response

Identify and restart the recovery coordinator.

This abnormal termination is most commonly associated with a termination of RRS. There might be additional CSQ3009E messages on the console log associated with the termination of RRS.

00F30053

An internal error has occurred.

System action

The request is not processed. A dump is taken, and an entry is written in SYS1.LOGREC.

System programmer response

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30067

An internal error has occurred.

System action

The connection request is not processed. A dump is taken, and an entry is written in SYS1.LOGREC.

System programmer response

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30070

Functional recovery for the connection processing could not be established. The executing module could not establish its ESTAE. This can occur if the current address space has insufficient storage. This might lead to an abnormal termination of the queue manager.

System action

The connection request is not processed. The caller is ended abnormally with completion code X'5C6' and this reason code.

System programmer response

Restart the queue manager if necessary. A dump should be taken for problem analysis.

Examine the usage and free areas in the LSQA portion of the current address space private area. If necessary, have the size of the private areas expanded.

The caller should produce a SYS1.LOGREC entry and an SVC dump, so that you can examine the LSQA area. You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

00F30071

An internal error has occurred.

System action

The connection request is not processed. A dump is taken, and an entry is written in SYS1.LOGREC.

System programmer response

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30075

An internal error has occurred.

System action

A dump is taken, and an entry is written in SYS1.LOGREC.

System programmer response

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30078

An internal error has occurred.

System action

The request is not processed. A dump is taken, and an entry is written in SYS1.LOGREC.

System programmer response

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30080

An internal error has occurred.

System action

The application program is ended abnormally with completion code X'5C6' and this reason code. A dump is taken, and an entry is written in SYS1.LOGREC.

System programmer response

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30091

The application program issued an RRSAF IDENTIFY function request, but RRS is not available.

System action

The IDENTIFY request is not processed.

00F30093

The application program issued an RRSAF TERMINATE THREAD or TERMINATE IDENTIFY function request, but the application has issued an MQ API request since the last invocation of SRRCMIT or SRRBACK and therefore is not at a point of consistency.

System action

The function request is not processed.

00F30095

An internal error was detected in either MQ or RRS.

System action

The application is ended abnormally. The error is recorded in the SYS1.LOGREC data set and an SVC dump is requested.

This error might, in many cases, eventually cause the queue manager to terminate abnormally.

System programmer response

This is probably either an error in MQ or in RRS.

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30096

An internal error was detected in either MQ or RRS Context Services.

System action

The application is ended abnormally. The error is recorded in the SYS1.LOGREC data set and an SVC dump is requested.

This error might, in many cases, eventually cause the queue manager to terminate abnormally.

System programmer response

This is probably either an error in MQ or in RRS.

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30101

The parameter contained in the IEFSSNxx member used to initialize MQ (and other subsystems) is in error. See message CSQ3101E for details.

System action

See message CSQ3101E.

System programmer response

See message CSQ3101E.

You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

00F30102

The parameter contained in the IEFSSNxx member used to initialize MQ (and other subsystems) is in error. The MQ command prefix (CPF) must not be blank. For details, see message CSQ3102E.

System action

See message CSQ3102E.

System programmer response

See message CSQ3102E.

You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

00F30103

The parameter contained in the IEFSSNxx member used to initialize MQ (and other subsystems) is in error or the named module is not resident in a library available during IPL. See message CSQ3103E for details.

System action

See message CSQ3103E.

System programmer response

See message CSQ3103E.

You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

00F30104

Module CSQ3UR00 was unable to obtain the affinity table index for the named subsystem. z/OS did not recognize the named subsystem. See message CSQ3109E for details.

System action

See message CSQ3109E.

System programmer response

See message CSQ3109E.

You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

00F30105

Module CSQ3UR00 was unable to load Early module CSQ3EPX. Either there was an I/O error, or the named module is not resident in a library available during IPL. See message CSQ3105E for details.

System action

See message CSQ3105E.

System programmer response

See message CSQ3105E.

You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

00F30106

The parameter contained in the IEFSSNxx member used to initialize MQ (and other subsystems) is in error. The scope of the MQ command prefix (CPF) is not valid. For details, see message CSQ3112E.

System action

See message CSQ3112E.

System programmer response

See message CSQ3112E.

You might find the items listed in "Diagnostics" on page 5181 useful in resolving the problem.

00F30107

An error occurred during command prefix registration.

System action

The MQ subsystem ends abnormally.

System programmer response

See the accompanying CSQ3xxx messages for information about the cause of the problem.

00F30210, 00F30211, 00F30212, 00F30213, 00F30214

An internal error has occurred.

System action

The caller is ended abnormally. An SVC dump and associated SYS1.LOGREC entries are produced.

System programmer response

Collect the items listed in "Diagnostics" on page 5181 and contact your IBM support center.

00F30216

An attempt to create a queue manager address space failed. This is probably because the user who issued the START QMGR command has insufficient authority.

System action

The current START command processing is terminated. An SVC dump and associated SYS1.LOGREC entries are produced.

System programmer response

Check the authority of users and consoles to issue commands. Retry the command.

You might find the items listed in "Diagnostics" on page 5181 useful in resolving the problem.

00F30217

The console ID for the z/OS console that entered the current command is not found in the z/OS unit control module (UCM) structure. An internal z/OS command might have been incorrectly issued by an application program that provided invalid input parameters.

System action

The caller is ended abnormally.

System programmer response

Retry the START QMGR command. If the command was unsuccessful, collect the items listed in "Diagnostics" on page 5181 and contact your IBM support center.

00F30218

An internal error has occurred.

System action

The current task is ended abnormally. The calling task might have requested an SVC dump or created associated SYS1.LOGREC entries.

System programmer response

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30219

An internal error has occurred.

System action

The calling task is ended abnormally. The calling task might have requested an SVC dump or created associated SYS1.LOGREC entries.

System programmer response

Cancel the queue manager. End-of-task processing might still work, and it does a more complete clean-up than end-of-memory processing does. If this does not work, issue the z/OS command FORCE for the queue manager. If the problem is still unresolved, it might be necessary to perform an IPL of your z/OS system.

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F3021A

An internal error has occurred.

System action

The calling task is ended abnormally. An SVC dump and associated SYS1.LOGREC entries are produced.

System programmer response

Stop the queue manager and reissue the START QMGR command.

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F3021C

An ESTAE could not be established. This can occur if the z/OS system address space that is broadcasting the command has insufficient storage.

System action

The caller is ended abnormally (without a dump). The current START command processing is terminated.

System programmer response

Retry the command. If the error persists, it might be necessary to perform an IPL of your z/OS system.

Examine the LOGREC entries, and the console log for indications of a z/OS error, and try increasing the storage.

If you are unable to resolve the problem, collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F3021D

An ESTAE could not be established during either the initialization or termination of the queue manager.

This can occur during initialization if the z/OS system address space that is broadcasting the first command (assumed to be the START command) has insufficient storage.

This can occur during termination if the current address space (usually the queue manager, or in the case of EOM broadcast, a z/OS system address space) has insufficient storage.

System action

The caller is ended abnormally without taking a system dump. The initialization stops, but termination proceeds.

System programmer response

Retry the command after the queue manager has terminated. If the problem persists, it might be necessary to perform an IPL of your z/OS system.

Examine the LOGREC entries, and the console log for indications of a z/OS error, and try increasing the storage.

If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5181 and contact your IBM support center.

00F3021E

An ESTAE could not be established while in the process of routing control to the actual ESTAE routine. The caller (RTM) is ended abnormally. This causes the original error to percolate to a higher-level recovery routine and causes this reason code to be shown in an RTM recovery environment.

This can occur if the current address space (usually an allied address space) has insufficient storage.

System action

The caller is ended abnormally and a dump is produced.

System programmer response

Examine the usage and free areas in the LSQA portion of the current address space private area. If necessary, have the size of the private area expanded.

You might find the items listed in "Diagnostics" on page 5181 useful in resolving the problem.

00F3021F, 00F30220

An internal error has occurred.

System action

The caller is not ended abnormally. A dump is taken, and an entry is written in SYS1.LOGREC.

System programmer response

Collect the items listed in "Diagnostics" on page 5181 and contact your IBM support center.

00F30230

An internal error has occurred.

System action

The connection between the allied address space and the queue manager terminated. A dump is taken, and an entry is written in SYS1.LOGREC.

System programmer response

Collect the items listed in "Diagnostics" on page 5181 and contact your IBM support center.

00F30310

An internal error has occurred.

System action

The invoker is ended abnormally. A dump is taken, and an entry is written in SYS1.LOGREC.

System programmer response

Collect the items listed in "Diagnostics" on page 5181 and contact your IBM support center.

00F30311

An ESTAE could not be established during the processing of a resolve-indoubt request. This can occur if the current address space has insufficient storage. This will probably cause an abnormal termination of the queue manager.

System action

The caller is ended abnormally.

System programmer response

Restart the queue manager if necessary.

Examine the usage and free areas in the local system queue area (LSQA) portion of the current address space private area. If necessary, have the size of the private area expanded.

The caller should produce a SYS1.LOGREC entry and an SVC dump, so that you can examine the LSQA area.

You might find the items listed in "Diagnostics" on page 5181 useful in resolving the problem.

00F30312

An ESTAE could not be established during the processing of a resolve-indoubt-UR request. This can occur if the current address space has insufficient storage.

System action

The caller is ended abnormally.

System programmer response

Examine the usage and free areas in the local system queue area (LSQA) portion of the current address space private area. If necessary, have the size of the private area expanded.

The caller should produce a SYS1.LOGREC entry and an SVC dump.

You might find the items listed in "Diagnostics" on page 5181 useful in resolving the problem.

00F30313

A control block could not be allocated. This could occur when the storage pool has no more free space available.

System action

The request is not processed. The application program is ended abnormally with completion code X'5C6' and this reason code.

System programmer response

A dump should be taken for problem analysis.

Check that you are running with the recommended region size, and if not, reset your system and retry. If you are unable to resolve the problem, collect the items listed in "Diagnostics" on page 5181 and contact your IBM support center.

00F30400, 00F30401, 00F30402

An internal error has occurred.

System action

The program which made the request might produce diagnostics to report the error.

System programmer response

Collect the diagnostics produced by the application program reporting the error, if any, and contact your IBM support center.

00F30406

The queue manager has gone to EOM (end-of-memory). This is probably because the z/OS command FORCE has been issued.

System action

The queue manager is terminated, and a dump is taken.

System programmer response

The queue manager can be restarted after termination completes.

Determine why the z/OS command FORCE was issued.

00F30409, 00F3040A

An internal error has occurred.

System action

The queue manager is terminated with an SVC dump.

System programmer response

The queue manager can be started again after it terminates.

Collect the items listed in "Diagnostics" on page 5181 and contact your IBM support center.

00F3040B

See message CSQ3001E.

System action

See message CSQ3001E.

System programmer response

See message CSQ3001E.

You might find the items listed in "Diagnostics" on page 5181 useful in resolving the problem.

00F3040C, 00F3040D

An internal error has occurred.

System action

The queue manager is terminated with an SVC dump.

System programmer response

The queue manager can be started again after it terminates.

Collect the items listed in "Diagnostics" on page 5181 and contact your IBM support center.

00F3040E

An internal error has occurred.

System action

The queue manager is terminated.

System programmer response

The queue manager should be restarted.

Collect the items listed in "Diagnostics" on page 5181 and contact your IBM support center.

00F3040F, 00F30410

An internal error has occurred.

System action

The queue manager is terminated.

System programmer response

The queue manager can be started again after it terminates.

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30411, 00F30412, 00F30413

An internal error has occurred.

System action

The queue manager is terminated.

System programmer response

The queue manager can be started again after it terminates.

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30414

An internal error has occurred.

System action

The queue manager is terminated.

System programmer response

The queue manager can be started again after it terminates. If the problem persists, request a stand-alone dump, and perform an IPL of your z/OS system.

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30415

An ESTAE could not be established during the processing of an EOM SSI broadcast. This is probably a z/OS problem, because these modules are executing in the z/OS master scheduler address space.

System action

The queue manager is terminated.

System programmer response

The queue manager can be started again after it terminates. If the problem persists, it might be necessary to perform an IPL of your z/OS system.

This can occur if the z/OS master scheduler address space has insufficient free storage. If such is the case, MQ is unable to write a SYS1.LOGREC record or request a dump. The z/OS master scheduler should have produced these diagnostic aids. Examine the dump to determine whether the problem is in z/OS or MQ. Other unrelated errors in the z/OS Master Scheduler address space would indicate a z/OS problem.

If the problem appears to be an MQ problem, collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30416

An ESTAE could not be established during the processing of an EOM for an allied address space.

System action

The queue manager is terminated.

System programmer response

The queue manager can be started again after it terminates. If the problem persists, it might be necessary to perform an IPL of your z/OS system.

This can occur if the z/OS master scheduler address space has insufficient free storage. If such is the case, MQ is unable to write a SYS1.LOGREC record or request a dump. The z/OS master scheduler should have produced these diagnostic aids. Examine the dump to determine whether the problem is in z/OS or MQ. Other unrelated errors in the z/OS Master Scheduler address space would indicate a z/OS problem.

If the problem appears to be an MQ problem, collect the items listed in "Diagnostics" on page 5181 and contact your IBM support center.

00F30417, 00F30418

An internal error has occurred.

System action

The queue manager is terminated.

System programmer response

The queue manager can be started again after it terminates.

Collect the items listed in "Diagnostics" on page 5181 and contact your IBM support center.

00F30419

An internal error has occurred.

System action

The queue manager is terminated with an SVC dump.

System programmer response

The queue manager can be started again after it terminates.

Collect the items listed in "Diagnostics" on page 5181 and contact your IBM support center.

00F3041A

An ESTAE could not be established by the deferred end-of-task (EOT) processor. This error could occur only during queue manager startup. Probably, an ESTAE could not be established because of a shortage of LSQA space.

System action

The queue manager is terminated.

System programmer response

Restart the queue manager.

If the problem persists, increase the size of the queue manager address space private area.

You might find the items listed in "Diagnostics" on page 5181 useful in resolving the problem.

00F3041B, 00F30420

An internal error has occurred.

System action

The queue manager is terminated. A SYS1.LOGREC entry and associated SVC dump were requested.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30429

An internal error has occurred.

System action

The queue manager is terminated with an SVC dump.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30450

An ESTAE could not be established during the processing of an identify SSI call. This can occur if the current address space has insufficient storage.

System action

The allied address space is ended abnormally (without a dump). A dump should be produced by the allied task.

System programmer response

The user can retry the identify request. If a dump is available, review the storage manager's control blocks to determine if all of the private area has been allocated. If necessary, increase the private area size of the allied address space.

You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

00F30451

An ESTAE could not be established during the processing of an identify SSI call. This can occur if the current address space has insufficient storage.

System action

The allied task is ended abnormally (without a dump). A dump should be produced by the allied task.

System programmer response

The user can retry the identify request. If a dump is available, review the storage manager's control blocks to determine if all of the private area has been allocated. If necessary, increase the private area size of the allied address space.

You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

00F30452

An ESTAE could not be established during the processing of an identify SSI call. This can occur if the current address space has insufficient storage.

System action

The allied task is ended abnormally (without a dump). A dump should be produced by the allied task.

System programmer response

The user can retry the identify request. If a dump is available, review the storage manager's control blocks to determine if all of the private area has been allocated. If necessary, increase the private area size of the allied address space.

You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

00F30453

ESTAEs could not be established during the processing of a n SSI call other than FEOT, EOM, HELP, COMMAND, and IDENTIFY. This can occur if the current address space has insufficient storage.

System action

The allied task is ended abnormally (without a dump). A dump should be produced by the allied task.

System programmer response

The user can retry the request. If a dump is available, review the storage manager's control blocks to determine if all of the private area has been allocated. If necessary, increase the private area size of the allied address space.

You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

00F30454

An internal error has occurred.

System action

The allied task is ended abnormally.

System programmer response

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30455

An ESTAE could not be established during the processing of an identify termination request. This can occur if the current address space has insufficient storage.

System action

The allied task is ended abnormally (without a dump). A dump should be produced by the allied task.

System programmer response

The user can retry the request. If a dump is available, review the storage manager's control blocks to determine if all of the private area has been allocated. If necessary, increase the private area size of the allied address space.

You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

00F30456

An internal error has occurred.

System action

The calling task is ended abnormally.

System programmer response

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30457

An internal error has occurred.

System action

The caller is ended abnormally. The error might, in many cases, eventually terminate the queue manager.

System programmer response

Restart the queue manager if necessary.

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30459

An internal error has occurred.

System action

The queue manager is terminated with a reason code of X'00F30420'.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30461

The queue manager was unable to successfully restart with RRS because of an internal error in either MQ or RRS.

System action

The queue manager is not connected to RRS and all services dependent on that connection are unavailable. This means that applications might not connect to the queue manager using RRSAF and that WLM-established address spaces might not be used for MQ stored procedures until the queue manager successfully restarts with RRS.

System programmer response

Stop and then start RRS. Stop and then start the queue manager. If the problem persists, perform an RRS cold start.

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30501, 00F30502

An internal error has occurred.

System action

The requester is ended abnormally, and the request is not processed.

System programmer response

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30503

CSQ6SYSP is missing from the system parameter load module.

System action

Queue manager start-up is terminated.

System programmer response

Re-create your system parameter load module (if a customized version is being used) and restart the queue manager. For information about the system parameter modules, see Tailor your system parameter module.

00F30573, 00F30574

An internal error has occurred.

System action

The requester is ended abnormally, and the request is not processed. A dump is taken, and an entry is written in SYS1.LOGREC.

System programmer response

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30580

An internal error has occurred.

System action

The requester is ended abnormally.

System programmer response

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30581

An internal error has occurred.

System action

The queue manager ends abnormally. The startup/shutdown ESTAE creates a SYS1.LOGREC entry and takes an SVC dump.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30597, 00F30598

An internal error has occurred.

System action

The allied task is ended abnormally, and the request is not processed.

System programmer response

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30599

An internal error has occurred.

System action

The connection name associated with the error is probably unable to continue communication with MQ until the queue manager is terminated and restarted.

System programmer response

If necessary, stop and restart the queue manager.

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30601

Asynchronous events occurred which caused the premature termination of the thread. The thread could not be recovered.

There might be other errors or messages concerning this allied user indicating what the asynchronous events were.

System action

The allied user is ended abnormally with completion code X'5C6' and this reason code.

System programmer response

You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

00F30610

An ESTAE could not be established during the processing of an 'end stop-work force' notification. This can occur if there is insufficient storage. This might lead to abnormal termination of the queue manager.

System action

The caller is ended abnormally. An SVC dump and related SYS1.LOGREC entry are requested.

System programmer response

If necessary, restart the queue manager.

If necessary, increase the private area size of the address space.

You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

00F30801

An internal error has occurred.

System action

The queue manager is terminated. An SVC dump is requested.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30802

An internal error has occurred.

System action

The task is not ended abnormally.

System programmer response

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30803

An ESTAE could not be established during the processing of an application program support call. This can occur if the current address space has insufficient storage.

System action

The allied task is ended abnormally. The allied task might have requested an SVC dump.

System programmer response

The user can retry the request. If necessary, increase the private area size of the application address space.

You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

00F30805

An internal error has occurred.

System action

The request might have been processed or rejected.

System programmer response

Collect the items listed in “Diagnostics” on page 5181 and contact your IBM support center.

00F30901

MQ has lost its cross-memory authority to an allied address space because the ally has released its authorization index.

System action

The allied address space is terminated.

System programmer response

You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

00F30902

MQ has detected a recursive error condition while processing End-of-Task for a task in an allied address space.

System action

The allied address space is terminated.

System programmer response

You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

00F30903

An error has occurred while processing End-of-Task for the queue manager address space.

System action

The address space is forced to 'end-of-memory' with this reason code.

System programmer response

You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

00F30904

End-of-Task occurred for the queue manager address space, and MQ could not establish an ESTAE to protect its processing. Insufficient storage might be the reason the ESTAE could not be established.

System action

The address space is forced to 'end-of-memory' with this reason code.

System programmer response

You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

Attempt to determine if one or more MQ address spaces is storage-constrained. Examination of the console output for the time period preceding this condition might reveal other messages or indications that the terminating address space was storage-constrained.

00F30905

End-of-Task occurred for the job step task in an allied address space. MQ would normally attempt to terminate the address space's connection to the queue manager but was unable to protect its processing by establishing an ESTAE. Insufficient storage might be the reason the ESTAE could not be established.

System action

The address space is forced to 'end-of-memory' with this reason code.

System programmer response

You might find the items listed in “Diagnostics” on page 5181 useful in resolving the problem.

Attempt to determine if one or more allied address spaces is storage-constrained. Examination of the console output for the time period preceding this condition might reveal other messages or indications that the terminating allied address space was storage-constrained.

00F33100

The MQ thread is read-only.

System action

A prepare issued by the application program was processed through Phase-1. MQ discovered there were no resources modified and no need for COMMIT or BACKOUT to be subsequently issued.

System programmer response

This might create a path length saving by not issuing the subsequent commit or backout which normally follows prepare. No further action is required to complete the unit of recovery; the unit of recovery is complete.

Db2 manager codes (X'F5'): z/OS

If a Db2 manager reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- A description of the actions that led to the error or, if applicable, either a listing of the application program or the input string to a utility program that was being run at the time of the error.
- The console output for the period leading up to the error.
- The queue manager job log.
- The system dump resulting from the error.
- A printout of SYS1.LOGREC.
- Appropriate WebSphere MQ, z/OS, Db2, CICS, and IMS service levels.

00F50000

An internal error has occurred.

System action

The queue manager terminates, a record is written to SYS1.LOGREC and a dump is taken.

System programmer response

Ensure that the QSGDATA system parameter is specified correctly and restart the queue manager.

If the problem persists, collect the items listed in “Diagnostics” and contact your IBM support center.

00F50001

An internal error has occurred.

System action

The queue manager terminates, a record is written to SYS1.LOGREC and a dump is taken.

System programmer response

Restart the queue manager.

If the problem persists, collect the items listed in “Diagnostics” and contact your IBM support center.

00F50002

An internal error has occurred.

System action

The task ends abnormally. Queue manager processing continues but the queue manager might not terminate normally and might not register DB2 termination.

System programmer response

Refer to *Db2 for z/OS Messages and Codes* for information about the completion and reason code in the accompanying message and collect the diagnostic data requested in the manual. In addition, collect the items listed in "Diagnostics" on page 5199 and contact your IBM support center.

00F50003

An internal error has occurred.

System action

The task ends abnormally. Queue manager processing continues.

System programmer response

Collect the items listed in "Diagnostics" on page 5199 and contact your IBM support center.

00F50004

An internal error has occurred.

System action

The queue manager terminates, a record is written to SYS1.LOGREC and a dump is taken.

System programmer response

Ensure that the following modules are available through the linklist or the steplib concatenation: DSNRLI, DSNHLIR, DSNWLIR, ATRCMIT and ATRBACK. Restart the queue manager.

If the problem persists, collect the items listed in "Diagnostics" on page 5199 and contact your IBM support center.

00F50006

An internal error has occurred.

System action

The queue manager terminates, a record is written to SYS1.LOGREC and a dump is taken.

System programmer response

All queue managers that are members of the same queue-sharing group must connect to the same Db2 data-sharing group. Check that all queue managers in the queue-sharing group have the same Db2 data-sharing group specified in the QSGDATA system parameter. Restart the queue manager.

Collect the items listed in "Diagnostics" on page 5199 and contact your IBM support center.

00F50007

An internal error has occurred.

System action

The queue manager terminates, a record is written to SYS1.LOGREC and a dump is taken.

System programmer response

Ensure that the Db2 subsystem(s) specified on the QSGDATA system parameter are members of the DB2 data-sharing group that is also specified on the QSGDATA system parameter. Restart the queue manager.

If the problem persists, refer to *Db2 for z/OS Messages and Codes* for information about the completion and reason code in the accompanying message and collect the diagnostic data requested in the manual. In addition, collect the items listed in "Diagnostics" on page 5199 and contact your IBM support center.

00F50008

An internal error has occurred.

System action

The task ends abnormally and processing continues.

System programmer response

Collect the items listed in "Diagnostics" on page 5199 and contact your IBM support center.

00F50009

An internal error has occurred.

System action

The queue manager terminates, a record is written to SYS1.LOGREC and a dump is taken.

System programmer response

Restart the queue manager.

Refer to *DB2 for z/OS Messages and Codes* for information about the completion and reason code in the accompanying message and collect the diagnostic data requested in the manual. In addition, collect the items listed in "Diagnostics" on page 5199 and contact your IBM support center.

00F50010

An internal error has occurred.

System action

The queue manager terminates, a record is written to SYS1.LOGREC and a dump is taken.

System programmer response

Restart the queue manager.

See z/OS MVS Programming: Sysplex Services Reference for an explanation of the error and the diagnostic information, if any, that you must collect. In addition, collect the items listed in "Diagnostics" on page 5199 and contact your IBM support center.

00F50013

No queue manager entry was found in the CSQ.ADMIN_B_QMGR table for this combination of queue manager and queue-sharing group, or the entry was incorrect.

System action

The queue manager terminates, a record is written to SYS1.LOGREC and a dump is taken.

System programmer response

Check the CSQ.ADMIN_B_QMGR table in the DB2 data-sharing group and ensure that an entry has been defined for the queue manager and it relates to the correct queue-sharing group.

If you are migrating from a previous release of MQ, check also that you have updated the DB2 tables to the format for the current release. See Maintaining and migrating, for information about migration and compatibility between releases.

Restart the queue manager. If the problem persists, collect the items listed in “Diagnostics” on page 5199 and contact your IBM support center.

00F50014

An internal error has occurred.

System action

The queue manager terminates, a record is written to SYS1.LOGREC and a dump is taken.

System programmer response

Check that the Db2 related installation and customization tasks have all completed successfully. Restart the queue manager.

If the problem persists, refer to *DB2 for z/OS Messages and Codes* for information about the completion and reason code in the accompanying message and collect the diagnostic data requested in the manual. In addition, collect the items listed in “Diagnostics” on page 5199 and contact your IBM support center.

00F50015

An internal error has occurred.

System action

The queue manager terminates, a record is written to SYS1.LOGREC and a dump is taken.

System programmer response

Restart the queue manager.

If the problem persists, refer to *Db2 for z/OS Messages and Codes* for information about the completion and reason code in the accompanying message and collect the diagnostic data requested in the manual. In addition, collect the items listed in “Diagnostics” on page 5199 and contact your IBM support center.

00F50016

An internal error has occurred.

System action

The queue manager terminates, a record is written to SYS1.LOGREC and a dump is taken.

System programmer response

Restart the queue manager.

If the problem persists, refer to *Db2 for z/OS Messages and Codes* for information about the completion and reason code in the accompanying message and collect the diagnostic data requested in the manual. In addition, collect the items listed in “Diagnostics” on page 5199 and contact your IBM support center.

00F50017

An internal error has occurred.

System action

The queue manager terminates, a record is written to SYS1.LOGREC and a dump is taken.

System programmer response

See z/OS MVS Programming: Sysplex Services Reference for information about the completion and reason code in the accompanying message.

Restart the queue manager. If the problem persists, collect the items listed in “Diagnostics” on page 5199 and contact your IBM support center.

This error may occur if one or more of the queue managers in a queue-sharing group (QSG) do not have a member entry in the XCF group for the QSG.

Enter the following z/OS command substituting the queue-sharing group name for xxxx:

```
D XCF,GRP,CSQGxxxx,ALL
```

This will list the members of the XCF group. If any queue managers are defined as a member of the QSG, but do not have an entry in the XCF Group, use the ADD QMGR command of the CSQ5PQSG utility to restore the XCF group entry for that queue manager. The utility should be run for each queue manager which does not have an entry in the XCF group.

00F50018

An internal error has occurred.

System action

The queue manager terminates, a record is written to SYS1.LOGREC and a dump is taken.

System programmer response

See z/OS MVS Programming: Sysplex Services Reference for information about the completion and reason code in the accompanying message.

Restart the queue manager. If the problem persists, collect the items listed in “Diagnostics” on page 5199 and contact your IBM support center.

00F50019

An internal error has occurred.

System action

The queue manager terminates, a record is written to SYS1.LOGREC and a dump is taken.

System programmer response

See z/OS MVS Programming: Sysplex Services Reference for information about the completion and reason code in the accompanying message.

Restart the queue manager. If the problem persists, collect the items listed in “Diagnostics” on page 5199 and contact your IBM support center.

00F5001C

CSQ5_DB2_UNAVAILABLE

System action

The queue manager terminates, a record is written to SYS1.LOGREC and a dump is taken.

System programmer response

See z/OS MVS Programming: Sysplex Services Reference for information about the completion and reason code in the accompanying message.

Restart the queue manager. If the problem persists, collect the items listed in “Diagnostics” on page 5199 and contact your IBM support center.

00F50021

An internal error has occurred.

System action

The queue manager terminates, a record is written to SYS1.LOGREC and a dump is taken.

System programmer response

See z/OS MVS Programming: Sysplex Services Reference for information about the completion and reason code in the accompanying message.

Restart the queue manager. If the problem persists, collect the items listed in “Diagnostics” on page 5199 and contact your IBM support center.

00F50024

An internal error has occurred.

System action

The task ends abnormally and a dump is taken.

System programmer response

If the problem persists, collect the items listed in “Diagnostics” on page 5199 and contact your IBM support center.

00F50025

An internal error has occurred.

System action

The task ends abnormally and a dump is taken.

System programmer response

Collect the items listed in “Diagnostics” on page 5199 and contact your IBM support center.

00F50026

An internal error has occurred.

System action

The task ends abnormally and a dump is taken.

System programmer response

Collect the items listed in “Diagnostics” on page 5199 and contact your IBM support center.

00F50027

An internal error has occurred.

System action

The task ends abnormally and a dump is taken.

System programmer response

Collect the items listed in “Diagnostics” on page 5199 and contact your IBM support center.

00F50028

An internal error has occurred.

System action

The task ends abnormally and a dump is taken.

System programmer response

This might be a temporary condition if DB2 or RRS has failed. If the problem persists, collect the items listed in “Diagnostics” on page 5199, together with output from Db2 command DISPLAY THREAD(*), and contact your IBM support center.

00F50029

The queue manager has detected a mismatch between its supported versions of MQ and those of other members of the queue-sharing group.

System action

The queue manager terminates, a record is written to SYS1.LOGREC and a dump is taken.

System programmer response

Verify the started task JCL procedure for the queue manager (xxxxMSTR) is executing the correct version of MQ. Restart the queue manager. If the correct version is being executed, collect the items listed in "Diagnostics" on page 5199, together with a printout of the CSQ.ADMIN_B_QMGR table from the Db2 data-sharing group to which the queue manager connected, and contact your IBM support center.

00F50901

An internal error has occurred.

System action

The job ends abnormally with a X'5C6' completion code and a dump is taken.

System programmer response

Collect the items listed in "Diagnostics" on page 5199 and contact your IBM support center.

00F51030

An internal error has occurred.

System action

The task ends abnormally and a dump is taken.

System programmer response

Restart RRS if it has terminated. If RRS has not terminated, collect the items listed in "Diagnostics" on page 5199 and contact your IBM support center.

00F51031

An internal error has occurred on a DB2 connection thread.

System action

The task ends abnormally and a new task is created. A dump is taken if there is an 'in-flight' Db2 request.

System programmer response

None. A new Db2 server task is automatically re-created to replace the task that was terminated. If the problem persists, collect the items listed in "Diagnostics" on page 5199 and contact your IBM support center.

Generalized command preprocessor codes (X'F9'): z/OS

If a command preprocessor reason code occurs that is not listed here, an internal error has occurred. Collect the following diagnostic items and contact your IBM support center.

Diagnostics

- A description of the actions that led to the error or, if applicable, either a listing of the application program or the input string to a utility program that was being run at the time of the error.
- The console output for the period leading up to the error.
- The queue manager job log.
- The system dump resulting from the error.
- Appropriate WebSphere MQ, z/OS, Db2, CICS, and IMS service levels.
- If you are using the WebSphere MQ Operations and Control panels, the ISPF panel name.
- The command issued before the error occurred.

00F90000

An internal error has occurred.

System action

Command execution was ended abnormally. If the command was properly entered, it might have been partially or completely executed.

System programmer response

Collect the items listed in "Diagnostics" and contact your IBM support center.

It might be necessary to restart the CICS or IMS adapter.

00F90001

An internal error has occurred.

System action

Command execution was ended abnormally. If the command was properly entered, it might have been partially or completely executed.

System programmer response

Collect the items listed in "Diagnostics" and contact your IBM support center.

It might be necessary to restart the CICS or IMS adapter.

00F90002

The routines of the multiple console support (MCS) service of z/OS. were unable to initialize. This condition might indicate an error in the address space.

System action

Initialization is stopped, causing the queue manager to terminate.

System programmer response

Collect the items listed in "Diagnostics" and contact your IBM support center.

Restart the queue manager.

00F90003

The routines of the multiple console support (MCS) service of z/OS were unable to initialize.

System action

If the error was issued by module CSQ9SCNM, queue manager initialization is stopped, causing the queue manager to terminate. If the error was issued by module CSQ9SCN6, the command from the associated console is executed, and should proceed normally.

System programmer response

Collect the items listed in "Diagnostics" on page 5206 and contact your IBM support center.

00F90004

The routines of the multiple console support (MCS) service of z/OS detected a logic error.

System action

The command was not executed.

System programmer response

Collect the items listed in "Diagnostics" on page 5206 and contact your IBM support center.

00F90005

A routine of the multiple console support (MCS) service of z/OS was not able to create an ESTAE recovery environment. This condition is detected when the ESTAE service of z/OS returns a nonzero return code. The command from the associated z/OS console is not executed. See the *MVS Programming: Assembler Services Reference* manual for an explanation of ESTAE return codes.

System action

Command processing is terminated.

System programmer response

Collect the items listed in "Diagnostics" on page 5206 and contact your IBM support center.

00F90006

An internal error has occurred.

System action

Agent allocation is terminated.

System programmer response

Collect the items listed in "Diagnostics" on page 5206 and contact your IBM support center.

00F90007

An internal error has occurred.

System action

The statistical update is not completed. The statistics block address is cleared from the CGDA to prevent future problems. No further command statistical counts are maintained. Processing for the command is retried and should complete normally.

System programmer response

Collect the items listed in "Diagnostics" on page 5206 and contact your IBM support center.

00F90008

An internal error has occurred.

System action

The function is ended abnormally.

System programmer response

Collect the items listed in "Diagnostics" on page 5206 and contact your IBM support center.

00F90009

This reason code is used to document that module CSQ9SCN9 has added information to the SDWA variable recording area (VRA) following the data provided by the CSQWRCRD service. If CSQ9SCN9 records an error in SYS1.LOGREC and the reason code in the VRA is not of the form X'00F9xxxx', the reason code is changed to X'00F90009'. This is done so that anyone examining a SYS1.LOGREC entry can determine, from the reason code, what additional data has been placed in the VRA. The reason code is the first data item in the VRA, as mapped by macro IHAVRA.

System programmer response

Collect the items listed in "Diagnostics" on page 5206 and contact your IBM support center.

00F9000A

An internal error has occurred.

System action

Command execution was ended abnormally. The command was not executed.

System programmer response

Collect the items listed in "Diagnostics" on page 5206 and contact your IBM support center.

00F9000B

An internal error occurred while attempting to obtain CSA storage. The storage request could not be satisfied, either because no CSA storage was available or because an unreasonably large amount of storage was requested. The amount of storage requested is determined by the length of the command being parsed. Normally, it is several hundred bytes.

System action

Command execution is ended abnormally.

System programmer response

It might be necessary to restart the CICS or IMS adapter, or the queue manager.

If the problem persists, collect the items listed in "Diagnostics" on page 5206 and contact your IBM support center.

00F9000C

An internal error has occurred.

The command processor invoked attempted to return a message formatted for inclusion in a z/OS multiple line WTO (write to operator).

System action

Command execution is ended abnormally.

System programmer response

The command in error is identified by message CSQ9017E. It might be necessary to restart the CICS or IMS adapter, or the queue manager.

Collect the items listed in "Diagnostics" on page 5206 and contact your IBM support center.

00F9000D

An internal error has occurred.

System action

The queue manager start-up is terminated.

System programmer response

Restart the queue manager.

Collect the items listed in “Diagnostics” on page 5206 and contact your IBM support center.

00F9000E

An internal error has occurred.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Collect the items listed in “Diagnostics” on page 5206 and contact your IBM support center.

00F9000F

MQ was unable to locate the default userid to be used on a command check. This indicates that CSQ6SYSP is not in the system parameter load module.

System action

The current execution unit terminates with completion code X'5C6'.

System programmer response

Ensure that CSQ6SYSP is in the system parameter load module. Restart the queue manager if necessary.

00F90010

An internal error has occurred while processing a command.

System action

Command execution was ended abnormally. The command was not executed.

System programmer response

Collect the items listed in “Diagnostics” on page 5206 and contact your IBM support center.

IBM MQ CICS adapter abend codes

z/OS

All the CICS versions supported by IBM MQ Version 9.0.0, and later, use the CICS supplied version of the adapter. See the Transaction abend codes section of the CICS documentation for further information.

IBM MQ CICS bridge abend codes

z/OS

All the CICS versions supported by IBM MQ Version 9.0.0, and later, use the CICS supplied version of the bridge. See the Transaction abend codes section of the CICS documentation for further information.

IBM MQ component identifiers

z/OS

IBM MQ for z/OS has a component-based architecture and each component uses a unique identifier code. These identifier codes are displayed in some of the informational messages.

Table 427. Component identifiers used in IBM MQ messages and codes

Component	ID	Hex ID
Batch adapter	B	X'C2'
CICS adapter	C	X'C3'
Coupling Facility manager	E	X'C5'
Message generator	F	X'C6'
Functional recovery manager	G	X'C7'
Security manager	H	X'C8'
Data manager	I	X'C9'
Recovery log manager	J	X'D1'
Lock manager	L	X'D3'
Connection manager	m	X'94'
Message manager	M	X'D4'
Command server	N	X'D5'
Operations and control	O	X'D6'
Buffer manager	P	X'D7'
IMS adapter	Q	X'D8'
Recovery manager	R	X'D9'
Storage manager	S	X'E2'
Timer services	T	X'E3'
Utilities	U	X'E4'
Agent services	V	X'E5'
Instrumentation facilities	W	X'E6'
Distributed queuing	X	X'E7'
Initialization procedures and general services	Y	X'E8'
System parameter manager	Z	X'E9'
Advanced message security	0 (zero)	X'F0'
Service facilities	1	X'F1'
IBM MQ - IMS bridge	2	X'F2'
Subsystem support	3	X'F3'
Db2 manager	5	X'F5'
Generalized command processor	9	X'F9'

Communications protocol return codes

z/OS

The communication protocols used by IBM MQ for z/OS can issue their own return codes. Use these tables to identify the return codes used by each protocol.

The tables in this topic show the common return codes from TCP/IP and APPC/MVS returned in messages from the distributed queuing component:

- "TCP/IP UNIX System Services Sockets return codes"
- APPC/MVS return codes

If the return code is not listed, or if you want more information, see to the documentation mentioned in each table.

If the return code you received is X'7D0' or more, it is one of the MQRC_* return codes issued by IBM MQ. These codes are listed in API completion and reason codes.

TCP/IP UNIX System Services Sockets return codes

See the *TCP/IP UNIX System Services Messages and Codes* manual for more information and for further return codes.

Table 428. UNIX System Services sockets return codes

Return code (Hexadecimal)	Explanation
0001	Error in the domain
0002	Result is too large
006F	Permission is denied
0070	The resource is temporarily unavailable
0071	The file descriptor is incorrect
0072	The resource is busy
0073	No child process exists
0074	A resource deadlock is avoided
0075	The file exists
0076	The address is incorrect
0077	The file is too large
0078	A function call is interrupted
0079	The parameter is incorrect
007A	An I/O error occurred
007B	The file specified is a directory
007C	Too many files are open for this process
007D	Too many links occurred
007E	The file name is too long
007F	Too many files are open in the system
0080	No such device exists
0081	No such file, directory, or IPC member exists
0082	The exec call contained a format error (DFSMS error)
0083	No locks are available

Table 428. UNIX System Services sockets return codes (continued)

Return code (Hexadecimal)	Explanation
0084	Not enough space is available
0085	No space is left on the device, or no space is available to create the IPC member ID
0086	The function is not implemented
0087	Not a directory
0088	The directory is not empty
0089	The I/O control operator is inappropriate
008A	No such device or address exists
008B	The operation is not permitted
008C	The pipe is broken
008D	The specified file system is read only
008E	The seek is incorrect
008F	No such process or thread exists
0090	A link to a file on another file system was attempted
0091	The parameter list is too long, or the message to receive was too large for the buffer
0092	A loop is encountered in symbolic links
0093	The byte sequence is incorrect
0095	A value is too large to be stored in the data type
0096	OpenMVS kernel is not active
0097	Dynamic allocation error
0098	Catalog Volume Access Facility error
0099	Catalog obtain error
009C	Process Initialization error
009D	An MVS environmental or internal error has occurred
009E	Bad parameters were passed to the service
009F	HFS encountered a permanent file error
00A2	HFS encountered a system error
00A3	SAF/RACF extract error
00A4	SAF/RACF error
00A7	Access to the OpenMVS version of the C RTL is denied
00A8	The password for the specified resource has expired
00A9	The new password specified is not valid
00AA	A WLM service ended in error
03EA	Socket number assigned by client interface code (for socket() and accept()) is out of range
03EB	Socket number assigned by client interface code is already in use
03ED	Offload box error
03EE	Offload box restarted
03EF	Offload box down
03F0	Already a conflicting call outstanding on socket
03F1	Request canceled using SOCKcallCANCEL request

Table 428. UNIX System Services sockets return codes (continued)

Return code (Hexadecimal)	Explanation
03F3	SetIbmOpt specified a name of a PFS that either was not configured or was not a Sockets PFS
044C	Block device required
044D	Text file busy
044E	The descriptor is marked nonblocking, and the requested function cannot complete immediately
044F	Operation now in progress
0450	Operation already in progress
0451	Socket operation on a non-socket
0452	Destination address required
0453	The message is too large to be sent in a single transmission, as required
0454	The socket type is incorrect
0455	Protocol or socket option unavailable
0456	Protocol not supported
0457	Socket type not supported
0458	The referenced socket is not a type that supports the requested function
0459	Protocol family not supported
045A	The address family is not supported
045B	The address is already in use
045C	Cannot assign requested address
045D	Network is down
045E	Network is unreachable
045F	Network dropped connection on reset
0460	Software caused connection abort
0461	Connection reset by peer
0462	Insufficient buffer space available
0463	The socket is already connected
0464	The socket is not connected
0465	Cannot send after socket shutdown
0466	Too many references: Cannot splice
0467	Connection timed out
0468	The attempt to connect was rejected
0469	Host is down
046A	No route to host
046B	Too many processes
046C	Too many users
046D	Disk quota exceeded
046E	Stale NFS file handle
046F	Too many levels of remote in path
0470	Device is not a stream

Table 428. UNIX System Services sockets return codes (continued)

Return code (Hexadecimal)	Explanation
0471	Timer expired
0472	Out of streams resources
0473	No message of the required type
0474	Trying to read unreadable message
0475	Identifier removed
0476	Machine is not on the network
0477	Object is remote
0478	The link has been severed
0479	Advertise error
047A	srmount error
047B	Communication error on send
047C	Protocol error
047D	Protocol error
047E	Cross mount point
047F	Remote address change
0480	The asynchronous I/O request has been canceled
0481	Socket send/receive gotten out of order
0482	Unattached streams error
0483	Streams push object error
0484	Streams closed error
0485	Streams link error
0486	Tcp error
Other	See the *OS/390 UNIX System Services Messages and Codes* manual

APPC/MVS return codes

The tables in this section document the following return codes:

- APPC return codes
- APPC allocate services return codes
- APPC reason codes

See the *Writing Transaction Programs for APPC/MVS* and *Writing Servers for APPC/MVS* documentation for more information.

APPC return codes

This table documents the return codes that can be returned from APPC/MVS in messages from the distributed queuing component if you are using APPC/MVS as your communications protocol. These return codes can be returned to the local program in response to a call.

Table 429. APPC return codes and their meanings

Return code (Hexadecimal)	Explanation
00	The call issued by the local program ran successfully. If the call specified a Notify_type of ECB, the call processing is performed asynchronously, and the ECB is posted when the processing is complete.
01	The caller specified an allocate_type that was other than *immediate*. Either APPC/MVS can not establish a session with the partner LU, or VTAM can not establish the conversation. In this case (when allocate_type is *immediate*), APPC/MVS converts this return code to “unsuccessful”.
02	The conversation cannot be allocated on a session because of a condition that might be temporary. The program can try again the allocation request. The system returns this code when the allocate_type specified on a CMALLOC verb is other then *immediate*.
03	The partner LU rejected the allocation request because the local program issued an Allocate call with the Conversation_type parameter set to either Basic_conversation or Mapped_conversation, and the partner program does not support the mapped or basic conversation protocol boundary. This return code is returned on a call made after the Allocate.
05	The partner LU rejected an ATBALLC or ATBALC2 (allocate) request because the partner program has one or more initialization parameter (PIP) variables defined. APPC/MVS does not support these parameters. This return code is returned on a call made after the Allocate. It is not returned for allocate requests made using CPI Communications.
06	The partner LU rejected the allocation request because the access security information is not valid. This return code is returned on a call subsequent to the Allocate.
08	The partner LU rejected the allocation request because the local program specified a synchronization level (with the Sync_level parameter) that the partner program does not support. This return code is returned on a call subsequent to the Allocate.
09	The partner LU rejected the allocation request because the local program specified a partner program that the partner LU does not recognize. This return code is returned on a call subsequent to the Allocate.
0A	The partner LU rejected the allocation request because the local program specified a partner program that the partner LU recognizes but cannot start. The condition is not temporary, and the program should not try again the allocation request. This return code is returned on a call subsequent to the Allocate.
0B	The partner LU rejected the allocation request because the local program specified a partner program that the partner LU recognizes but currently cannot start. The condition might be temporary, and the program can try again the allocation request. This return code is returned on a call subsequent to the Allocate.
11	The partner program issued a Deallocate call with a Deallocate_type of Deallocate_abend, or the partner LU has done so because of a partner program abnormal ending condition. If the partner program was in receive state when the call was issued, information sent by the local program and not yet received by the partner program is purged. This return code is reported to the local program on a call the program issues in Send or Receive state.
12	The partner program issued a Deallocate call on a basic or mapped conversation with a Deallocate_type of Deallocate_sync_level or Deallocate_flush. This return code is reported to the local program on a call the program issues in Receive state.

Table 429. APPC return codes and their meanings (continued)

Return code (Hexadecimal)	Explanation
13	The local program issued a call specifying an argument that was not valid. Specific reasons for the return code apply to the following callable services: **ATBALC2 or ATBALLC (LU 6.2 Allocate)** • The TP name was not 1 - 64 characters long • Either the SYMDEST name or the TP name length were not specified • SNASVCMG is specified as mode name • X'06' is used as the first character of a TP name • An SNA service TP name is used with a mapped conversation verb • The partner LU name was not valid • The mode name was not valid • The local LU name specified is either undefined or not permitted **CMALLC (CPI-C Allocate)** • SNASVCMG is specified as mode name • X'06'is used as the first character of a TP name • An SNA service TP name is used with a mapped conversation verb • The mode name was not valid
14	A product-specific error has been detected. The system writes symptom records that describe the error to SYS1.LOGREC.
15	Indicates one of the following: • The partner program made a Send_error call on a mapped conversation and the conversation for the partner program was in Send state. No truncation occurs at the mapped conversation protocol boundary. This return code is reported to the local program on a Receive call before receiving any data records or after receiving one or more data records. • The partner program made a Send_error call specifying the Type parameter with a value of PROG, the conversation for the partner program was in Send state, and the call did not truncate a logical record. No truncation occurs at the basic conversation protocol boundary when a program performs a Send_error before sending any logical records, or after sending a complete logical record. This return code is reported to the local program on a Receive call before receiving any logical records or after receiving one or more complete logical records.
16	The partner program made a Send_error call on a mapped conversation, or made a Send_error call on a basic conversation specifying the Type parameter with a value of PROG, and the conversation for the partner program was in Receive or Confirm state. The call might have caused information to be purged. Purging occurs when a program issued Send_error in receive state before receiving all the information sent by its partner program. No purging occurs when a program issues the call in Confirm state or in Receive state after receiving all the information sent by its partner program. The return code is normally reported to the local program on a call it issues before sending any information, depending on the call and when it is made.
17	The partner program made a Send_error call specifying the Type parameter with a value of PROG, the conversation for the partner program was in Send state, and the call truncated a logical record. Truncation occurs at the basic conversation protocol boundary when a program begins sending a logical record and then makes a Send_error call before sending the complete logical record. This return code is reported to the local program on a Receive call it issues after receiving the truncated logical record.

Table 429. APPC return codes and their meanings (continued)

Return code (Hexadecimal)	Explanation
18	The local program issued a call in which a programming error has been found in one or more parameters. Specific reasons for the return code apply to the following callable services: **ATBALC2 or ATBALLC (LU 6.2 Allocate)** • An unauthorized caller passed a nonzero TP_ID • For Sec_pgm-type security, both the user ID and password were not specified • For Sec_Pgm-type security, a user ID was specified with a blank password, or a password was specified with a blank user ID • The SYMDEST name was not found in the side information • The specified TP_ID is not associated with the address space • An unauthorized caller specified a Notify_Type of ECB **ATBCFM (LU 6.2 Allocate)** • An unauthorized caller specified a Notify_type of ECB • The Sync_Level field for the conversation was equal to sync_level_none **ATBDEAL (LU 6.2 Allocate)** • A Deallocate_type of deallocate_confirm was specified, and the Sync_Level field for the conversation was equal to sync_level_none **ATBPTR (LU 6.2 Prepare to Receive)** • A Prepare_To_Receive_Type of Prep_to_receive_sync_level was specified, and the Sync_Level field for the conversation was equal to sync_level_none **ATBSEND (LU 6.2 Send)** • The value in the 2 byte LL field was not valid • A Send_Type of Send_and_Confirm was specified, and the Sync_Level field for the conversation was equal to sync_level_none **CMINIT (CPI-C Initialize Conversation)** The SYMDEST name was not found in the side information
19	The local program issued a call in a state that was not valid for that call. The program should not examine any other returned variables associated with the call as nothing is placed in the variables. The state of the conversation remains unchanged. If the error occurs in one of the following callable services, the conversation was in send state and the program started, but the program did not finish sending a logical record: • ATBCFM (LU 6.2 Allocate) • ATBDEAL (LU 6.2 Allocate) • ATBPTR (LU 6.2 Allocate) • ATBRCVW and ATBRCVI (LU 6.2 Receive and Wait and Receive Immediate) • ATBSEND (LU 6.2 Send)
1A	A failure occurred that caused the conversation to be prematurely terminated. The condition is not temporary, and the program should not try the transaction again until the condition is corrected.
1B	A failure occurred that caused the conversation to be prematurely terminated. The condition might be temporary, and the program can try the transaction again.

Table 429. APPC return codes and their meanings (continued)

Return code (Hexadecimal)	Explanation
1C	The call issued by the local program did not run successfully. This return code is returned on the unsuccessful call. If this code is returned by the ATBRCVI (LU 6.2 Receive_Immediate) callable service, there is no data to be returned.
1E	The partner program issued a Deallocate call with a Deallocate_type of Deallocate_abend_SVC. If the partner program was in Receive state when the call was issued, information sent by the local program and not yet received by the partner program is purged. This return code is reported to the local program on a call the program issues in Send or Receive state.
1F	The partner program issued a Deallocate call with a Deallocate_type of Deallocate_abend_timer. If the partner program was in Receive state when the call was issued, information sent by the local program and not yet received by the partner program is purged. This return code is reported to the local program on a call the program issues in Send or Receive state.
20	The partner program issued a Send_error call specifying a Type parameter of SVC, the conversation for the partner program was in Send state, and the call did not truncate a logical record. This return code is returned on a Receive call. It is not returned for Send_error requests using CPI Communications.
21	The partner program issued a Send_error call specifying a Type parameter of SVC, the conversation for the partner program was in Receive, Confirm, or Sync_Point state, and the call might have caused information to be purged. This return code is normally returned to the local program on a call that the local program issues after sending some information to the partner program. However the return code can be returned on a call that the local program issues before sending any information, depending on when the call is issued. This code is not returned for Send_error requests using CPI Communications.
22	The partner program issued a Send_error call specifying a Type parameter of SVC, the conversation for the partner program was in Send state, and the call truncated a logical record. Truncation occurs when a program begins sending a logical record and then issues Send_error before sending the complete record. This return code is returned to the local program on a Receive call that the local program issues after receiving the truncated logical record. The code is not returned for Send_error requests using CPI Communications.
40	APPC/MVS is not currently active. Call the service again after APPC is available.
Other	See the *Writing Transaction Programs for APPC/MVS* and *Writing Servers for APPC/MVS* manuals.

APPC allocate services return codes

This table documents the return codes that can be returned from APPC/MVS allocate queue services in messages from the distributed queuing component if you are using APPC/MVS as your communications protocol.

Table 430. APPC allocate services return codes and their meanings

Return code (Hex)	Explanation
0	The service completed as requested.
4	The service completed, but possibly not as expected. See the reason code parameter for a description of the warning condition.
8	A user-supplied parameter was found to be in error. For example, a parameter contains characters not in the required character set. See the reason code parameter to determine which parameter is in error.
10	The service was unsuccessful. The cause is most likely a parameter error other than a syntax error, or an environmental error. For example, a syntactically valid LU name was specified, but the LU is not defined to APPC/MVS. An example of an environmental error is that the caller called the service while holding locks. See the reason code parameter for the specific cause of the error, and to determine whether the error can be corrected and the service issued again.
20	APPC/MVS service failure. Record the return and reason code, and give them to your system programmer, who should contact the appropriate IBM support personnel.
40	APPC/MVS is not currently active. Call the service again after APPC is available.
Other	See the *Writing Transaction Programs for APPC/MVS* and *Writing Servers for APPC/MVS* manuals.

APPC reason codes

This table documents the reason codes that can be returned from APPC/MVS allocate queue services in messages from the distributed queuing component if you are using APPC/MVS as your communications protocol.

Note: Some of the APPC return codes are not accompanied by a reason code; in these cases, the value in the reason code field can be ignored. See the documentation shown in "APPC/MVS return codes" on page 5214 for more information.

Table 431. APPC reason codes and their meanings

Return code (Hex)	Explanation
1	The address space issued a Register_For_Allocates call that duplicated a previous Register_For_Allocate call (that is, the values specified for TP name, local LU name, partner LU name, user ID, and profile all matched those specified on a previous call to the Register_For_Allocates service).
2	A TP name is required, but none was specified.
3	The specified TP name contains characters that are not valid
4	The specified TP name length is outside the allowable range.
5	A local LU name is required, but none was specified.
7	An asynchronous call failed because a specified parameter was found to be inaccessible.
8	The caller held one or more locks when calling the service.
0A	A transaction scheduler called the Register_For_Allocate service, which is not allowed
0B	The specified symbolic destination name can not be found in the side information data set.
0C	The specified local LU is undefined.
0D	The specified local LU is not receiving inbound allocate requests.
0E	The Register_For_Allocate service was called, but the caller is not authorized to serve the specified TP name on the specified local LU.

Table 431. APPC reason codes and their meanings (continued)

Return code (Hex)	Explanation
0F	The specified local LU is inaccessible to the caller.
10	The service failed because of an APPC failure.
11	The specified allocate queue token does not represent an allocate queue for which this address space is registered.
12	The specified notify type is not valid.
13	The specified timeout value is not valid.
14	The request was canceled while in progress. This might have been caused by a call to the Unregister_For_Allocates service, or the termination of the caller's address space.
15	A Receive_Allocate call completed, but no allocate request was available to be received.
1A	The specified event notification type is not valid.
1B	The specified event code is not supported or is not valid for this service.
1C	The netid retrieved from the side information data set does not match the local netid.
1D	The specified event code qualifier is not valid or supported.
1E	The Get_Event call completed, but no event element was available to be received.
1F	The call to the Get_Event service was interrupted because all event notification requests were canceled for this address space.
20	The call to the Get_Event service was rejected because a previous Get_Event call is currently outstanding.
21	The Get_Event call was rejected because no event notification is in effect for this address space.
22	The specified allocate queue keep time is outside the allowable range.
24	A call to the Unregister_For_Allocates service specified “unregister all” (that is, the allocate_queue_token was set to binary zeros), but this address space is not registered for any allocate queues.
25	The specified event get type is not valid.
26	The specified receive allocate type is not valid.
27	APPC/MVS cannot determine if the specified netid is valid.
29	The service failed because the supplied buffer was not large enough to contain the requested information.
Other	See the *Writing Transaction Programs for APPC/MVS* and *Writing Servers for APPC/MVS* manuals.

Transport Layer Security (TLS) return codes for z/OS

z/OS

IBM MQ for z/OS can use TLS with the various communication protocols. Use this topic to identify the error codes that can be returned by TLS.

Table 432 in this appendix documents the return codes, in decimal form, from the TLS that can be returned in messages from the distributed queuing component.

Table 433 on page 5223 in this appendix documents the return codes, in hexadecimal form, from the TLS function 'gsk_fips_state_set' that can be returned in messages from the distributed queuing component.

If the return code is not listed, or if you want more information, see z/OS Cryptographic Services System SSL Programming - SSL Function Return Codes.

Table 432. SSL return codes

Return code (decimal)	Explanation
1	Handle is not valid.
3	An internal error has occurred.
4	Insufficient storage is available
5	Handle is in the incorrect state.
6	Key label is not found.
7	No certificates available.
8	Certificate validation error.
9	Cryptographic processing error.
10	ASN processing error.
11	LDAP processing error.
12	An unexpected error has occurred.
102	Error detected while reading key database or SAF key ring.
103	Incorrect key database record format.
106	Incorrect key database password.
109	No certificate authority certificates.
201	No key database password supplied.
202	Error detected while opening the key database.
203	Unable to generate temporary key pair
204	Key database password is expired.
302	Connection is active.
401	Certificate is expired or is not valid yet.
402	No TLS cipher specifications.
403	No certificate received from partner.
405	Certificate format is not supported.
406	Error while reading or writing data.
407	Key label does not exist.
408	Key database password is not correct.
410	TLS message format is incorrect.

Table 432. SSL return codes (continued)

Return code (decimal)	Explanation
411	Message authentication code is incorrect.
412	TLS protocol or certificate type is not supported.
413	Certificate signature is incorrect.
414	Certificate is not valid.
415	TLS protocol violation.
416	Permission denied.
417	Self-signed certificate cannot be validated.
420	Socket closed by remote partner.
421	SSL V2 cipher is not valid.
422	SSL V3 cipher is not valid.
427	LDAP is not available.
428	Key entry does not contain a private key.
429	SSL V2 header is not valid.
431	Certificate is revoked.
432	Session renegotiation is not allowed.
433	Key exceeds allowable export size.
434	Certificate key is not compatible with cipher suite.
435	certificate authority is unknown.
436	Certificate revocation list cannot be found.
437	Connection closed.
438	Internal error reported by remote partner.
439	Unknown alert received from remote partner.
440	Incorrect key usage.
442	Multiple certificates exist for label.
443	Multiple keys are marked as the default.
444	Error encountered generaing random bytes.
445	Key database is not a FIPS mode database.
446	TLS extension mismatch has been encountered.
447	Required TLS extension has been rejected.
448	Requested server name is not recognized.
449	Unsupported fragment length was received.
450	TLS extension length field is not valid.
451	Elliptic Curve is not supported.
452	EC Parameters not supplied.
453	Signature not supplied.
454	Elliptic Curve parameters are not valid.
455	ICSF services are not available.
456	ICSF callable services returned a error.
457	ICSF PKCS#11 not operating in FIPS mode.

Table 432. SSL return codes (continued)

Return code (decimal)	Explanation
458	The SSL V3 expanded cipher is not valid.
459	Elliptic Curve is not supported in FIPS mode.
460	Required TLS Renegotiation Indication not received.
461	EC domain parameter format is not supported.
462	Elliptic Curve point format is not supported.
463	Cryptographic hardware does not support service or algorithmn.
464	Elliptic curve list is not valid.
466	Signature algorithm pairs list is not valid.
467	Signature algorithm not in signature algorithm pairs list.
468	Certificate key algorithm not in signature algorithm pairs list.
501	Buffer size is not valid.
502	Socket request would block.
503	Socket read request would block.
504	Socket write request would block.
505	Record overflow.
601	Protocol is not TLS V1.0, TLS V1.1, or TLS V1.2.
602	Function identifier is not valid.
603	Specified function enumeration is not valid.
604	Send sequence number is near maxumum value.
701	Attribute identifier is not valid.
702	Attribute length is not valid.
703	Enumeration is not valid.
704	Session identifier cache callback is not valid.
705	Numeric value is not valid.
706	Attribute parameter is not valid.
707	TLS extension type is not valid.
708	Supplied TLS extension data is not valid.

Table 433. SSL return codes from 'gsk_fips_state_set'

Return code (hexadecimal)	Explanation
03353050	The enumeration value is not valid or it cannot be set due to the current state.
0335306B	The System SSL FIPS mode state cannot be changed to FIPS mode because it is currently not in FIPS mode.
0335306C	The request to execute in FIPS mode failed because the Cryptographic Services Security Level 3 FMID is not installed so that the required System SSL DLLs could not be loaded.
03353067	The power on known answer tests failed. FIPS mode cannot be set.

Distributed queuing message codes

z/OS

Distributed queuing is one of the components of IBM MQ for z/OS. Use this topic to interpret the message codes issued by the distributed queuing component.

Distributed queuing message codes are in the form *s* 0009 *nnn* (in hexadecimal). The error they identify is described in detail by error message CSQX *nnn*, although there are some exceptions. The following table shows the full correspondence. Distributed queuing message codes are used in some error messages, and in the event data for the MQRC_CHANNEL_STOPPED event. The event data also contains message inserts. The meanings of the inserts depend on the message code, and are shown in the following table, in the form in which they are given in the message explanation. Where no meaning is shown, the insert is not relevant to the message code, and the value set in the event message is unpredictable.

Note: *trptype* can be shown in various forms:

Message insert
Event data

TCP TCP/IP

LU62 LU 6.2, APPC, CPI-C

Message code (*nnn*)	Message number	Integer insert 1	Integer insert 2	Character insert 1	Character insert 2	Character insert 3
001	CSQX501I			channel-name		
181	CSQX181E	response		exit-name		
182	CSQX182E	response		exit-name		
184	CSQX184E	address		exit-name		
189	CSQX189E	length		exit-name		
196	CSQX196E	data-length	agent-buffer length	exit-name		
197	CSQX197E	data-length	exit-buffer length	exit-name		
201	CSQX201E	return-code		conn-id	trptype	
202	CSQX202E	return-code		conn-id	trptype	
203	CSQX203E	return-code		conn-id	trptype	
204	CSQX204E	return-code		conn-id	trptype	
205	CSQX205E	return-code		conn-id	trptype	
206	CSQX206E	return-code		conn-id	trptype	
207	CSQX207E			conn-id	trptype	
208	CSQX208E	return-code		conn-id	trptype	
209	CSQX209E			conn-id	trptype	
211	CSQX027E					
212	CSQX212E	return-code				
213	CSQX213E	return-code			trptype	
237	CSQX203E	return-code	reason	conn-id	trptype	
238	CSQX213E	return-code	reason		trptype	
403	CSQX403I			channel-name	exit-name	

Message code (*nnn*)	Message number	Integer insert 1	Integer insert 2	Character insert 1	Character insert 2	Character insert 3
496	CSQX496I			channel-name		
498	CSQX498E	fieldvalue		channel-name		
506	CSQX506E			channel-name		
510	CSQX037E	mqrc			name	
511	CSQX038E	mqrc			name	
514	CSQX514E			channel-name		
519	CSQX519E			channel-name		
520	CSQX520E			channel-name		
525	CSQX525E			channel-name		
526	CSQX526E	msg-seqno	exp-seqno	channel-name		
527	CSQX527E			channel-name		
528	CSQX528I			channel-name		
533	CSQX533I			channel-name		
534	CSQX534E			channel-name		
536	CSQX536I			channel-name	exit-name	
540	CSQX540E	mqrc		commit identifier which includes channel-name		
542	the queue manager is stopping (no corresponding error message)					
544	see integer insert 1	1 - see message CSQX548E 2 - see message CSQX544E		channel-name		
545	CSQX545I			channel-name		
546	code 00E70546					
558	CSQX558E			channel-name		
565	CSQX565E			channel-name	qmgr-name	
569	CSQX569E			channel-name		
570	CSQX570E			channel-name		
572	CSQX572E			channel-name		
573	CSQX573E			channel-name		
574	CSQX574I			channel-name		
575	CSQX575E					
613	CSQX613E			channel-name		
620	CSQX620E	return-code		SSL-function		
631	CSQX631E			channel-name	local cipher spec	remote cipher spec
633	CSQX633E			channel-name		

Message code (*nnn*)	Message number	Integer insert 1	Integer insert 2	Character insert 1	Character insert 2	Character insert 3
634	CSQX634E			channel-name		
635	CSQX635E			channel-name		cipher spec
636	CSQX636E			channel-name	dist-name	
637	CSQX637E			channel-name		
638	CSQX638E			channel-name		
639	CSQX639E			channel-name		
640	CSQX640E			channel-name		key-name
641	CSQX641E			channel-name		
642	CSQX642E			channel-name		
643	CSQX643E			channel-name		
644	CSQX644E			channel-name		
999	CSQX599E			channel-name		

Queued Publish/Subscribe message codes

z/OS

Queued Publish/Subscribe is a component of IBM MQ for z/OS. Use this topic to interpret the message codes issued by the queued Publish/Subscribe component.

Queued publish/subscribe message codes are in the form 5 *nnn* (in hexadecimal), and the error they identify is described in detail by error message CSQT *nnn*, although there are some exceptions. The following table shows the full correspondence. Queued publish/subscribe message codes are used in some error messages.

Message Code (*nnn*)	Message Number	Description
800	No equivalent message	Unexpected error
87F	CSQX036E	Failed

Messages from other products

z/OS

Software products on the z/OS platform issue messages and each product uses a unique identifier. Use this topic to identify the different z/OS products using the unique identifier.

The following table shows the message prefixes for other products that you might receive while using IBM MQ for z/OS.

Table 434. Message prefixes

Prefix	Component	Procedure
AMQ	IBM MQ (not z/OS)	Consult Reason codes
ATB	APPC	Consult *MVS System Messages*
ATR	Resource recovery services	Consult *MVS System Messages*
CBC	C/C++	Consult *C/MVS User's Guide*
CEE	Language Environment	Consult *Language Environment for z/OS Debugging Guide and Runtime Messages*

Table 434. Message prefixes (continued)

Prefix	Component	Procedure
CSQ	IBM MQ for z/OS	Consult this documentation
CSV	Contents supervision	Consult *MVS System Messages*
DFH	CICS	Consult *CICS Messages and Codes*
DFS	IMS	Consult *IMS Messages and Codes*
DSN	Db2	Consult *Db2 Messages and Codes*
EDC	Language Environment	Consult *Language Environment for z/OS Debugging Guide and Runtime Messages*
EZA, EZB, EZY	TCP/IP	Consult *TCP/IP for MVS Messages and Codes*
IBM	Language Environment	Consult *Language Environment for z/OS Debugging Guide and Runtime Messages*
ICH	RACF	Consult *RACF Messages and Codes*
IDC	Access method services	Consult *MVS System Messages*
IEA	z/OS system services	Consult *MVS System Messages*
IEC	Data management services	Consult *MVS System Messages*
IEE,IEF	z/OS system services	Consult *MVS System Messages*
IKJ	TSO	Consult *MVS System Messages*
IST	VTAM	Consult *VTAM Messages and Codes*
IWM	z/OS workload management services	Consult *MVS System Messages*
IXC	Cross-system coupling facility (XCF)	Consult *MVS System Messages*
IXL	Cross-system extended services (XES)	Consult *MVS System Messages*

Index

Special characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

Programming interface information, if provided, is intended to help you create application software for use with this program.

This book contains information on intended programming interfaces that allow the customer to write programs to obtain the services of WebSphere MQ.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Important: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks

IBM, the IBM logo, ibm.com®, are trademarks of IBM Corporation, registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" www.ibm.com/legal/copytrade.shtml. Other product and service names might be trademarks of IBM or other companies.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

This product includes software developed by the Eclipse Project (http://www.eclipse.org/).

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Sending your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or give us any other feedback that you might have.

Use one of the following methods to send us your comments:

- Send an email to ibmkc@us.ibm.com
- Use the form on the web here: www.ibm.com/software/data/rcf/

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

Include the following information:

- Your name and address
- Your email address
- Your telephone or fax number
- The publication title and order number
- The topic and page number related to your comment
- The text of your comment

IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you submit.

Thank you for your participation.